Journal of King Saud University - Computer and Information Sciences 36 (2024) 102082

HOSTED BY

Contents lists available at ScienceDirect

Journal of King Saud University - Computer and Information Sciences

journal homepage: www.sciencedirect.com

Full length article

Energy-aware task scheduling for streaming applications on NoC-based MPSoCs

Suhaimi Abd Ishak [a,*], Hui Wu [b], Umair Ullah Tariq [c]

[a] Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia, 86400 Parit Raja, Johor, Malaysia
[b] School of Computer Science and Engineering, The University of New South Wales, 2052 Sydney, Australia
[c] College of ICT, School of Engineering and Technology, Central Queensland University, 4701 Rockhampton, Queensland, Australia

ARTICLE INFO

Keywords:
Network-on-chip
MPSoC
Communication contention
DVFS
Energy consumption
Memory constraint

ABSTRACT

Streaming applications are being extensively run on portable embedded systems, which are battery-operated and with limited memory. Thus, minimizing the total energy consumption of such a system is important. We investigate the problem of offline scheduling for streaming applications composed of non-preemptible periodic dependent tasks on homogeneous Network-on-Chip (NoC)-based Multiprocessor System-on-Chip (MPSoCs) such that their total energy consumption is minimized under memory constraints. We propose a novel unified approach that integrates task-level software pipelining with Dynamic Voltage and Frequency Scaling (DVFS) to solve the problem. Our approach is supported by a set of novel techniques, which include constructing an initial schedule based on a list scheduling where the priority of each task is its approximate successor-tree-consistent deadline such that the workload across all the processors is balanced, a retiming heuristic to transform intra-period dependencies into inter-period dependencies for enhancing parallelism, assigning an optimal discrete frequency for each task and each message using a Non-Linear Programming (NLP)-based algorithm and an Integer-Linear Programming (ILP)-based algorithm, and an incremental approach to reduce the memory usage of the retimed schedule in case of memory size violations. Using a set of real and synthetic benchmarks, we have implemented and compared our unified approach with two state-of-the-art approaches, RDAG+GeneS (Wang et al., 2011) , and JCCTS (Wang et al., 2013a). Experimental results show that our approach's maximum, average, and minimum improvements over RDAG+GeneS (Wang et al., 2011) are 31.72%, 14.05%, and 7.00%, respectively. Our approach's maximum, average, and minimum improvement over JCCTS (Wang et al., 2013a) are 35.58%, 17.04%, and 8.21%, respectively.

1. Introduction

Streaming applications have become increasingly important and widespread, particularly on portable embedded systems that rely on batteries and with limited memory. In real-time systems, a streaming application can be modeled as periodic tasks with precedence and deadline constraints. Within a certain period, a streaming application shall entertain a stream of data, such as decoding a frame in video streaming, and this happens continuously until finished. Streaming applications are mostly compute-intensive and could be exploited to increase processing parallelism. Consequently, they are very suitable to be executed on MPSoCs. One example is the video surveillance system used in industries for monitoring and analyzing live video feeds. A homogeneous NoC-based MPSoC can be employed to process and analyze video streams from multiple cameras simultaneously. Tasks include object detection, tracking, and event recognition, all requiring real-time processing for timely responses.

An MPSoC is a system-on-a-chip with multiple processors. Examples include TI DaVinci processors, ARM ARM11 MPCore, and Intel Atom processors. Network-on-Chip (NoC) is a communication subsystem on an integrated circuit. NoC not only provides higher bandwidths but is also scalable. Therefore, NoC-based MPSoCs are being increasingly used in embedded systems.

* Corresponding author.
E-mail addresses: suhaimiabd@uthm.edu.my (S. Abd Ishak), huiw@unsw.edu.au (H. Wu), u.tariq@cqu.edu.au (U.U. Tariq).
Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

https://doi.org/10.1016/j.jksuci.2024.102082
Received 30 December 2023; Received in revised form 15 May 2024; Accepted 29 May 2024
Available online 31 May 2024

Energy consumption is a crucial design concern for modern embedded systems as escalating power density and, hence, temperature variation continue to accelerate wear-out, increasing device defects. Thus, it is crucial to design an energy-efficient embedded system that not only reduces the operating cost but also improves the system's reliability and lifetime. Dynamic Voltage and Frequency Scaling (DVFS) is an efficient technique to reduce energy consumption. DVFS alters operational frequencies and thus supply voltages based on computation demands.

In general, the performance of applications can be improved by exploiting their inherent horizontal or vertical parallelism (Henkel and Parameswaran, 2007). Horizontal parallelism occurs when multiple independent tasks can be executed simultaneously using multiprocessors. Vertical parallelism occurs when different stages of a sequence of operations can be overlapped and can be utilized through pipelining, which can be done with retiming techniques. Furthermore, with improved performance, static slacks might be derived, and thus, specific tasks can be executed with lower operating frequencies to reduce the total energy consumption. However, pipelining may cause memory overhead. This is because extra memory space is required to keep the data across multiple periods i.e., the sender and receiver tasks are executed on different periods. Thus, it is vital to construct energy-efficient schedules considering the system memory constraints.

In this paper, we study the following energy-aware task scheduling problem. Given a set of non-preemptible periodic tasks with precedence and deadline constraints, construct a feasible schedule on a homogeneous NoC-based MPSoC with discrete frequencies such that the total energy consumption made up of computation and communication energy is minimized. We make the following significant contributions.

- We propose a novel unified energy-aware scheduling approach that integrates task-level software pipelining with DVFS under memory constraints. To the best of our knowledge, our study is the first one that investigates the problem of scheduling a set of periodic dependent tasks on NoC-based MPSoCs such that the total energy consumption is minimized considering the memory constraints of the system.
- We present a novel task mapping and scheduling approach to construct an initial schedule under maximum frequencies, which further reduces the total energy consumption when combined with retiming and DVFS. We employ a list scheduling where the priority of each task and each message is its approximate successor-tree-consistent deadline.
- We introduce a heuristic to compute the memory usage of the schedule. We correlate the problem to compute the memory usage of a schedule into the problem of Maximum Weight Clique, which is NP-hard (Jiang et al., 2017).
- We have implemented and compared our approach with two state-of-the-art approaches, RDAG+GeneS (Wang et al., 2011) and JCCTS (Wang et al., 2013a) using a set of real and synthetic benchmarks. Experimental results show that the maximum improvement, the average improvement, and the minimum improvement of our approach over the RDAG+GeneS approach (Wang et al., 2011) are 31.72%, 14.05%, and 7.00%, respectively. The maximum improvement, average improvement, and minimum improvement of our approach over the JCCTS approach (Wang et al., 2013a) are 35.58%, 17.04%, and 8.21%, respectively.

The rest of this paper is organized as follows. Section 2 surveys the related work. Section 3 describes system model and key definitions. Section 4 provides some motivational examples. Section 5 proposes our unified scheduling approach. Section 6 presents the experimental results. Section 7 concludes this paper.

2. Related work

Various approaches have been proposed to minimize the energy consumption of multiprocessor systems. In this section, we survey the previous energy-aware task scheduling approaches for periodic dependent task sets with timing constraints on either homogeneous or heterogeneous distributed multiprocessor platforms from 2000 until recently in chronological order.

Luo and Jha (2000) consider the problem of reducing dynamic energy for scheduling multiple task graphs consisting of periodic and aperiodic tasks with individual deadlines. They assume homogeneously distributed multiprocessors with a continuous frequency model. The key idea of their approach is to distribute available static slacks proportionately among all tasks. However, this approach does not necessarily lead to the best energy savings since every task contributes differently to the overall energy savings.

Schmitz and Al-Hashimi (2001) study the problem of minimizing the total dynamic energy consumption for periodic dependent tasks with individual deadlines on heterogeneous distributed multiprocessors with continuous frequencies. They propose a PV-DVS algorithm that considers power variations among tasks. It distributes static slacks to tasks based on their energy gradient.

Zhang et al. (2002) present a two-phase framework that integrates task assignment, ordering, and voltage selection to minimize energy consumption of real-time dependent tasks executing on a given number of variable voltage processors. They use the Earliest Deadline First (EDF) algorithm to schedule all tasks during the first phase and Integer Programming for assigning a voltage for each task in the second phase.

Schmitz et al. (2002) extend their work in Schmitz and Al-Hashimi (2001). They present an energy-efficient genetic list scheduling algorithm (EE-GLSA) that constructs and evaluates different schedules during an iterative optimization. Each schedule is assessed using a fitness function based on its total dynamic energy consumption and time penalty. However, they consider a continuous frequency model.

Luo and Jha (2002) study the problem of minimizing the total dynamic energy consumption of multi-rate periodic task sets, i.e., multiple task graphs with different periods and aperiodic task sets with individual deadlines on heterogeneous multiprocessors embedded system with continuous frequencies. They propose a heuristic based on critical path analysis and task execution order refinement. Firstly, the heuristic identifies the critical path from an initial schedule and computes its speed reduction ratio. Then, it refines the task execution order if further energy reduction is possible.

Luo and Jha (2003) address the problem of minimizing the total dynamic energy consumption on heterogeneous multiprocessors with continuous voltage connected via a shared bus. They assume multi-rate periodic tasks. Firstly, they assume a given task assignment. Next, they construct a schedule using a list scheduling based on the critical path. Then, the algorithm allocates slack to a task with the highest energy gradient iteratively until no available slacks are left.

Luo and Jha (2007) investigate the problem of voltage scheduling for a set of dependent periodic tasks with individual deadlines on heterogeneous distributed multiprocessor embedded systems with discrete frequencies. All the processors are connected through a shared bus. They propose a list scheduling approach based on the critical path. In this study, they assume that task and communication assignment is given as an initial schedule. Next, they employ an execution-order optimization algorithm using Simulated Annealing (SA) on the initial schedule to reorder tasks. The reordering may introduce extra slacks, which can be utilized. Then, they perform power-profile and timing-constraint-driven slack allocation on the schedule to minimize the total dynamic energy consumption.

Xu et al. (2007) study the problem to minimize the total computation energy consumption (dynamic energy plus static energy) while satisfying both throughput and response time with pipelining. They consider stream applications composed of dependent periodic tasks

with a common deadline executed on distributed homogeneous Chip Multiprocessors (CMPs) with discrete frequencies. They present two integrated task mapping and frequency assignment approaches: Scheduling1D for linear task graphs and Scheduling2D for general task graphs. The Scheduling2D transforms task graphs into a two-dimensional structure, X-load and Y-load. For X-load, they use parallel processing to reduce energy consumption while satisfying the deadline constraint. As for Y-load, they employ pipelining to reduce energy consumption while satisfying the throughput requirement.

Watanabe et al. (2007) study the problem of minimizing the total energy consumption (computation plus communication) for periodic dependent tasks with latency and throughput constraints on homogeneous Globally Asynchronous Locally Synchronous (GALS) MPSoCs with discrete frequencies. They propose a pipelined scheduling approach using Mixed Integer Linear Programming (MILP) to find an optimal solution. In addition, they propose an approach based on Simulated Annealing (SA) to find a near-optimal solution but with a quicker running time.

Kumar and Manimaran (2007) investigate the problem of assigning frequency and modulation levels to tasks to minimize the total energy consumption (computation and communication) of periodic dependent tasks on heterogeneous multiprocessors connected via a wireless network. They propose a slack allocation scheme for tasks and messages based on their normalized energy gain. The heuristic incrementally allocates slack to each task. In each iteration, slack is given to the task (message) with the highest normalized energy gain.

Liu et al. (2008) study the problem of joint energy and performance optimization. They focus on periodic dependent tasks on homogeneous distributed multiprocessors. They present an approach that combines task-level software pipelining with Dynamic Voltage Scaling (DVS). Firstly, they use a retiming technique to transform all intra-period dependencies into inter-period dependencies. Secondly, they use a heuristic named SpringS combining DVFS to perform task scheduling and voltage selection such that the total energy consumption is minimized. Results show that their approach can achieve better energy savings and better schedulability than Luo and Jha (2007) approach, which does not employ task-level software pipelining.

Yang et al. (2009) work on the problem of partitioning a set of periodic independent tasks on a heterogeneous distributed multiprocessor such that the total computation energy consumption is minimized. They propose two partition scheduling approaches. First, they employ a dynamic programming approach to compute the optimal partition, but with exponential time complexity. Second, they present a polynomial-time approximation algorithm to compute a near-optimal solution.

Chen et al. (2011) investigate a combinatorial optimization problem consisting of task mapping and energy objectives. For the task mapping objective, they attempt to find a solution such that the cumulative utilization of any processors does not exceed a utilization bound. In contrast, the energy objective strives to minimize the cumulative energy consumption of all assigned tasks. They focus on periodic independent tasks with timing constraints to execute on a heterogeneous multiprocessor with continuous frequencies. They propose a meta-heuristic based on Ant Colony Optimization (ACO) to solve the problem.

Wang et al. (2011) investigate the energy-aware task scheduling problem for periodic dependent tasks on homogeneous distributed multiprocessors. They present a two-phase approach that combines task-level software pipelining with DVFS named RDAG+GeneS. Firstly, they use a retiming technique to transform all intra-period dependencies of the task graphs into inter-period dependencies. Secondly, they employ GA to find the best task mapping and frequency assignment for all the tasks. Results show that their approach can significantly reduce the total energy consumption compared to the approaches in Liu et al. (2008) and Zhang et al. (2002). In addition, their approach performs better in task schedulability than the two approaches. However, their approach requires more memory than the approach in Zhang et al. (2002).

Qiu et al. (2012) propose a three-phase scheme to minimize the total dynamic energy consumption for periodic dependent tasks with individual deadlines on homogeneous CMPs with discrete frequency levels. Firstly, they construct an initial schedule under maximum frequencies using the Min–Min algorithm (Li et al., 2011). Secondly, it computes an optimal frequency for each task based on the extendable factor of its path. Thirdly, they employ a naive approach to assign a discrete frequency to each task. Based on the optimal frequencies for all the tasks computed during the second phase, they select the higher discrete frequency to avoid deadline violations.

Huang et al. (2013) present an energy-efficient scheduling approach for streaming applications composed of periodic dependent tasks with deadline constraints on heterogeneous multiprocessors. Initially, they assume a given task assignment and ordering. Then, they formulate the discrete frequency assignment for the systems with local DVFS switches using Mixed Integer Linear Programming (MILP) and a three-phase heuristic combining the MILP for the systems with a global DVFS switch to minimize the total dynamic energy consumption.

Wang et al. (2013a) investigate the problem of removing the inter-processor communication overhead. They focus on periodic dependent tasks on homogeneous multiprocessors connected via a shared bus. They present an algorithm named JCCTS, which includes a retiming approach by using Integer Linear Programming (ILP). Firstly, they do a schedulability analysis to compute the minimum and maximum retiming value for each task. Then, the ILP-based algorithm incorporates the bounds as constraints to minimize the maximum retiming value. Their approach can be extended to minimize the total dynamic energy consumption by applying other DVFS techniques. The slacks caused by the inter-core communication overhead can be fully utilized to minimize the total computation energy consumption.

Singh et al. (2013) consider concurrent multimedia applications composed of periodic, cyclic, and multi-rate dependencies between tasks modeled as Synchronous Dataflow Graph (SDFG). The target platform is heterogeneous NoC-based multiprocessors. They propose a design-time strategy that generates a set of task mappings with different resource requirements, throughput, and energy consumption (dynamic energy plus communication energy) to handle dynamism during runtime. A runtime algorithm then selects a mapping according to its requirements with the minimum energy consumption.

Liu et al. (2015) investigate the problem of minimizing the total energy consumption while guaranteeing latency and throughput constraints. They consider streaming applications composed of periodic dependent tasks with deadline constraints on a cluster heterogeneous MPSoC. Each cluster consists of either identical performance-efficient processors or identical energy-efficient processors. They propose a combined partitioned scheduling and global scheduling in which tasks are statically assigned to a cluster and globally scheduled within a cluster. Tasks are assigned to clusters based on the First-Fit-Decreasing (FFD) algorithm to form an initial schedule. Then, they remap tasks to the unused clusters to balance the workload to further scale down the clusters' operating frequencies, thus reducing the total energy consumption.

Gammoudi et al. (2018) propose a deterministic strategy to guarantee that the energy is sufficient to run all the tasks without missing their deadlines by the next battery recharge. However, they consider reconfigurable applications composed of a set of periodic tasks that can be added or removed on homogeneous NoC-based multicore processors.

Ali et al. (2019) propose a two-phase approach named ETSH to reduce the total energy consumption for running a set of periodic dependent tasks on heterogeneous NoC-based multiprocessors. Firstly, the approach constructs a retimed schedule using R-DAG (Wang et al., 2011) algorithm. Then, the energy of the retimed schedule is optimized using a Genetic Algorithm (GA). Results show that ETSH achieves an average of at least 20% energy savings compared to a previous approach (Li and Wu, 2016).

Tariq et al. (2020) propose an EMRCTG approach to minimize the total energy consumption of homogeneous NoC-based multiprocessors under memory constraints. Results show that the EMRCTG approach outperforms the previous RDAG+GeneS (Wang et al., 2011) and JC-CTS (Wang et al., 2013a) approaches. However, EMRCTG focuses on real-time applications with conditional tasks.

Tariq et al. (2021) propose an energy-aware scheduling approach for periodic conditional task set on Voltage Frequency Island (VFI)-based heterogeneous MPSoCs. The approach consists of a retiming algorithm and a Non-Linear Programming (NLP)-based algorithm. Results show that the approach achieves more energy efficiency than the previous approaches: CA-TMES-Search and CA-TMES-Quick in Han et al. (2014).

Roeder et al. (2021) devise an energy-aware scheduling approach for dependent tasks on heterogeneous multiprocessor systems. Their approach is based on Forward List Scheduling (FLS). Firstly, the FLS orders all the tasks using Breadth-First Search (BFS) and Depth-First Search (DFS). Next, it assigns each task to a processor and computes its total dynamic and static energy consumption. The greedy algorithm selects the best mapping and version with the minimum energy consumption.

Zhang and Chen (2022) give an overview of energy-aware scheduling for mixed-criticality systems composed of periodic tasks considering reliability factors. They defined the reliability of each task as its probability of completing its execution within the deadline.

Mo et al. (2022) work on periodic tasks on NoC-based homogeneous multicore processors. They formulate a Mixed Integer Linear Programming (MILP) approach to schedule each task and assign an optimal frequency to each task, with task duplication and communication path selection to reduce energy consumption. However, they do not consider memory constraints.

Chen et al. (2022) work on the energy-aware scheduling problem of dependent tasks in heterogeneous multiprocessor systems to minimize the schedule length under energy constraints. They propose an approach named LESA consisting of three phases. Firstly, LESA assigns a priority for each task using (Topcuoglu et al., 2002) approach. Secondly, it computes an energy limitation value for each task. Thirdly, each task is assigned to a processor that can give the earliest finish time while satisfying the preallocated energy limitation.

Roy et al. (2022) focuses on multiple independent periodic task sets on bus-based heterogeneous multiprocessors. They propose a three-phase heuristic named SAFLA to minimize the dynamic energy consumption of the system. Firstly, it computes an initial schedule of all the tasks with the highest frequencies. Secondly, it assigns each task to a processor and each message to the bus according to its earliest execution start time. Lastly, it assigns each task with an appropriate operating frequency recursively.

Our approach differs from all the previous approaches in three major aspects. First, we correlate a workload balance mapping that is based on an approximate successor-tree-consistent deadline with a retiming technique. Second, our approach handles NoC and takes link contentions into account. Third, our approach collectively optimizes the frequencies of processors and NoC links to minimize the total energy consumption of the MPSoC.

3. Problem, definitions and models

The problem we investigate is described as follows. Given a set of n non-preemptible periodic tasks $T = \{t_1, \ldots, t_n\}$ subject to precedence and deadline constraints, find a feasible schedule with a discrete voltage and frequency assignment to each task and each message on a NoC-based MPSoC such that the total energy consumption of all the tasks and messages is minimized. A feasible schedule is a schedule that satisfies all the constraints.

3.1. System model

The target MPSoC consists of m DVFS-enabled identical processors $P = \{p_1, p_2, \ldots, p_m\}$ interconnected via a 2-dimensional mesh NoC such as in Fig. 1(b). Each tile has a coordinate and is composed of a processor (P), local memory (M), and a network interface (NI). All the processors are connected through routers (R). We represent the platform with an undirected graph $A = (P, L, R)$. Each node in P denotes a processor, each edge in L denotes the communication link between two processors, and each edge weight in R denotes the corresponding communication link transmission rate. Each communication link is DVFS-enabled. All the routers are identical, and all the communication links are identical. Each processor has its own local memory and can run on a set $\{(v_{k,1}, f_{k,1}), \ldots, (v_{k,n_k}, f_{k,n_k}) : v_{k,1} < \cdots < v_{k,n_k}, f_{k,1} < \cdots < f_{k,n_k}\}$ of n_k discrete voltage and frequency levels. Each communication link can operate on $\{(v_1, f_1), \cdots, (v_p, f_p) : v_1 < \cdots < v_p, f_1 < \cdots < f_p\}$ of p discrete voltage and frequency levels. We assume XY routing, which is based on Cartesian coordinates, to route a message from the source router to the destination router. The length $h_{s,d}$ of the routing path between processor p_s and processor p_d, namely, the Manhattan distance, is computed below.

$$h_{s,d} = |x_d - x_s| + |y_d - y_s| \tag{1}$$

where (x_s, y_s) is the coordinate of p_s while (x_d, y_d) is the coordinate of p_d.

The target streaming application is represented by a weighted Directed Acyclic Graph (DAG), $G = (T, E, W, C, \rho)$, where each node in T denotes a task, each edge in $E \subseteq T \times T$ denotes the precedence between two tasks, and each node weight in W denotes the worst-case execution time (WCET) in cycles of the corresponding task. Each edge weight in C denotes the message size in unit data to be transferred between two tasks. ρ represents the period between two invocations of the DAG, and we assume that the relative deadline equals the period. Each task $t_i \in T$ has a preassigned deadline d_i. We assume that all the tasks are non-preemptible.

A task t_i sends its data to a task t_j via a message $M_{i,j}$. However, if t_i and t_j are on the same processor, no message is required. We assume that all the communication links on the routing path for sending the data of t_i to t_j have the same frequency which is called the frequency of $M_{i,j}$. The communication time for transferring one unit of data depends on the transmission rate $b_{s,d} = 1/(\lambda f_{s,d})$ of the communication link between p_s and p_d, where λ and $f_{s,d}$ are the link data width and link frequency, respectively. Thus, if a task t_i on p_s sends $c_{i,j}$ units of data to another task t_j on $p_d, (p_s \neq p_d)$, the time it takes to transfer the data is computed as follows.

$$\varsigma_{i,j} = c_{i,j} b_{s,d} \tag{2}$$

where $c_{i,j}$ is the message size.

A data buffer is needed for each edge $(t_i, t_j) \in E$. It keeps the data from when it is produced by t_i until the time point when t_j is finished.

3.2. Power model

The total power consumption of a processor consists of dynamic power due to switching activity and static power due to leakage. The dynamic power P_{dyn} is formulated as follows (Teng et al., 2022).

$$P_{dyn} = C_{eff} v_s^2 f \tag{3}$$

where C_{eff} is the average switched capacitance, v_s is the supply voltage and f is the operating frequency. We assume C_{eff} is a constant for all the tasks. The static power P_{sta} is as follows (Teng et al., 2022).

$$P_{sta} = L_g(v_s K_3 e^{K_4 v_s} e^{K_5 v_{bs}} + |v_{bs}| I_j) \tag{4}$$

where L_g is the number of logic gates in the circuit, K_3, K_4 and K_5 are parameters for a specific processor technology, and v_{bs} and I_j are

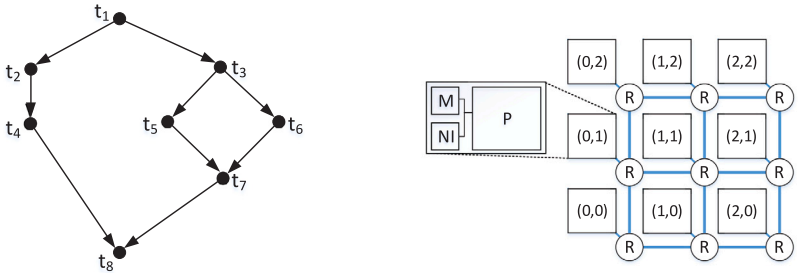

Fig. 1. Examples of (a) an application task graph, (b) a 3-by-3 mesh NoC architecture.

the body-bias voltage and body junction leakage current, respectively. Thus, we can compute the total power as follows:

$$P_{tot} = P_{dyn} + P_{sta} \tag{5}$$

The relation between supply voltage, v_s and frequency, f is given as follows (Andrei et al., 2007).

$$f = ((1 + K_1)v_s + K_2 v_{bs} - v_{th1})^{\alpha}/(L_d K_6) \tag{6}$$

where K_1, K_2, K_6 and α are the processor constants, v_{th1} is the threshold voltage, and L_d is the logic depth.

Notice that both the dynamic and total power functions are convex. Under the total power model, there is an optimal minimum processor speed f^{crit} such that the processor total energy increases as the processor frequency lower than f^{crit} further decreases (Teng et al., 2022).

3.3. Successor-tree-consistent deadline

Let $pred(t_i)$ be a set of all the predecessors of a task t_i and $succ(t_i)$ be a set of all the successors of a task t_i.

Definition 1. Given a task graph $G = (T, E, W, C, \rho)$ and a task $t_i \in T$, the successor-tree of t_i is a weighted tree ST(G, t_i)= (T', E', W', C'), where $T' = \{t_i\} \bigcup \text{succ}(t_i)$, $E' = \{(t_i, t_j) : t_j \in \text{succ}(t_i)\}$, $W' = W$ and $C' = \{c'_{i,j}$: if t_j is an immediate successor of t_i, $c'_{i,j} = c_{i,j}$; otherwise, $c'_{i,j} = 0\}$

Definition 2. Given a problem instance P, the successor-tree-consistent deadline of a task t_i, denoted by d'_i, is recursively defined as follows. If t_i is a sink where t_i does not have any successors, d'_i is equal to its preassigned deadline d_i; otherwise, d'_i is the upper bound on the latest completion time of t_i in any feasible schedule for the relaxed problem instance $P(t_i)$: a set $T' = \{t_i\} \bigcup \text{succ}(t_i)$ of tasks with precedence constraints in the form of the successor-tree of t_i. Formally, $d'_i = \min\{d_i, \max\{\sigma_j(t_i) + w_i\} : \sigma_j$ is a feasible schedule for $P(t_i)\}$.

We compute the approximate successor-tree-consistent deadline of each task as the problem of constructing a schedule on multiple processors with minimum makespan is NP-complete (Chu et al., 2018). We use the approximate successor-tree-consistent deadline of each task as its priority when assigning each task to a processor.

3.4. Retiming

Retiming involves shifting a parent task a few periods earlier than its child node, ensuring that the necessary data from its parent node is accessible at the onset of the period. Consequently, the child task can commence execution before the completion of its parent task. Leiserson and Saxe (1991) pioneered retiming to decrease the cycle period of synchronous circuits, while Wang et al. (2013b) has expanded the application of retiming to scheduling tasks represented by a DAG model on MPSoCs.

Definition 3. Given a task graph $G = (T, E, W, C, \rho)$, retiming is a function that maps each task, $t_i \in T$ and each message, $M_{i,j} : (t_i, t_j) \in E$ to an integer R_i and $R_{i,j}$, respectively, where R_i and $R_{i,j}$ are the number of periods of t_i and $M_{i,j}$ reschedule in prologue. Retiming t_i and $M_{i,j}$ once, if legal, reschedule one period of t_i and $M_{i,j}$ into the prologue.

From a program perspective, retiming reorganizes the loop body so that intra-period data dependencies become inter-period data dependencies. The retiming function remains valid if it does not reference data from future periods. The definition of a valid retiming function is as follows.

Definition 4. Given a task graph $G = (T, E, W, C, \rho)$, for each edge $(t_i, t_j) \in E$ and $t_i, t_j \in T$, the retiming function R is valid if $R_i - R_j \geq 0$. Otherwise, the retiming is illegal as it implies a reference to unavailable data from the future.

4. Motivational examples

In this section, we illustrate three motivational examples. We use the task graph shown in Fig. 1(a) executed on a two-processor platform. The worst-case execution time (WCET) of each task $t_i (i = 1, \ldots, 8)$ is $w_1 = w_2 = w_3 = w_4 = w_5 = w_6 = w_7 = w_8 = 1$ while the period, ρ of the task graph is 6. For simplicity, we ignore the inter-processor communication to focus on the computation tasks for both examples. Also, we assume that the relative deadline is equal to the period.

Firstly, we illustrate the benefit of a retiming technique. Fig. 2(a) shows one possible schedule of a periodic application represented in Fig. 1(a). This figure shows all the tasks run repeatedly within four periods, and to avoid confusion, we denote each task instance as t_a^b, where a is the task number and b is the instance number of a task. This figure identifies unusable slack in processor $P2$ due to the precedence constraint between t_1 and t_3. On the other hand, Fig. 2(b) shows its corresponding retimed schedule. This figure turns unusable slack into usable slack, which can be used later for frequency scaling. However, retiming adds a preprocessing stage called the prologue, which is a one-time delay as it executes only once. Throughout this paper, we consider only the steady period, which is the first period after the prologue.

Secondly, we show that two different retiming techniques may produce different memory capacity overhead. A retiming technique that relies only on the height of each task in the task graph to transform all intra-period dependencies into inter-period dependencies would create unnecessary extra memory usage. Assume that we attempt to achieve the retimed schedule as in Fig. 4(a). Consider a first scenario where we only rely on the structure of the task graph, i.e. the height of each task when computing the retiming values, the retiming value of each task is $R_i (i = 1, \ldots, 8)$ is $R_1 = 4, R_2 = 2, R_3 = 3, R_4 = 1, R_5 = 2, R_6 = 2, R_7 = 1, R_8 = 0$. Then, consider a second scenario, which is computed by our approach. We compute the retiming values of all the tasks by considering the task mapping, and we get $R_1 = 3, R_2 = 1, R_3 = 2, R_4 = 0, R_5 = 2, R_6 = 1, R_7 = 1, R_8 = 0$. The maximum retiming value or the number of prologues required by the first scenario is 4, while

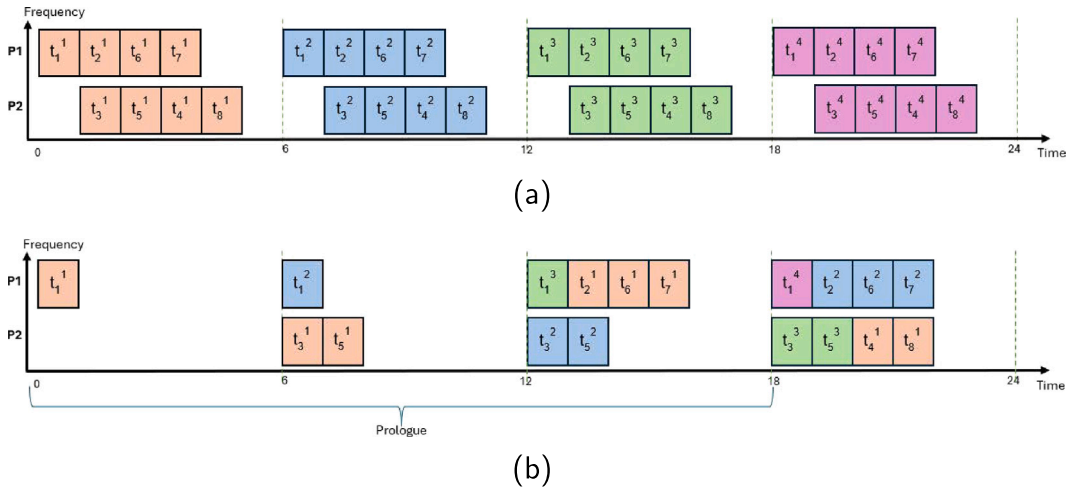

Fig. 2. The impact of retiming (a) A schedule of the periodic application in Fig. 1(a). (b) A retimed schedule of the periodic application.

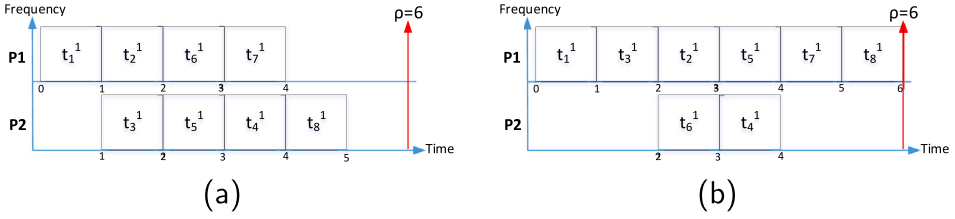

Fig. 3. Two different initial schedules, (a) with a balanced workload and (b) with an unbalanced workload, across the two processors.

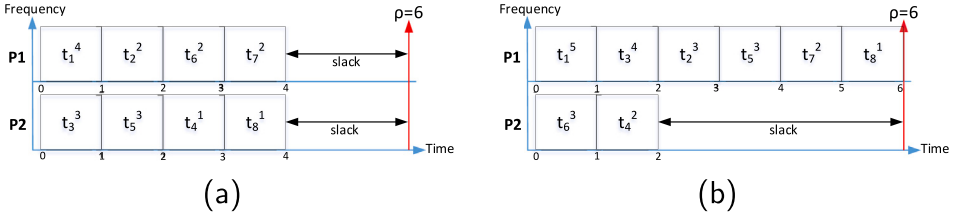

Fig. 4. The corresponding retimed schedules from Fig. 3, (a) A retimed schedule of Fig. 3(a), (b) A retimed schedule of Fig. 3(b).

the second scenario is only 3. Consequently, the first scenario requires more memory capacity overhead to store data compared to the second scenario, albeit both scenarios can achieve the same objective schedule as in Fig. 4(a).

Thirdly, we show that combining a workload-balanced initial schedule with retiming can produce a better energy-efficient schedule. Figs. 3(a), 4(a) and 5(a) show an initial schedule with a balanced workload across the two processors, its corresponding retimed schedule and its schedule after frequency scaling, respectively. Figs. 3(b), 4(b) and 5(b) show an initial schedule with an unbalanced workload across the two processors, its corresponding retimed schedule and its schedule after frequency scaling, respectively. Assume that the maximum frequency of both processors is 1 and simply use $\epsilon^{comp} = \sum_{i=1}^{|T|} f_i^3 e_i$ to compute the total energy where ϵ^{comp}, f_i, e_i are the total computation (dynamic) energy, the operating frequency of task t_i and the execution time of task t_i, respectively. Slack reclamation in Fig. 5(a) shows that all the tasks can be assigned with an optimal frequency of 0.67, generating approximately 3.609 J. On the other hand, Fig. 5(b) shows that only tasks t_6 and t_4 can be scaled to the optimal frequency of 0.33 while the other tasks use the maximum frequency and thus generate approximately 6.216 J. From this example, we can see that a workload-balanced initial schedule, combined with retiming, can produce usable slacks for all the processors, which can then be utilized for frequency scaling to reduce their total energy consumption.

5. Proposed approach

Given a streaming application and a NoC-based MPSoC to execute the application, our main objective is to minimize the total energy consumption of the system under memory constraints. Our approach constructs an initial schedule under maximum frequencies such that the workload across all the processors is balanced. Then, we employ a retiming heuristic on the initial schedule to transform certain intra-period dependencies into inter-period dependencies. Next, we assign an optimal discrete frequency to each task and each message using an NLP-based algorithm and an ILP-based algorithm. Then, we compute the resultant schedule's total energy and memory usage. If there are no memory capacity violations, our approach terminates. Otherwise, our approach iteratively fixes the retiming values based on the initial schedule and reruns the NLP-based and ILP-based algorithms to assign optimal discrete frequencies to all tasks and messages until the memory constraint is satisfied. We specify our approach as follows:

1. Compute the priority of each task and each message based on its approximate successor-tree-consistent deadline.
2. Based on the priorities of tasks, assign and schedule each task on a processor under maximum frequency such that the total utilization of all the processors is minimized. Hence, the workload across all the processors is balanced.

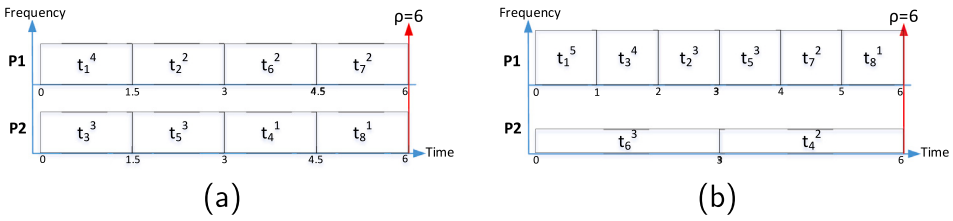

Fig. 5. Slack reclamation and frequency scaling, (a) based on the retimed schedule in Fig. 4(a)(b) based on the retimed schedule in Fig. 4(b) .

3. Retime the initial schedule by computing a retiming value for each task and each message according to its schedule.
4. Assign an optimal discrete frequency for each task and each message using an NLP-based algorithm and an ILP-based algorithm and then compute the total energy consumption and memory usage of the resultant schedule. If there are no memory spills, i.e., memory capacity violations, our heuristic terminates. Otherwise, repeat the following until the memory constraint is satisfied.
   (a) Fix the retiming value of the initial schedule considering the clique with maximum total weight problem. The clique is formed by a set of lifetime segments which overlap between each other and the total data size of the set is maximum.
   (b) Assign an optimal discrete frequency for each task and each message using an NLP-based algorithm and an ILP-based algorithm and then, compute the total energy consumption and memory usage of the resultant schedule.

In the subsequent subsections, we describe on how to compute the approximate successor-tree-consistent deadlines, task assignment and scheduling, retiming, optimal discrete frequency selection, and the repair step to fix the retiming values in case of memory violation.

5.1. Computing priorities

The priority of each task $t_i \in T$ and each edge $(t_i, t_j) \in E$ are their approximate successor-tree-consistent deadlines, d'_i and $d'_{i,j}$, respectively. A smaller deadline implies a higher priority. The approximate successor-tree-consistent deadlines of all the tasks are computed in reverse topological order, while the approximate successor-tree-consistent deadlines of all the edges are computed following their source tasks. We compute these priorities under the maximum frequencies of the processor and communication links. For each task t_i, if it is a sink task, its successor-tree-consistent deadline is equal to its preassigned deadline d_i. Otherwise, the approximate successor-tree-consistent deadline d'_i of t_i is computed as follows.

1. Construct the successor-tree of t_i.
2. If t_i has only one successor, $d'_i = d'_j - w_j$, where t_j is the successor of t_i. The approximate successor-tree-consistent deadline $d'_{i,j}$ of the edge incident from t_i is set equal to the start time of t_j, $d'_{i,j} = d'_j - w_j$.
3. Otherwise, do the following.
   (a) Partition all the successors of t_i into two disjoint sets U and V. Set U consists of all the tasks, each of which does not receive any data from t_i, and the set V contains all the successors of t_i that are not in U.
   (b) Sort all the tasks in U in non-increasing order of their approximate successor-tree-consistent deadlines. For the tasks with the same approximate successor-tree-consistent deadlines, further sort them in non-decreasing order of their worst-case execution times.
   (c) Schedule each task in U on a processor such that its start time is maximized.
   (d) Sort all the tasks in V in non-increasing order of their approximate successor-tree-consistent deadlines. For the tasks with the same approximate successor-tree-consistent deadlines, sort them in a non-increasing order of their edge weights. For the tasks with the same edge weight, further sort them in a non-decreasing order of their worst-case execution times.
   (e) Schedule each task in V on a processor such that its start time is maximized.
   (f) For each edge incident from t_i and incident to $t_j \in V$, set its $d'_{i,j}$ equal to the start time of t_j.
   (g) Find the latest completion time of t_i in the schedule for the tasks in $U \cup V$, respecting the precedence constraints specified by the successor-tree of t_i.
   (h) Set the approximate successor-tree-consistent deadline d'_i of t_i to the smaller one of its preassigned deadline and its latest completion time.

An illustrative example to compute the approximate successor-tree-consistent deadline for each task is presented in Abd Ishak and Wu (2016).

5.2. Task assignment and scheduling

Our scheduling step aims to construct a workload-balanced schedule under maximum frequencies within the first period. Our approach assigns each task $t_i \in T$ to a processor p_k and constructs an initial schedule G' for all the tasks and messages, assuming the maximum frequencies, as below.

- Repeat the following steps until each task $t_i \in T$ is scheduled.
  1. Select a ready task t_i with the smallest approximate successor-tree-consistent deadline among all the unscheduled tasks.
  2. For each processor $p_k \in P$, do the following.
     (a) Tentatively assign t_i to p_k.
     (b) For each immediate predecessor t_j of t_i assigned on a different processor than t_i, schedule the message $M_{j,i}$ using Earliest Deadline First (EDF) considering communication contention.
  3. Assign t_i to a processor p_k with minimum utilization among all the processors. If there are multiple options, choose the one that offers the earliest start time of t_i.
  4. Set t_i as scheduled.

The utilization μ_k of a processor p_k is computed as follows.

$$\mu_k = \sum_{t_i \in T_k}^{|T_k|} w_i / \rho \tag{7}$$

where T_k is a set of all the tasks assigned on processor p_k, w_i is the worst-case execution time of task $t_i \in T_k$ and ρ is the period of the task graph.

Since each task is executed periodically, let r_i^1 be the release time of task t_i in the first period and let ρ be the period. Then, the release

time of t_i in the η period ($\eta \geq 1$) is $r_i^\eta = r_i^1 + (\eta - 1)\rho$. Similarly, for a message $M_{i,j}$ between tasks t_i and t_j, let $r_{i,j}^1$ be the release time of $M_{i,j}$ in the first period, then its release time in the η period ($\eta \geq 1$) is $r_{i,j}^\eta = r_{i,j}^1 + (\eta - 1)\rho$.

Note that two messages have a communication contention if their routing paths overlap and they want to use a shared link simultaneously. For messages having a communication contention, we use EDF to serialize their simultaneous accesses to the shared communication links.

Eventually, for all the scheduled tasks and messages, we construct a new node-weighted graph $G' = \{T \cup M', E'\}$, where T, M', E' represent a set of scheduled tasks, a set of scheduled messages, and a set of control edges, respectively. M' and E' are constructed as follows:

1. For each edge $(t_i, t_j) \in E(t_i, t_j \in T)$, if t_i and t_j are on two different processors, add a message node $M_{i,j}$ to M', and two edges $(t_i, M_{i,j})$ and $(M_{i,j}, t_j)$ to E'. Otherwise, add an edge (t_i, t_j) to E'.
2. For each pair of messages $M_{i,j}$ and $M_{s,t}$ that have overlapping routing paths based on the XY routing strategy, do the following. If the modified deadline of $M_{i,j}$ is not greater than that of $M_{s,t}$ and there is no path from $M_{i,j}$ to $M_{s,t}$ in G', insert an edge $(M_{i,j}, M_{s,t})$ to E' in order to avoid communication contention. If the modified deadline of $M_{s,t}$ is not greater than that of $M_{i,j}$ and there is no path from $M_{s,t}$ to $M_{i,j}$ in G', insert an edge $(M_{s,t}, M_{i,j})$ to E'.

5.3. Retiming

Given a set of scheduled tasks and messages under maximum frequencies, we attempt to reschedule certain instances of tasks and messages so that non-productive slacks can be turned into productive slacks for frequency scaling. This can be done by transforming certain intra-period dependencies into inter-period dependencies. In other words, we regroup instances of tasks and messages from different periods into the same period, but with an introduction of latency periods called prologue.

In this step, we present a retiming technique to allocate each task and each message with a retiming value. The retiming value of a task or a message represents the number of its instances executed in the prologue. The retiming value of each task (message) has the following constraints.

- The retiming value of a task (message) must be a nonnegative integer.
- The retiming value of a parent node (task (message)) must be greater than or equal to its children nodes (task (message)).

A retiming value must be valid to preserve semantic correctness. If the retiming value of a parent node t_i is lesser than the retiming value of its child node t_j, it indicates that the data generated by the parent in the current period is needed to execute a child node in the previous period, which is incorrect.

We compute the retiming value of each task (message) in G' using a breadth-first search starting from a sink node, as follows.

1. Initialize the retiming value R_i of each node u_i representing t_i or $M_{i,j}$ in G' as 0.
2. Construct a set $S = \{u_k : u_k \text{ is a sink node in G'}\}$. Keep the last element of S in a variable X.
3. Repeat the following until S is empty.
   (a) Select the last element u_j in set S and then remove it from S.
   (b) For each predecessor u_i of $u_j \in S$, do the following.
      i. If u_i is assigned to a different processor (link) than u_j, update $u_i = max(u_i, u_j + 1)$.
      ii. If u_i is assigned to the same processor (link) as u_j, update $u_i = u_j$.
      iii. If X is not equal to u_i, add u_i to S. Update X as u_i.

In other words, we assign a retiming value for each task and each message according to the following conditions.

$$R_i = \begin{cases} max\{R_i, R_j + 1\}, & \text{if } u_j \text{ is a child node of } u_i \text{ and both are on different processors.} \\ R_j, & \text{if } u_j \text{ is a child node of } u_i \text{ and both are on the same processor.} \\ 0, & \text{if } u_i \text{ is a sink node.} \end{cases} \quad (8)$$

Next, we form a retimed graph $G'_r = (T \cup M', E')$ to represent the new intra-period dependencies after the retiming. Firstly, we make a copy G'_r of G'. Then, we remove certain edges in G'_r to represent the new intra-period dependencies of G'. An edge $(u_i, u_j) \in G'_r$ is removed between nodes u_i and u_j if the following constraints hold simultaneously:

- u_i and u_j have different retiming values.
- u_i and u_j are on different resources (processor or link).

Notice that we keep the mapping and ordering of tasks (messages) on each processor (link) by preserving the additional control edges of G'.

Furthermore, we compute the retimed schedule of each task (message) in the first steady period following the retimed graph G'_r. We compute the earliest start time of each task and each message according to the retimed graph G'_r.

5.4. Discrete frequency selection

We have a set of scheduled tasks and messages, their retiming values, and the retimed graph G'_r. Our objective is to assign an optimal discrete frequency to each task and each message under a discrete frequency model such that the total energy consumption of all the tasks and messages is minimized. Firstly, we employ an NLP-based algorithm to compute the optimal frequencies for all tasks and messages under a continuous frequency model. Secondly, based on the optimal continuous frequency of each task and each message computed by the NLP-based algorithm, we assign its optimal discrete frequency using an ILP-based algorithm.

5.4.1. Computing optimal continuous frequencies

We have $G'_r = (T \cup M', E'_r)$ where T, M and E'_r be a set of all the tasks, a set of all the messages, and a set of all the edges. Next, we derive all the constraints based on G'_r as follows:

1. The execution time constraint for each task t_i:

$$e_i = c_i / f_i \quad \forall t_i \in T \quad (9)$$

   where e_i, c_i, and f_i are the execution time, worst-case execution cycles, and processor frequency of t_i, respectively.
2. The communication time constraint for each message $M_{i,j}$:

$$e_{i,j} = c_{i,j} / (\lambda f_{i,j}) \quad \forall M_{i,j} \in M'_r \quad (10)$$

   where $e_{i,j}$, $c_{i,j}$, λ, and $f_{i,j}$ are the communication time, data size, link data width and link frequency, respectively.
3. The precedence constraints:

$$s_i + e_i(e_{i,j}) \leq s_j \quad \forall (u_i, u_j) \in E'_r \quad (11)$$

   where s_i is the start time of a task or a message, and $e_i(e_{i,j})$ is the execution time of the task (the communication time of the message).

4. The supply voltage range constraints and the frequency range constraints for all the tasks:

$$v_{min} \le v_{s_i} \le v_{max} \quad \forall t_i \in T \tag{12}$$

$$f_{min} \le f_i \le f_{max} \quad \forall t_i \in T \tag{13}$$

where v_{min}, v_{max}, f_{min} and f_{max} are the minimum voltage, maximum voltage, minimum frequency and maximum frequency of the processor p_k running t_i, respectively.

5. The supply voltage range constraint and the frequency range constraints for each link:

$$v^l_{min} \le v_{s_{i,j}} \le v^l_{max} \quad \forall M_{i,j} \in M'_r \tag{14}$$

$$f^l_{min} \le f_{i,j} \le f^l_{max} \quad \forall M_{i,j} \in M'_r \tag{15}$$

where v^l_{min}, v^l_{max}, f^l_{min} and f^l_{max} are the minimum supply voltage, the maximum supply voltage, the minimum frequency and the maximum frequency of each link, respectively.

6. Deadline constraint for each task t_i:

$$s_i + e_i \le d'_i \quad \forall t_i \in T \tag{16}$$

where d'_i is the approximate successor-tree-consistent deadline of t_i.

Furthermore, for each supply voltage and frequency pair for all the tasks and messages, Constraint (6) in Section 3.2 is used.

The objective function is shown as follows:

$$\min\{ \underbrace{\sum_{t_i \in T} \varepsilon_i(v_{s_i}, f_i)}_{\text{computation energy}} + \underbrace{\sum_{M_{i,j} \in M'_r} \varepsilon_{i,j}(v_{s_{i,j}})}_{\text{communication energy}} \} \tag{17}$$

Since the objective function and the constraints are convex, this NLP problem can be solved in polynomial time (Andrei et al., 2007).

5.4.2. Computing optimal discrete frequencies

We continue with an ILP-based algorithm to assign an optimal discrete frequency to each task and each message, aiming at minimizing the total energy consumption of all the tasks and messages.

For each task $t_i \in T$ and each message $M_{i,j}$, let f^{opt}_i and $f^{opt}_{i,j}$ be the optimal frequencies of t_i and $M_{i,j}$, respectively, computed by our approach after the scheduling phase. We distinguish between the following two cases.

1. The optimal frequency for t_i ($M_{i,j}$) is equal to a discrete frequency of the processor (link). In this case, we assign the optimal frequency to t_i ($M_{i,j}$).
2. The optimal frequency for t_i ($M_{i,j}$) is not a discrete frequency of the processor (link). Let f^l_i be the largest frequency of the processor where $t_i \in T$ is assigned such that f^l_i is less than f^{opt}_i, and f^{l+1}_i be the frequency at a higher level of that processor. Similarly, let $f^l_{i,j}$ be the largest frequency of a link that is less than f^{opt}_i, and $f^{l+1}_{i,j}$ be the frequency at a higher level of a link. Clearly, in an optimal schedule, the frequency for t_i must be either f^l_i or f^{l+1}_i, and the frequency for $M_{i,j}$ must be either $f^l_{i,j}$ or $f^{l+1}_{i,j}$.

Therefore, we introduce a binary decision variable x_i to choose between f^l_i or f^{l+1}_i for each task $t_i \in T$ and $f^l_{i,j}$ or $f^{l+1}_{i,j}$ for each message $M_{i,j} \in M'_r$ as follows.

$$x_i = \begin{cases} 0, & \text{if } (v^l_{s_i}, f^l_i)\ ((v^l_{s_{i,j}}, f^l_{i,j})) \text{ is used} \\ 1, & \text{if } (v^{l+1}_{s_i}, f^{l+1}_i)\ ((v^{l+1}_{s_{i,j}}, f^{l+1}_{i,j})) \text{ is used} \end{cases} \tag{18}$$

where $v^l_{s_i}$ and $v^{l+1}_{s_i}$ are the corresponding supply voltages of f^l_i and f^{l+1}_i, and $v^l_{s_{i,j}}$ and $v^{l+1}_{s_{i,j}}$ are the corresponding supply voltages of $f^l_{i,j}$ and $f^{l+1}_{i,j}$.

The total computation energy consumption ε^{comp} of all tasks is formulated as follows:

$$\varepsilon^{comp} = \sum_{t_i \in T'} (1 - x_i)\varepsilon_i(v^l_{s_i}, f^l_i) + x_i\varepsilon_i(v^{l+1}_{s_i}, f^{l+1}_i) \tag{19}$$

where $\varepsilon_i(v^l_{s_i}, f^l_i)$ and $\varepsilon_i(v^{l+1}_{s_i}, f^{l+1}_i)$ are the total computation energy consumption of t_i at the frequency f^l_i and at the frequency f^{l+1}_i, respectively. Notice that both $\varepsilon_i(v^l_{s_i}, f^l_i)$ and $\varepsilon_i(v^{l+1}_{s_i}, f^{l+1}_i)$ are constants.

Similarly, the total communication energy ε^{comm} is calculated as follows:

$$\varepsilon^{comm} = \sum_{M_{i,j} \in M'} (1 - x_i)\varepsilon_{i,j}(v^l_{s_{i,j}}) + x_i\varepsilon_{i,j}(v^{l+1}_{s_{i,j}}) \tag{20}$$

where $v^l_{s_{i,j}}$ and $v^{l+1}_{s_{i,j}}$ are the corresponding supply voltages of $f^l_{i,j}$ and $f^{l+1}_{i,j}$, respectively.

The objective function is shown as below.

$$\min\{\varepsilon^{comp} + \varepsilon^{comm}\} \tag{21}$$

The constraints include the execution time constraints for each task and each message:

$$e_i = (1 - x_i)\epsilon^l_i + x_i\epsilon^{l+1}_i \quad \forall t_i \in T \tag{22}$$

$$e_{i,j} = (1 - x_{i,j})\epsilon^l_i + x_{i,j}\epsilon^{l+1}_i \quad \forall M_{i,j} \in M'_r \tag{23}$$

The precedence constraint (11) for each edge and the deadline constraint (16) for each task in the NLP formulation still hold. From this formulation, we get the amount of total energy consumption of the schedule. Next, we compute the memory usage of the schedule as in Section 5.4.3.

5.4.3. Computing memory usage

Note that since the memory is distributed, the data buffer depends on the following two cases:

- If a node in G' is a task, t_i, the data is stored in the processor's memory where t_i is assigned. In this case, the data buffer is needed from the time t_i completes execution to the time t_j completes execution ($(t_i, t_j) \in E'$).
- If a node in G' is a message, $M_{i,j}$, the data is stored in the processor's memory where t_j is assigned. In this case, the data buffer is needed from when t_i starts execution to when t_j completes execution.

The duration for which the data buffer is needed is called a lifetime segment.

Given a set of scheduled tasks and messages, we compute the memory usage of the schedule in two steps. Firstly, we find all the lifetime segments within the prologue and the first steady period according to the schedule task graph G' since it preserves all the original intra-period dependencies. Secondly, we solve the problem to compute the maximum memory usage into the Maximum Weight Clique problem (Jiang et al., 2017). Next, we describe each step. The first step works as follows:

- For each edge $(u_i, u_j) \in E'$ ($u_i, u_j \in T \bigcup M'$) excluding all control edges, do the following.
  1. Let $E'_i = (u_s, u_t)$ be the first edge.
  2. Compute the periodic delay of edge E'_i, which is the difference between the retiming value of node u_s and u_t, $\delta_i = R_s - R_t$.
  3. For each period $\theta_l (l = 0, \ldots, R_s)$, i.e. within the prologue and the first steady period, do the following.
     (a) Add a lifetime segment between node u_i and node u_j, $li_k = (\vdash, \dashv, s, \Omega, size, loc)$, where $li_k.\vdash$, $li_k.\dashv$, $li_k.s$, $li_k.\Omega$, $li_k.size$ and $li_k.loc$ are the sender task,

receiver task, start time, end time, data size and location of the segment. We compute the real start time and real end time of $li_k.s$, $li_k.\Omega$ by appending the prologue with $R^{max}\rho$ time unit as in Eq. (24) and Eq. (25), respectively.

$$li_k.s = \begin{cases} (R^{max} - l)\rho + s_i, & \text{if } u_i \text{ is a message node} \\ (R^{max} - l)\rho + s_i + e_i, & \text{if } u_i \text{ is a task node} \end{cases} \tag{24}$$

$$li_k.\Omega = (R^{max} - l + \delta_i)\rho + s_j + e_j, \forall u_j \tag{25}$$

(b) If $li_k.\Omega$ is greater than $(R_{max} + 1)\rho$, set $li_k.\Omega$ equal to $(R_{max} + 1)\rho$, where R_{max} and ρ are the maximum retiming value of all the nodes and the period, respectively.

Notice that it is sufficient to consider all lifetime segments within the prologue and the first steady period because the schedule is repetitive.

In the second step, we compute the maximum memory usage in each data buffer $b_k \in B$. We consider this problem as the Maximum Weight Clique problem (Jiang et al., 2017). It works as follows.

- Let a memory usage variable MEM^{max} be initialized as 0.
- For each data buffer $b_k \in B$, do the following.

1. Let $L_k = \{li_i : li_i.loc == b_k\}$ be the set of all lifetime segments in data buffer b_k.
2. Find a set of distinct time points, Γ of the start time and end time of all the lifetime segments in L_k. Sort Γ in non-decreasing order.
3. For each pair of time points Γ_j and $\Gamma_{j+1} (j = 0, 1, \ldots, |\Gamma|-2)$, do the following.
   (a) Find a set of lifetime segments intersecting within interval $[\Gamma_j, \Gamma_{j+1}]$ $(L_k^{j+1}) \subseteq L_k$.
   (b) Add an interval $lt_{j+1} = (\Gamma_j, \Gamma_{j+1}, L_k^{j+1}, totalsize)$. Γ_j, Γ_{j+1}, L_k^{j+1} and $totalsize$ represent the start time, end time, a set of intersecting lifetime segments within interval $[\Gamma_j, \Gamma_{j+1}]$ and the total data size of the lifetime segments in L_k^{j+1}, respectively.
4. Find an interval lt_s such that its total data size is maximum. If there are multiple options, choose the one with the minimum number of intersecting lifetime segments.
5. Construct an undirected graph $G_s = (V_s, E_s)$ representing the lifetime segments in lt_s, where each node in V_s represents a lifetime segment in $lt_s.L_k^{j+1}$ and E_s is a set of all edges such that every pair of distinct nodes in V_s is connected.
6. If $MEM^{max} < lt_s.totalsize$, update MEM^{max} as $lt_s.totalsize$ and set $G_{C_{max}}$ as G_s. $G_{C_{max}}$ is the Maximum Weight Clique graph.

Notice that a set of lifetime segments intersecting within interval $[\Gamma_j, \Gamma_{j+1}]$ interval can be considered a clique. A clique can be represented as an undirected graph where each lifetime segment is a node and all the nodes are connected to each other to form a complete subgraph.

5.5. Repair approach

We employ an algorithm to fix the schedule in case of memory violations. We aim is to recompute the retiming values of all tasks and messages such that the schedule can meet the system memory capacity constraints.

We introduce two metrics for each producer or sender task t_i: the total energy difference and memory difference. These metrics are used to evaluate the impact when reducing the retiming value of t_i on the memory space reduction and the total energy. We attempt to identify a significant t_i, when reducing its current retiming value by one, can reduce the memory usage the most, but with a minimum increase in total energy consumption.

We compute the total energy difference metric as follows.

$$\Delta E_i = \varepsilon_{\sigma'} - \varepsilon_{\sigma} \tag{26}$$

where ε_σ and $\varepsilon_{\sigma'}$ are the total energy of the current schedule and the total energy when we reduce the retiming value of t_i by 1, respectively.

We compute the memory requirement difference metric as below.

$$\Delta MEM_i = MEM_{\sigma'}^{max} - MEM_{\sigma}^{max} \tag{27}$$

where MEM_σ^{max} and $MEM_{\sigma'}^{max}$ are the maximum memory usage of the current schedule and the maximum memory usage when we reduce the retiming value of t_i by 1, respectively.

Given a Maximum Weight Clique graph $G_{C_{max}}$ with the corresponding lifetime segment attributes, we rank all sender tasks based on metrics in Eq. (26) and Eq. (27) to evaluate their effectiveness in reducing the clique total weight. We repeat the following until the memory capacity constraint is satisfied.

- Find a set of distinct sender tasks from $G_{C_{max}}$.
- For each sender task, do the following.

1. Let $li_k = (\vdash, \dashv, s, \Omega, size, loc)$ be the first lifetime segment. Let the sender task $li_k.\vdash$ as t_k.
2. Make a copy R_{new} of R_{cur}, where R_{cur} is the current retiming values of all tasks and messages.
3. Tentatively reduce the retiming value R_{new_k} of t_k by 1 and update its successors' retiming values accordingly in R_{new} as follows.
   (a) Let $t_i(M_{i,j}) \in G'$ be the task (message) such that its retiming value R_{cur_i} needs to be reduced by 1.
   (b) If R_{new_i} is equal to 0, terminates. Otherwise, update R_{new_i} as $R_{cur_i} - 1$.
   (c) Find a set S, consisting the immediate successors of $t_i(M_{i,j})$ such that their retiming values are equal to R_{cur_i}.
   (d) For each $t_j(M_{s,t}) \in S$, repeat steps (a) until (c).
4. Construct a retimed graph G'_r based on R_{new} as in Section 5.3.
5. Assign optimal discrete frequencies to all tasks and messages based on G'_r as in Section 5.4.
6. Compute the total energy and maximum memory usage of the new schedule. Then, compute the energy difference ΔE_k, memory requirement difference ΔMEM_k and latency l_k of the new schedule, and compare them to the current schedule measurements.
7. Save the new tentative schedule, R_{new}, its total energy and its maximum memory usage in $TABLE$.

- Rank all the sender tasks based on non-decreasing order of ΔMEM_k, non-decreasing order of ΔE_k and non-decreasing order of l_k.
- Select the one with the highest rank and retrieve the new schedule from $TABLE$.
- Update the schedule.

The time complexity to construct an initial schedule in Sections 5.1 and 5.2 is dominated by computing the approximate successor-tree-consistent deadlines. It takes $O(|T \parallel E|)$ time to calculate the approximate successor-tree-consistent deadlines, where $|T|$ is the number of

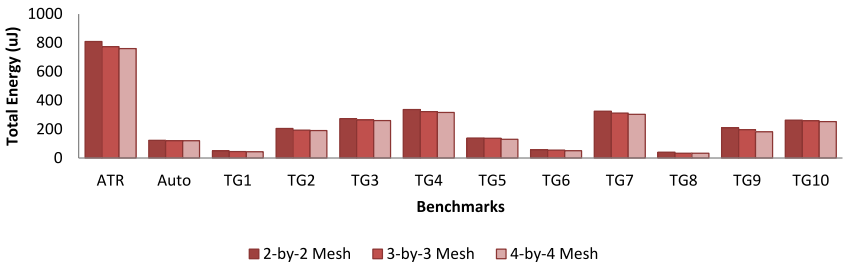

Fig. 6. The total energy of each benchmark on three architectures using Ours.

tasks and $|E|$ is the number of edges in the task graph. Next, the time complexity of the retiming algorithm for the initial schedule represented by a task graph in Section 5.3 is generally dominated by traversing the graph using the breadth-first search. Thus, the retiming algorithm takes $O(|T|+|E|)$, where $|T|$ is the number of tasks and $|E|$ is the number of edges in the task graph.

6. Experimental results

To evaluate our approach, we compare them with two state-of-the-art approaches, RDAG+GeneS (Wang et al., 2011) and JCCTS (Wang et al., 2013a), using a set of real and synthetic benchmarks. We chose the two because they are the latest approaches considering a similar problem structure as ours, which concentrates on the real-time periodic dependent task set on homogeneous multiprocessors under memory constraints. This experiment aims to quantify energy consumption from different approaches and provides insights into the key energy consumption factors of a NoC-based MPSoC. In addition, the running times of the algorithm are measured.

6.1. Experimental settings

We choose two real benchmarks, which are Automatic Target Recognition (ATR) and automotive, with another ten synthetic benchmarks (TG1 until TG10) in Canon et al. (2019). All ten synthetic benchmarks are randomly generated task graphs with their structure based on specified parameters. Table 1 shows the characteristics of each benchmark in terms of the number of tasks $|T|$, the number of edges $|E|$, the communication to computation ratio (CCR), which is defined as total communication divided by average computation and its period ρ. The period, the required number of computation clock cycles of each task and the communication data size of each edge are given in the source.

We configure three sets of NoC-based architecture which are 2×2 mesh NoC, 3×3 mesh NoC and 4×4 mesh NoC as the target MPSoC. Each processor is based on the 0.07 μm technology with its parameters provided in Ali et al. (2021), Martin et al. (2002), that follow the power model in Section 3.2. Each processor has five voltage and frequency levels. The minimum supply voltage v_{min_k} and maximum supply voltage v_{max_k} follow the minimum supply voltage and maximum supply voltage of their corresponding discrete set. Communication links have five voltage and frequency levels. To compute the communication energy, we use the parameters in Li and Wu (2016), Andrei et al. (2007).

We define our proposed approach as Ours. We implement Ours, RDAG+GeneS (Wang et al., 2011) and JCCTS (Wang et al., 2013a) on Matlab version R2016a. We utilize the Matlab fmincon solver for the NLP problem and Matlab intlinprog solver for the ILP problem. We perform the experiments on a hardware platform with Intel(R) Core(TM) i5-4570 CPU and a clock frequency of 3.20 GHz, 8.00 GB memory and 3 MB cache.

Table 1
The characteristics of benchmarks.

Benchmarks	$\|T\|/\|E\|$	CCR	ρ
ATR	17/16	16.99	30
Automotive	9/9	0.49	0.0009
TG1	30/33	10.14	365
TG2	28/27	8.00	291
TG3	18/24	4.60	90
TG4	14/20	3.54	90
TG5	26/28	8.06	250
TG6	15/19	3.00	139
TG7	20/27	4.60	77
TG8	18/26	3.64	129
TG9	13/18	2.75	112
TG10	11/14	2.19	70

6.2. Results and discussions

In this section, we discuss the simulation results in terms of three metrics: total energy consumption, memory usage and algorithm running time.

First, we discuss on the impact of processing parallelism of each benchmark on the total energy consumption. Fig. 6 shows the total energy consumption (computation plus communication energy) of all the benchmarks using Ours on three different platforms. Each vertical axis denotes the total energy consumption of each benchmark using Ours, and each horizontal axis denotes the benchmarks. The simulation results show that in most scenarios, the total energy consumption decreases as the number of processors increases. This is because our approach constructs initial schedules such that the total utilization of all the processors is minimized. Thus, a larger number of processors causes the processing parallelism to increase and creates more static slacks, resulting in lower frequencies for the tasks and messages.

Second, we compare the total energy consumption of each benchmark using the Ours, RDAG+GeneS (Wang et al., 2011) and JCCTS (Wang et al., 2013a). Since Ours considers memory capacity constraints, we set the memory capacity constraint of each scenario to be equal to the maximum memory usage of each scenario computed using RDAG+Genes. Fig. 7 shows the total energy consumption (computation plus communication energy) of all the benchmarks using Ours, RDAG+GeneS (Wang et al., 2011) and JCCTS (Wang et al., 2013a). Each vertical axis denotes the total energy consumption of each benchmark using a specific approach, and each horizontal axis denotes the benchmarks. Overall, the results show that Ours has the advantage in reducing the total energy consumption of all the benchmarks compared to the other approaches. Specifically, as compared to RDAG+GeneS (Wang et al., 2011), the largest improvement of 31.72% occurred for synthetic benchmark TG8 on a 4-by-4 mesh NoC, the least improvement of 7.00% for synthetic benchmark TG4 on a 2-by-2 mesh NoC, while the average improvement of all scenarios over RDAG+GeneS (Wang et al., 2011) is 14.05%.

The main reason of Ours advantage over RDAG+GeneS is that RDAG+Genes employs the Genetic Algorithm (GA) to schedule and assign discrete frequencies to all tasks. GA is based on natural selection and does not necessarily find the optimal solutions. On the contrary,

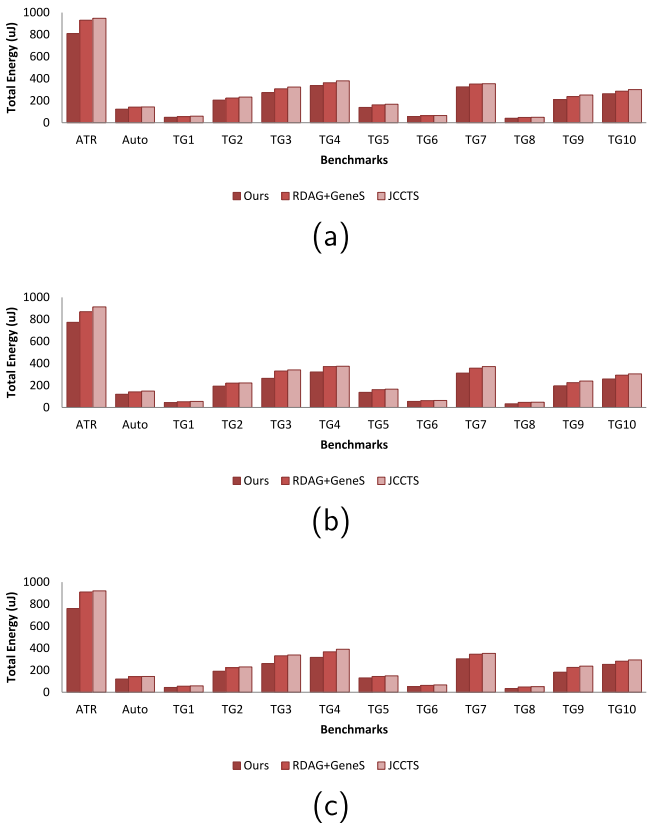

Fig. 7. Comparison of total energy between Ours, RDAG+GeneS (Wang et al., 2011) and JCCTS (Wang et al., 2013a) on (a) 2 × 2 mesh NoC (b) 3 × 3 mesh NoC and (c) 4 × 4 mesh NoC.

Ours uses an NLP-based algorithm and an ILP-based algorithm to assign an optimal discrete frequency for each task and each message.

Furthermore, the maximum improvement, the average improvement and the minimum improvement of our approach over JCCTS (Wang et al., 2013a) are 35.58%, 17.04% and 8.21%, respectively. The maximum improvement happens on synthetic benchmark TG8 on a 4-by-4 mesh NoC and the minimum improvement occurred for synthetic benchmark TG7 on a 2-by-2 mesh NoC.

The main reason is that JCCTS attempts to minimize the prologue length while totally removing the inter-processor communication overhead. We observe that their approach might work to remove the communication overhead, but not in terms of total energy. This is because significant intra-period dependencies still exist in order to minimize the maximum retiming value or the prologue length. Consequently, these dependencies block the utilization of slacks for frequency scaling and thus limit the opportunity for energy reductions. On the contrary, Ours attempt to exploit the non-productive slacks for frequency scaling fully.

Third, we evaluate the benefits of retiming. We compare Ours with a modified version of Ours, Our-WR (Without Retiming) which does not employ retiming. We construct the schedules using Ours-WR as follows.

1. Construct an initial schedule as in Section 5.2.
2. Assign an optimal discrete frequency for each task and message as in Section 5.4.

Figs. 8, 9, 10 show the total energy consumption and the corresponding maximum memory usages of all benchmarks using Ours and Ours-WR. For the figure displaying total energy consumption, each vertical axis denotes the total energy consumption of each benchmark using a specific approach, and each horizontal axis denotes the benchmarks. For the figure showing maximum memory usage, each vertical axis denotes the memory size requirements of each benchmark using a specific approach, and each horizontal axis denotes the benchmarks. The simulation results show that retiming can effectively reduce total energy consumption. Ours is advantageous in reducing the total energy consumption of all scenarios. However, Ours-WR constructs schedules that require the least memory usage. This is because Ours-WR does not implement pipelining and thus requires no extra memory capacity overhead. Our retiming technique can improve the total energy consumption by up to 77%.

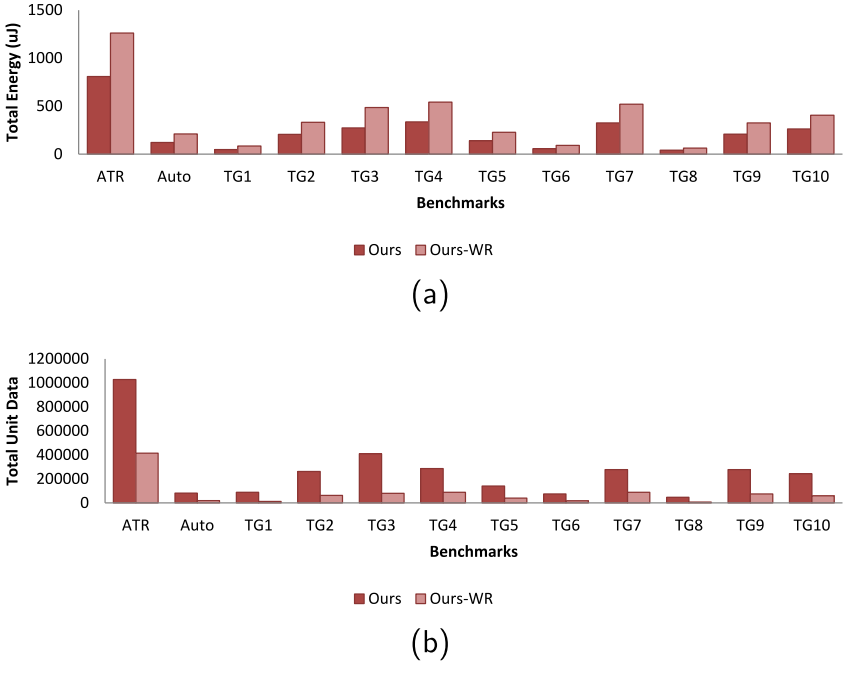

Fig. 8. Comparison between Ours and Ours-WR on 2-by-2 Mesh NoC in terms of (a) total energy consumption (b) maximum memory usage.

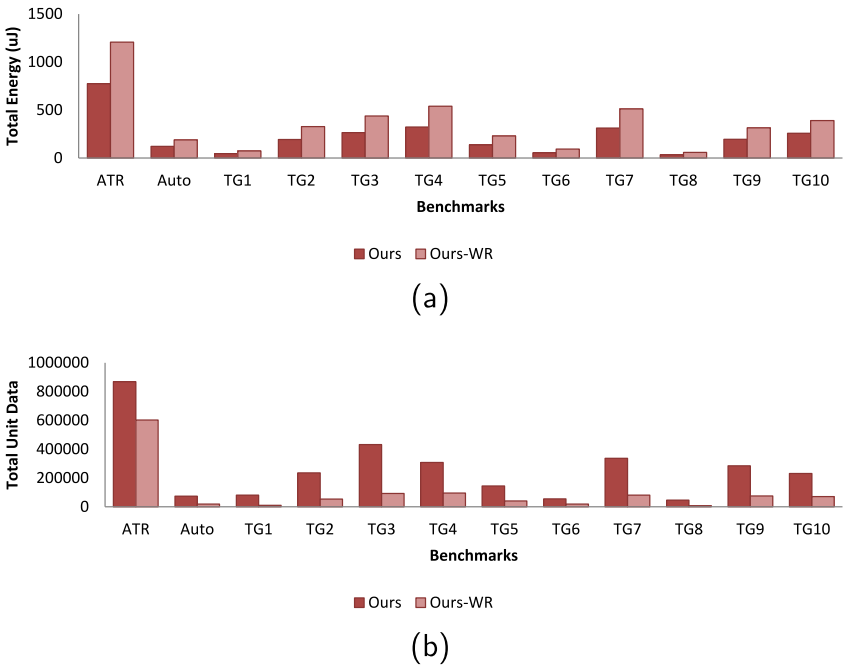

Fig. 9. Comparison between Ours and Ours-WR on 3-by-3 Mesh NoC in terms of (a) total energy consumption (b) maximum memory usage.

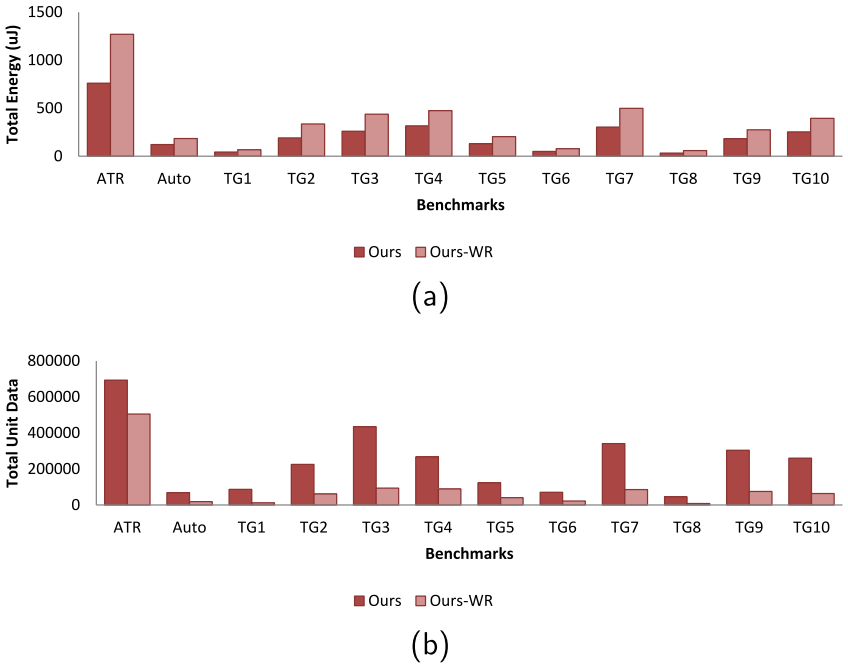

Fig. 10. Comparison between Ours and Ours-WR on 4-by-4 Mesh NoC in terms of (a) total energy consumption (b) maximum memory usage.

6.3. Algorithm running time

We show the running times to construct the schedules produced by Ours, RDAG+GeneS (Wang et al., 2011) and JCCTS (Wang et al., 2013a). Table 2 summarizes the average running times of Ours, RDAG+GeneS (Wang et al., 2011) and JCCTS (Wang et al., 2013a) approaches on each NoC architecture.

For all the benchmarks, the average running time ratio of the RDAG+GeneS (Wang et al., 2011) approach over Ours is 97.97% while the average running time ratio of JCCTS (Wang et al., 2013a) over Ours is 91.74%. As we can see, our approaches are much slower than the RDAG+GeneS (Wang et al., 2011) and JCCTS (Wang et al., 2013a) approaches. We found repetitive NLP-based algorithm and ILP-based algorithm dominates the total running time of Ours. The longer running times of the NLP-based algorithm and the ILP-based algorithm can

Table 2
Average running times of Ours, RDAG+GeneS (Wang et al., 2011) and JCCTS (Wang et al., 2013a) on each architecture.

Architecture	Ours (s)	RDAG+GeneS (s)	JCCTS (s)
2-by-2	264.71	266.59	248.70
3-by-3	280.21	276.45	253.07
4-by-4	297.33	282.14	255.28

be attributed to the inherent complexity of the optimization problems to solve, as well as the computational challenges associated with exploring large solution spaces, handling numerical instability, and addressing integer constraints. Although the run time is slower, our approach computes much better energy-efficient schedules than the RDAG+GeneS (Wang et al., 2011) and JCCTS (Wang et al., 2013a) approaches do. Notice that our approaches target embedded systems. Therefore, the running time for constructing an offline schedule at the design stage is not an issue.

7. Conclusion

We present a novel approach to the problem of scheduling a set of non-preemptible periodic dependent tasks with precedence and deadline constraints on a homogeneous NoC-based MPSoC with discrete frequencies such that the total energy consumption of all the tasks is minimized under memory constraints. Our approach consists of a set of novel techniques which are constructing an initial schedule based on a list scheduling where the priority of each task is its approximate successor-tree-consistent deadline such that the workload across all the processors is balanced, a retiming heuristic to transform intra-period dependencies into inter-period dependencies for enhancing parallelism, assigning an optimal discrete frequency for each task and each message using NLP-based algorithm and ILP-based algorithm, and an incremental approach to reducing the memory usage of the retimed schedule in case of memory violation. We evaluated and compared our approach with two state-of-the-art approaches, RDAG+GeneS (Wang et al., 2011), and JCCTS (Wang et al., 2013a). Experimental results show that our approach performs significantly better than the two approaches in terms of total energy consumption while satisfying the memory constraints.

CRediT authorship contribution statement

Suhaimi Abd Ishak: Writing – review & editing, Writing – original draft, Validation, Methodology, Funding acquisition. **Hui Wu:** Writing – review & editing, Supervision, Methodology, Formal analysis. **Umair Ullah Tariq:** Visualization, Resources, Investigation.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Suhaimi Abd Ishak reports financial support was provided by Universiti Tun Hussein Onn Malaysia. Suhaimi Abd Ishak reports financial support was provided by Malaysia Ministry of Higher Education. Suhaimi Abd Ishak reports a relationship with Malaysia Ministry of Higher Education that includes: employment and funding grants. Suhaimi Abd Ishak reports a relationship with Universiti Tun Hussein Onn Malaysia that includes: employment and funding grants. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research was supported by Universiti Tun Hussein Onn Malaysia (UTHM) through Tier 1 (vot Q430) and Ministry of Higher Education (MOHE), Malaysia through Fundamental Research Grant Scheme (FRGS) (FRGS/1/2019/ICT05/UTHM/03/1)

References

Abd Ishak, S., Wu, H., 2016. Energy-aware task scheduling with precedence and deadline constraints on MPSoCs. In: 2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems. HPCC/SmartCity/DSS, IEEE, pp. 1163–1172.

Ali, H., Tariq, U.U., Hardy, J., Zhai, X., Lu, L., Zheng, Y., Bensaali, F., Amira, A., Fatema, K., Antonopoulos, N., 2021. A survey on system level energy optimisation for MPSoCs in IoT and consumer electronics. Comp. Sci. Rev. 41, 100416.

Ali, H., Tariq, U.U., Liu, L., Panneerselvam, J., Zhai, X., 2019. Energy optimization of streaming applications in IoT on NoC based heterogeneous MPSoCs using re-timing and dvfs. In: 2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation. SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI, IEEE, pp. 1297–1304.

Andrei, A., Eles, P., Peng, Z., Schmitz, M.T., Al Hashimi, B.M., 2007. Energy optimization of multiprocessor systems on chip by voltage selection. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 15 (3), 262–275.

Canon, L.-C., Sayah, M.E., Héam, P.-C., 2019. A comparison of random task graph generation methods for scheduling problems. In: Euro-Par 2019: Parallel Processing: 25th International Conference on Parallel and Distributed Computing, GÖTtingen, Germany, August 26–30, 2019, Proceedings 25. Springer, pp. 61–73.

Chen, H., Cheng, A.M.K., Kuo, Y.-W., 2011. Assigning real-time tasks to heterogeneous processors by applying ant colony optimization. J. Parall. Distribut. Comput. 71 (1), 132–142.

Chen, J., He, Y., Zhang, Y., Han, P., Du, C., 2022. Energy-aware scheduling for dependent tasks in heterogeneous multiprocessor systems. J. Syst. Archit. 129, 102598.

Chu, H.-M., Yang, S.-W., Pillai, P., Chen, Y.-K., 2018. Scheduling in visual fog computing: NP-completeness and practical efficient solutions. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, (no. 1).

Gammoudi, A., Benzina, A., Khalgui, M., Chillet, D., 2018. Energy-efficient scheduling of real-time tasks in reconfigurable homogeneous multicore platforms. IEEE Trans. Syst. Man Cybern.: Syst. 50 (12), 5092–5105.

Han, J.-J., Lin, M., Zhu, D., Yang, L.T., 2014. Contention-aware energy management scheme for NoC-based multicore real-time systems. IEEE Trans. Parallel Distrib. Syst. 26 (3), 691–701.

Henkel, J., Parameswaran, S., 2007. Designing Embedded Processors: A Low Power Perspective. Springer Science & Business Media.

Huang, P., Moreira, O., Goossens, K., Molnos, A., 2013. Throughput-constrained voltage and frequency scaling for real-time heterogeneous multiprocessors. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing. ACM, pp. 1517–1524.

Jiang, H., Li, C.-M., Manya, F., 2017. An exact algorithm for the maximum weight clique problem in large graphs. In: AAAI. pp. 830–838.

Kumar, G.S.A., Manimaran, G., 2007. Energy-aware scheduling of real-time tasks in wireless networked embedded systems. In: Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International. IEEE, pp. 15–24.

Leiserson, C.E., Saxe, J.B., 1991. Retiming synchronous circuitry. Algorithmica 6 (1), 5–35.

Li, J., Qiu, M., Niu, J.-W., Chen, T., 2011. Battery-aware task scheduling in distributed mobile systems with lifetime constraint. In: Design Automation Conference (ASP-DAC), 2011 16th Asia and South Pacific. IEEE, pp. 743–748.

Li, D., Wu, J., 2016. Energy-efficient contention-aware application mapping and scheduling on NoC-based MPSoCs. J. Parallel Distrib. Comput. 96, 1–11.

Liu, H., Shao, Z., Wang, M., Chen, P., 2008. Overhead-aware system-level joint energy and performance optimization for streaming applications on multiprocessor systems-on-chip. In: Real-Time Systems, 2008. ECRTS'08. Euromicro Conference on. IEEE, pp. 92–101.

Liu, D., Spasic, J., Chen, G., Stefanov, T., 2015. Energy-efficient mapping of real-time streaming applications on cluster heterogeneous mpsocs. In: Embedded Systems for Real-Time Multimedia (ESTIMedia), 2015 13th IEEE Symposium on. IEEE, pp. 1–10.

Luo, J., Jha, N.K., 2000. Power-conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems. In: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design. pp. 357–364.

Luo, J., Jha, N.K., 2002. Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems. In: Proceedings of the 2002 Asia and South Pacific Design Automation Conference. IEEE Computer Society, p. 719.

Luo, J., Jha, N.K., 2003. Power-profile driven variable voltage scaling for heterogeneous distributed real-time embedded systems. In: 16th International Conference on VLSI Design, 2003. Proceedings. IEEE, pp. 369–375.

Luo, J., Jha, N.K., 2007. Power-efficient scheduling for heterogeneous distributed real-time embedded systems. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 26 (6), 1161–1170.

Martin, S.M., Flautner, K., Mudge, T., Blaauw, D., 2002. Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. In: IEEE/ACM International Conference on Computer-Aided Design. pp. 721–725.

Mo, L., Zhou, Q., Kritikakou, A., Liu, J., 2022. Energy efficient, real-time and reliable task deployment on noc-based multicores with DVFS. In: 2022 Design, Automation & Test in Europe Conference & Exhibition. DATE, IEEE, pp. 1347–1352.

Qiu, M., Ming, Z., Li, J., Liu, S., Wang, B., Lu, Z., 2012. Three-phase time-aware energy minimization with DVFS and unrolling for chip multiprocessors. J. Syst. Archit. 58 (10), 439–445.

Roeder, J., Rouxel, B., Altmeyer, S., Grelck, C., 2021. Energy-aware scheduling of multi-version tasks on heterogeneous real-time systems. In: Proceedings of the 36th Annual ACM Symposium on Applied Computing. pp. 501–510.

Roy, S.K., Devaraj, R., Sarkar, A., 2022. SAFLA: Scheduling multiple real-time periodic task graphs on heterogeneous systems. IEEE Trans. Comput. 72 (4), 1067–1080.

Schmitz, M.T., Al-Hashimi, B.M., 2001. Considering power variations of DVS processing elements for energy minimisation in distributed systems. In: Proceedings of the 14th International Symposium on Systems Synthesis. ACM, pp. 250–255.

Schmitz, M.T., Al-Hashimi, B.M., Eles, P., 2002. Energy-efficient mapping and scheduling for DVS enabled distributed embedded systems. In: Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings. IEEE, pp. 514–521.

Singh, A.K., Kumar, A., Srikanthan, T., 2013. Accelerating throughput-aware runtime mapping for heterogeneous MPSoCs. ACM Trans. Des. Autom. Electron. Syst. (TODAES) 18 (1), 9.

Tariq, U.U., Ali, H., Liu, L., Hardy, J., Kazim, M., Ahmed, W., 2021. Energy-aware scheduling of streaming applications on edge-devices in IoT-based healthcare. IEEE Trans. Green Commun. Netw. 5 (2), 803–815.

Tariq, U.U., Wu, H., Abd Ishak, S., 2020. Energy and memory-aware software pipelining streaming applications on NoC-based MPSoCs. Future Gener. Comput. Syst. 111, 1–16.

Teng, F., Yu, L., Liu, X., Lai, P., 2022. Tight lower bound on power consumption for scheduling real-time periodic tasks in core-level DVFS systems. Parallel Comput. 110, 102892.

Topcuoglu, H., Hariri, S., Wu, M.-y., 2002. Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. 13 (3), 260–274.

Wang, Y., Liu, H., Liu, D., Qin, Z., Shao, Z., Sha, E.H.-M., 2011. Overhead-aware energy optimization for real-time streaming applications on multiprocessor system-on-chip. ACM Trans. Des. Autom. Electron. Syst. (TODAES) 16 (2), 14.

Wang, Y., Liu, D., Qin, Z., Shao, Z., 2013a. Optimally removing intercore communication overhead for streaming applications on mpsocs. IEEE Trans. Comput. 62 (2), 336–350.

Wang, Y., Shao, Z., Chan, H.C., Liu, D., Guan, Y., 2013b. Memory-aware task scheduling with communication overhead minimization for streaming applications on bus-based multiprocessor system-on-chips. IEEE Trans. Parallel Distrib. Syst. 25 (7), 1797–1807.

Watanabe, R., Kondo, M., Imai, M., Nakamura, H., Nanya, T., 2007. Task scheduling under performance constraints for reducing the energy consumption of the GALS multi-processor SoC. In: DATE. pp. 1–6.

Xu, R., Melhem, R., Mosse, D., 2007. Energy-aware scheduling for streaming applications on chip multiprocessors. In: RTSS. pp. 25–38.

Yang, C.-Y., Chen, J.-J., Kuo, T.-W., Thiele, L., 2009. An approximation scheme for energy-efficient scheduling of real-time tasks in heterogeneous multiprocessor systems. In: Proceedings of the Conference on Design, Automation and Test in Europe. European Design and Automation Association, pp. 694–699.

Zhang, Y.-W., Chen, R.-K., 2022. A survey of energy-aware scheduling in mixed-criticality systems. J. Syst. Archit. 127, 102524.

Zhang, Y., Hu, X.S., Chen, D.Z., 2002. Task scheduling and voltage selection for energy minimization. In: Proceedings of the 39th Annual Design Automation Conference. ACM, pp. 183–188.