Energy-Aware Task Scheduling for MPSoC-based Embedded Systems

Author:
Abd Ishak, Suhaimi

Publication Date:
2018

DOI:
https://doi.org/10.26190/unsworks/20627

Energy-Aware Task Scheduling for MPSoC-based Embedded Systems

Suhaimi Abd Ishak

A thesis in fulfillment of the requirements for the degree of

Doctor of Philosophy

UNSW
AUSTRALIA

School of Computer Science and Engineering

Faculty of Engineering

The University of New South Wales

July 2018

Thesis/Dissertation Sheet

Surname/Family Name	:	**Abd Ishak**
Given Name/s	:	**Suhaimi**
Abbreviation for degree as give in the University calendar	:	**PhD**
Faculty	:	**Faculty of Engineering**
School	:	**School of Computer Science and Engineering**
Thesis Title	:	**Energy-Aware Task Scheduling for MPSoC-based Embedded Systems**

Abstract 350 words maximum: (PLEASE TYPE)

Energy reduction is a critical factor in designing embedded systems. One technique to reduce the total energy consumption of the systems is Dynamic Voltage and Frequency Scaling (DVFS). However, different models require different approaches to maximize its effectiveness in reducing energy consumption. In this thesis, we investigate the following three problems: energy-aware scheduling for applications with precedence and deadline constraints on homogeneous Multiprocessor Systems-on-Chips (MPSoCs), energy-aware scheduling for applications with precedence and deadline constraints on heterogeneous Network-on-Chip (NoC)-based MPSoCs and energy-aware scheduling for streaming applications on NoC-based MPSoCs. Firstly, we propose an energy-aware task scheduling approach to the problem of reducing the energy consumption for homogeneous MPSoCs assuming continuous frequencies under two power models: total dynamic power and total power. This approach uses a novel priority scheme for task assignment and a convex Non-Linear Programming (NLP) to assign an optimal execution frequency to each task. Secondly, we propose two energy-aware task scheduling approaches for heterogeneous MPSoCs considering discrete frequency model. Initially, both approaches use a heuristic to assign each task to a processor, and compute an optimal frequency for each task and message under the continuous frequency model using convex NLP. Based on the frequencies of each task and message under the continuous frequency model, the first approach uses Integer Linear Programming (ILP) to select an optimal discrete frequency, and the second approach uses a polynomial-time heuristic to select a discrete frequency for each task and message. Thirdly, we propose an energy-aware task scheduling approach for streaming applications on homogeneous MPSoCs considering discrete frequencies under memory capacity constraints. This approach integrates task-level software pipelining with DVFS and features a novel retiming technique to transform intra-period dependencies into inter-period dependencies considering the task mapping to enhance parallelism. It uses NLP and ILP to assign discrete frequencies to all tasks and messages. An iterative approach is employed to resolve memory capacity violation. We have implemented our proposed approaches and compared them with state-of-the-art approaches. Results indicate that the proposed approaches perform significantly better, in terms of total energy consumption.

FOR OFFICE USE ONLY Date of completion of requirements for Award:

ORIGINALITY STATEMENT

‘I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at UNSW or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at UNSW or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.’

Signed …….....

Date …….....

Copyright Statement

_____

Suhaimi Abd Ishak
July 24, 2018

Authenticity Statement

_____

Suhaimi Abd Ishak
July 24, 2018

Buat Mak dan Ayah
(To My Parents)

Acknowledgments

I would like to take this opportunity to appreciate many people who has supported me either directly or indirectly throughout my PhD life. This thesis would not have reached this point without their continuous support.

First of all, I would like to express my heartfelt gratitude to my supervisor Dr. Hui Wu for his invaluable advices, guidance and support. His patience and commitment are truly admirable. Also, thanks to my second supervisor, Prof. Dr. Boualem Benatallah for his time and suggestions during the early time of my PhD studies.

Not to forget, thanks to all my colleagues and friends in the university with whom I have exchanged helpful thoughts and feedback. I really appreciate the productive culture.

Thanks to my sponsors, the Ministry of Higher Education Malaysia and Universiti Tun Hussein Onn Malaysia for giving me this great opportunity to learn abroad, which helped me achieving my goals toward a successful career.

Last but not the least, I would like to thank my whole family members for their constant love and encouragement. Specially for my wonderful wife; Mrs Zainab Jaafar, my lovely children; Syamil, Syafi, Fatihah and Ariana, my parents; Hj Abd Ishak Maasab, Hjh Supiah Iksan, and my parents-in-law; Hj Jaafar Md Tap, Hjh Satariah Hasan, for their love and prayer. You are truly my source of inspiration and motivation.

Publications

- Suhaimi Abd Ishak, Hui Wu. Energy-Aware Task Scheduling with Precedence and Deadline Constraints on MPSoCs. The 18th IEEE International Conference on High Performance Computing and Communications (HPCC), 2016.
- Suhaimi Abd Ishak, Hui Wu, Umair Ullah Tariq. Energy-Aware Task Scheduling on Heterogeneous NoC-Based MPSoCs. The 35th IEEE International Conference on Computer Design (ICCD), 2017.
- Umair Ullah Tariq, Hui Wu, Suhaimi Abd Ishak. Energy-Aware Scheduling of Conditional Task Graphs on NoC-Based MPSoCs. The 51st IEEE Hawaii International Conference on System Sciences (HICSS), 2018.

Abstract

Energy reduction is a critical factor in designing embedded systems. One technique used to reduce the total energy consumption of the system is Dynamic Voltage and Frequency Scaling (DVFS). However, different system models require different approaches to maximize its potential and effectiveness in reducing the energy consumption. In this thesis, we investigate the following three problems: energy-aware task scheduling for applications with precedence and deadline constraints on homogeneous Multiprocessor Systems-on-Chips (MPSoCs) assuming continuous frequencies, energy-aware task scheduling for applications with precedence and deadline constraints on heterogeneous Network-on-Chip (NoC)-based MPSoCs considering discrete frequencies and the energy-aware task scheduling for streaming applications on NoC-based MPSoCs.

Firstly, we propose an energy-aware task scheduling approach to the problem of reducing the energy consumption for homogeneous MPSoCs assuming continuous frequencies under two power models: total dynamic power and total power. Our approach employs a novel priority scheme for task assignment and uses a convex Non-Linear Programming (NLP)-based algorithm to assign an optimal execution frequency to each task. We have evaluated this approach and compared it with two state-of-the-art approaches, LL-ES-GREEDY shorten as LESG [1] which considers the dynamic energy dissipation, and EES [2] which considers the total energy consumption, using a set of synthetic and real-world benchmarks. The experimental results indicate that the maximum, average and minimum improvements of our proposed approach compared to the LESG approach are 42.44%, 30.46% and 9.46%, respectively, and the maximum, average and minimum improvements of our proposed approach compared to the EES approach are 75.98%, 39.74% and 7.08%, respectively.

Secondly, we propose two energy-aware task scheduling approaches for heterogeneous Network-on-Chip (NoC)-based MPSoCs considering discrete frequency model. Initially, both approaches use a heuristic to assign each task to a processor, and compute an optimal frequency for each task and each message under the continuous frequency model using convex NLP. Based on the frequencies of each task and each message under the continuous frequency model, the first approach uses Integer Linear Programming (ILP)-based algorithm to select an optimal discrete frequency, and the second approach uses a polynomial-time heuristic to select a discrete frequency to each task and each message. We have implemented our approaches and compared them with two state-of-the-art ap-

proaches, ETFGBF [3] and CA-TMES-Search [4], using a set of synthetic benchmarks. Experimental results indicate that the maximum, average and minimum improvements of our proposed approach using ILP compared to the ETFGBF are 69.40%, 46.30% and 18.45%, respectively. The maximum, average and minimum improvements of our proposed approach using ILP compared to the CA-TMES-Search are 48.35%, 34.98% and 13.52%, respectively. Moreover, the performance of our proposed approach using the heuristic is very close to that of our proposed approach using ILP.

Thirdly, we propose an energy-aware task scheduling approach for streaming applications on homogeneous NoC-based MPSoCs considering discrete frequencies under memory capacity constraints. Our proposed integrated approach uses task-level software pipelining with DVFS. Initially, our approach constructs an initial schedule under maximum frequencies such that the workload across all the processors is balanced. Next, we employ a novel retiming technique to transform intra-period dependencies into inter-period dependencies considering the task mapping to enhance parallelism. Then, our approach uses NLP and ILP to assign optimal discrete frequencies to all the tasks and messages. An iterative algorithm is employed to resolve memory capacity violations. We have implemented this approach and compared it with two state-of-the-art approaches, RDAG+GeneS [5] and JCCTS [6], using a set of real and synthetic benchmarks. Experimental results suggest that the maximum, average and minimum improvements of our proposed approach compared to RDAG+GeneS are 40.82%, 17.31% and 7.53%, respectively. The maximum, average and minimum improvements of our proposed approach compared to the JCCTS are 46.46%, 21.67% and 10.75%, respectively.

Keywords

Embedded Systems, *MPSoC*, *NoC*, *Task Scheduling*, *Energy*, *Inter-processor Communication*, *Non-Linear Programming*,*Integer Linear Programming*, *Streaming Applications*, *Software Pipelining*, *Retiming*, *Heuristic*, *Polynomial Time*

Contents

Abbreviations

AET Actual Execution Time

DAG Directed Acyclic Graph

DPM Dynamic Power Management

DVFS Dynamic Voltage and Frequency Scaling

EDF Earliest Deadline First

ILP Integer Linear Programming

MPSoC Multiprocessor System-on-Chip

NLP Non-Linear Programming

NoC Network-on-Chip

WCET Worst-Case Execution Time

List of Figures

List of Tables

Chapter 1

Introduction

This chapter discusses the advantages of Multiprocessor System-on-Chip (MPSoC) in current applications of embedded systems, the importance of the problems investigated in this thesis and our major contributions to the problems. In addition, it provides the thesis structure.

Multiprocessor Systems-on-Chips (MPSoCs) are ideal and being increasingly used in complex embedded systems. An MPSoC is an integrated circuit consisting of multiple processors. It is designed for a specific application with special hardware and software. Examples of commercial MPSoCs include Samsung Exynos SoC [11] which powers the Samsung Galaxy smart phone series and Xilinx Zynq UltraScale+ MPSoC devices [7] which have been used in robots. MPSoCs have the potential to achieve the best trade-off among computational performance, energy efficiency, and costs, when designed carefully.

Current embedded system applications are typically complex and have precise real-time performance requirements. Applications such as high-speed data communication and multimedia not only require high performance but also require the implementations to meet strict design constraints such as time, energy and heat constraints. Multiprocessor architectures provide parallel processing capability to handle concurrent application tasks in

real time to meet timing constraints. In contrast, a system with a single processor architecture is more suited to run a simple application. With parallel processing capability, an MPSoC could increase its throughput. In addition, with the whole system on the same chip, an MPSoC reduces communication overheads among its components.

An MPSoC also consumes low energy. With all the components are on the same chip and tightly integrated, the required power supply is small. Moreover, it can conserve the energy in many ways and at all levels of abstraction i.e. on device, circuit, and logic levels. Dynamic Power Management (DPM) reduces the energy consumption by switching off system components when they are idle. Dynamic Voltage and Frequency Scaling (DVFS) reduces energy consumption by lowering supply voltage and frequency. By disabling portions of a circuit so that the flip-flops in them do not have to switch states, clock-gating reduces parts of power consumption of the circuit [12]. An MPSoC uses specialized mechanisms such as specialized memory systems, specialized logic or application-specific instructions tailored for a specific application. Various optimisations can be employed to simplify the hardware, further reducing the energy consumption.

In addition, an MPSoC can reduce die area and improve system reliability through extensive analyses and optimizations on the hardware, including interconnects and interfaces as the requirements of its application are known at design stage. Furthermore, via low power design, an MPSoC can significantly reduce operating costs.

Embedded systems have a wide range of applications, from large industrial control systems, and autonomous mobile robots, to small wearable devices. Energy consumption of the system is a critical factor and an important consideration for system engineers when designing embedded real-time systems. Most systems are powered by batteries and are deployed in remote locations. In general, standalone systems that are autonomous in terms of energy are affected by the disparity between their energy needs and their available power resources. The number of battery powered embedded devices, as well as their complexity, continues to expand. Complex real-time applications require high computational

resources, with compute-intensive tasks usually deplete batteries faster. Consequently, the requirement for energy is increasing at a faster rate. These systems use electronic components, such as processors, that operate at increasingly high frequencies to increase their performance, at the expense of energy consumption. This is evident as the power required by new devices increases by 35% to 40% per annum but the capacity of batteries increases by only 10% to 15% per annum [13]. For MPSoC, the energy consumed by processors and communication subsystems consumes significant percentage of the energy consumed by the whole system [14].

Therefore, instead of attempting to increase the capacity of batteries, another direction to address this problem is to reduce the energy consumption of the system. An energy-aware embedded system increases its battery lifetime, due to the reduction in current drawn from the batteries.

Moreover, the cost of replacing battery packs is high when the batteries that power these systems fail. Therefore, it is not always practical to replace or recharge the battery packs. Even for an embedded system which has continuous power support connected directly to the electrical grid, it is desirable to minimize its energy consumption since it shall reduce the cost of operation.

Furthermore, a significant amount of the energy consumed is converted to heat. If this is not controlled, heating can cause defects in the electronic chips, resulting in the reliability of the system to be compromised [15]. Consequently, extra efforts and additional costs are needed to cancel out the heat such as attaching cooling subsystem to the embedded system.

While minimizing the energy consumption for real-time embedded systems is of great importance, the accuracy of the system must not be compromised. Besides, this must be balanced against the need for real-time responsiveness. The software of a real-time embedded system typically consists of a set of real-time application tasks with deadlines

and precedence constraints. Deadlines of application tasks are defined to guarantee the responsiveness of the system. Real-time systems can be grouped into three categories as follows [16].

- Hard real-time systems: an overrun in response time leads to catastrophic consequences. An example is an avionic control system, in which a failure may cause an accident.
- Firm real-time systems: the result produced by the corresponding task ceases to be useful as soon as the deadline expires, but the consequences of not achieving the deadline are not severe. An example is a weather forecasting system, in which an analysis is obsolete over time.
- Soft real-time systems: the use of results produced by a task with a soft deadline decreases over time after the deadline expires. It is tolerable, but not desired. An example is a video streaming application, when the deadline is not met, may not have serious impacts, but do affect the quality of the output.

There is communication among application tasks in the form of precedence constraints. These constraints enforce the ordering of application tasks. A child task requiring an output from its parent task cannot be executed until the completion of its parent task and upon receiving the output. Accordingly, both tasks cannot run in parallel on different processors. These dependencies need to be taken into account when attempting to minimize the energy consumption.

The execution of application tasks on processors or communication messages on communication subsystems, uses higher processing power than what is necessary needed. This is due to the processors and communication subsystems being idle during certain time intervals. Conceptually the operating frequency and its corresponding supply voltage of the processors and communication subsystems for executing a task or a message could be

lower provided that its deadline constraint is satisfied. The availability of the idle time intervals, also known as slack provides the opportunity and potential for a reduction in energy consumption.

Dynamic Voltage and Frequency Scaling (DVFS) is a powerful technique used to reduce the energy consumption of real-time embedded systems. Modern processors and communication subsystems are equipped with DVFS. This technique facilitates the adjustment of supply voltage and operating frequency of the processors and communication subsystems to support the system requirements. However, the deployment of DVFS affects the execution time of application tasks and communication messages. This is due to the execution time being inversely proportional to the operating frequency. Therefore, the DVFS should be performed carefully with respect to the task scheduler, to ensure predefined constraints of the embedded application are maintained.

The challenge in scheduling a set of real-time tasks on multiprocessors is to determine where and when each task is executed. It is a well known Non-deterministic Polynomial time (NP)-complete problem [17], which is intractable as there may not exist an algorithm which can find an optimal solution within polynomial time. As a result, previous studies focus on heuristics to address the computational complexity. Considering energy reduction on top of the scheduling adds a new dimension to these design issues.

Numerous approaches have been proposed to construct an energy-efficient task schedule. In general, the approaches can be classified into two categories, those are: offline approaches and online approaches. The main difference between these two categories is that the offline scheduling approach constructs a schedule for a set of application tasks during the design stage, while an online scheduling approach schedules tasks during the runtime. In offline scheduling, timing information of all application tasks is made available during the design stage, thus enabling the system performance and energy needs to be predicted before implementation. The actual execution time (AET) of each task cannot be predicted in advance, resulting in the offline scheduling approaches to use the worst-case execution

time (WCET), which is defined as the maximum length of time a task could take to run under maximum frequency on a processor.

The energy-aware task scheduling problem involves three major sub-problems as follows:

- Mapping: determine the assignment of each application task to a processor and communication messages to communication links.
- Scheduling: determine the execution sequence of application tasks i.e. the start time of each application task on its assigned processor and communication messages on communication links.
- Slack allocation and frequency assignment: determine the amount of processor slack to be given to tasks or communication slack to be given to communication messages. Also, to assign an appropriate operating frequency and its corresponding supply voltage to each task and each communication message for slack reclamation.

It is important to map each task to an appropriate processor. For distributed systems, the mapping has a significant impact on the inter-processor communication overhead. Besides, the combination of mapping and scheduling have large impacts on the creation of reusable slack thus achievable energy savings. There are two kinds of slack available during the scheduling of a real-time system: static slack and dynamic slack. Static slack is produced as a result of under-utilized system workloads, assuming all the tasks are executed at the worst-case execution time. Dynamic slack is produced as a result of variations in execution times of the tasks i.e. when a task completes earlier than expected during the runtime. An offline scheduling approach is used to claim static slack, and an online scheduling algorithm is used to claim dynamic slack. Furthermore, there are two passes in selecting an operating frequency for each application task and each communication message. For a scheduling approach assuming the continuous frequency model, a task may be assigned with any frequency within a specific range of frequencies. For a scheduling approach

considering discrete frequency model, a task can only use a specific frequency which may not be able to claim the whole slack.

It is also important to compute the right continuous frequency or discrete frequency for each task and each message during slack reclamation. This is because lowering the frequency and its corresponding supply voltage indefinitely may not be beneficial in reducing the total energy consumption and affects the feasibility of the schedule.

The problem of energy-aware task scheduling is not only applicable to embedded systems but also to other general purpose systems or domains, e.g. high-performance computers, cloud computing and grid computing. This thesis focuses the discussion on embedded systems. Embedded systems must not only implement the desired function, but must also satisfy a diversity of constraints, such as power consumption, performance, safety, size, cost, and flexibility, which usually compete with each other.

Therefore, with all the aforementioned facts, it is an important research problem to optimize the total energy consumption of the processors and communication subsystems of the MPSoCs. Moreover, there is no one-size-fits-all solution to the energy-aware task scheduling problem, since the requirements of each system model are different.

We address three different problems in this thesis as follows.

1. Energy-aware task scheduling for a set of application tasks with precedence and deadline constraints on a homogeneous MPSoC assuming continuous frequency model.

2. Energy-aware task scheduling for a set of application tasks with precedence and deadline constraints on a heterogeneous NoC-based MPSoC considering discrete frequency model.

3. Energy-aware task scheduling for a set of periodic application tasks with precedence and deadline constraints on a homogeneous NoC-based MPSoC considering discrete frequency model.

We make the following major contributions in this thesis as follows.

1. Firstly, we propose a unified approach to the problem of scheduling a set of application tasks with precedence and deadline constraints on a homogeneous MPSoC assuming continuous frequencies such that the energy consumption of all the tasks is minimized under two power models specifically the total dynamic power and the total power. We present a novel priority scheme to construct an initial schedule under maximum frequencies. The priority of each task is its approximate successor-tree-consistent deadline, which is the approximate upper bound on its latest completion time in any feasible schedule for a relaxed problem where only the precedence constraints between the task and all its successors are considered. Furthermore, we present an approach by using a convex Non-Linear Programming (NLP)-based algorithm to assign an optimal supply voltage and frequency to each task. Furthermore, we present experimental results indicating the effectiveness of this approach in minimizing not only the total dynamic energy, but also the total energy consumption.

2. Secondly, we propose two unified approaches to the problem of scheduling a set of application tasks with precedence and deadline constraints on a heterogeneous NoC-based MPSoC considering discrete frequencies such that the total energy consumption of all the tasks is minimized. Both approaches use an iterative NLP-based algorithm for assigning and scheduling each task to a processor, and computing its optimal continuous frequency. Then, for the selection of discrete frequencies, one approach uses a novel Integer Linear Programming (ILP)-based algorithm to select an optimal discrete frequency for each task and each communication message. The second approach uses a polynomial time heuristic to select a discrete frequency for each task and each communication message. Moreover, we present experimental results showing that our proposed approaches perform significantly better than state-of-the-art approaches in terms of total energy consumption. Besides, the performance of our proposed approach using the polynomial time heuristic is very close to that of our

proposed approach using the ILP-based algorithm for computing discrete frequencies for all tasks and messages, and achieving a better running time.

3. Lastly, we propose a novel approach combining task-level software pipelining with DVFS to the problem of scheduling a set of periodic dependent application tasks on a homogeneous NoC-based MPSoC considering discrete frequencies such that the total energy consumption of all the tasks is minimized under memory capacity constraints. To the best of our knowledge, this study is the first that investigates this problem. This approach is supported by a set of novel techniques, which include constructing an initial schedule under maximum frequencies such that the workload across all the processors is balanced using a list-based scheduling, where the priority of each task is its approximate successor-tree-consistent deadline, a retiming approach based on the task mapping 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 iterative algorithm to resolve memory capacity violations. In addition, we correlate the problem to compute the memory usage of a schedule into the problem of Maximum Weight Clique which is NP-hard [18]. Also, we present extensive experimental results showing the effectiveness of this approach in terms of minimizing the total energy consumption while satisfying all the constraints.

This thesis is organized as follows. Chapter 2 presents the background. Chapter 3 gives a survey of the related work on energy-aware task scheduling. Chapter 4 describes our proposed unified approach for minimizing the total energy consumption of a set of application tasks with precedence and deadline constraints on a homogeneous MPSoC. Chapter 5 proposes our two approaches for minimizing the total energy consumption of a set of application tasks with precedence and deadline constraints on a heterogeneous NoC-based MPSoC. Chapter 6 explores the problem of minimizing the total energy consumption of

real-time streaming applications on a homogeneous NoC-based MPSoC under memory capacity constraints. Finally, Chapter 7 presents the conclusion and future work.

Chapter 2

Background

This chapter elaborates on the background knowledge of the Multiprocessor System-on-Chip (MPSoC) with Dynamic Voltage and Frequency Scaling (DVFS) and details the key challenges of the problems we investigate in this thesis.

An embedded system is a microprocessor-based system combining computer hardware and application software. It is designed to perform a specific function or a set of functions and interacts with its surrounding environment. A real-time embedded system has additional constraints such as precedence constraints and timing constraints, which are required to be satisfied in addition to the functional aspects, for the overall system to be considered correct. Embedded systems are widely deployed on small and large devices. Typical examples include applications in vehicles, airplanes or robots, as shown in Figure 2.1. Traditionally, an embedded system is powered by a single processor. The growing demands for more complex applications, has resulted in most modern embedded systems to be driven by multiprocessor architectures. Moreover, multiprocessor architectures such as Multiprocessor System-on-Chip (MPSoC) have received attention due to their high performance and low power requirements.

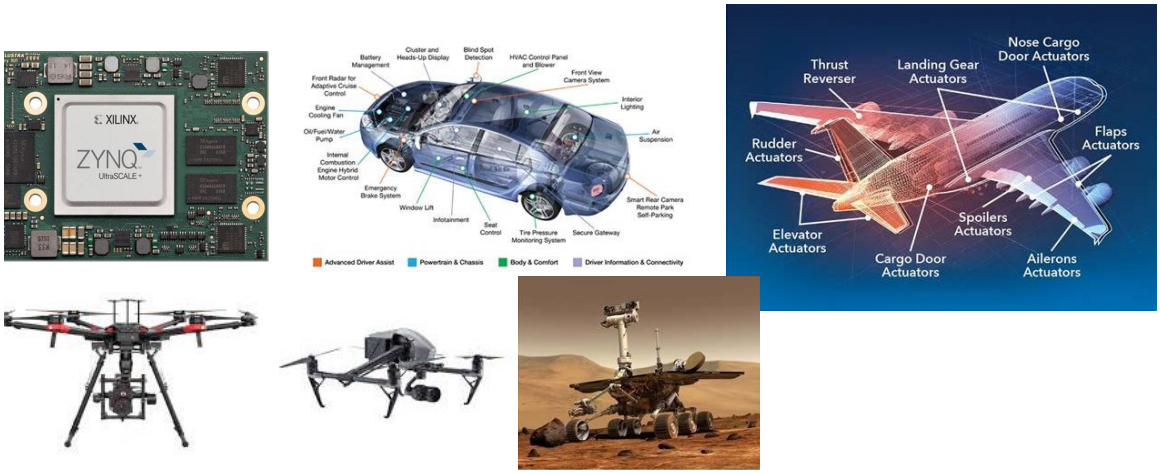


Figure 2.1: A Xilinx Zynq UltraScale+ MPSoC [7] and embedded system applications in automotive [8], avionics [9] and robotics [10].

2.1 Multiprocessor System-on-Chip (MPSoC)

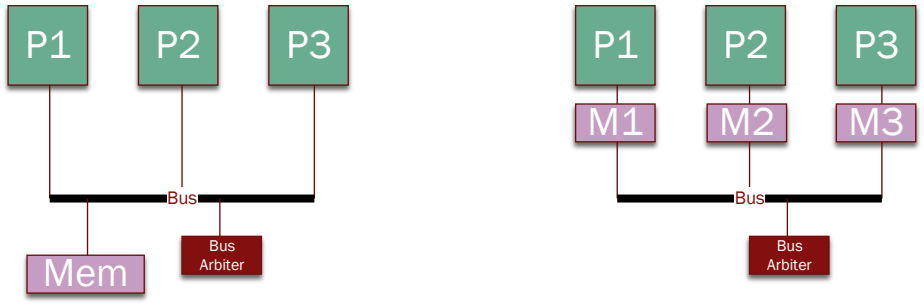
A Multiprocessor System-on-Chip (MPSoC) is an integrated circuit with multiple processors. However, it is not simply a traditional multiprocessor fitted into a single chip, but designed to complete the unique requirements of embedded applications in terms of computational performance, power efficiency and real-time performance. All the processors may implement the same or different Instruction Set Architecture (ISA). ISA defines the arithmetic and logic operations, data handling, memory operations, control flow and other necessary operations.

When all the processors implement the same ISA, it is considered a homogeneous architecture. All the processors have the same computational performance and power profile. However, when all the processors employ different ISA, it forms a heterogeneous architecture. The computational performance, as well as the power profile of all the processors are different.

MPSoCs can be classified into two distinct models as follows.

Shared Memory Model: This model enables all processors to share the same location in memory for read and write operations. Communication among processors is done by means of access to shared variables in the shared memory. The general layout of this model is shown in Figure 2.2a. The processors, represented by P1, P2 and P3 access a single memory (Mem) through a shared bus architecture.

Distributed Memory Model: This model relies on explicit message-passing among processors, each of which possesses a local private memory. Message-passing is performed through communication networks. The general layout of this model is shown in Figure 2.2b. The processors, represented by P1, P2 and P3 have access to their own local memory M1, M2 and M3, respectively, while communication among processors is achieved using a shared bus or communication links.



(a) General layout of a shared memory model.

(b) General layout of a distributed memory model.

Figure 2.2: Two distinct architecture models of multiprocessor embedded systems.

Most modern MPSoC processors are Dynamic Voltage and Frequency Scaling (DVFS)-enabled, which allows the adjustment of frequencies and corresponding supply voltages dynamically to reduce energy consumption.

2.1.1 MPSoC Communication Subsystems

All the processors in an MPSoC are connected through a system-on-chip interconnect. There are various types of MPSoC interconnect architectures such as single shared-bus, crossbar and Network-on-Chip. Traditionally, all processors are connected via a shared bus. However, single shared-bus architecture can quickly turn into a bottleneck. In addition, this architecture faces scalability issues. An improved crossbar architecture can guarantee maximum bandwidth, but large crossbars run into a problem known as spaghetti wiring issues [19], which hinder the achievable frequency of operation and pose severe physical design problems. Furthermore, large crossbars are expensive to implement. Network-on-Chip (NoC) is inspired from the wide area network which is based on packet-switched paradigm. NoC offers virtually unlimited scalability, higher bandwidth and also allows customization. NoC is composed of three main components as follows.

- Communication links: A communication link connects two routers in the network. It consists of one or more channels, each of which is composed of a set of wires.
- Router: A NoC router is composed of a set of global input and output ports which connect to other routers, a switching matrix connecting the input ports to the output ports, and local input and output ports to connect to the processor attached to the router. It includes several policies for data communication, such as flow control, routing algorithms and buffering policies.
- Network interface: This module provides a logical, as well as physical connections between a processor and the network.

A two-dimensional $|r|$-by-$|c|$ NoC architecture consists of $|r||c|$ number of processors each of which associated with a router, where $|r|$ and $|c|$ are the number of rows and number of columns, respectively. A router has at most five ports: one local port and four global ports. A communication link connecting between two routers is known as a global link

while a communication link connecting a router to a processor is known as a local link. An example of a 3-by-3 mesh NoC is shown in Figure 2.3 which has nine processors $\{P1, \cdots, P9\}$ each with its Cartesian coordinate, network interfaces (NI) and routers (R).

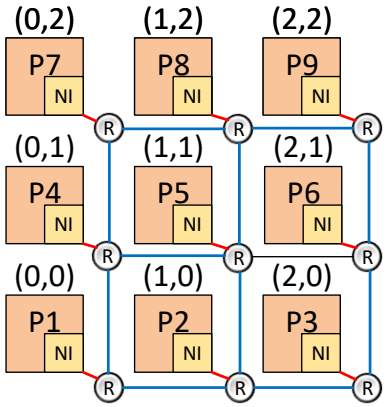


Figure 2.3: A diagram of a two-dimension NoC.

From the programmer's perspective, a communication message is needed to relay an output of a source task to a destination task if both the source task and destination task reside on different processors. It is first generated after the completion of the source task. Then, it is transmitted from the source processor where the application task is executed, to its associated router via a network interface (NI). On the router, the communication message is kept in a buffer for servicing. The router makes a routing decision i.e. the path to the destination based on its policies. Next, the communication message moves to the next router. These steps are repeated until the communication message reaches the destination processor. The communication latency much depends on the characteristics of the application such as the message data size and the network characteristics such as the network bandwidth and the router buffer size [20] as well as the communication traffic in the network.

There are two major policies on how NoC works on servicing a communication message: switching and routing.

Switching: A switching mechanism determines how and when an input channel is con-

nected to an output channel for data transmission. Switching mechanisms can be classified into two categories: circuit switching and packet switching [21]. In circuit switching, the communication path is determined by a routing algorithm. Thus, a single communication message takes the same circuit or route. On the other hand, packet switching breaks a communication message into several fixed-sized packets, each containing header flit and data flit. Routing decisions are made separately for every packet. There are various packet switching strategies with the most popular are store-and-forward, virtual cut-through and wormhole described as follows [22].

Store-and-forward: In this strategy, the router stores the complete packet before forwarding it to the next router in the path. The buffer size at each router should be sufficient to keep the whole packet. Otherwise, the packet is stalled.

Virtual cut-through: In this strategy, a router first checks whether a whole packet can be accepted by the next router. If so, the router forwards the packet to the next router. Otherwise, the whole packet is stored in the current router buffer without blocking any communication links. This strategy requires significant router buffer size but has lower latency and higher link utilization.

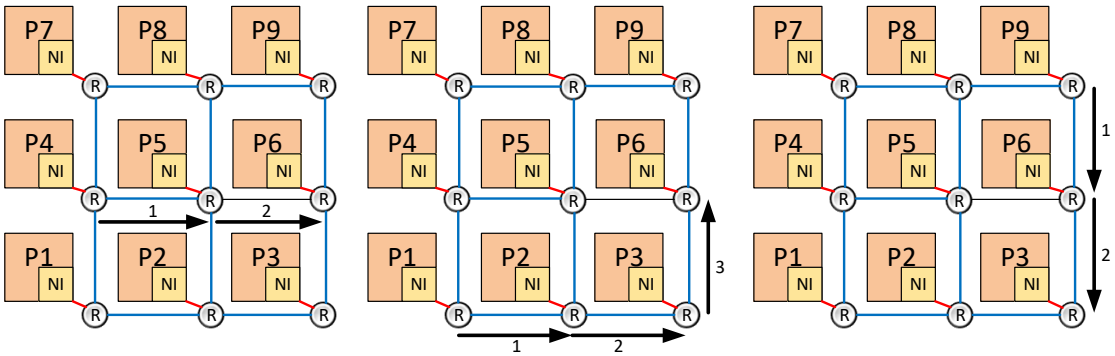
Wormhole: In this strategy, a router immediately forwards a packet as soon as its header flit arrives and the subsequent flits follow i.e. worm its way through the network. The disadvantage with this strategy is that when contention occurs, a stalling packet blocks all the communication links that the worm spans, resulting in low link utilization.

Routing: A routing algorithm decides the path of a communication message from the source processor (via its associated router) to the destination processor (via its associated router). Consider the NoC architecture in Figure 2.3, to relay a message from P1 to P8, can go through the routers of $\{P1, P2, P5, P8\}$ or routers of $\{P1, P4, P7, P8\}$ or routers of $\{P1, P4, P5, P8\}$. Routing schemes can be classified into two categories: deterministic and adaptive [22].

Deterministic routing: In this scheme, packet always uses the same path between the source and destination routers. An example is XY routing. XY routing scheme works based on the Cartesian coordinates. If the source router and the destination router are on different X-axis and Y-axis, the packet traverses the X-axis first, then moves along the Y-axis towards the destination router. Otherwise, if both routers are on the same X-axis or Y-axis, the packet only traverses along the X-axis or Y-axis, respectively. Figure 2.4 shows examples of XY routing scheme. Figure 2.4a shows the path from P4 to P6, which only traverses along X-axis, Figure 2.4b shows the path from P1 to P6, which traverses both X-axis and Y-axis, and Figure 2.4c shows the path from P9 to P3, which only traverses Y-axis.

Adaptive routing: This scheme involves a dynamic evaluation of the link load and implies a dynamic load balancing strategy. In other words, different path between the source and destination routers may be used in case of the original path is congested.

Most NoCs employ deterministic routing in particular the XY routing because of its simplicity.



(a) Routing from P4 to P6. (b) Routing from P1 to P6. (c) Routing from P9 to P3.

Figure 2.4: Examples of XY routing.

Like processors, advanced communication subsystems of MPSoCs are also DVFS-enabled. This allows the reduction of not only the computational energy, but also communication energy of MPSoCs.

2.2 Real-Time Embedded Applications

The application software of an embedded system is tightly coupled with the system hardware. It contains all the programs, known as the firmware, to give functionality to the system hardware. This firmware is stored in a non-volatile memory and is not able to be modified by the end users.

The firmware can be extracted into a number of smaller programs, called tasks. Each task performs a specific function in the system and requires the use of hardware resources. The set of application tasks can be represented as a Directed Acyclic Graph (DAG), also known as task graph. A vertex in the graph denotes a task and a directed edge between two vertices represents the dependency and precedence constraint between the two corresponding tasks. Figure 2.5 shows an example of the software extraction into a task graph, with t_1, t_2, t_3 and t_4 representing the tasks.

One important feature emerging from this software extraction is the potential parallelism between application tasks. In a multiprocessor architecture, application tasks that are not restricted by dependencies can be executed in parallel, resulting in better performance.

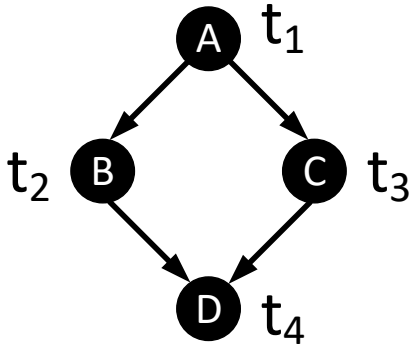
Typical parameters used for real-time application software are as follows.

Release Time: This parameter defines the time at which an application task is ready for execution.

Deadline: This parameter defines the time after a triggered event by which an application task should complete its execution.

```
1   #include <iostream>
2
3   using namespace std;
4
5   int main()
6   {
7       cout << "Hello world!" << endl;
8       N = 1000;
9       for (i=0; i<N; i++)
10      {
11          A[i] = i * 300;
12          B[i] = A[i] - 2;
13          C[i] = A[1] + 1;
14          D[i] = B[i] + C[i];
15      }
16      return 0;
17  }
```

(a) An example of application codes.



(b) An example of a task graph.

Figure 2.5: Examples of application codes and the corresponding task graph.

Period: This parameter defines the exact inter-arrival time between successive invocations of an application task.

Worst-Case Execution Time: This parameter refers to an estimated maximum execution time required to complete an application task under maximum frequency, assuming that its execution is not preempted. It is expressed in the same units as deadline and period.

2.2.1 Task Model

A task is a sequential program that is triggered for execution by the occurrence of a particular event. Several task models are described as below.

Precedent-Tasks Model: This model dictates the ordering of tasks. A set of tasks under this model can be represented as a weighted Directed Acyclic Graph (DAG) or task graph, in which a vertex denotes a task and a directed edge is added between two tasks to denote the precedence constraint between the two tasks.

Periodic Tasks Model: A periodic task is a continuous stream of task instances initi-

ated at a regular time interval. Each task has an exact inter-arrival time between its successive instances which is defined by its period.

Sporadic Tasks Model: A sporadic task is a continuous stream of task instances. The period defines the minimum inter-arrival time between its consecutive instances, thus removing the restriction of generating instances at a regular time interval.

Aperiodic Tasks Model: An aperiodic task model does not have a period parameter.

A task is known as a constrained-deadline task if the period is greater than or equal to its deadline. It is known as implicit-deadline task if the period is equal to its deadline.

The demand for more complex systems requiring higher power has increased significantly over recent years. However, the development of the capacity of batteries has lagged behind. Therefore, it is important to minimize the energy consumption required by embedded systems. We elaborate on the general framework for energy-aware task scheduling and two major techniques: the Dynamic Voltage and Frequency Scaling (DVFS) and task-level software pipelining which aid in the conservation of energy consumption in the following sections.

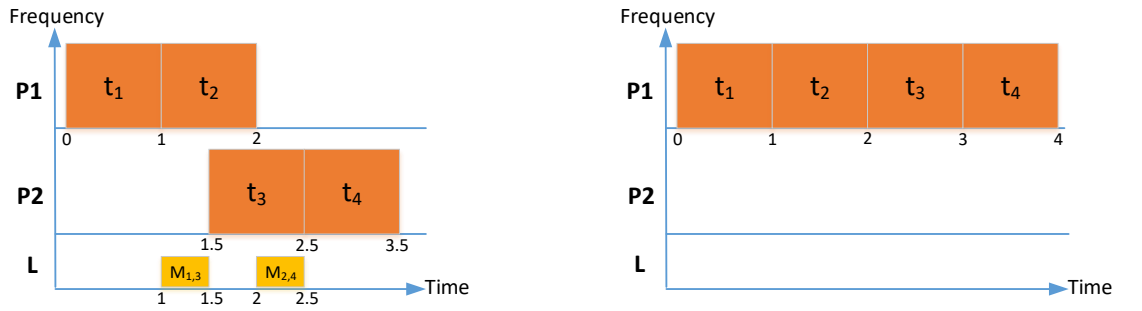
2.3 Energy-aware Task Scheduling

In general, the problem of energy-aware scheduling for a set of tasks with precedence and deadline constraints involves three common sub-problems as follows.

Task mapping: determine the assignment of each application task to a processor and communication messages to communication links.

Task scheduling: determine the execution sequence of application tasks i.e. the start time of each application task on its assigned processor and communication messages on communication links.

Slack allocation and frequency assignment: determine the amount of processor slack to be given to tasks or communication slack to be given to communication messages. Using Dynamic Voltage and Frequency Scaling (DVFS), assign an appropriate operating frequency and its corresponding supply voltage to each task and each communication message for slack reclamation.



(a) Mapping option 1.

(b) Mapping option 2.

(c) Mapping option 3.

(d) Mapping option 4.

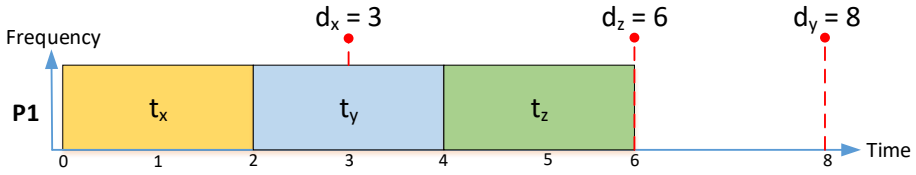
Figure 2.6: Several mapping options of a task graph in Figure 2.5b on two processors.

For a uniprocessor architecture, only task scheduling and slack allocation and frequency assignment are required. A multiprocessor architecture requires all three sub-problems. Consequently, the quality of an energy-aware schedule is dependent on three major factors: the mapping of application tasks to the processors, the ordering of the application tasks on processors, and the selection of frequencies and supply voltages of all the application tasks. Each of this sub-problem brings about its own challenges.

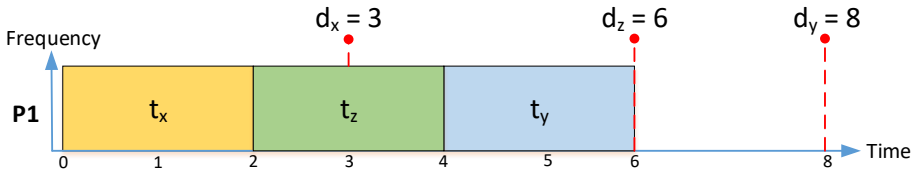
Consider a simple scenario to map an application consisting four dependent tasks as in Figure 2.5 on two processors. The worst-case execution time of each task is one time unit. Each task has two available processors for its execution, thus the design space of this simple scenario is $2^4 = 16$. The design space grows with increasing number of tasks and processors. Figure 2.6 depicts Gantt charts of four possible mappings of the simple scenario. The height of each task and each message resembling Y-axis represents its operating frequency while X-axis represents the time. A resource label with a letter P denotes a processor and a resource label with a letter L denotes a communication link. In case the common deadline of the application is 3.5 time unit, solutions in Figures 2.6b, 2.6c and 2.6d are infeasible. Furthermore, the figures also show that different mappings incur different inter-processor communication overhead. Moreover, the mapping affects the availability of slack. If the common deadline of the application is 4 time unit, only mapping in Figure 2.6a has a static slack of 0.5 which can be utilized for frequency scaling. Hence, an efficient algorithm is needed to obtain a good mapping.

Furthermore, the order of tasks also affects the feasibility of the schedule as well as the potential for reducing the energy consumption. Consider a scenario with three tasks t_x, t_y and t_z each of which has an individual deadline of 3, 8 and 6 time units, respectively. All the three tasks have the same worst-case execution time of two time units and they have been mapped to a processor P1. Figure 2.7 shows two possible orderings of the scenario each having different energy reduction potential. On one hand, Figure 2.7a depicts that the available slack cannot be utilized by all the three tasks because t_z has reached its deadline. Tasks t_x and t_y are blocked from stretching their executions. On the other hand, Figure 2.7b depicts a situation which allows all the three tasks to slowdown their executions until their deadlines to reduce the energy consumption.

Considering each sub-problem separately may produce a suboptimal solution. Therefore, a unified algorithm is needed to solve the energy-aware task scheduling problem.



(a) Ordering option 1.



(b) Ordering option 2.

Figure 2.7: Two possible task orderings.

2.4 Dynamic Voltage and Frequency Scaling (DVFS)

Dynamic Voltage and Frequency Scaling (DVFS) is a powerful technique for reducing the energy consumption of real-time embedded systems. This technique was originally designed for a uniprocessor architecture, but later extended to multiprocessor platforms. It is widely used in modern processors, as well as communication subsystems. Some examples of processors with DVFS capability are Intel Speedstep, Marvells XScale R technology-based embedded processors, ARM, AMD PowerNow! and Transmeta Crusoe. The fundamental idea behind this technique is based on two properties as follows.

- The dynamic power consumption of CMOS circuits have a direct relationship with the square of the supply voltage multiplied by the operating frequency.
- The processors or communication subsystems may be idle during certain time intervals, also known as slack, which can be utilized to slowdown the execution of application tasks or communication messages.

Enabling the adjustment of the frequency and supply voltage based on the system re-

quirements over time provides an opportunity to conserve energy. However, lowering the operational frequency and supply voltage of an application task or communication message indefinitely may not be beneficial in reducing the energy consumption due to several reasons as follows.

Critical frequency: There exists a minimum frequency, which is known as the critical frequency, for each application task or each communication message. Previously reported studies demonstrate the existence of critical frequency for an application task beyond which the processor slowdown is no longer beneficial. Not only the performance of the task degraded, but its energy consumption is also increased [23].

Leakage energy: A longer execution time which decreases the dynamic energy consumption, increases the leakage energy. Leakage current, which is the source of static power, is increasing with the advances of CMOS technology miniaturization. It is predicted that the static power increases five fold over each generation of technology [24]. Hence, it is important to take into account the existence of static energy.

The power consumption of a CMOS circuit can be categorised into three types which are dynamic power, static power and short-circuit power [25]. Dynamic power resulting from switching activities within the circuit and is approximated to be proportional to the square of the supply voltage multiplied by the operating frequency. Static power is due to the leakage current that exists even in the absence of switching activities in the circuit. Short-circuit power is dissipated during signal transitions which is small, and therefore negligible.

The selection of a frequency and supply voltage for an application task and a communication message has a direct impact on its execution time and communication time, respectively. Hence, DVFS needs to be tightly coupled to a task scheduler to ensure the feasibility of the application task.

2.4.1 Task Mapping and Scheduling

One of the key challenges to this problem is to generate a feasible task schedule with minimum energy consumption in accordance with all the constraints. A schedule is feasible if the precedence and deadline constraints are satisfied. The schedule can then be generated by compilers and statically loaded into processors or integrated into real-time operating systems. Without considering the energy, task scheduling is already a well known Non-deterministic Polynomial time (NP)-complete problem [17]. This means the problem is intractable as there may not exist an algorithm which can find an optimal solution within polynomial time.

Task scheduling on a multiprocessor architecture poses more challenges than task scheduling on a single processor. Not only does it need an additional step for task-to-processor mapping, but also more difficult to be solved as the design space is larger. Moreover, the precedence and deadline constraints limit the flexibility in scheduling all the tasks.

Therefore, in order to guarantee the feasibility of the schedule, common approaches tend to firstly construct an initial schedule under the maximum frequencies. If there exists a feasible schedule, the next step is to claim available slack with frequency scaling for energy reduction.

Task mapping: Given a set of tasks with precedence and deadline constraints, each task needs to be mapped to a specific processor. Different task mappings lead to different schedules with different amounts of slack available. Poor task mapping may lead to an infeasible schedule and impact the potential for reducing the energy consumption. Therefore, an efficient algorithm is required to map each task to a processor such that a feasible schedule can be constructed.

Task scheduling: For each set of tasks mapped to a specific processor, a feasible schedule satisfying all the timing constraints needs to be constructed. The sequence of tasks

impacts the feasibility and the potential for reducing the energy consumption.

Offline task scheduling can be grouped into two categories: heuristic-based and guided random search-based algorithms. Heuristic-based algorithms can be further grouped into three categories as follows [26].

List-based Scheduling: A heuristic in this group involves a step to prioritize all the application tasks. It maintains a list of all the tasks according to their priorities. It then schedules each task on a processor and a time slot based on its readiness and priority.

Clustering-based Scheduling: A heuristic under this group assigns all the application tasks to a number of clusters. All the tasks in the same cluster are mapped to the same processor. All the tasks are then ordered on their respective processors.

Duplication-based Scheduling: This heuristic makes several copies of a task which are executed on different processors in order to remove the inter-processor communication overhead.

Guided Random Search-based Scheduling: An algorithm under this group improves the scheduling result iteratively until it satisfies a stop criteria. It uses the knowledge gained from previous search steps and make a reconfiguration to generate a new solution according to random features. It then accepts a scheduling result based on a predefined metric. Examples include the Genetic Algorithm (GA) and Simulated Annealing (SA). These algorithms may produce a good scheduling result. Nevertheless, their time complexities are much greater than heuristic-based scheduling and the control parameters, that lead to the best solution need to be determined accordingly.

Another major challenge in task scheduling is the communication message scheduling. A communication message is required to pass the data of a parent application task to a

child application task if both tasks are mapped on distinct processors. Different from task scheduling which assigns an application task to a processor, message scheduling may assigns a message on several communication links. This is because a message may traverse several communication links resembling a path to reach its destination. Hence, a message should be scheduled on all the links on its path following the order. Also, the correctness of the message schedule depends on causality conditions [21] as follows.

- The start time of a message on its subsequent links should not be earlier than its start time on the current link.
- The finish time of a message on its subsequent links should not be earlier than its start time on the current link.

Figure 2.8 shows a simple scenario for task and message scheduling without contention. An application task t_1 on processor $P1$ transmits a message $M_{1,3}$ to an application task t_3 on processor $P6$ on a NoC platform as in Figure 2.3. Hence, the message traverses links $L_{1,2}$, $L_{2,3}$ and $L_{3,6}$ in which the subscript indexes denote the source processor (via its associated router) and destination processor (via its associated router).

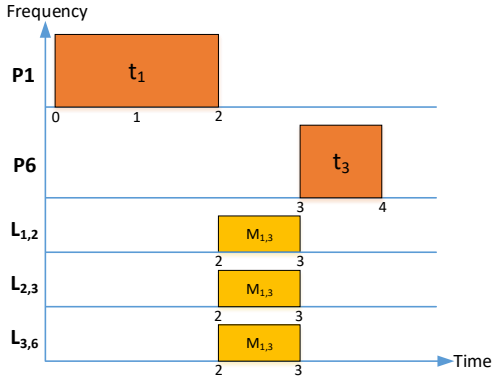


Figure 2.8: Communication message scheduling without contention.

Communication contention may exists when a message encounter other messages along its path. There are two major issues that need to be handled in contention-aware scheduling.

Firstly, among the contended messages, which one should be given the higher priority for accessing the network resources i.e. communication link. Secondly, on how to determine the start time and finish time of those messages.

There are two common models for the determination of the scheduling times on a path in contention-aware scheduling: non-aligned model and aligned model [21] described as follows.

Non-aligned model: In this model, the start times and finish times of a message on all links of its path are not synchronize, but follow the causality conditions. The message is delayed on the link with contention. Figure 2.9a depicts a simple scenario using non-aligned model. It shows that the message $M_{1,3}$ on link $L_{1,2}$ starts at time 2, but on the subsequent link $L_{2,3}$ it is delayed by message $M_{x,y}$ thus forcing it to start at time 2.5.

Aligned model: In this model, the contention on all its link is resolved first. Then, its start times and finish times on all the links of its path are synchronized. Figure 2.9b depicts a simple scenario using aligned model. It shows that the start times of message $M_{1,3}$ on all links of its path are the same at time 2.5.

2.4.2 Slack allocation and frequency assignment

There are two types of slack: static slack and dynamic slack. Static slack is produced due to an under-utilized system workload assuming all the tasks are executed for the worst-case execution time. Dynamic slack is produced due to variations in the execution time of tasks i.e. when a task completes earlier than expected during runtime. There are two distinct schemes for slack reclamation which are inter-task DVFS and intra-task DVFS as below.

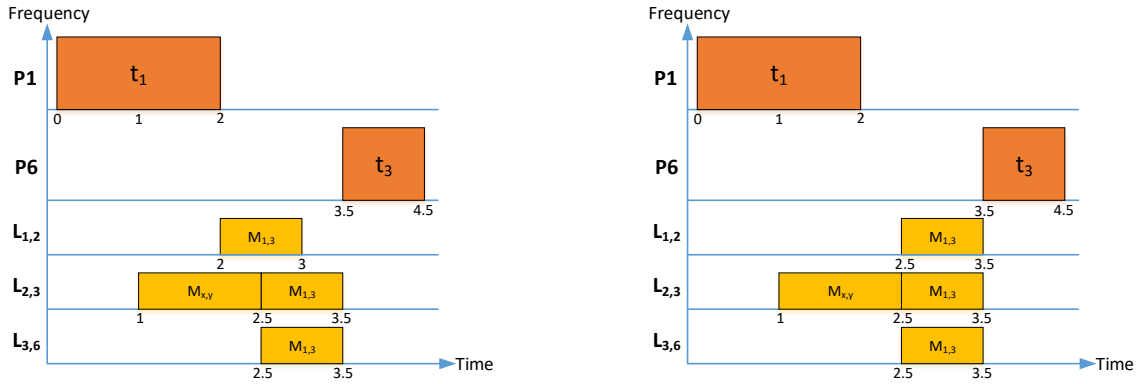


Figure 2.9: Contention-aware communication message scheduling: (a) non-aligned message scheduling model, and (b) aligned message scheduling model.

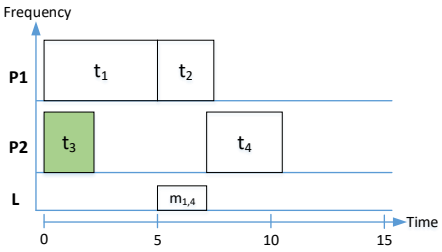
Inter-task DVFS Scheme

Inter-task DVFS scheme controls the frequency and supply voltage on a task-by-task basis. The scheme assigns a constant frequency and its corresponding voltage during the execution of each task. The transition energy overhead and transition time overhead occur between two tasks with different frequencies. Examples of this scheme are shown in Figure 2.10b, which assumes a continuous frequency model and Figure 2.10d, which assumes a discrete frequency model.

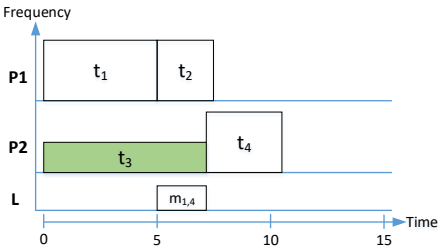
Intra-task DVFS Scheme

Intra-task DVFS scheme determines the frequency and supply voltage within an individual task boundary. This scheme assigns multiple frequencies during the execution of a task. However, it has been shown that at most two frequencies and supply voltages minimize the energy consumption of a single task under a timing constraint [27]. This scheme is efficient, in terms of energy reduction, for a system with a single task and when the execution time of any one task dominates the whole execution time of all tasks [28]. Nevertheless, the impact

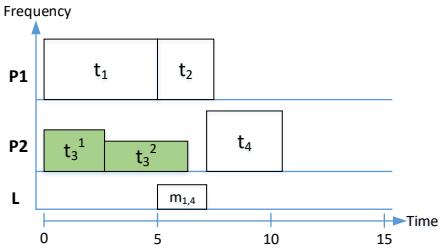
of enabling the adjustment of frequency within task executions may increase transition energy overhead, as well as the transition time overhead, which occur during the change between frequencies.



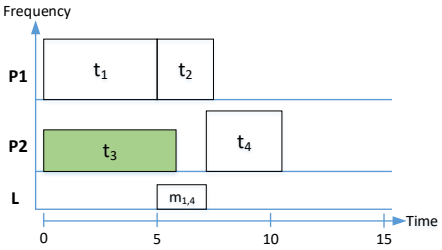
(a) An initial schedule.



(b) Lowering the frequency of t_3 under a continuous frequency model.



(c) Lowering the frequency of t_3 using combination of discrete frequencies i.e. intra-task scaling.



(d) Lowering the frequency of t_3 using a discrete frequency i.e. inter-task scaling.

Figure 2.10: Examples of DVFS approaches.

2.5 Task-level Software Pipelining

Task-level software pipelining is another technique used to supplement the reduction of energy consumption for real-time embedded systems. The potential of reducing the energy consumption can be increased by enhancing parallel processing on multiprocessors. This technique is applicable to periodic dependent tasks, which process a continuous stream of data repetitively. The fundamental idea of this technique is to change intra-period dependencies into inter-period dependencies by scheduling certain task instances to ear-

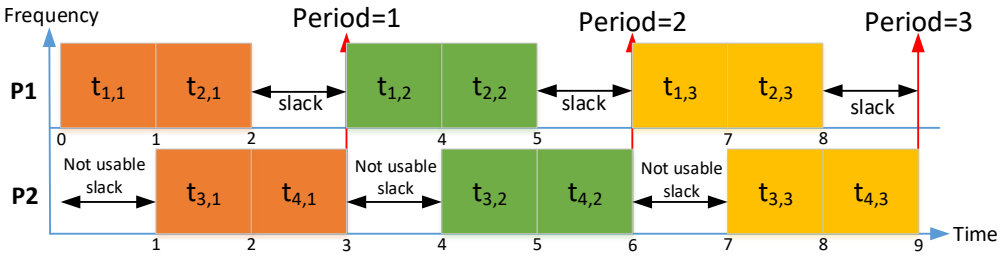
lier periods. Consequently, the precedence constraints among tasks within a period are removed, allowing some flexibility in using the available slack for frequency scaling. A technique to make this possible is known as retiming.

Retiming was first proposed to minimize the cycle period of a synchronous circuit [29]. It distributes registers evenly on a circuit while preserving the functional behaviour of the entire circuit. As for the problem of energy-aware task scheduling, it redistributes the instances of tasks to different periods, while ensuring the dependencies among the tasks are preserved. These dependencies then appear between different periods. With this, the amount of usable slack can be increased and utilized during frequency scaling. Examples are shown in Figure 2.11. Figure 2.11a shows a common initial schedule of the task graph in Figure 2.5b with its period equal to 3 time units on two processors. The subscript of each task instance denotes the task index followed by its instance index. The schedule produces only one time unit of slack within each period. After redistributing the task instances of $t_{1,1}$, $t_{1,2}$, $t_{2,1}$ and $t_{3,1}$ to earlier periods named prologue, the available slack become two time units, each on different processors. This implies that retiming may be beneficial in reducing the energy consumption.

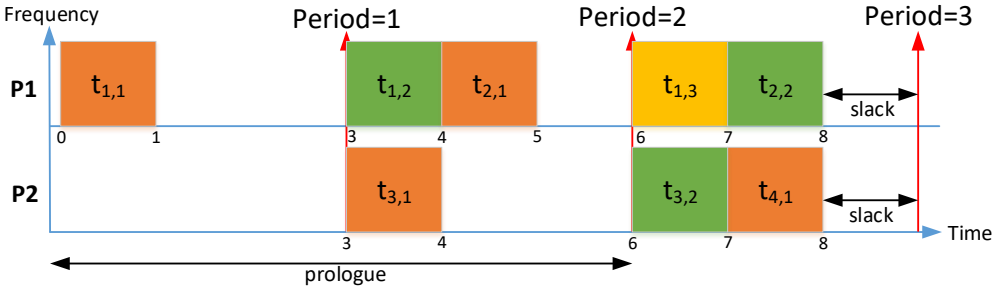
However, this technique comes with extra costs as follows.

Prologue latency: The latency is caused by rescheduling certain task instances to earlier periods. As shown in Figure 2.11b, the latency is the number of periods in the prologue multiplied by the period length which is equal to 6 time units.

Memory capacity overhead: Extra memory space is required to keep data caused by grouping different task instances from different periods into one period. Data may be kept for a longer time in memory as the sender task instance and the receiver task instance are on different periods. For example, in Figure 2.11, the time to keep the data of task instance $t_{2,1}$ to task instance $t_{4,1}$ in memory before the retiming is one time unit, which is computed as the difference between the completion time of a



(a)



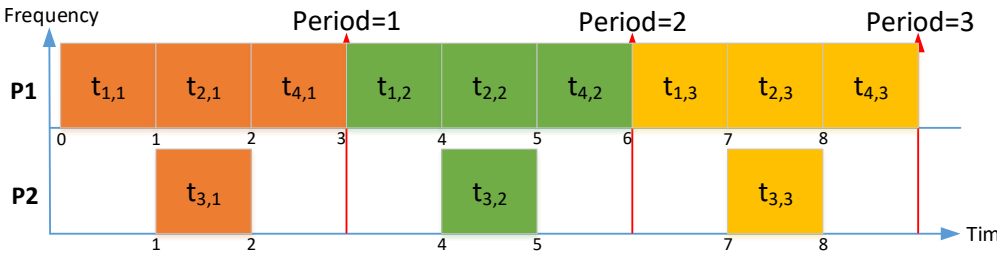
(b)

Figure 2.11: Examples of Scenario 1: (a) an initial schedule before retiming, and (b) the initial schedule after retiming.

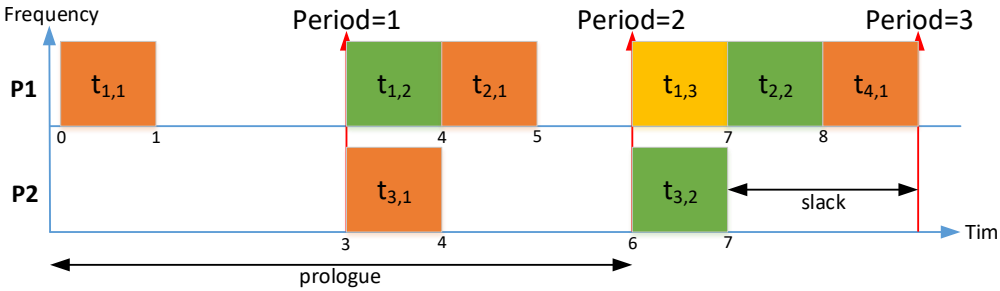
child task and the completion of a parent task. These times are the time when the data is produced by the parent task and the time when the child task has consumed the data. After retiming, it takes three time units.

The quality of the solution, in terms of increasing the parallelism under memory capacity constraints, depends on how the assignment of a retiming value for each task is done. The retiming value of a task represents the number of times it is executed in the prologue. In addition, the maximum retiming value of all the tasks can show the amount of latency incurred by the schedule, when using retiming.

The distribution of application tasks among processors has an influence on how well retiming increases the parallelism to reduce the energy consumption. Consider a different



(a) An initial schedule before retiming.



(b) An initial schedule after retiming.

Figure 2.12: Examples of Scenario 2: (a) an initial schedule before retiming, and (b) the initial schedule after retiming.

schedule as in Figure 2.12. Compared to Figure 2.11, task t_4 is now mapped on processor P1. The available slack after retiming under this scenario is also two time units, but only on processor P2. The energy consumption of processor P1 cannot be reduced due to the task execution reaching the limit.

As discussed above, the problems of energy-aware task scheduling for a set of tasks with precedence and deadline constraints on a homogeneous MPSoC, energy-aware task scheduling for a set of tasks with precedence and deadline constraints on a heterogeneous NoC-based MPSoC and energy-aware task scheduling for a set of periodic dependent tasks on a homogeneous NoC-based MPSoC pose many challenges which are difficult to address. This thesis proposes a set of efficient approaches to address these problems.

Chapter 3

Literature Review

This chapter provides a survey on the previous energy-aware task scheduling approaches focusing on applications with precedence and deadline constraints on multiprocessors. It discusses the key idea, results and limitations of each approach.

A considerable amount of literature has been published describing energy-aware task scheduling since Yao et al. [23] seminal work on this topic for a single processor assuming continuous frequencies. They consider the problem of scheduling a set of tasks with release times, deadlines and amount of works in clock cycles. Different from the traditional task scheduling studies, the solution is not only the time at which each task is scheduled, but also the frequency at which the task is executed such that the energy consumption is minimized. They propose a simple polynomial-time algorithm in which the optimality of its solution depends upon the convexity of the objective function.

This chapter divides the previous approaches into two broad categories based on the platform model, which are the energy-aware task scheduling on homogeneous multiprocessors, and energy-aware task scheduling on heterogeneous multiprocessors, each of which is further divided into two task models, namely aperiodic tasks with precedence constraints and periodic tasks with precedence constraints.

3.1 Energy-aware Task Scheduling on Homogeneous Multiprocessors

3.1.1 Aperiodic Tasks with Precedence Constraints

Zhang et al. [30] propose PEDF which integrates task scheduling with voltage scaling to minimize the total dynamic energy consumption. They consider a set of tasks with precedence and individual deadline constraints on a homogeneous multiprocessors platform with discrete voltage levels. They propose a two-phase framework. In the first phase, they attempt to construct an initial schedule under maximum frequencies with high potential during slack reclamation. To achieve this, tasks are prioritize based on the deadline, dependencies and usage of processors. Next, they employ an intra-task voltage scaling by using an Integer Linear Programming (ILP)-based algorithm. Results indicate that PEDF gives more opportunity for slowing down compared to an Earliest Deadline First (EDF) algorithm for multiprocessors. However, using an intra-task voltage scaling may introduce additional transition overhead which affect the feasibility of the schedule. Moreover, their study does not account for inter-processor communication overhead.

Mishra et al. [31] investigate the problem of slack reclamation for task schedules consisting of a set of dependent tasks with a common deadline to minimize the total dynamic energy consumption. They focus on homogeneous distributed multiprocessors assuming continuous frequency model. They propose a static slack allocation approach named P-SPM and two dynamic slack allocation approaches. P-SPM iteratively distributes static slack to different sections of the schedule according to the degree of parallelism. In each step, the heuristic distributes more slack to the sections with a greater degree of parallelism to reduce the total dynamic energy consumption. The results indicate that this approach can save an average 10% more energy compared to an approach that distributes static slack proportionally among the tasks. However, not all available static slack can be claimed by P-SPM, impacting the potential for reducing the total dynamic energy consumption.

Leung et al. [32] study the problem of minimizing the total dynamic energy consumption for a set of aperiodic dependent tasks with individual deadlines on homogeneous multiprocessors assuming ideal frequencies. They propose a unified approach for task mapping, scheduling and voltage assignment using Mixed Integer Non-Linear Programming (MINLP)-based algorithm. In addition, they propose a polynomial-time heuristic based on a divide-and-conquer strategy, due to the MINLP-based algorithm is not scalable. Simulation results suggest that these approaches perform better, in terms of total dynamic energy consumption compared to the PEDF approach [30]. However, their study also considers only the total dynamic energy consumption and assumes that communication time and communication energy are negligible.

Alexandru et al. [33] focus on the problem of allocating slack to a set of tasks with precedence and a common deadline constraints. They present four different voltage selection schemes to minimize the total computation energy consumption by using a combination of Dynamic Voltage Scaling (DVS) and Adaptive Body Biasing (ABB). The DVS reduces the dynamic power and the ABB reduces the static power. The first scheme assumes continuous frequencies without transition overhead. The second scheme considers continuous frequencies with transition overhead. The third scheme assumes discrete frequencies without transition overhead. The last scheme considers discrete frequencies with transition overhead. The first and second schemes which assume continuous frequencies to compute an optimal frequency for each task use Non-Linear Programming (NLP)-based algorithm. The third and fourth schemes which assume discrete frequencies use Mixed Integer Linear Programming (MILP)-based algorithm, to select an optimal discrete frequency for each task. However, they do not consider inter-processor communication overhead which may deteriorates the resultant schedule in terms of its feasibility.

Su et al. [2] study a similar problem of minimizing the total computation energy consumption for High Performance Computing (HPC) systems. They propose a three-phase approach consisting of task mapping under maximum frequencies, slack-room distribution

and slack reclamation. In the first phase, tasks are assigned to processors by using the HEFT algorithm [26]. In the second phase, the static slack between the makespan and common deadline is proportionally distributed across the tasks on the critical-path. During the last phase, the heuristic reallocates slack to tasks in a global manner and assigns an optimal operating frequency to each task. The experimental results indicate that the heuristic conserves more energy, compared to the greedy slack reclamation approach [34] and the path-based slack reclamation approach [35].

Han et al. [4] explore both offline and online contention-aware energy management schemes for homogeneous Voltage Frequency Island (VFI) and NoC-based multicore processors with discrete frequencies. They consider dependent tasks with a common deadline. They propose two offline heuristics: CA-TMES-Quick and CA-TMES-Search. CA-TMES-Quick initially constructs an initial schedule under maximum frequencies by considering the worst-case traffic congestion. It uses a list-based scheduling where the priority of each task is its longest path to a sink task while the priority of each communication message is its ready time and its longest path to a sink task. The CA-TMES-Search exhaustively search for the best task and message schedule in terms of makespan but has greater time complexity compared to CA-TMES-Quick. After task assignment, both heuristics adopt a uniform slowdown strategy which assigns the same low frequencies to all processors and communication links. Results suggest that these offline approaches perform better, in terms of energy consumption depending on the factor of network congestion, compared to an approach which employs task mapping based on Integer Non-Linear Programming (INLP).

Li [1] studies a similar problem for clouds and data centers. He considers two problems: minimizing the total dynamic energy consumption with a deadline constraint and minimizing the schedule length with an energy constraint. The target platform is a homogeneous multiprocessors with continuous frequencies. For the problem of minimizing the energy consumption of applications with precedence and a deadline constraints, he

presents LL-ES-GREEDY, shorten as LESG heuristic. This approach schedules as many tasks as possible for simultaneous executions and is based on level-by-level scheduling technique. Firstly, it partitions the task graph into levels. All tasks at the same level are independent of each other. Secondly, each level is allocated a time slot such that all the tasks at the same level are executed within the time slot. Lastly, all the tasks on each level are assigned with the same frequencies as long as the deadline is met. Analytical results indicate that this approach can minimize the total dynamic energy consumption. However, this study does not come with simulation results and it works only to reduce the total dynamic energy consumption.

Li and Wu [3] deal with the problem to minimize the total energy consumption of a set of dependent tasks with a common deadline on a homogeneous NoC-based MPSoC with discrete frequencies. They propose a two-phase approach. First, they employ a Quadratic Programming (QP)-based algorithm to map each task to a processor, with the aim of minimizing the total weighted distance of communication among the tasks. The total weighted distance from task t_i to task t_j is computed as $L_{i,j}c_{i,j}$, where $L_{i,j}$ is the distance while $c_{i,j}$ is the communication data size between t_i and t_j. Genetic Algorithm (GA) is then used to schedule each task on its mapped processor and assign a discrete frequency to each task and each message. The GA iteratively constructs a schedule using a list-based scheduling with an Earliest Task First (ETF) strategy. In each step, the following fitness function is used to evaluate the schedule s.

$$fitness_s = \begin{cases} \frac{1}{\varepsilon_s}, & \text{if } L_s \leq D \\ \frac{\frac{1}{\varepsilon_s}}{10(\frac{L_s}{D})^2}, & \text{if } L_s > D \end{cases} \tag{3.1}$$

where L_s and ε_s are the makespan and total energy consumption, respectively, of a schedule s, and D is the common deadline. The fitness implies that a feasible schedule consuming less total energy consumption is preferred, compared to one that is not feasible or with a larger total energy consumption. They evaluate this approach with four different variants and the results indicate that it can reduce the total energy consumption. However, their

mapping approach is guided by communication rather than energy consumption which may degrades the solution in terms of energy consumption. Furthermore, GA is based on natural selection and it does not necessarily produce an optimal result.

3.1.2 Periodic Tasks with Precedence Constraints

Luo and Jha [36] examine the problem of minimizing the total dynamic energy consumption for multiple applications comprising periodic tasks and aperiodic tasks on homogeneous distributed MPSoCs assuming continuous frequencies. The periodic tasks are assumed to be hard deadline while the aperiodic tasks may have hard or soft deadlines. Firstly, it constructs an initial schedule for tasks and messages along hyper-period using a slack-based list scheduling under maximum frequencies. This is done to guarantee the feasibility of the schedule. Then, it applies DVS to distribute available static slack proportionately among all tasks and scale the supply voltage of each task accordingly. Results suggest this approach can save up to 68% of total dynamic energy compared to non-power-aware approaches. Nevertheless, this approach does not necessarily lead to the best energy savings, as every task contributes differently to the overall energy savings.

Xu et al. [37] study the problem to minimize the total computation energy consumption composed of dynamic energy and static energy for streaming applications while satisfying both throughput and response time. They consider a homogeneous distributed Chip Multiprocessors (CMPs) with discrete frequency model. They present two integrated task mapping and frequency assignment approaches: Scheduling1D for linear task graphs and Scheduling2D for general task graphs. The Scheduling2D works on two dimensions: pipelining and parallel processing. Firstly, it partitions all the tasks into levels. Secondly, it computes the optimal number of pipeline stages which guarantees the timing constraint and assigns tasks on each level to a pipeline stage. Thirdly, for each pipeline stage, it computes the optimal number of processors, assigns each task to a processor and selects a discrete frequency for each task. Results suggest that Scheduling2D reduces the total en-

ergy consumption more effectively, compared to non-energy-aware approaches. However, this study does not explicitly consider the impact of the communication which may affect the feasibility of the schedule and the utilization of slack for frequency scaling.

Watanabe et al. [38] investigate the problem of minimizing the total energy consumption comprising computation energy and communication energy for periodic dependent tasks with latency and throughput constraints on homogeneous Globally Asynchronous Locally Synchronous (GALS) MPSoCs considering discrete frequencies. They propose a pipelined scheduling approach using a Mixed Integer Linear Programming (MILP)-based algorithm to compute an optimal solution. In addition, they propose an approach based on Simulated Annealing (SA) to find a near-optimal solution. Results indicate that the solution of their SA-based algorithm is close to the optimal solution computed by the MILP-based algorithm, with a faster running time.

Liu et al. [39] work on the problem of joint energy and performance optimization. They focus on periodic dependent tasks on homogeneous distributed multiprocessors considering discrete frequencies. They present a two-phase approach named RDAG+SpringS, that combines task-level software pipelining with Dynamic Voltage Scaling (DVS). Firstly, they use a retiming technique, RDAG, to transform all intra-period dependencies into inter-period dependencies. It assigns an integer for each task to represent the retiming value which is based on its height in the task graph. Starting from a sink task, they calculate the retiming value R_i of each task t_i in a breadth-first manner as follows.

$$R_i = \begin{cases} \max(R_i, R_j + 1), & \text{if task } t_i \text{ is a parent of task } t_j \\ 0, & \text{if task } t_i \text{ is a sink task} \end{cases} \tag{3.2}$$

Secondly, they use a heuristic named SpringS which iteratively generates a feasible schedule by adjusting the voltage, assignment and ordering of each task simultaneously. Results show that this approach can achieve better energy savings and better schedulability, compared to an approach which does not employ task-level software pipelining [40]. However, transforming all the intra-period dependencies into inter-period dependencies consumes

large memory overhead. In addition, the proposed retiming function, which only considers the height factor of tasks, may introduce unnecessary memory capacity overhead.

Wang et al. [5] investigate the energy-aware task scheduling problem for streaming application composed of periodic dependent tasks on homogeneous distributed multiprocessors such that the total energy consumption is minimized. They present a two-step approach, RDAG+GeneS that combines task-level software pipelining with DVFS. Firstly, they employ a retiming technique, RDAG from a previous work in [39] to transform all intra-period dependencies in task graphs into inter-period dependencies. Secondly, they employ Genetic Algorithm (GA) to find the best task mapping and frequency assignment of each task. The GA runs iteratively until a termination condition is met. At each step, each schedule s is assessed based on its fitness as below.

$$fitness_s = \begin{cases} \frac{1}{E_s}, & \text{if } D \leq L_s \\ 0, & \text{if } D > L_s \end{cases} \tag{3.3}$$

where E_s, L_s and D are the total energy consumption of a schedule s, the makespan of a schedule s and the common deadline, respectively. Results suggest this approach can significantly reduce the total energy consumption and increase task schedulability, but requires more memory capacity overhead, compared to the approaches in [39], which uses a heuristic for task scheduling and [30], which does not employ task-level software pipelining.

Qiu et al. [41] study the problem of minimizing the total dynamic energy consumption for periodic dependent tasks on homogeneous chip multiprocessors (CMPs) with discrete frequency model. All processors are connected through a shared bus. They assume the relative deadline of the application is greater than one period. They propose a three-phase scheme. First, it constructs an initial schedule under maximum frequencies within one period using the Min-Min algorithm [42]. Second, it iteratively reduces the discrete frequency of each task by one level, provided that the schedule is feasible. In each step, it tentatively computes an optimal frequency for each task based on the extendable factor

of its path and selects a discrete frequency not less than its optimal frequency. Then, it chooses the task with a potentially higher energy saving to use the next lower discrete frequency. Third, it extends the schedule task graph to represent its execution in several periods to accommodate the tasks which have their deadlines larger than one period. Then, it repeats the second step for this new extended task graph to further reduce the energy consumption. The approach is compared to another approach which considers one period and the results indicate improvement in total dynamic energy consumption. However, an explanation of the need of the third step is not given, and it is not understood why the third step is required given the requirements of the application are fulfilled by the first two steps. Also, they employ a naive approach in selecting a discrete frequency which may degrade the quality of the solution.

Wang et al. [6] explore the problem to totally remove the inter-processor communication overhead. They focus on periodic dependent tasks on homogeneous multiprocessors connected via a shared bus. They present a retiming approach named JCCTS by using Integer Linear Programming (ILP). Firstly, they do schedulability analysis to compute the minimum and maximum retiming values for each task. Then, the ILP-based algorithm incorporates the bounds as constraints in order to minimize the maximum retiming value. The approach can be extended to minimize the total dynamic energy consumption by applying any DVFS techniques. The slack resulting from the removal of inter-processor communication overhead can be fully utilized to minimize the total computation energy consumption.

3.2 Energy-aware Task Scheduling on Heterogeneous Multiprocessors

3.2.1 Aperiodic Tasks with Precedence Constraints

Hu and Marculescu [43] investigate the problem of scheduling both tasks and communication messages such that the total energy consumption (computation energy plus communication energy) is minimized on a heterogeneous NoC-based MPSoC. They assume dependent tasks with individual deadlines. They propose a heuristic, EAS, composed of three major steps: budget slack allocation for each task, level-based scheduling and, a search and repair step. The key idea of this approach is to distribute more slack to tasks whose mapping onto processors has a potentially larger impact on the total energy consumption and the performance of the application. The first step computes the weight, which is the priority of each task and is also used for allocating the slack to each task. It is computed as follows.

$$\omega_i = \sigma_i^E \sigma_i^R \tag{3.4}$$

where σ_i^E and σ_i^R are the variance of the energy consumption of task t_i on different processors and the variance of the execution time of t_i on different processors, respectively. The second step simultaneously schedules tasks and communication messages. It computes the earliest finish time which includes the data ready time of each task among different processors and assigns each task to a processor with minimum total energy consumption. The third step is initiated when the first two steps results in deadline violations. It iteratively swaps the ordering of tasks or migrates tasks to other processors, until all deadlines are satisfied. Results show that this approach significantly outperforms the schedule generated by an Earliest-Deadline First (EDF) approach in terms of total energy consumption. Nevertheless, task priorities based only upon the variance of energy and execution time only may result in deadline violations, and fixing the schedule may take significant running time.

Yan et al. [44] propose a two-phase approach to minimize the total computation energy consumption. They consider a set of tasks with precedence and deadline constraints on heterogeneous multiprocessors assuming continuous frequencies. Firstly, they construct an initial schedule by using list-based scheduling under maximum frequencies. The priority of each task is the inverse of its as-late-as-possible start time. Then, they employ an iterative slack allocation algorithm with the combination of Dynamic Voltage Scaling (DVS) and Adaptive Body Biasing (ABB) to minimize both dynamic power and static power, simultaneously. In each iteration, it selects a task with the highest energy gradient for voltage scaling. It stops when there is no slack to be utilized. Results indicate that this approach can further reduce the total energy consumption, compared to an approach that only uses DVS.

Kianzad et al. [45] deal with a set of tasks with precedence and deadline constraints on either homogeneous or heterogeneous multiprocessors with discrete voltage levels. The processors are connected through a shared bus. They present an approach named CASPER, which consolidates task mapping, scheduling and Dynamic Voltage Scaling (DVS) under Genetic Algorithm (GA) to search for the most energy-efficient solution. The GA runs iteratively. In each step, it constructs an initial schedule under maximum frequencies where the ordering of each task is based on its height in the task graph. Then, it uses a slack allocation algorithm, depending on the platform to minimize the total dynamic energy consumption. On homogeneous platform, they use the algorithm in [46] while on heterogeneous platform, they use a power variation DVS algorithm in [47]. Next, it evaluates the schedule based on its fitness. The GA stops when the termination condition is met, i.e. the maximum number of generations is reached or the total dynamic energy saving in two consecutive generations is less than 1%. Results indicate that this single loop CASPER saves more energy, compared to the approach which combines task scheduling and DVS in the inner loop and leaves task assignment in the outer loop [48].

Gorjiara and Bagherzadeh [49] investigate the problem of minimizing the total dynamic

energy consumption for tasks with precedence and hard deadline constraints on homogeneous distributed multiprocessors considering discrete frequencies. They suggest that a greedy slack allocation scheme which depends only on the energy gradient does not necessarily produce good results for discrete frequency model. Thus, they present ASG-VTS, a stochastic-based scheduling heuristic which not only considers energy gradient, but also execution delay simultaneously. Given a static schedule under maximum frequencies, the heuristic iteratively allocates slack to tasks. At each step, task that can save the most energy and with the least execution delay is assigned a higher slowdown probability. The heuristic stops when there is no more usable slack. Results indicate that this approach produces a slightly better schedule, in terms of energy savings, with fewer number of discrete voltage transitions, compared to the approach in [48]. In addition, their approach results in near-optimal solution as compared to the approach in [33] for a real benchmark. However, the study would have been more convincing if they assess the actual transition overhead in terms of time and energy instead of evaluating the number of transitions only.

Liu et al. [50] study the problem of minimizing the total dynamic energy consumption for a set of tasks with precedence and individual deadline constraints. The target system is a heterogeneous distributed multiprocessors assuming continuous frequencies. They propose two algorithms: CPSS for offline scheduling and CPDS for online scheduling. The key idea of the CPSS is to allocate slack evenly among tasks based on their path scaling factor. The scaling factor of a path is defined as follows.

$$S_k = \frac{d_k - r_k - (\sum_{t_i \in k} w_i + \sum_{M_{i,j} \in k} w_{i,j})}{\sum_{t_i \in k} w_i} \tag{3.5}$$

where d_k, r_k are the deadline of a sink task and the arrival time of a source task in path k, respectively. Terms $\sum_{t_i \in k} w_i$ and $\sum_{M_{i,j} \in k} w_{i,j}$ are the total worst-case execution time (wcet) of all tasks and the total worst-case communication time on path k, respectively. Firstly, they construct an initial schedule under maximum frequencies using a list-based scheduling, where the priority of each task is the inverse of its slack-time ratio. Secondly, they employ CPSS which iteratively distribute slack to tasks. At each step, it identifies

a critical path which is defined as the path with minimum scaling factor, computes its scaling factor and allocates the slack for each tasks on the path according to the scaling factor. The path is then removed. The heuristic stops when the frequencies of all the tasks are determined. Results suggest that CPSS performs slightly better in terms of energy savings and is significantly better in terms of running time than the approach in [51], which also perform frequency scaling based on critical-path information.

Chang et al. [52] propose a heuristic named ETAHM to minimize the total dynamic energy consumption. They consider tasks with precedence and deadline constraints on a heterogeneous multiprocessors with discrete voltage levels. The processors are connected through a shared bus. They employ ant-colony optimization (ACO) for simultaneous task mapping, scheduling and voltage scaling. It runs iteratively until the improvement rate is too slow or reaches the maximum repetitions. In each iteration, it constructs a schedule according to the task merit, and then assess its total dynamic energy consumption. Results suggest that ETAHM performs better than [45] in terms of the total dynamic energy consumption, but requires greater running time. However, ACO-based approach may not necessarily find an optimal solution as some parameters are randomly chosen.

Goh et al. [53] develop two heuristics, EGMS and EGMSIV to solve the problem of minimizing the total dynamic energy for dependent tasks with a common deadline. They consider a heterogeneous MPSoC with discrete supply voltage. The processors are connected to a shared bus. Both heuristics integrate task mapping, scheduling and voltage scaling simultaneously. They construct an initial schedule under maximum frequencies based on information from the critical-path. Then, they improve the schedule iteratively. In each step, they select a task to be reassigned to another processor or another voltage level such that the total energy consumption is reduced without any deadline violations. For EGMS, it employs inter-task voltage scaling and the reassignment is decided based on the energy gradient, while the EGMSIV uses intra-task voltage scaling and uses a Linear Programming (LP)-based algorithm. Both heuristics stop when the improvement rate is

low. Results indicate that both heuristics improve the total dynamic energy and running time as compared to the approaches in [48], [49], [54].

Ghosh et al. [55] study the problem of minimizing the total energy consumption of a set of tasks with precedence and a common deadline on heterogeneous NoC-based MPSoC with discrete frequencies. They propose two approaches; an MILP-based algorithm to compute an optimal solution and a unified heuristic employing MILP relaxation and randomized rounding. They consider solving the following sub-problems in a unified manner instead of a sequential manner: mapping of tasks to processors, mapping of processors to the routers, assigning voltages to tasks and routing of communication data. Results suggest that their unified heuristic is slightly better than an approach using sequential steps in reducing the total energy consumption. However, this study does not offer an adequate explanation on the heuristic that uses MILP relaxation and randomized rounding.

Huang et al. [56] investigate the problem of minimizing the total energy consumption (computation plus communication) for a set of dependent tasks with individual deadlines on a heterogeneous NoC-based MPSoC. Initially, they extend an Integer Linear Programming (ILP)-based algorithm to take into account computation energy and communication energy. Then, based on the analyses of the ILP-based algorithm, they propose a Simulated Annealing with Timing Adjustment (SA-TA) heuristic. The heuristic runs iteratively and instead of starting the SA optimization from a random mapping, they first compute a baseline mapping. The baseline mapping is constructed using a list-based scheduling. The priority of each task is the difference between its minimum execution time and the second minimum execution time across all processors. In each step, a probability P_s to determine if a schedule is accepted is used. The probability is computed as follows.

$$P_s = \begin{cases} 1, & \text{if } E_{new} < E_{cur} \\ 10^{(E_{cur} - E_{new})/T}, & \text{Otherwise} \end{cases} \tag{3.6}$$

where E_{cur}, E_{new}, T are the total energy consumption of the current schedule, the total energy consumption of the new schedule and a temperature parameter, respectively. This

probability ensures that if the new schedule has lower energy, it is always accepted. Otherwise, a worse schedule could be accepted with some probability to prevent it from being stuck at a local optimum. The heuristic stops when a maximum iteration is achieved. The temperature parameter T is decreased after each iteration by multiplying itself with a cooling factor. Experimental results suggest that the SA-TA heuristic performs similarly to the optimal result computed by the ILP-based algorithm, but with better running time.

He et al. [57] work on the problem to optimize the total energy consumption and schedule length for dependent tasks with a common deadline on a general NoC-based MPSoCs. They present a new graph model, Labelled Graph, considers a general network and estimates the communication energy and latency. The model is used for a Mixed Integer Linear Programming (MILP)-based algorithm, a unified approach to assign each task to a processor and scheduling. In addition, they propose a polynomial-time heuristic to reduce the running time. Results indicate that their approach can produce higher performance schedule with comparable energy consumption as compared to approaches in [58] and [59] for custom, regular mesh and irregular mesh NoC.

Pietri and Sakellariou [60] study similar problem for minimizing the total energy consumption of scientific workflow applications on heterogeneous distributed multiprocessors with discrete frequencies. They consider scientific workflow applications consisting of dependent tasks with a common deadline. They propose a two-step approach. Firstly, they construct an initial schedule under maximum frequencies by using the HEFT [26] algorithm. Secondly, they present an algorithm named ESFS to scale the frequencies of all the tasks to reduce the energy consumption. It performs iteratively, starting from the maximum frequencies downwards. In each iteration, a task can use the next lower discrete frequency instead of its current frequency, if this results in the highest energy saving compared to other tasks. It stops when there is no more slack to be utilized. Results suggest that this approach can further reduce the energy consumption compared to the approach in [61] which does not account for the difference in energy-profile of heterogeneous systems.

Lin et al. [62] consider similar problem for mobile cloud computing. They assume applications with precedence and a common deadline constraints on heterogeneous multiprocessors with discrete frequencies. They propose an energy-efficient task scheduling heuristic consisting of three steps: constructing an initial schedule under maximum frequencies using HEFT algorithm [26], an iterative procedure for migrating tasks to another processors under maximum frequencies to reduce the total dynamic energy consumption and, reclaiming static slack to further reduce the total dynamic energy consumption.

Zheng and Huang [63] consider the same problem as in [60], and propose an efficient heuristic which takes the system and application characteristics and the overall energy consumption into account when making a frequency scaling decision. They present a three-phase heuristic named AGTI. Firstly, they construct an initial schedule under maximum frequencies by using the HEFT [26] algorithm. Secondly, it scales the frequencies of tasks by group (processor) if the schedule task graph satisfies a rule based on its structure and the number of processors. In each step during this phase, it reduces the discrete frequencies of all the tasks on a processor with the maximum energy saving potential, by one level. Thirdly, it scales the frequency of each task individually and iteratively finds tasks with maximum energy savings and rescaling the tasks whilst maintaining the feasibility of the schedule. Results suggest that this approach performs better than the approaches in [61] and [60], in terms of conserving the total energy consumption.

Singh et al. [64] work on the problem of minimizing the total energy consumption composing of computation energy and communication energy for a set of dependent tasks with individual deadlines on a heterogeneous distributed MPSoC. They propose CEEDMIP, a duplication-based approach using Mixed Integer Programming (MIP). Besides, they present a clustering-based heuristic, FastCEED to reduce the time complexity in solving the problem. The key idea of CEEDMIP is to duplicate the execution of tasks on different processors to reduce communication congestion on links as well as the communication energy consumption. Results indicate that FastCEED can reduce the total energy con-

sumption of communication intensive applications compared to other duplication-based approaches in [65] and [66]. However, duplication-based approach may not work for computation intensive applications. Duplicating tasks may increase the computation energy consumption significantly, particularly on heterogeneous multiprocessors where each processor has its own energy profile. Consequently, the total energy could increase.

Zhao et al. [67] propose a clustering approach to minimize inter-processor communications between tasks. They consider the total computation energy consumption (dynamic plus static energy). The key idea of this approach is to map each task to a processor with the most of its required input data. In addition, they propose a metric named Task Requirement Degree (TRD) to improve the utilization of processors through load balancing and reduce the energy consumption during scheduling.

Tang et al. [68] present DEWTS consisting of three phases: constructing an initial schedule using the HEFT [26] algorithm, processors merging phase to minimize the number of processors being used, and task slacking phase to minimize the total dynamic energy consumption. This approach shuts down under-utilized processors by merging the processors provided that the schedule does not violate the deadline.

Xie et al. [69] work on the problem of minimizing the total dynamic energy consumption of dependent tasks on heterogeneous distributed multiprocessors with discrete frequencies. They propose an approach named NDES+GDES. Firstly, it constructs an initial schedule under maximum frequencies by using HEFT algorithm [26]. Secondly, if there exists available static slack, it iteratively reassigns tasks to processors to find a schedule with minimum total dynamic energy. However, they make no attempt to explain on what basis they reassign task since enumerating all possible schedule require exponential time. Results suggest that this approach outperforms the approach in [68] in terms of total dynamic energy consumption.

3.2.2 Periodic Tasks with Precedence Constraints

Schmitz and Al-Hashimi [70] 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 PV-DVS algorithm which considers power variations among tasks. It distributes static slack to tasks based on their energy gradients.

Schmitz et al. [48] extend their work in [70]. They present an energy-efficient genetic list scheduling algorithms (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 its time penalty. However they consider continuous frequency model.

Luo et al. [51] investigate 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 frequency reduction ratio. Then, it refines the task execution order if further energy reduction is possible.

Luo and Jha [71] address the problem to minimize 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 iteratively allocates slack to a task with the highest energy gradient.

Schmitz et al. [47] explore the problem of minimizing the total dynamic energy consumption for multi-rate applications on distributed MPSoCs assuming continuous frequency

model. They consider all the processors are connected through a single bus. They propose an integrated approach for task mapping, scheduling and voltage assignments. The heuristic is built using two nested genetic algorithms (GAs) where the outer GA generates the task assignments and the inner GA examines various task orderings. In each step where a schedule is generated, they employ a PV-DVS heuristic [70] which iteratively allocates slack to tasks with the highest energy savings. An energy difference metric, $\triangle E_i$ for each task t_i is defined as follows.

$$\triangle E_i = E_i(e) - E_i(e + \triangle e) \tag{3.7}$$

where $E_i(e)$ and $E_i(e + \triangle e)$ are the energy consumption of task t_i with execution time e and when extended to $e + \triangle e$, respectively. Results show that their approach is better than an approach that distributes slack evenly among tasks in terms of energy savings. However, the time complexity of their approach is higher because they attempt to find various energy-efficient schedules. Moreover, GA is based on natural selections which does not necessarily find an optimal solution.

Shin and Kim [72] investigate the problem of minimizing the communication energy consumption of heterogeneous NoC-based MPSoCs. They consider periodic dependent real-time applications with deadline constraints. They propose an approach consisting of four components as follows.

- Task mapping using a GA-based task assignment algorithm (GA-TA) to assign each task to a processor.
- Network assignment using a GA-based tile mapping algorithm (GA-TM) to map each processor to a tile and a GA-based routing path allocation algorithm (GA-RPA).
- List-based task scheduling where the priority of each task is its mobility defined as the difference between its as-soon-as-possible (ASAP) start time and the as-late-as-possible (ALAP) finish time.

- Frequency scaling for communication messages by using the approach in [70].

However, they offer no explanation on the fitness function of each GA in order to evaluate the solution. Results indicate that this approach could significantly reduce the total communication energy, compared to an extended approach in [73].

Luo and Jha [40] investigate the problem of minimizing the total dynamic energy consumption of multi-rate applications with individual deadlines on heterogeneous MPSoCs with discrete voltages. All the processors are connected through a shared bus. Firstly, they employ a critical-path-based list scheduling as in [74] to construct an initial schedule within one hyper-period under maximum frequencies. Then, they employ an iterative power-profile and timing-constraint driven approach to select a discrete voltage for each task. In each step, a task with the highest energy gradient is chosen to use the next lower voltage than its current voltage. The algorithm stops when there is no usable slack in the schedule. Secondly, they attempt to improve the resultant schedule by using an iterative Simulated-Annealing (SA)-based approach. In each step, the priority of each task is modified using a random parameter, reordering of the tasks on their respective processors based on the new priorities, reassignment of discrete voltages to all the tasks, and reassessment of the energy consumption of the schedule. The following probability is used as follows.

$$1/(1 + 10^{(E_{new} - E_{cur})/T}) \tag{3.8}$$

where E_{new}, E_{cur} and T are the energy consumption of the new solution and the current solution, respectively, while T is a temperature parameter which is initialized and decreases in every step. This algorithm stops when there is no further significant improvement in the total dynamic energy consumption or the temperature parameter is equal to zero. Results indicate that this approach performs better than the approach in [70]. Nevertheless, SA-based algorithm accepts an inferior solution with some probability, which may affect the quality of the solution.

Kumar et al. [75] explore the problem of assigning frequency and modulation levels to tasks

in order to minimize the total energy consumption (computation and communication) of periodic dependent tasks on heterogeneous multiprocessors connected via wireless network. They propose a slack allocation scheme for tasks and messages based on their normalized energy gain. The heuristic allocates slack in an incrementally fashion. In each iteration, slack is given to the task or message with the highest normalized energy gain.

Huang et al. [76] address the problem of minimizing the total dynamic energy consumption for streaming applications composed of periodic dependent tasks under throughput constraint on heterogeneous distributed multiprocessors. They consider a given initial schedule with maximum frequencies. Then, they formulate the slack allocation problem for multiprocessors with local DVFS switches using a Mixed Integer Linear Programming (MILP)-based algorithm. For multiprocessors with a global DVFS switch, they propose a three-phase heuristic combining the MILP. Results suggest that this approach perform better in reducing the total dynamic energy consumption for streaming applications with cross-period precedence constraints as compared to an approach that adapting a deadline-constrained within one period. This is due to the manipulation of available static slack that exists across different periods. However, the impact of significant inter-processor communication towards the resultant schedule in terms of total energy consumption has not been addressed in this study, since the applications considered not only have intra-period dependencies but also cross-period dependencies.

Singh et al. [77] consider concurrent multimedia applications composed of periodic, cyclic and multi-rate dependencies between tasks modelled 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 map according to its requirements with minimum energy consumption.

Liu et al. [78] investigate the problem to minimize 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 in order to further scale down the clusters operating frequencies and thus reducing the total energy consumption.

Chapter 4

Energy-aware Task Scheduling on Homogeneous MPSoCs

This chapter elaborates our energy-aware task scheduling approach for applications consisting tasks with precedence and individual deadline constraints on homogeneous DVFS-enabled MPSoCs. We propose a list-based scheduling algorithm to construct an initial schedule under maximum frequencies and a Non-Linear Programming (NLP)-based algorithm for minimizing the energy of the initial schedule. This chapter is organized as follows. Section 4.1 introduces the objectives of our tasks scheduling and describes our major contributions. Section 4.2 specifies the system models and introduces some definitions. Section 4.3 provides a motivational example of our novel priority scheme. Section 4.4 proposes our algorithm for constructing the offline schedule. Section 4.5 presents the experimental results and analyses. Section 4.6 summarizes this chapter.

4.1 Introduction

There are three major components to the problem of constructing an offline task schedule such that its energy is minimized. Those components are mapping each task to a processor, ordering each task on its assigned processor and selecting an operating frequency and its corresponding supply voltage of each task. The task mapping and ordering aim to construct a feasible schedule. Next, operating frequencies of all tasks can be assigned considering the available slack and thus the energy can be minimized. Hence, it is important to compute a good initial schedule during task mapping and ordering that can provide enough slack for frequency scaling. This translates into the problem of constructing a schedule with the minimum completion time which is NP-complete [17].

For a list-based scheduling, the priority of each task plays an important role for task mapping and task ordering. Most of the previous approaches employ a priority for each task according to its critical-path length. The length is computed either from an entrance task to the task, known as downward rank of the task, or from the task to an exit task, known as upward rank of the task. Nonetheless, the priority based on the critical-path does not necessarily represent the importance of each task.

In this chapter, we investigate the problem of constructing an offline schedule for a set of non-preemptible tasks with precedence and individual deadline constraints to be executed on a distributed MPSoC with continuous frequencies such that the total processor energy consumption of all the tasks is minimized under two power models, namely the dynamic power model and the total power model. We make the following major contributions.

1. We present a novel priority scheme for task assignment. The priority of each task is its approximate successor-tree-consistent deadline which is the approximate upper bound on its latest completion time in any feasible schedule for a relaxed problem, where only the precedence constraints between the task and all its successors are

considered.

2. We propose a unified approach by using a convex Non-Linear Programming (NLP) formulation to assign an optimal execution frequency and its corresponding supply voltage to each task.

3. We have evaluated our approach and compared it with two state-of-the-art approaches, the LL-ES-GREEDY approach, shorten as LESG [1] which considers the dynamic energy dissipation, and the EES approach [2] which considers the total energy consumption, by using a set of synthetic and real-world benchmarks. The experimental results show that the maximum improvement, the average improvement and the minimum improvement of our approach over the LL-ES-GREEDY approach are 42.44%, 30.46% and 9.46%, respectively, and the maximum improvement, the average improvement and the minimum improvement of our approach over the EES approach are 75.98%, 39.74% and 7.08%, respectively.

4.2 Problem, Definitions and Models

The problem we investigate is described as follows. Given a set $T = \{t_1, \cdots, t_n\}$ of n non-preemptible tasks with precedence constraints and individual deadlines, find a feasible schedule on a target MPSoC such that the total dynamic processor energy or the total processor energy of all the tasks is minimized. A feasible schedule is a schedule that satisfies all the constraints.

Table 4.1 shows a list of notations used throughout this chapter.

4.2.1 System Model

The target MPSoC is composed of a set $P = \{ p_1, p_2, \cdots, p_m\}$ of m identical processors interconnected via a network. There is no shared memory among the processors. We

Table 4.1: Notations used.

Notation	Description	Notation	Description
T	A set of application tasks	P	A set of processors
t_i	The i$^{\text{th}}$ application task	p_k	The k$^{\text{th}}$ processor
d_i	Predefined deadline of t_i	d'_i	The successor-tree-consistent deadline of t_i
$\lambda_{i,j}$	Communication time of transferring one unit of data from p_i to p_j	c_{ij}	Data size between t_i and t_j
$WCET(t_i)$	Worst-case execution time of t_i	s_i	Start time of t_i
e_i	Execution time of t_i	cc_i	Number of clock cycles of t_i
$pred(t_i)$	All predecessors of t_i	$succ(t_i)$	All successors of t_i
$V_{dd_{min}}$	The minimum supply voltage of a processor	$V_{dd_{max}}$	The maximum supply voltage of a processor
V_{dd_i}	The supply voltage of t_i	f_i	The operating frequency of t_i
E_{dyn}	The total dynamic energy consumption	E_{tot}	The total energy consumption
C_{eff}	The average switched capacitance of a processor	L_g	Number of logic gates in a circuit
V_{th1}	The threshold voltage	V_{bs}	The body-bias voltage
L_d	The logic depth	I_j	The body junction leakage current
$K_l(l = 1, \cdots, 6)$,α Processor technology constants			

ignore the transition overhead between two different frequencies as it takes only 30 - 150 μs [79]. The operating frequency of each processor can be continuously adjusted. We employ a communication cost matrix λ, where $\lambda_{i,j}$ denotes the communication time of transferring one unit of data from processor p_i to processor p_j. We assume that $\lambda_{i,j}$ does

not depend on the operating frequencies of p_i and p_j. If a task t_i on processor p_s sends c_{ij} units of data to another task t_j on processor $p_t, (p_s \neq p_t)$, the communication time is $c_{ij}\lambda_{s,t}$. If t_i and t_j are scheduled on the same processor, the communication time between t_i and t_j is 0.

4.2.2 Task Model

The embedded software of the target embedded system consists of a set of tasks with precedence constraints and individual deadlines. The task set is represented by a weighted DAG $G = (T, E, W, C)$, where each node in T denotes a task, each edge in E denotes precedence between two tasks, each node weight in W represents the worst-case execution time (WCET) of the corresponding task, and each edge weight in C denotes the size of the data transferred between the two tasks via the network. We assume that all the tasks are non-preemptible.

4.2.3 Power Model

The total power of a processor is the sum of the dynamic power due to switching activity, and the static power as a result of leakage. The dynamic power, denoted by P_{dyn}, is specified as follows [25].

$$P_{dyn} = C_{eff} V_{dd}^2 f \tag{4.1}$$

where C_{eff} is the average switched capacitance, V_{dd} is the supply voltage and f is the operating frequency. The total power, denoted by P_{tot}, can be calculated by the following equation [25].

$$P_{tot} = C_{eff} V_{dd}^2 f + L_g(V_{dd} K_3 e^{K_4 V_{dd}} e^{K_5 V_{bs}} + |V_{bs}| I_j) \tag{4.2}$$

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, V_{bs} and I_j are the body-bias voltage and body junction leakage current, respectively. The relation between the operating frequency and the supply voltage is given by the following equation [25].

$$f = ((1 + K_1)V_{dd} + K_2V_{bs} - V_{th1})^{\alpha}/(L_dK_6) \tag{4.3}$$

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 power function and the total power function are convex. Under the dynamic power model, the processor dynamic energy decreases as the processor frequency decreases. Under the total power model, there is an optimal minimum processor frequency f_{min} such that the processor total energy increases as the processor frequency lower than f_{min} further decreases [25].

4.2.4 Successor-Tree-Consistent Deadline

Let $pred(t_i)$ be the set of all the predecessors of a task t_i, $succ(t_i)$ the set of all the successors of a task t_i, and $WCET(t_i)$ the worst-case execution time of a task t_i at the maximum operating frequency.

Definition 4.2.1 *Given a weighted task graph $G = (T, E, W, C)$ 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 $T' = \{t_i\} \bigcup succ(t_i)$, $E' = \{(t_i,t_j) : t_j \in succ(t_i)\}$, $W' = \{WCET(t_j) : WCET(t_j) \in W$ and $t_j \in T'\}$ and $C' = \{c'_{ij}$: if t_j is an immediate successor of t_i, $c'_{ij} = c_{ij}$; otherwise, $c'_{ij} = 0\}$.*

Definition 4.2.2 *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 task without any successor, d'_i is equal to its pre-assigned 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 $T' = \{t_i\} \bigcup succ(t_i)$ of tasks with precedence constraints in the form of the successor-tree of t_i and individual deadlines where the deadline of each successor of t_i is equal to its successor-tree-consistent deadline, and the same MPSoC. Formally, $d'_i = \min\{d_i, \max\{\sigma_j(t_i) + WCET(t_i) : \sigma_j \text{ is a feasible schedule for } P(t_i)\}\}$.

Notice that it is NP-complete to compute the successor-tree-consistent deadline of a task as the problem of constructing a schedule with the minimum makespan on multiple processors is NP-complete [17]. Therefore, we compute the approximate successor-tree-consistent deadline of each task and use it as the priority of the task. Compared to the priority schemes used in the previous task scheduling approaches, the approximate successor-tree-consistent deadlines not only consider the precedence constraints, but also take the resource constraints into account. As a result, they capture the importance of each task better than the previous priority schemes.

4.3 A Motivational Example

In this section, we illustrate that it is essential to take into account the resource constraints when computing the priority of each task for list-based scheduling.

(a) (b)

Figure 4.1: Examples of (a) a Directed Acyclic Graph (DAG), (b) the successor-tree of t_2 of the DAG.

Consider a task graph as in Figure 4.1a with each task t_i has its worst-case execution time

(wcet) of three time units $WCET(t_i) = 3(i = 1, \ldots, 11)$ and a common deadline of 30 time units to be scheduled on a 3-processor platform. Intuitively, task t_5 should be given higher priority than task t_4 due to the fact that it has more successors and offers higher degree of parallelism.

One approach which does not consider resource constraints is to use the longest path as a guide to prioritize each task. For example, the bottom-level priority bl_i of each task t_i is the length of the longest path between itself to a sink task. Higher value implies higher priority. Based on this concept, the priority of each task $t_i(i = 1, \ldots, 11)$ in Figure 4.1a is $bl_1 = bl_2 = bl_3 = 9, bl_4 = bl_5 = 6, bl_7 = bl_8 = bl_9 = bl_{10} = bl_{11} = 3$.

Next, consider when the priority scheme takes into account not only the precedence constraints and deadline constraints, but also the resource constraints such as our approach using approximate successor-tree-consistent deadline. Our approach defines a concept of successor-tree as in Figure 4.1b and considers the availability of the three processors. Smaller value implies higher priority. Based on this aspect, the priority of each task $t_i(i = 1, \ldots, 11)$ in Figure 4.1a is $d'_1 = 23, d'_2 = 20, d'_3 = 21, d'_4 = 26, d'_5 = 24, d'_6 = d'_7 = d'_8 = d'_9 = d'_{10} = d'_{11} = 30$.

According to these two priorities, the one that considers the resource constraints perform better as it assigns a higher priority to task t_5 than t_4 as task t_5 has more influence than task t_4 in terms of its parallelism degree. On the other hand, bottom-level approach needs to find another metric to break the tie between task t_4 and task t_5. Thus, it is important to consider not only the precedence and deadline constraints, but also the resource constraints when computing the priority of each task.

4.4 Scheduling Approach

Our scheduling algorithm takes into consideration task mapping, task ordering and frequency scaling in an integrated manner. It consists of two major phases: task scheduling phase and voltage and frequency selection phase. Next, we describe each phase in detail.

4.4.1 Task Scheduling Phase

In the task scheduling phase, our approach computes a priority for each task. The priority of each task is its approximate successor-tree-consistent deadline, where a smaller deadline implies a higher priority. After computing the priority of each task, our approach assigns each task to a processor based on its priority, and constructs a schedule based on the priorities and precedence constraints. Next, we show how to compute the approximate successor-tree consistent deadline of each task.

When computing the approximate successor-tree-consistent deadlines of all the tasks, we assume that the target MPSoC uses the maximum processor operating frequency. The approximate successor-tree-consistent deadlines of all the tasks are computed in reverse topological order. For each task t_i, if it is a sink task, its successor-tree-consistent deadline is equal to its pre-assigned deadline d_i. Otherwise, the approximate successor-tree-consistent deadline of t_i is computed as follows.

1. Construct the successor-tree of t_i.
2. If t_i has only one immediate successor, its approximate successor-tree-consistent deadline is equal to $d'_j - WCET(t_j)$, where d'_j and $WCET(t_j)$ are the approximate successor-tree-consistent deadline and the worst-case execution time of its immediate successor t_j, respectively. 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 as late as possible.

(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, further sort them in non-increasing order of their edge weights. For the tasks with the same edge weight, further sort them in non-decreasing order of their worst-case execution times.

(e) Schedule each task in V on a processor as late as possible.

(f) 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.

(g) Set the approximate successor-tree-consistent deadline of t_i to the smaller one of its preassigned deadline and its latest completion time.

After computing the approximate successor-tree-consistent deadlines, our approach repeatedly selects a ready task with the smallest approximate successor-tree-consistent deadline among all the ready tasks, and assigns it to a processor such that its start time is minimized.

After assigning each task to a processor, our approach constructs a local schedule for each processor by using the earliest approximate successor-tree-consistent deadline first strategy. Next, our approach constructs an extended task graph by adding additional edges to the task graph G as follows.

- For each processor p_i do the following.

1. Let S_i be the set of tasks assigned to processor p_i and σ_i be the local schedule for all the tasks on p_i.
2. Construct a list L of all the tasks in S_i in non-decreasing order of their start times in σ_i.
3. For each pair of tasks L_j and $L_{j+1}(j = 0, 1, \cdots, |L| - 2)$, if there is no path from L_j to L_{j+1} in G, add a directed edge from L_j to L_{j+1} to G.

The new DAG G is used to formulate the optimal frequency assignment problem into an NLP problem. The time complexity of our task assignment algorithm is dominated by computing the approximate successor-tree-consistent deadlines. It takes $O(ne)$ time to compute the approximate successor-tree-consistent deadlines, where n is the number of tasks and e is the number of edges in the task graph. As a result, the time complexity of the task scheduling phase is $O(ne)$.

4.4.2 Voltage and Frequency Selection Phase

In this phase, we formulate the voltage and frequency selection problem into an NLP problem to find an optimal frequency for each task based on the extended task graph constructed before.

Let V_{dd_i}, f_i, cc_i, s_i and e_i be the supply voltage, operating frequency, number of clock cycles, start time and execution time of t_i, respectively. The total energy E_{tot} and the total dynamic energy E_{dyn} for the whole schedule are computed as follows.

$$E_{tot} = \{\sum_{i=1}^{|T|}(cc_i C_{eff} V_{dd_i}^2 + L_g(V_{dd_i} K_3 e^{K_4 V_{dd_i}} e^{K_5 V_{bs}} + |V_{bs}| I_j) e_i)\} \tag{4.4}$$

$$E_{dyn} = \sum_{i=1}^{|T|} cc_i C_{eff} V_{dd_i}^2 \tag{4.5}$$

Therefore, we have the following two objective functions for the total power model and the dynamic power model.

$$min\ E_{tot} \tag{4.6}$$

$$min\ E_{dyn} \tag{4.7}$$

The execution time e_i of task t_i is equal to cc_i/f_i. By using (4.3), we have the following constraint.

$$e_i = cc_i K_6 L_d / ((1 + K_1)V_{dd_i} + K_2 V_{bs} - V_{th1})^{\alpha} \tag{4.8}$$

For each edge (t_i, t_j) in the new DAG G, assume that t_i and t_j are assigned to processor p_u and processor p_v in the previous phase, respectively. The precedence constraint between t_i and t_j is modelled as follows.

$$s_i + e_i + \lambda_{u,v} c_{ij} \leq s_j, \qquad \forall (t_i, t_j) \in E \tag{4.9}$$

where $\lambda_{u,v} c_{ij}$ is the communication time between t_i and t_j as discussed in Section 4.2.1.

For each task t_i, the deadline constraint is specified as follows.

$$s_i + e_i \leq d_i \tag{4.10}$$

For each power model, there is a minimum supply voltage and a maximum supply voltage. However, the minimum supply voltage for the dynamic power model may be different from that for total power model. Let $V_{dd_{min}}$ and $V_{dd_{max}}$ denote the minimum supply voltage and the maximum supply voltage, respectively for each power model. We have the following constraint.

$$V_{dd_{min}} \leq V_{dd_i} \leq V_{dd_{max}} \tag{4.11}$$

The decision variables that need to be determined from the NLP are the start time s_i, the execution time e_i and the supply voltage V_{dd_i}. After the supply voltage V_{dd_i} for each task t_i is known, the processor operating frequency for task t_i can be computed by using (4.3).

The objective function for the total processor energy as well as the execution time constraint in (4.8) are convex non-linear functions. Therefore, the problem can be solved in polynomial time [33].

4.4.3 An Illustrative Example

Figure 4.2 shows a sample task graph where the number on each edge is the communication data size. The WCET at the maximum processor operating frequency of each task t_i in the sample task graph is shown in Table 4.2.

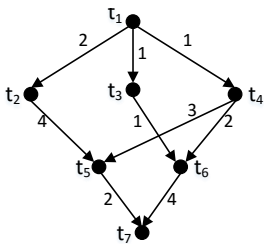


Figure 4.2: Example of a task graph.

Table 4.2: WCETs of the tasks for the task graph in Figure 4.2.

Tasks	t_1	t_2	t_3	t_4	t_5	t_6	t_7
WCET	5	4	3	7	2	6	3

We use the sample task graph shown in Figure 4.2 to illustrate how our approach works. Assume that the target MPSoC has two identical processors with continuous frequencies. Also assume that all the tasks has a common deadline of 30 and our algorithm has computed d'_i for each task t_i ($i = 7, 6, 5, 4, 3, 2$), where $d'_7 = 30$, $d'_6 = d'_5 = 27$, $d'_4 = d'_3 = 21$, and $d'_2 = 25$. Next, we show how our heuristic computes the approximate successor-tree-consistent deadline of t_1.

The approximate successor-tree-consistent deadline d'_1 is the approximate upper bound on the latest completion time of t_1 in any feasible schedule for the relaxed problem instance

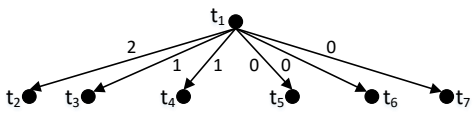


Figure 4.3: The successor-tree of t_1 of the DAG.

$P(t_1)$. The successor-tree of t_1 is shown in Figure 4.3. Based on the successor-tree, our heuristic constructs a backward schedule for all its successors where each successor is scheduled as late as possible. The backward schedule is shown in Figure 4.4. The heuristic finds the latest completion time of t_1 in the backward schedule respecting the successor-tree of t_1, which is 14. Therefore, the approximate successor-tree-consistent deadline of t_1 is 14.

Figure 4.5 shows the resulting initial schedule. After constructing the initial schedule, our approach expands the task graph by adding additional edges. Each additional edge denotes a new ordering between the two tasks enforced by the initial schedule. From Figure 4.5, our approach adds an edge from t_3 to t_2 in G.

Lastly, our approach constructs an NLP formulation to compute the optimal voltage and frequency of each task. The initial schedule is stretched according to the objective functions and constraints defined in 4.4.2. Figure 4.6 shows the final schedule after assigning an optimal voltage and frequency to each task.

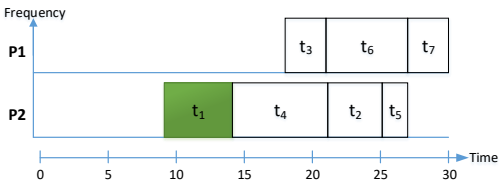


Figure 4.4: The backward schedule of the successor-tree of t_1 in Figure 4.3.

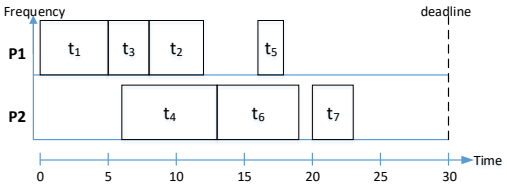


Figure 4.5: The initial schedule based on the successor-tree-consistent deadlines.

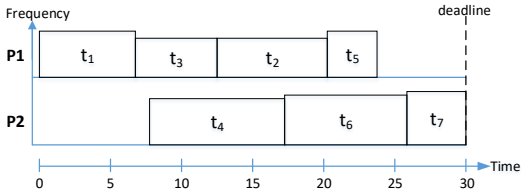


Figure 4.6: The schedule after the voltage and frequency selection.

4.5 Experimental Results

In order to evaluate our approach, we choose two state-of-the-art approaches, namely the LESG approach proposed in [1] and the EES proposed in [2], to make comparisons. We have implemented our approach, the LESG approach and the EES approach in Matlab version R2015a. We use Matlab fmincon to solve the NLP problem. The hardware platform for the simulation is an Intel(R) Core(TM) i5-4570 CPU with a clock frequency of 3.20 GHz, 8.00 GB memory and 3 MB caches.

Table 4.3: Constants of 0.07 μm processor technology.

Variable	Value	Variable	Value	Variable	Value
K_1	0.063	K_6	$5.26\text{x}10^{-12}$	v_{bs}	-0.70
K_2	0.153	C_{eff}	$4.30\text{x}10^{-10}$	v_{th1}	0.244
K_3	$5.38\text{x}10^{-7}$	I_j	$4.80\text{x}10^{-10}$	α	2.00
K_4	1.83	L_d	37.00		
K_5	4.19	L_g	$4.00\text{x}10^{6}$		

Each processor in our experiments is the 0.07μm Transmeta Crusoe processor equipped

with DVFS. The parameters used to compute the dynamic power and the total power are listed in Table 4.3, as given in [25, 80]. The accuracy of these technology parameters has been verified through SPICE simulations [25]. Also, they were widely used in the previous energy-aware task scheduling research [33, 80–83]. The continuous voltage range is set to $0.65V \leq V_{dd_i} \leq 0.85V$. We choose three sets of processors (three processors, five processors and ten processors) for the target MPSoC.

Table 4.4: The characteristics of benchmarks.

Benchmarks	$\|T\|/\|E\|$	$\overline{WCET}$	$\overline{c_{ij}}$	D
TG1	50/82	10.02	1.80	218
TG2	50/206	10.83	1.70	370
TG3	50/143	10.13	1.69	256
TG4	50/249	4.78	0.62	210
TG5	50/211	10.48	1.85	376
TG6	50/232	11.06	1.60	308
TG7	50/255	11.10	1.69	320
TG8	50/321	10.92	1.61	370
TG9	50/309	10.5	2.37	304
TG10	50/174	8.34	1.36	306
robot	88/131	27.61	3.99	1214
sparse matrix	96/67	20.17	3.48	800
ATR	17/16	657.59	133.77	8000

4.5.1 Experimental Setup

We use a set of ten synthetic benchmarks, two real benchmarks taken from [84], and one real benchmark which is Automated Target Recognition (ATR) application extracted from [85]. The two real applications taken from [84] are robot control and sparse matrix solver. The robot control application consists of 88 tasks while the sparse matrix solver

has 96 tasks. The real benchmark ATR contains 17 tasks. These real benchmarks are modelled from actual application programs. ATR is widely used for mobile military systems and usually requires real time processing [86]. It is used to recognize objects based on data obtained from sensors. Each synthetic task graph has 50 tasks with different task dependencies and distributions. These sets of benchmarks are widely used by the embedded systems community for task scheduling research.

Each synthetic task graph has 50 tasks with different task dependencies and distributions. For TG1, TG2, TG3, TG5, TG6, TG7, TG8, and TG9, each WCET is set to a random number from 1 to 20. For TG4 and TG10, each WCET is set to a random number from 1 to 10. This benchmark set does not provide the communication data sizes and deadlines.

Table 4.5: Average running times of our approach and LESG for dynamic energy simulations

No of processors	Ours (s)	LESG (s)
5	7.61	3.69
8	10.30	3.57
10	11.13	3.58

Table 4.6: Average running times of our approach and EES for total energy simulations

No of processors	Ours (s)	EES (s)
5	8.37	0.08
8	10.93	0.09
10	12.05	0.09

We assign a communication data size c_{ij} to each edge (t_i, t_j) as follows.

$$c_{ij} = \beta * WCET(t_i) \qquad \forall (t_i, t_j) \in E \tag{4.12}$$

where β is a constant between 0.01 to 0.30, and $WCET(t_i)$ is the worst-case execution time of task t_i.

This benchmark set also does not provide deadlines for tasks. For each benchmark, we assign a common deadline as follows.

1. Compute the total worst-case execution time γ of all the tasks at the maximum operating frequency.
2. Set the deadlines of all the tasks of the benchmark to γ.
3. Construct an initial schedule on five processors using our approach.
4. Set the common deadline for the benchmark to twice the makespan of the initial schedule.

The characteristics of each benchmark in terms of the number of tasks $|T|$, the number of edges $|E|$, the mean WCET $\overline{WCET}$, the mean communication data size $\overline{c_{ij}}$ and the deadline D are shown in Table 4.4.

4.5.2 Results and Discussions

In this section, we show the running times and the energy consumption (the total dynamic energy consumption and the total energy consumption) of the schedules produced by our approach, the LESG approach and the EES approach. Table 4.5 shows the average running times of our approach and the LESG approach.

Table 4.6 shows the average running times of our approach and the EES approach. For all the benchmarks, the average running time ratio of the EES approach over our approach is 0.8% while the average running time ratio of the LESG approach over our approach is 37.9%. As we can see, our approach is much slower than the EES approach. However, our approach computes much better energy-efficient schedules than the EES approach does. Notice that our approach targets embedded systems. Therefore, the running time for constructing an offline schedule at the design stage should not be an issue.

Figures 4.7, 4.8, 4.9 and 4.10 show the comparison between our approach and the LESG approach in terms of the total dynamic energy consumption. Each vertical axis denotes the total dynamic energy consumption and each horizontal axis represents benchmarks. The experimental results indicate that the maximum improvement, the average improvement and the minimum improvement of our approach compared to the LESG approach are 42.44%, 30.46% and 9.46%, respectively. The maximum improvement occurs at TG5 on 5-processors platform. The least improvement is observed at the real benchmark, sparse matrix on a 5-processors platform.

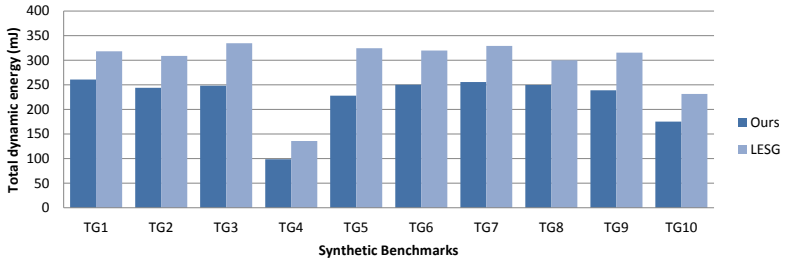


Figure 4.7: Total dynamic energy consumption of our approach and LESG [1] for synthetic benchmarks on 5-processors platform.

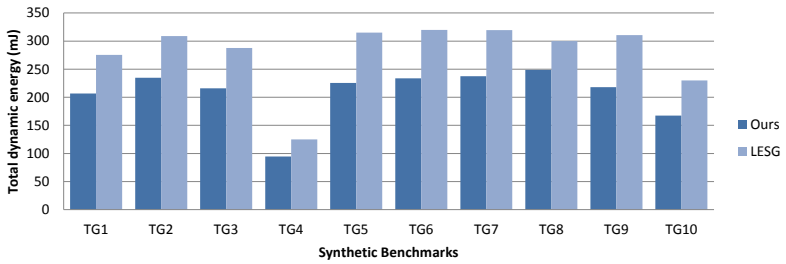


Figure 4.8: Total dynamic energy consumption of our approach and LESG [1] for synthetic benchmarks on 8-processors platform.

There are two key reasons that our approach significantly outperforms the LESG approach. Firstly, our approach uses approximate successor-tree-consistent deadlines to assign tasks to individual processors such that the workloads of all the processors are balanced. Secondly, our approach assigns an optimal frequency to each task. Whereas, the LESG approach assigns the same frequency to all the tasks at the same level.

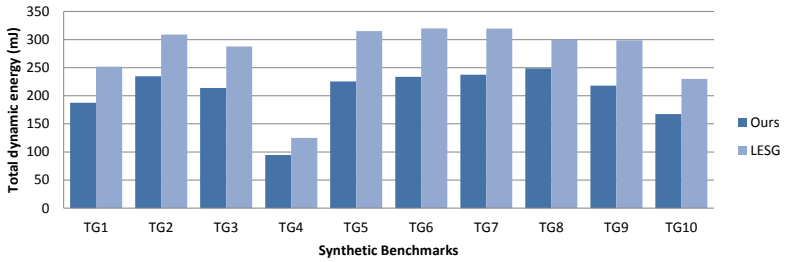


Figure 4.9: Total dynamic energy consumption of our approach and LESG [1] for synthetic benchmarks on 10-processors platform.

Figures 4.11, 4.12, 4.13 and 4.14 show the comparison between our approach and the EES approach in terms of the total energy consumption, where each vertical axis represents the total energy consumption and each horizontal axis represents benchmarks. The experimental results indicate that the maximum improvement, the average improvement and the minimum improvement of our approach over the EES approach are 75.98%, 39.74% and 7.08%, respectively. The greatest improvement of 75.98% occurs at synthetic benchmark TG4 on 8-processors platform and 10-processors platform, and the least improvement occurs at the real benchmark, sparse matrix on 5-processors platform.

Overall, the previous two reasons for the good performance of our approach still hold. It is observed that in some scenarios the schedules produced by the EES approach have some idle slots unused after frequency scaling. This is primarily caused by the task dependencies between different processors. Due to task dependencies, a newly scaled task may push a successor of a previously scaled task scheduled on the same processor to start at a later time, creating a local slack between the previously scaled task and the successor. This is in contrast to our approach where the convex NLP performs a global optimization to determine an optimal frequency for each task.

We have compared the benchmark structures of TG4 and sparse matrix for which our approach achieves the maximum improvement and the minimum improvement, respectively, over EES. It is observed that TG4 has a higher degree of parallelism than sparse matrix. When the degree of parallelism is high, more local slack may be available. As a result, the

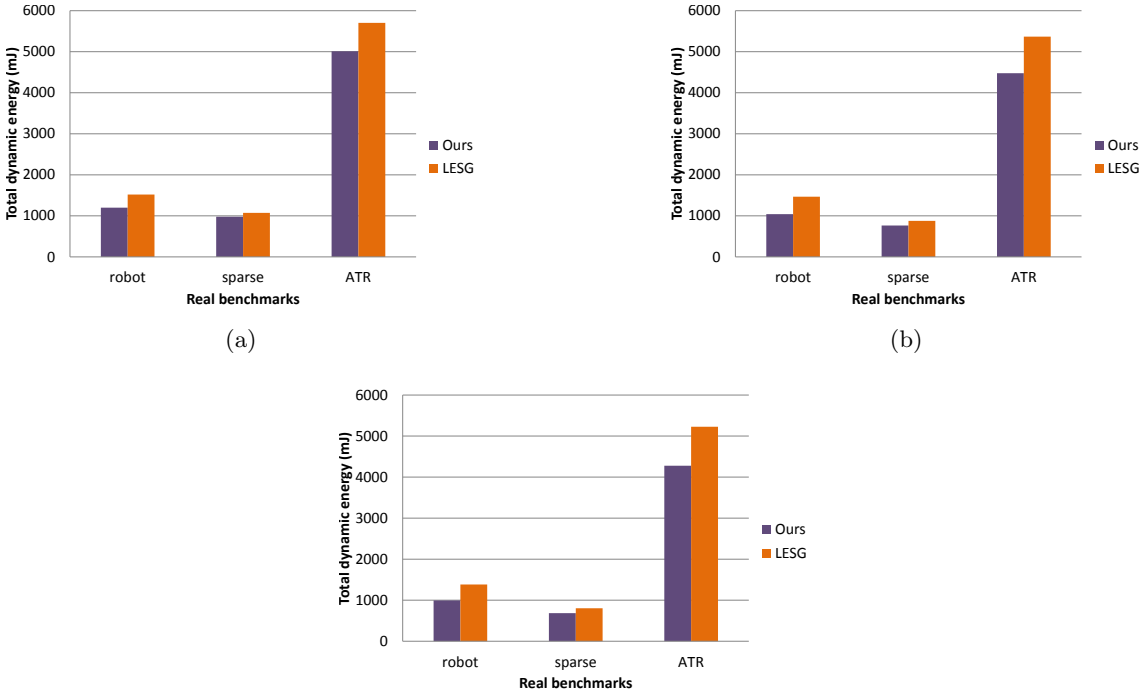


Figure 4.10: Total dynamic energy consumption of our approach and LESG [1] for real benchmarks on (a) 5-processors platform, (b) 8-processors platform and (c) 10-processors platform.

global optimization of our NLP approach constructs a better schedule.

Both the total dynamic energy consumption and the total energy consumption decrease as the number of processors increases. This can be explained as more processors may reduce the makespan and create more static slack given a common deadline, resulting in lower frequencies for the tasks. Therefore, both the total dynamic energy consumption and the total energy consumption reduce.

Furthermore, we have calculated the static processor energy consumption of each schedule under the total power model. The experimental results suggest that on average the static processor energy consumption accounts for 18.72% of the total energy consumption for all the thirty nine scenarios.

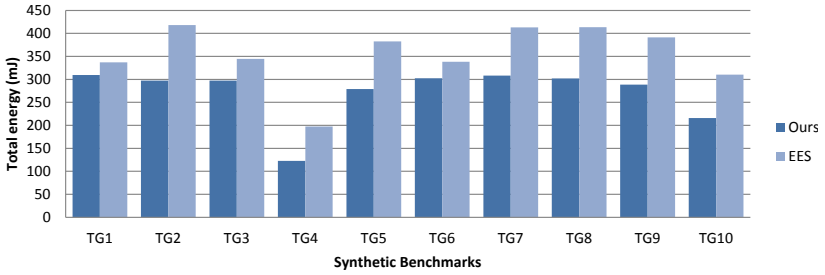


Figure 4.11: Total energy consumption of our approach and EES [2] for synthetic benchmarks on 5-processors platform.

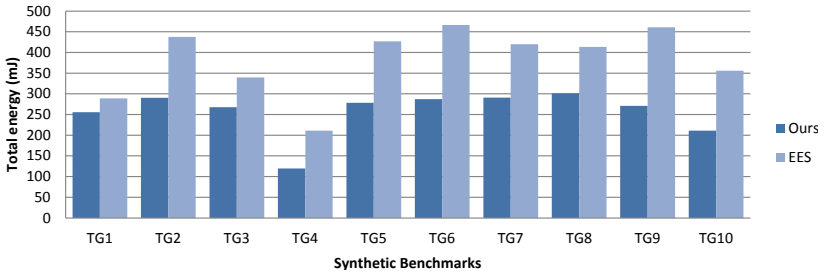


Figure 4.12: Total energy consumption of our approach and EES [2] for synthetic benchmarks on 8-processors platform.

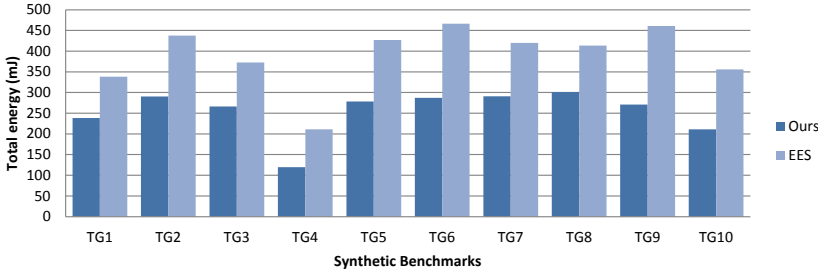


Figure 4.13: Total energy consumption of our approach and EES [2] for synthetic benchmarks on 10-processors platform.

4.6 Summary

In this chapter, we present a unified approach to the problem of scheduling a set of non-preemptible tasks with precedence constraints and individual deadlines on an MPSoC with continuous frequencies such that the total dynamic processor energy consumption or the

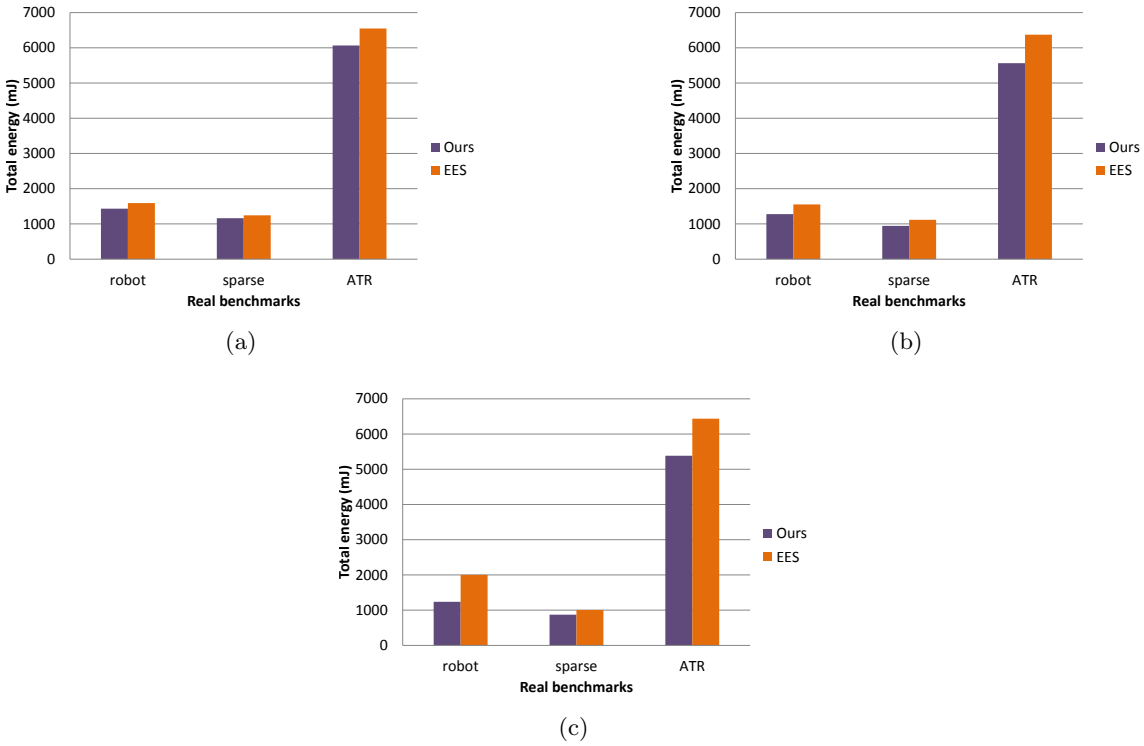


Figure 4.14: Total energy consumption of our approach and EES [2] for real benchmarks on (a) 5-processors platform, (b) 8-processors platform and (c) 10-processors platform.

total processor energy consumption of all the tasks is minimized. Our scheduling approach has two major features. Firstly, it uses the approximate successor-tree-consistent deadline of each task for task assignment and scheduling. Secondly, it formulates the problem of selecting an optimal frequency and its corresponding supply voltage for each task into an NLP problem. Experimental results show that our approach significantly outperforms the two state-of-the-art approaches, the EES approach [2] and the LESG approach [1].

Chapter 5

Energy-aware Task Scheduling on Heterogeneous MPSoCs

In this chapter, we investigate the energy-aware task scheduling problem for heterogeneous DVFS-enabled MPSoCs considering discrete frequency model, with communication contention consciousness. This chapter is structured as follows. Section 5.1 describes our major contributions. Section 5.2 defines the system models and some definitions. Section 5.3 provides some motivational examples. Section 5.4 presents our iterative NLP-based algorithm for scheduling tasks as well as computing the optimal frequencies for all tasks and messages. Section 5.5 explains our two approaches to compute discrete frequencies of all the tasks and communication messages: an ILP-based algorithm to select an optimal discrete frequency for each task and each message and a polynomial-time heuristic to assign a discrete frequency to each task and each message. Section 5.6 shows the experimental results and analyses. Section 5.7 summarizes this chapter.

5.1 Introduction

In this chapter, we investigate the following energy-aware task scheduling problem. Given a set of tasks with precedence and individual deadlines constraints, construct a feasible schedule on a heterogeneous, NoC-based, Dynamic Voltage and Frequency Scaling (DVFS)-enabled MPSoC with discrete frequencies such that the total computation and communication energy consumption of all the tasks is minimized. We make the following major contributions:

- We propose a novel offline energy-aware scheduling heuristic that assigns each task to a processor, constructs a feasible schedule for all the tasks and messages, and assigns an optimal frequency to each task and each message using convex NLP under continuous frequency model.
- We present an ILP-based algorithm to assign an optimal discrete frequency to each task and each communication link, and a novel polynomial-time heuristic to assign a discrete frequency to each task and each message.
- We have implemented our approaches and compared them with two state-of-the-art approaches, ETFGBF proposed by Li and Wu [3] and CA-TMES-Search proposed by Han et al. [4] by using a set of synthetic benchmarks. Experimental results show that the maximum improvement, the average improvement and the minimum improvement of our approach using ILP over the ETFGBF are 69.40%, 46.30% and 18.45%, respectively. The maximum improvement, the average improvement and the minimum improvement of our approach using ILP over the CA-TMES-Search are 48.35%, 34.98% and 13.52%, respectively. Moreover, the performance of our approach using the heuristic is very close to that of our approach using ILP.

5.2 Problem, Definitions and Models

Table 5.1 shows a list of notations used throughout this chapter.

The target MPSoC consists of m heterogeneous processors $P = \{p_1, p_2, \cdots, p_m\}$ interconnected via a 2D mesh homogeneous NoC. Each processor has its own local memory and may employ a distinct instruction set architecture (ISA). Each processor and each communication link is DVFS-enabled. Each processor and each communication link could function on a set of voltage and frequency levels.

The target application is represented by a weighted directed acyclic graph (DAG) $G = (T, E, W, C)$, where each node in T denotes a non-preemptible task, each edge in E denotes the precedence between two tasks, and each node weight in W is a m-tuple, denoting the WCETs in cycles of the corresponding task on the m different processors, and each edge weight in C denotes the size of the message in unit data to be transferred between two tasks. 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 for transferring the data is $e_{i,j} = c_{i,j}/(\lambda f_{i,j})$, where $c_{i,j}$, λ and $f_{i,j}$ are the message size, link data width and link frequency, respectively. If task t_i and task t_j are on the same processor, $e_{i,j} = 0$. When a message $M_{i,j}$ is sent from task t_i to task t_j on different processors, XY routing is used and all the communication links on the routing path for $M_{i,j}$ have the same frequency. Each task has a predefined deadline.

We consider the total energy consumption that is composed of the computation energy and communication energy. Given a supply voltage v_{s_i} and an operating frequency f_i, the total computation power $\rho_{tot}(v_{s_i}, f_i)$ of a task t_i is computed as follows [25]:

$$\rho_{tot}(v_{s_i}, f_i) = \underbrace{C_e v_{s_i}^2 f_i}_{\text{dynamic power}} + \underbrace{L_g(v_{s_i} K_3 e^{K_4 v_{s_i}} e^{K_5 v_b} + |v_b| I_j)}_{\text{static power}} \tag{5.1}$$

where C_e is the average switched capacitance, 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, v_b and I_j are the body-

Table 5.1: Notations used.

Notation	Description	Notation	Description
T	A set of application tasks	P	A set of processors
t_i	The i$^{\text{th}}$ application task	p_k	The k$^{\text{th}}$ processor
$M_{i,j}$	The message from t_i to t_j	$c_{i,j}$	Data size of $M_{i,j}$
d_i	Predefined deadline of t_i	$d'_i(d'_{i,j})$	The successor-tree-consistent deadline of $t_i(M_{i,j})$
$v_{s_i}(v_{s_{i,j}})$	The supply voltage of $t_i(M_{i,j})$	$f_i(f_{i,j})$	The operating frequency of $t_i(M_{i,j})$
$\varepsilon_i(v_{s_i}, f_i)$	Total computation energy of t_i	$\varepsilon_{i,j}(v_{s_{i,j}})$	Total communication energy of $M_{i,j}$
λ	Communication link data width	$h_{s,d}$	The Manhattan distance between p_s and p_d
w_i	Worst-case number of cycles of t_i	s_i	Start time of a task or message
e_i	Execution time of t_i	$e_{i,j}$	Communication time of $M_{i,j}$
$pred(t_i)$	All predecessors of t_i	$succ(t_i)$	All successors of t_i
x_i	Binary decision variable of a task or message	$f_i^{opt}(f_{i,j}^{opt})$	The optimal frequency of $t_i(M_{i,j})$
$f_i^l(f_{i,j}^l)$	The lower frequency of $t_i(M_{i,j})$	$f_i^{l+1}(f_{i,j}^{l+1})$	The higher frequency of $t_i(M_{i,j})$
$v_{s_i}^l(v_{s_{i,j}}^l)$	The lower voltage of $t_i(M_{i,j})$	$v_{s_i}^{l+1}(v_{s_{i,j}}^{l+1})$	The higher voltage of $t_i(M_{i,j})$
$\epsilon_i^l(\epsilon_i^{l+1})$	Execution time of t_i when using $f_i^l(f_i^{l+1})$	$\epsilon_{i,j}^l(\epsilon_{i,j}^{l+1})$	Communication time of $M_{i,j}$ when using $f_{i,j}^l(f_{i,j}^{l+1})$
Ω_i	Finish time of t_i	ω	The time gain metric
γ	The normalize gain metric	$\triangle\varepsilon_i(\triangle\varepsilon_{i,j})$	The energy difference metric of $t_i(M_{i,j})$

Notation	Description	Notation	Description
N_{rep}	The number of repeaters of a routing path	C_{rep}	Capacity load of a repeater
s_a	The average switching activity of a link	I_j	The body junction leakage current
C_e	The average switched capacitance of a processor	L_g	Number of logic gates in a circuit
ε^R	The energy of one unit data on a router	v_b	The body-bias voltage
v_{min_k}	The minimum supply voltage of p_k	v_{max_k}	The maximum supply voltage of p_k
f_{min_k}	The minimum frequency of p_k	f_{max_k}	The maximum frequency of p_k
v^l_{min}	The minimum supply voltage of a link	v^l_{max}	The maximum supply voltage of a link
f^l_{min}	The minimum frequency of a link	f^l_{max}	The maximum frequency of a link
$K_l (l = 3, \cdots, 5)$ Processor technology constants			

bias voltage and body junction leakage current, respectively. The total computation energy of a task t_i is calculated as follows:

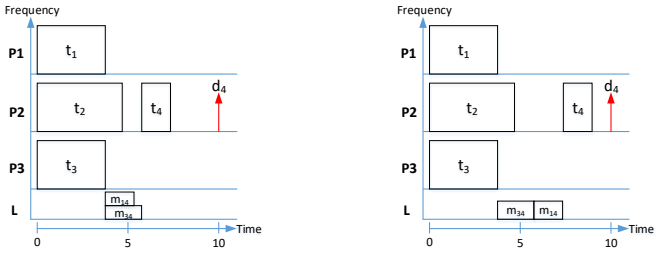
$$\varepsilon_i(v_{s_i}, f_i) = \rho_{tot}(v_{s_i}, f_i)e_i \tag{5.2}$$

where e_i is the execution time of task t_i.

The total communication energy $\varepsilon_{i,j}$ of a message $M_{i,j}$ over the links with a voltage $v_{s_{i,j}}$ is computed as follows [3, 4, 87, 88]:

$$\begin{aligned}\varepsilon_{i,j}(v_{s_{i,j}}) = c_{i,j}(h_{s,d}+1)\varepsilon^R + h_{s,d}(\sum^{N_{rep}} \frac{c_{i,j}}{\lambda} s_a C_{rep} v^2_{s_{i,j}} \\ + L_g(v_{s_{i,j}} K_3 e^{K_4 v_{s_{i,j}}} e^{K_5 v_b} + |v_b| I_j) e_{i,j})\end{aligned} \tag{5.3}$$

where $c_{i,j}$, $h_{s,d}$, N_{rep}, ε^R, and λ are the data size of $M_{i,j}$, the Manhattan distance, the number of repeaters of the routing path, the energy consumption of one unit data on a



(a) (b)

Figure 5.1: (a) A non-contention-aware schedule, and (b) a contention-aware schedule.

router, and the link data width, respectively, and s_a and C_{rep} are the average switching activity and capacity load of a repeater, respectively.

5.3 Motivational Examples

In this section, we illustrate two examples. Firstly, on why it is important to consider communication contention in task scheduling. Secondly, we depict the difference between major approaches for selecting a discrete frequency of each task.

5.3.1 Communication Contention Awareness

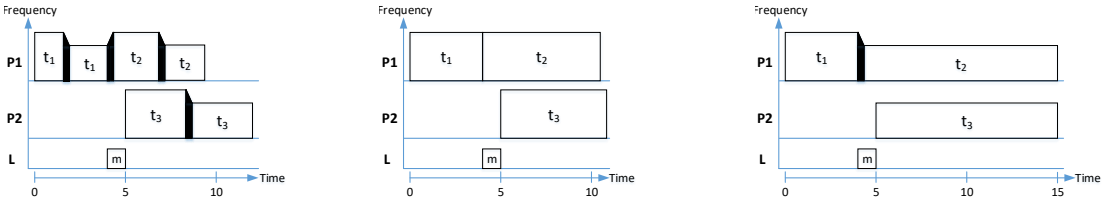
It is important to explicitly consider message scheduling when computing an offline schedule. This is because it affects not only the timing accuracy of the schedule, but also its total energy consumption when scaling is performed. Consider the examples in Figure 5.1. When the schedule is constructed without the consideration of communication contention, the available static slack of the schedule is larger as in Figure 5.1a, than the available slack in a real scenario with the consideration of contention as in Figure 5.1b. Thus, the amount of energy savings would be optimistically computed when communication conflicts are not taken into account.

5.3.2 Discrete Voltage and Frequency Selection

There are two major approaches to assign an optimal discrete frequency for each task given its optimal continuous frequency; intra-task and inter-task. We illustrate the results with a simple example.

Intra-task discrete assignment approach breaks each task execution cycle into two parts. One part uses the minimum frequency higher than its optimal continuous frequency and another uses the maximum frequency not greater than its optimal continuous frequency. Figure 5.2a shows that although the intra-task discrete assignment approach produces an optimal result but it may introduce larger transition overhead in terms of time and energy. This is because the frequency changes not only between the boundary of tasks but also within the boundary of each task.

On the other hand, a naive inter-task approach is to assign each task with the minimum frequency greater than its optimal continuous frequency. Figure 5.2b shows that although this approach guarantees the deadline of each task, it unnecessarily increases the energy consumption of the schedule. Our approach attempts to find an optimal result based on inter-task discrete assignment. Firstly, it tries to assign to each task the maximum frequency not greater than its optimal continuous frequency. If the schedule is feasible, we use the schedule. However, if the schedule is not feasible, our approach finds a minimum set of tasks that uses the minimum frequency greater than their optimal continuous frequencies. Then, we adjust the schedule accordingly. It is clear from Figure 5.2c that our approach managed to minimize the number of transition overhead compared to the approach in Figure 5.2a and significantly reduces the energy consumption as compared to the naive approach in Figure 5.2b.



(a) (b) (c)

Figure 5.2: Different approaches to discrete voltage and frequency assignment: (a) intra-task assignment, (b) a naive inter-task assignment using higher frequencies, (c) inter-task assignment using our approach.

5.4 Task Scheduling

Conceptually, the scheduling phase has three major steps: computing the priorities of each task and each message, assigning each task to a processor based on its priority and constructing a schedule based on Earliest Deadline First (EDF) strategy assuming the maximum frequencies for the tasks and communication links considering communication link contentions, and computing the optimal frequency for each task and each message using NLP. However, the second step and the third step are combined.

5.4.1 Computing Priorities

Our approach computes the approximate successor-tree-consistent deadline d'_i defined in Section 5.2 as a priority for each task $t_i \in T$ and $d'_{i,j}$ for each edge $(t_i, t_j) \in E$. A smaller deadline implies a higher priority. We used the term approximate successor-tree-consistent deadline interchangeably with modified deadline throughout this chapter. We sought to compute the successor-tree-consistent deadlines for each edge as to resolve the message sequence, in the case of communication contention. The basic idea is to let the message destined for a high priority task to have higher priority than other competing messages. The approximate successor-tree-consistent deadlines of all the tasks are computed in re-

verse topological order while the approximate successor-tree-consistent deadline 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, $d'_i = d'_j - \min_{\forall p_k \in P}(e_j)$, where t_j is the successor of t_i and t_j has the minimum execution time under maximum frequency among all the processors in P. The approximate successor-tree-consistent deadline $d'_{i,j}$ of the edge (t_i, t_j) is set to $d'_{i,j} = d'_i$.
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.
   (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 non-increasing order of their edge weights.
   (e) Schedule each task in V on a processor such that its start time is maximized.
   (f) For each edge (t_i, t_j), incident from t_i and incident to $t_j \in V$ in the schedule, $d'_{i,j} = d'_j - e_j$, where the execution time e_j is computed assuming maximum frequency of t_j processor.

(g) Find the latest completion time of t_i in the schedule for the tasks in $U \cup V$, respecting the 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.

5.4.2 Task Assignment and Scheduling

Our approach assigns each task t_i to a processor p_k and constructs a schedule for all the tasks and messages assuming the maximum frequencies as follows.

- 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 the processor p_k.
     (b) Compute the schedule of t_i and assign optimal frequencies for t_i and all tasks and messages currently scheduled under continuous frequency model such that the total energy consumption of all the tasks and messages is minimized by solving an NLP formulation described in Section 5.4.3.
  3. Select a partial schedule σ_k with minimum total energy among all the schedules each of which results from assigning t_i to each processor p_k. If there exists multiple options, select the processor with the maximum number of t_i predecessors.
  4. Set t_i as scheduled.

5.4.3 Optimal Frequency Assignment

We have a set of scheduled tasks and messages. Our objective is to assign an optimal frequency to each task and each message under continuous frequency model such that the

total energy consumption of all the tasks and messages is minimized. Next, we formulate this problem as an NLP problem.

Firstly, for all the scheduled tasks and messages, we construct a new node-weighted graph $G' = \{T' \cup M', E'\}$, where T', M' represent a set of scheduled tasks, a set of schedule messages, respectively, and E' is a set of edges. 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' to resolve 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'.

Note that G' can be constructed more efficiently in an incremental way. We formulate the optimal frequency assignment problem into an NLP problem as follows:

The objective function of the NLP formulation is shown as below.

$$\min\{\underbrace{\sum_{t_i \in T'} \varepsilon_i(v_{s_i}, f_i)}_{\text{computation energy}} + \underbrace{\sum_{M_{i,j} \in M'} \varepsilon_{i,j}(v_{s_{i,j}})}_{\text{communication energy}}\} \tag{5.4}$$

Next, we derive all the constraints based on G' as follows:

1. The execution time constraint for each task $t_i \in T'$:

$$e_i = w_i / f_i \tag{5.5}$$

where e_i, w_i, and f_i are the execution time, worst-case number of cycles, and processor frequency of t_i, respectively.

2. The communication time constraint for each message $M_{i,j} \in M'$:

$$e_{i,j} = c_{i,j}/(\lambda f_{i,j}) \tag{5.6}$$

where $e_{i,j}$, $c_{i,j}$, λ, and $f_{i,j}$ are the communication time, data size, link data width and link frequency of $M_{i,j}$, respectively.

3. The precedence constraints:

$$s_i + e_i\ (e_{i,j}) \leq s_j \quad \forall (u_i, u_j) \in E' \tag{5.7}$$

where s_i is the start time of a task or a message, and e_i ($e_{i,j}$) is the execution (communication) time of $t_i \in T'$ ($M_{i,j} \in M'$).

4. The supply voltage range constraints and the frequency range constraints for each task $t_i \in T'$:

$$v_{min_k} \leq v_{s_i} \leq v_{max_k} \tag{5.8}$$

$$f_{min_k} \leq f_i \leq f_{max_k} \tag{5.9}$$

where v_{min_k}, v_{max_k}, f_{min_k} and f_{max_k} 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 constraints and the frequency range constraints for each message $M_{i,j} \in M'$:

$$v^l_{min} \leq v_{s_{i,j}} \leq v^l_{max} \tag{5.10}$$

$$f^l_{min} \leq f_{i,j} \leq f^l_{max} \tag{5.11}$$

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

6. Deadline constraint for each task $t_i \in T'$:

$$s_i + e_i \leq d'_i \tag{5.12}$$

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 (4.3) in Section 5.2 is used.

The objective function of the NLP formulation is convex, and therefore could be solved in polynomial time [87].

5.5 Discrete Voltage and Frequency Selection

In this section, we propose an ILP-based approach and a heuristic approach to assign a discrete frequency to each task and each message, aiming at minimizing the total energy consumption of all the tasks and messages. Both our ILP-based algorithm and our heuristic are based on the final graph $G' = \{T' \cup M', E'\}$ constructed by our NLP-based algorithm.

5.5.1 ILP-based Approach

For each task t_i and each message $M_{i,j}$, let f_i^{opt} and $f_{i,j}^{opt}$ 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_i^l be the largest frequency of the processor where t_i is assigned such that f_i^l

is less than f_i^{opt}, and f_i^{l+1} be the frequency at a higher level of that processor. Similarly, let $f_{i,j}^{l}$ be the largest frequency of a link that is less than $f_{i,j}^{opt}$, and $f_{i,j}^{l+1}$ be the frequency at a higher level of that link. Clearly, in an optimal schedule, the frequency for t_i must be either f_i^l or f_i^{l+1}, and the frequency for $M_{i,j}$ must be either $f_{i,j}^{l}$ or $f_{i,j}^{l+1}$.

Therefore, we introduce a binary decision variable x_i to choose either f_i^l or f_i^{l+1} for each task $t_i \in T'$ (either $f_{i,j}^{l}$ or $f_{i,j}^{l+1}$ for each message $M_{i,j} \in M'$) as follows:

$$x_i = \begin{cases} 0, & \text{if } (v_{s_i}^l,\, f_i^l)\ ((v_{s_{i,j}}^l,\, f_{i,j}^l)) \text{ is used} \\ 1, & \text{if } (v_{s_i}^{l+1},\, f_i^{l+1})\ ((v_{s_{i,j}}^{l+1},\, f_{i,j}^{l+1})) \text{ is used} \end{cases} \tag{5.13}$$

where $v_{s_i}^l$ and $v_{s_i}^{l+1}$ are the corresponding supply voltages of f_i^l and f_i^{l+1}, respectively, and $v_{s_{i,j}}^l$ and $v_{s_{i,j}}^{l+1}$ are the corresponding supply voltages of $f_{i,j}^l$ and $f_{i,j}^{l+1}$, respectively.

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

$$\varepsilon^{comp} = \sum_{t_i \in T'} (1 - x_i)\varepsilon_i(v_{s_i}^l, f_i^l) + x_i\varepsilon_i(v_{s_i}^{l+1}, f_i^{l+1}) \tag{5.14}$$

where $\varepsilon_i(v_{s_i}^l, f_i^l)$ and $\varepsilon_i(v_{s_i}^{l+1}, f_i^{l+1})$ are the total computation energy consumption of t_i at the frequency f_i^l and at the frequency f_i^{l+1}, respectively. Notice that both $\varepsilon_i(v_{s_i}^l, f_i^l)$ and $\varepsilon_i(v_{s_i}^{l+1}, f_i^{l+1})$ are constants.

Similarly, the total communication energy is calculated as follows:

$$\varepsilon^{comm} = \sum_{M_{i,j} \in M'} (1 - x_i)\varepsilon_{i,j}(v_{s_{i,j}}^l) + x_i\varepsilon_{i,j}(v_{s_{i,j}}^{l+1}) \tag{5.15}$$

where $v_{s_{i,j}}^l$ and $v_{s_{i,j}}^{l+1}$ are the corresponding supply voltages of $f_{i,j}^l$ and $f_{i,j}^{l+1}$, respectively.

The objective function of the ILP formulation is shown as below.

$$\min\{\varepsilon^{comp} + \varepsilon^{comm}\} \tag{5.16}$$

The execution time constraint for each task $t_i \in T'$ are as follows.

$$e_i = (1 - x_i)\epsilon_i^l + x_i\epsilon_i^{l+1} \tag{5.17}$$

where ϵ_i^l and ϵ_i^{l+1} are the execution times of t_i when using f_i^l and f_i^{l+1}, respectively. Notice that ϵ_i^l and ϵ_i^{l+1} are constants.

Similarly, the communication time constraint for each message $M_{i,j} \in M'$ are as follows.

$$e_{i,j} = (1 - x_i)\epsilon_{i,j}^l + x_i\epsilon_{i,j}^{l+1} \tag{5.18}$$

where $\epsilon_{i,j}^l$ and $\epsilon_{i,j}^{l+1}$ are the communication times of $M_{i,j}$ when using $f_{i,j}^l$ and $f_{i,j}^{l+1}$, respectively.

The precedence constraint (6.11) for each edge and the deadline constraint (6.16) for each task in the NLP formulation still hold.

5.5.2 Heuristic

The ILP-based algorithm is not scalable as the ILP problem is a well known NP-complete problem. Therefore, we propose a heuristic to effectively solve the discrete voltage and frequency assignment problem.

Our heuristic is based on the initial feasible schedule constructed by our iterative NLP formulation described in Section 5.4. It works as follows:

1. For each t_i ($M_{i,j}$), if its optimal frequency computed by our NLP formulation is equal to a discrete frequency of the processor (link) where t_i ($M_{i,j}$) is mapped, assign the optimal frequency to t_i ($M_{i,j}$). Otherwise, assign f_i^l ($f_{i,j}^l$) to t_i ($M_{i,j}$).

2. Construct a schedule using the frequency assigned to each task (message) such that the relative order between any two nodes on each processor (link) remains the same as in the initial schedule constructed by our NLP-based algorithm.

3. If there is no late task in the schedule, our heuristic terminates. Otherwise, repeat the following until no late task exists.

   (a) Let t_k be the first late task. Repeat the following steps until t_k is not late.

      i. Construct a subgraph $G_1 = (\{t_k\} \cup pred(t_k),\ E_1)$ where $pred(t_k)$ is a set of predecessors of t_k and E_1 contains all the edges with their end nodes in $\{t_k\} \cup pred(t_k)$.

      ii. Compute the node cuts of G_1.

      iii. Find a set $B \subseteq \{t_k\} \cup pred(t_k)$ such that the discrete frequency of each t_i $(M_{i,j}) \in B$ has not been adjusted before and is not equal to the optimal frequency computed by NLP, and a set K of late tasks.

      iv. For each node $x_i \in B$, compute its rank. The rank of x_i is a 2-tuple $(1/\gamma(C_p), 1/\gamma(x_i))$, where C_p is a cut containing x_i, $\gamma(C_p)$ and $\gamma(x_i)$ are the normalized gains of C_p and x_i.

      v. If there exists a set of tasks (messages) in B such that their time gains are greater than or equal to the maximum tardiness of all the late tasks in K, select a task (message) with minimum normalized gain. Change its frequency to f_i^{l+1} $(f_{i,j}^{l+1})$ and update the schedule. Otherwise, do the following.

         A. Select a task (message) with the highest rank by comparing ranks lexicographically. Change its frequency to $f_i^{l+1}(f_{i,j}^{l+1})$.

         B. Update the schedule.

We compute the node cuts $C_p(p = 1, 2, \cdots)$ of graph G_1 as follows:

1. Create a copy G_2 of G_1 and repeat the following steps until G_2 is empty.

   (a) Create a new empty node cut C_p.

   (b) Add all the source nodes with zero indegrees in G_2 to C_p.

(c) Remove all the source nodes and their incident edges from G_2.

We introduce the time gain $\omega(t_i, K)$ ($\omega(M_{i,j}, K)$) for each task (message). This metric reflects the amount of time (benefits in terms of time) that could be utilized if t_i ($M_{i,j}$) uses f_i^{l+1} ($f_{i,j}^{l+1}$) in shifting a set of late tasks K in the schedule to finish at earlier times.

$$\omega(t_i, K) = \sum_{k=1}^{|K|} \Omega_k - \Omega'_k \quad \forall t_i, t_k \in T' \tag{5.19}$$

$$\omega(M_{i,j}, K) = \sum_{k=1}^{|K|} \Omega_k - \Omega'_k \quad \forall M_{i,j} \in M', t_k \in T' \tag{5.20}$$

where Ω_k and Ω'_k are the finish times of a late task $t_k \in K$ in the current schedule and the schedule when the frequency of t_i ($M_{i,j}$) is changed to f_i^{l+1} ($f_{i,j}^{l+1}$), respectively.

Furthermore, we define the energy difference $\triangle\varepsilon_i$ ($\triangle\varepsilon_{i,j}$) for each task (message). This metric measures the impact on the total energy of the schedule should t_i ($M_{i,j}$) uses f_i^{l+1} ($f_{i,j}^{l+1}$).

$$\triangle\varepsilon_i = \varepsilon_i^{l+1} - \varepsilon_i^l \tag{5.21}$$

$$\triangle\varepsilon_{i,j} = \varepsilon_{i,j}^{l+1} - \varepsilon_{i,j}^l \tag{5.22}$$

where ε_i^l ($\varepsilon_{i,j}^l$) and ε_i^{l+1} ($\varepsilon_{i,j}^{l+1}$) are the computation (communication) energy of t_i ($M_{i,j}$) using the frequency f_i^l ($f_{i,j}^l$) and when the frequency of t_i ($M_{i,j}$) is changed to f_i^{l+1} ($f_{i,j}^{l+1}$), respectively.

Next, we compute a normalize gain metric of t_i based on its time gain metric and its energy difference metric as follows.

$$\gamma(t_i) = \triangle\varepsilon_i/\omega(t_i, K) \quad \forall t_i \in T' \tag{5.23}$$

Similarly, a normalize gain metric for each message $M_{i,j}$ is computed as follows.

$$\gamma(M_{i,j}) = \triangle\varepsilon_{i,j}/\omega(M_{i,j}, K) \quad \forall M_{i,j} \in M' \tag{5.24}$$

In some cases, we observe that, we need to take into account a subset of tasks (messages) collectively. Thus, we define the normalized gain of a cut C_p consisting a set of tasks (messages) as follows.

$$\gamma(C_p) = \sum_{t_i \ (M_{i,j}) \in C_p} \triangle\varepsilon_i/\omega(C_p, K) \tag{5.25}$$

where we consider the total energy difference of all the tasks (messages) in cut C_p, when the frequencies of t_i $(M_{i,j}) \in C_p$ are changed to f_i^{l+1} $(f_{i,j}^{l+1})$ and the time gain, when the frequencies of t_i $(M_{i,j}) \in C_p$ are changed to f_i^{l+1} $(f_{i,j}^{l+1})$.

The time complexity of our heuristic is $O(ne)$, where n is the number of nodes, and e is the number of edges in the task graph.

5.6 Experimental Results

In order to evaluate our approaches, we compare them with two state-of-the-art approaches, ETFGBF [3] and CA-TMES-Search [4] by using a set of synthetic benchmarks. We choose ETFGBF [3] and CA-TMES-Search [4] because they are the latest approaches considering the similar problem structure as ours. Since CA-TMES-Search [4] is applicable for homogeneous VFI-based platform, we consider each processor as a voltage-frequency island (has individual voltage and frequency) in the simulations in this study.

5.6.1 Experimental Setup

We use a set of 18 synthetic benchmarks provided in [89]. We randomly pick these benchmarks to represent three groups of task graph sizes i.e. small (TG16, TG5, TG4, TG3), medium (TG2, TG17, TG18, TG1, TG6, TG7, TG8, TG9) and large (TG12, TG13, TG15, TG11, TG10, TG14). In addition, they represent various structures of task graphs in terms of the number of tasks, number of edges, the degree of parallelism and dependencies. These sets of benchmarks are widely used by the embedded systems community for task scheduling research.

We configure 23 scenarios to evaluate the performance of our algorithms. The characteristics of each benchmark in terms of the number of tasks $|T|$, the number of edges $|E|$, the CCR (communication to computation ratio) which is defined as the total communication divided by the average computation, and a common deadline D for each benchmark are shown in Table 5.2. The common deadline, the required number of computation clock cycles of each task on one processor and communication data size of each edge are provided from the source. We use a factor β to represent the computation heterogeneity of each task among different processors. The worst-case execution cycles $w_{i,k}$ of a task t_i on processor p_k is a random value in the following range.

$$w_i(1 - \beta/3) \leq w_{i,k} \leq w_i(1 + \beta/3) \tag{5.26}$$

where w_i is the provided worst-case execution cycles of a task t_i. For this experiment, we set $\beta = 1$.

We choose three sets of NoC-based architecture which are 2-by-2 mesh NoC, 3-by-3 mesh NoC and 4-by-4 mesh NoC as the target MPSoC. Each tile in the NoC is attached with different processors. We define two types of processors based on 0.07 μm technology in Table 5.3 and 0.18 μm technology in Table 5.4 [25] that follow the power model in Section 5.2. The accuracy of these technology parameters has been verified through SPICE

Table 5.2: The characteristics of benchmarks.

Benchmark	$\|T\|/\|E\|$	CCR	D	Benchmark	$\|T\|/\|E\|$	CCR	D
TG1	14/20	3.54	107	TG10	20/27	4.60	77
TG2	11/14	2.19	70	TG11	28/28	7.10	370
TG3	9/11	2.47	62	TG12	30/33	10.14	365
TG4	8/9	1.94	71	TG13	29/27	6.60	431
TG5	7/9	1.72	41	TG14	26/28	8.06	250
TG6	18/24	4.60	90	TG15	28/27	8.00	291
TG7	18/26	3.64	129	TG16	6/7	1.03	61
TG8	19/27	4.52	112	TG17	13/18	2.75	112
TG9	17/24	4.50	132	TG18	15/19	3.00	139

simulations [25]. Also, they were widely used in the previous energy-aware task scheduling research [33, 80–83].

We set both types of processors to have five voltage and frequency levels as in Table 5.5. The minimum supply voltage v_{min_k} and maximum supply voltage v_{max_k} follows the minimum supply voltage and maximum supply voltage of their corresponding discrete set. We arrange the processors with different types in alternate manner. Communication links have five voltage and frequency levels. To compute the communication energy, we use the parameters in Table 5.6 [3, 87].

Table 5.3: Constants of 0.07 μm processor technology.

Variable	Value	Variable	Value	Variable	Value
K_1	0.063	K_6	$5.26\text{x}10^{-12}$	v_{bs}	-0.70
K_2	0.153	C_{eff}	$4.30\text{x}10^{-10}$	v_{th1}	0.244
K_3	$5.38\text{x}10^{-7}$	I_j	$4.80\text{x}10^{-10}$	α	1.50
K_4	1.83	L_d	37.00		
K_5	4.19	L_g	$4.00\text{x}10^{6}$		

Table 5.4: Constants of 0.18 μm processor technology.

Variable	Value	Variable	Value	Variable	Value
K_1	0.053	K_6	$51.00\text{x}10^{-12}$	v_{bs}	-0.70
K_2	0.140	C_{eff}	$1.11\text{x}10^{-9}$	v_{th1}	0.359
K_3	$3.00\text{x}10^{-9}$	I_j	$2.40\text{x}10^{-10}$	α	1.50
K_4	1.63	L_d	37.00		
K_5	3.65	L_g	$4.00\text{x}10^{6}$		

Table 5.5: Voltage and frequency levels of 0.07 μm processor and 0.18 μm processor.

	0.07 μm technology			0.18 μm technology		
Level	Volt. (V)	Freq. (MHz)	Power (mW)	Volt. (V)	Freq. (MHz)	Power (mW)
1	0.85	2100	727.3	1.88	1000	4015.5
2	0.80	1810	557.2	1.68	800	2549.3
3	0.75	1530	415.5	1.47	600	1443.3
4	0.70	1260	299.3	1.22	400	666.9
5	0.65	1010	208.9	0.84	150	119.8

Table 5.6: Constants of repeater-based communication link.

Variable	Value	Variable	Value	Variable	Value
λ	32	ε^R	$0.01\text{x}10^{-9}$	C_{rep}	$0.11\text{x}10^{-8}$
N_{rep}	27	s_a	0.5		

We generate 23 scenarios based on the 18 benchmarks and 3 different NoC architectures. The configurations for each scenario are shown in Table 5.7.

We define Ours-ILP approach as the combination of our task mapping and scheduling approach with the ILP-based algorithm and Ours-Heu approach as the combination of our task mapping and scheduling approach with the heuristic described in Section 5.5.2.

We implement our approaches, ETFGBF [3] and CA-TMES-Search [4] on Matlab version

Table 5.7: Scenario configurations.

Scenario	Benchmark	Platform	Scenario	Benchmark	Platform
S1	TG1	2-by-2 Mesh	S13	TG13	4-by-4 Mesh
S2	TG2	2-by-2 Mesh	S14	TG14	4-by-4 Mesh
S3	TG3	2-by-2 Mesh	S15	TG15	4-by-4 Mesh
S4	TG4	2-by-2 Mesh	S16	TG1	4-by-4 Mesh
S5	TG5	2-by-2 Mesh	S17	TG2	4-by-4 Mesh
S6	TG6	3-by-3 Mesh	S18	TG3	4-by-4 Mesh
S7	TG7	3-by-3 Mesh	S19	TG4	4-by-4 Mesh
S8	TG8	3-by-3 Mesh	S20	TG5	4-by-4 Mesh
S9	TG9	3-by-3 Mesh	S21	TG16	4-by-4 Mesh
S10	TG10	3-by-3 Mesh	S22	TG17	4-by-4 Mesh
S11	TG11	4-by-4 Mesh	S23	TG18	4-by-4 Mesh
S12	TG12	4-by-4 Mesh			

R2016a. We utilize the Matlab fmincon solver for the NLP problem and Matlab intlinprog solver for the ILP problem. In addition, we use the Matlab quadprog solver for the task mapping algorithm of ETFGBF approach. 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 caches.

5.6.2 Results and Discussions

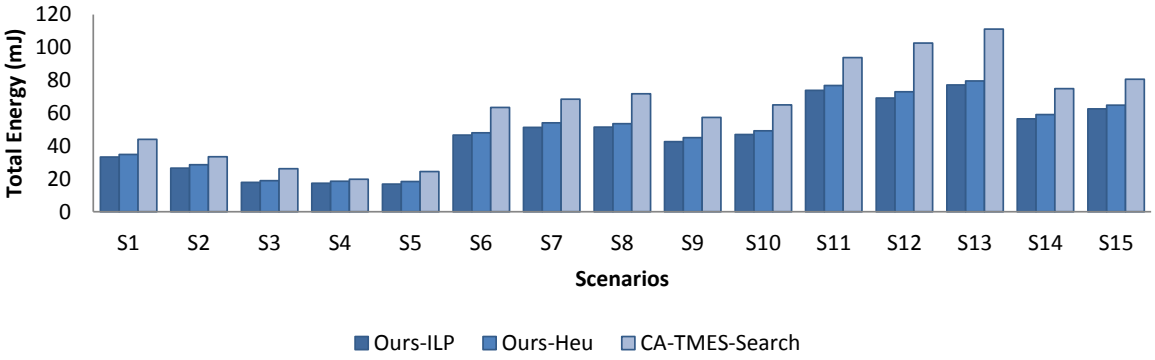
In this section, we discuss the simulation results based on three metrics: total energy consumption, total communication energy and algorithm running time. In order to compare with CA-TMES-Search [4], we run 15 scenarios ($S1, S2, \cdots, S15$). Figure 5.3a shows the total energy consumption (computation plus communication energy) of the scenarios by using Ours-ILP, Ours-Heu and CA-TMES-Search [4]. Each vertical axis denotes the

total energy consumption of each scenario using a specific approach, and each horizontal axis denotes the scenarios. The simulation results show that the maximum improvement, the average improvement and the minimum improvement of Ours-ILP compared to the CA-TMES-Search [4] are 48.35%, 34.98% and 13.52%, respectively. The greatest improvement of 48.35% occurs for scenario $S12$ (benchmark TG12 on a 4-by-4 mesh architecture) while the least improvement occurs for scenario $S4$ (benchmark TG4 on a 2-by-2 mesh architecture).

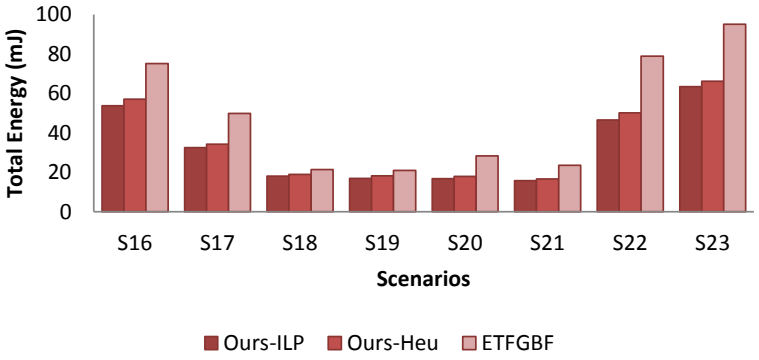
There are three key reasons on their poor performance on heterogeneous platforms. Firstly, their approach does not take into account the energy profile of each processor during task mapping while our approach guides the task mapping with the aim to minimize the total energy consumption (computation energy plus communication energy). Secondly, the execution time of each task varies among processors in a heterogeneous platform. Thus, their approach suffers when they do mapping based only on the earliest start time. Thirdly, the poor task mapping affects the potential of scaling the tasks and messages.

Figure 5.4a shows the total communication energy consumption of the different scenarios using Ours-ILP and CA-TMES-Search [4]. Each vertical axis denotes the total communication energy of each scenario using a specific approach, and each horizontal axis denotes the scenarios. It indicates that Ours-ILP has advantages compared to CA-TMES-Search [4] in all the scenarios except for scenarios $S1$ (TG1 on a 2-by-2 mesh architecture) and $S2$ (TG2 on a 2-by-2 mesh architecture). However, in this case where computation energy significantly dominates the total energy consumption, our approach still has the upper hand than CA-TMES-Search [4] in terms of total energy consumption. We observe that in certain cases, our approach assigns all the tasks only on one processor. This is due to allocating dependent tasks among different processors introduce message transfers through communication links, thus limiting the available slack and generating communication energy.

As for the comparison with ETFGBF [3], we are more interested to know the amount of



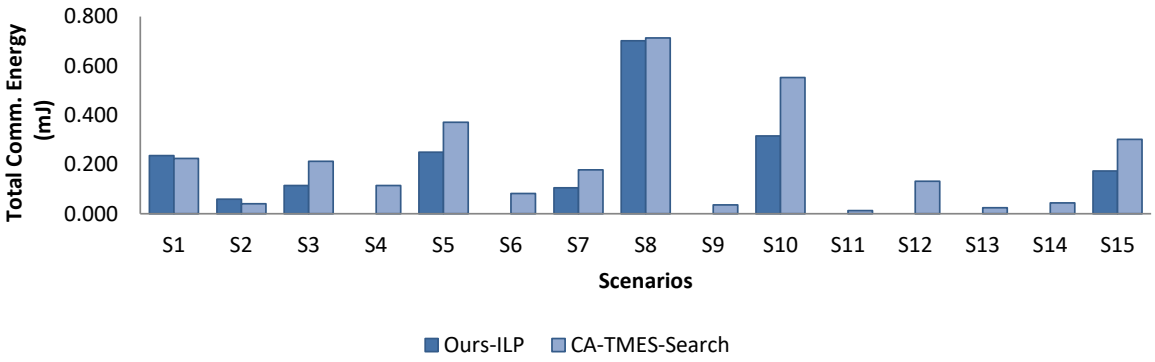
(a)



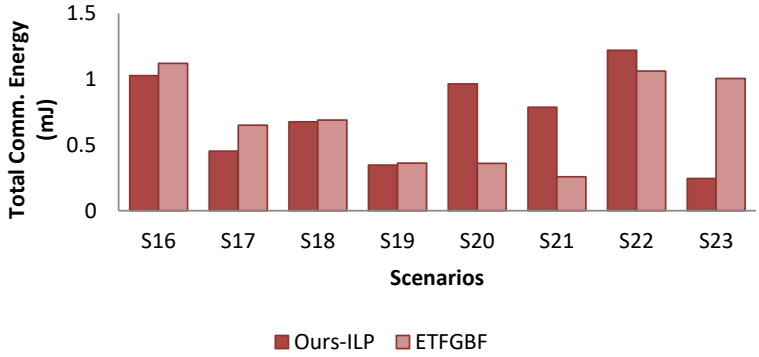
(b)

Figure 5.3: Total energy consumption of (a) Ours-ILP, Ours-Heu and CA-TMES-Search [4] and (b) Ours-ILP, Ours-Heu and ETFGBF [3].

communications produce by Ours-ILP and ETFGBF [3]. This is due to their approach which is set with constraints such that each processor could be assigned with at most one task. This avoids all tasks being placed on one processor and enables significant communications among tasks. In order to have a fair evaluation with ETFGBF [3], we configure Ours-ILP and Ours-Heu to have an additional constraint during mapping, where each processor could be assigned with only one task. We choose all the benchmarks having less than 16 tasks to run on a 4-by-4 mesh architecture which form the scenarios $S16, S17, \cdots, S23$. Figure 5.3b shows the total energy consumption (computation plus communication energy) of the scenarios using Ours-ILP, Ours-Heu and ETFGBF [3]. Each vertical axis denotes the total energy consumption of each scenario using a specific approach, and each horizon-



(a)



(b)

Figure 5.4: Total communication energy consumption of (a) Ours-ILP and CA-TMES-Search [4] and (b) Ours-ILP and ETFGBF [3].

tal axis denotes the scenarios. The results show that Ours-ILP outperforms ETFGBF [3] in terms of total energy for all the scenarios. Ours-ILP achieves an average of 46.30% for all scenarios with maximum improvement of 69.40% in scenario $S22$ (benchmark TG17 on a 4-by-4 mesh architecture) and the least improvement of 18.45% for scenario S18 (benchmark TG3 on a 4-by-4 mesh architecture). Figure 5.4b shows the total communication energy consumption of the scenarios using Ours-ILP and ETFGBF [3]. Each vertical axis denotes the total communication energy of each scenario using a specific approach, and each horizontal axis denotes the scenarios. The results show that Ours-ILP achieves improvements in all the scenarios, with the exception of scenarios $S20$, $S21$ and $S22$.

In general, the previous three reasons for the good performance of our approach still hold. Our task assignment strives to map each task to a processor with a minimum total energy. On the other hand, the mapping in ETFGBF [3] aims at minimizing the communication. It attempts to reduce the total weighted distances which is the sum of communication data size multiplied by the distance of each edge. In other words, the approach places tasks with large communication data size on nearby processors. This results in negative impacts when computing the discrete frequencies as the mapping only optimizes the communications rather than energy consumption. Moreover, the GA-based scaling approach cannot find the exact optimal solutions due to it being based on natural selection.

We compare the performance of our heuristic with our ILP-based algorithm for assigning a discrete frequency to each task. The simulation results in Figure 5.3 shows that the maximum total energy difference, the average total energy difference and the minimum total energy difference of Ours-Heu approach compared to the Ours-ILP approach are 8.45%, 5.07% and 2.75%, respectively. It shows that our heuristic is very efficient in computing near-optimal solutions for all scenarios.

We observe that it is important to explicitly consider communication contention when computing an offline schedule. This is because it affects not only the timing accuracy of the schedule but also its total energy consumption when scaling is performed. This is because, the amount of slack differs for schedule considering communication contention and the one without considering communication contention.

In addition, we show the running times to construct the schedules produced by these approaches: Ours-ILP, Ours-Heu, CA-TMES-Search [4] and ETFGBF [3]. Table 5.8 summarizes the average running times of Ours-ILP, Ours-Heu, CA-TMES-Search [4] and ETFGBF [3] approaches on each NoC architecture. For all the benchmarks, the average running time ratio of the CA-TMES-Search [4] approach compared to the Ours-ILP is 1.03% while the average running time ratio of ETFGBF [3] compared to the Ours-ILP

is 7.41%. As we can see, our approaches are much slower than the CA-TMES-Search [4] and ETFGBF [3] approaches. We found that the running time of our iterative NLP-based mapping algorithm dominates approximately 98% of our approaches. Table 5.9 shows the average running time ratio of the heuristic compared to the ILP is 70.39%. Although the execution time is slower, our approaches compute much better energy-efficient schedules compared to the CA-TMES-Search [4] and ETFGBF [3] approaches do. Notice that our approaches target embedded systems. Therefore, the running time for constructing an offline schedule at design stage is not an issue.

Table 5.8: Average running times of our approaches, CA-TMES-Search [4] and ETFGBF [3] on each architecture.

Architecture	Ours-ILP (s)	Ours-Heu (s)	CA-TMES-Search (s)	ETFGBF (s)
2-by-2	114.530	114.196	10.650	-
3-by-3	1279.948	1264.127	13.852	-
4-by-4	2602.872	2593.947	16.631	192.951

Table 5.9: Average running times of our discrete frequency assignment approaches, ILP and Heuristic on each architecture.

Architecture	ILP (s)	Heuristic (s)
2-by-2	0.082	0.039
3-by-3	0.062	0.052
4-by-4	0.062	0.054

5.7 Summary

In this chapter, we present two novel approaches addressing the problem of scheduling a set of non-preemptible tasks with precedence constraints and individual deadlines on a heterogeneous NoC-based MPSoC with discrete frequencies such that the total energy con-

sumption of all the tasks is minimized. We explicitly consider communication contention. Our two approaches consist of an iterative NLP-based algorithm for task assignment and continuous frequency selection of all tasks and messages. One approach uses an ILP-based algorithm for selecting optimal discrete frequencies to all tasks and messages, while the other uses a polynomial-time heuristic for selecting discrete frequencies to all tasks and messages. We have evaluated our approach and compared it with two state-of-the-art approaches, ETFGBF [3] and CA-TMES-Search [4]. Experimental results show that our approaches perform significantly better than the previously reported approaches, in terms of total energy consumption.

Chapter 6

Energy-aware Task Scheduling for Streaming Applications

This chapter describes our energy-aware task scheduling approach for streaming applications on NoC-based MPSoCs under memory capacity constraints. We propose an integrated approach consisting of task-level software pipelining and DVFS. This chapter is organized as follows. Section 6.1 describes our major contributions. Section 6.2 specifies the system models and some definitions. Section 6.3 illustrates motivational examples. Section 6.4 proposes the framework of our unified approach which includes constructing a workload balanced initial schedule, a retiming algorithm for increasing processing parallelism and an iterative algorithm for fixing the schedule in case of memory capacity violations. Section 6.5 discusses the experimental results and analyses. Section 6.6 summarizes this chapter.

6.1 Introduction

In this chapter, 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 major contributions.

- We propose a novel unified energy-aware scheduling approach which integrates task-level software pipelining with DVFS under memory capacity 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 capacity constraints of the system.

- We present a novel task mapping and scheduling approach to construct an initial schedule under maximum frequencies in which, when combined with retiming and DVFS further reduces the total energy consumption. 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 [18].

- We have implemented our approach and compared them with two state-of-the-art approaches, RDAG+GeneS [5] and JCCTS [6] by 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 compared to the RDAG+GeneS [5] are 40.82%, 17.31% and 7.53%, respectively. The maximum

improvement, the average improvement and the minimum improvement of our approach compared to the JCCTS [6] are 46.46%, 21.67% and 10.75%, respectively.

6.2 Problem, Definitions and Models

The problem we investigate is described as follows. Given a set of n non-preemptible periodic tasks $T = \{t_1, \cdots, t_n\}$ 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 under memory capacity constraints. A feasible schedule is a schedule that satisfies all the constraints.

Table 6.1 shows a list of notations used throughout this chapter.

6.2.1 System Model

The target MPSoC consists of m DVFS-enabled identical processors $P = \{p_1, p_2, \cdots, p_m\}$ interconnected via a 2-dimensional mesh NoC such as in Figure 6.1b. 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 $\{(v_1, f_1), \cdots, (v_{n_k}, f_{n_k}) : v_1 < \cdots < v_{n_k}, f_1 < \cdots < f_{n_k}\}$ of n_k discrete voltage and frequency levels. Each communication link could operate on $\{(v_1, f_1), \cdots, (v_p, f_p) : v_1 < \cdots < v_p$ and $f_1 < \cdots < f_p$ of p discrete voltage and frequency levels.

Table 6.1: Notations used.

Notation	Description	Notation	Description
T	A set of application tasks	P	A set of processors
t_i	The i$^{\text{th}}$ application task	p_k	The k$^{\text{th}}$ processor
$M_{i,j}$	The message from t_i to t_j	$c_{i,j}$	Data size of $M_{i,j}$
d_i	Predefined deadline of t_i	$d'_i(d'_{i,j})$	The successor-tree-consistent deadline of $t_i(M_{i,j})$
ρ	The application period	μ_k	The utilization of p_k
$r_i^{\eta}(r_{i,j}^{\eta})$	The release time of $t_i(M_{i,j})$ in the η^{th} period	w_i	Worst-case execution time of t_i
R_i	The retiming value of a task or message	li_i	The i$^{\text{th}}$ lifetime segment
b_k	The k$^{\text{th}}$ data buffer	R^{max}	The maximum retiming value of all tasks and messages
$v_{s_i}(v_{s_{i,j}})$	The supply voltage of $t_i(M_{i,j})$	$f_i(f_{i,j})$	The operating frequency of $t_i(M_{i,j})$
$\varepsilon_i(v_{s_i}, f_i)$	Total computation energy of t_i	$\varepsilon_{i,j}(v_{s_{i,j}})$	Total communication energy of $M_{i,j}$
λ	Communication link data width	$h_{s,d}$	The Manhattan distance between p_s and p_d
$e_i(e_{i,j})$	Execution(Communication) time of $t_i(M_{i,j})$	s_i	Start time of a task or message
$pred(t_i)$	All predecessors of t_i	$succ(t_i)$	All successors of t_i
x_i	Binary decision variable of a task or message	$f_i^{opt}(f_{i,j}^{opt})$	The optimal frequency of $t_i(M_{i,j})$
$f_i^l(f_{i,j}^l)$	The lower frequency of $t_i(M_{i,j})$	$f_i^{l+1}(f_{i,j}^{l+1})$	The higher frequency of $t_i(M_{i,j})$
$v_{s_i}^l(v_{s_{i,j}}^l)$	The lower voltage of $t_i(M_{i,j})$	$v_{s_i}^{l+1}(v_{s_{i,j}}^{l+1})$	The higher voltage of $t_i(M_{i,j})$

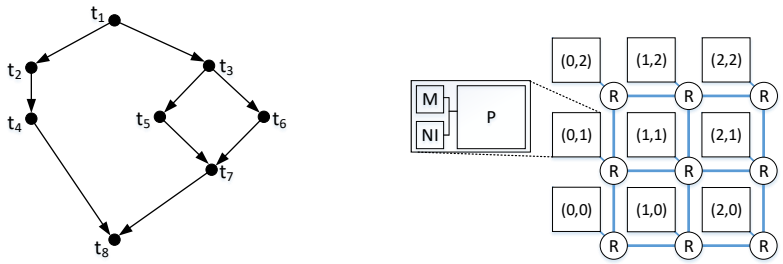
Notation	Description	Notation	Description
$\epsilon_i^l(\epsilon_i^{l+1})$	Execution time of t_i when using $f_i^l(f_i^{l+1})$	$\epsilon_{i,j}^l(\epsilon_{i,j}^{l+1})$	Communication time of $M_{i,j}$ when using $f_{i,j}^l(f_{i,j}^{l+1})$
ΔE_i	The total energy difference metric of a task or message	ΔMEM_i	The memory size difference metric of a task or message
C_{eff}	The average switched capacitance of a processor	L_g	Number of logic gates in a circuit
I_j	The body junction leakage current	v_{bs}	The body-bias voltage
v_{th1}	The threshold voltage	L_d	The logic depth
v_{min}	The minimum supply voltage of p_k	v_{max}	The maximum supply voltage of p_k
f_{min}	The minimum frequency of p_k	f_{max}	The maximum frequency of p_k
v_{min}^l	The minimum supply voltage of a link	v_{max}^l	The maximum supply voltage of a link
f_{min}^l	The minimum frequency of a link	f_{max}^l	The maximum frequency of a link
$K_l(l = 1, \cdots, 6)$,α Processor technology constants			

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 as below.

$$h_{s,d} = |x_d - x_s| + |y_d - y_s| \tag{6.1}$$

where (x_s, y_s) is the coordinate of p_s and (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-



(a) (b)

Figure 6.1: Examples of (a) a task graph, and (b) a 3-by-3 mesh NoC architecture

case execution time (WCET) of the corresponding task, and each edge weight in C denotes the size of the message 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 is equal to 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.

$$e_{i,j} = c_{i,j} b_{s,d} \tag{6.2}$$

where $c_{i,j}$ and $b_{s,d}$ are the message size and link transmission rate respectively. Otherwise, if task t_i and task t_j are assigned on the same processor, it is considered as intra-processor communication and thus, the communication time between task t_i and task t_j via the network is 0.

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

6.2.2 Power Model

The total power consumption of a processor consists of dynamic power due to switching activity, and static power as a result of leakage. The dynamic power P_{dyn} is formulated as follows [25].

$$P_{dyn} = C_{eff} v_s^2 f \tag{6.3}$$

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 given as follows [25].

$$P_{sta} = L_g(v_s K_3 e^{K_4 v_s} e^{K_5 v_{bs}} + |v_{bs}| I_j) \tag{6.4}$$

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

$$P_{tot} = P_{dyn} + P_{sta} \tag{6.5}$$

The relation between supply voltage v and frequency f is given as follows.

$$f = ((1 + K_1)v_s + K_2 v_{bs} - v_{th1})^{\alpha} / (L_d K_6) \tag{6.6}$$

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 power function and the total power function are convex. Under the total power model, there is an optimal minimum processor frequency f^{crit} such that the processor total energy increases as the processor frequency lower than f^{crit} further decreases [25].

We use equation (5.3) to compute the total communication energy of a message $M_{i,j}$ over communication links.

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 6.2.1 *Given a task graph $G = (T, E, W, C)$ 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 $T' = \{t_i\} \bigcup succ(t_i)$, $E' = \{(t_i,t_j) : t_j \in succ(t_i)\}$, $W' = \{w_j : w_j \in W$ and $t_j \in T'\}$ 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 6.2.2 *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 $T' = \{t_i\} \bigcup succ(t_i)$ 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 [17]. We use the approximate successor-tree-consistent deadline of each task as its priority when assigning each task to a processor.

6.3 Motivational Examples

In this section, we present two motivational examples. Firstly, a combination between a workload-balanced schedule and retiming could produce better energy-efficient schedule.

Secondly, a retiming technique based on task mapping may eliminate unnecessary memory usage when compared to an approach which transforms all intra-period dependencies into inter-period dependencies.

We use the task graph in Figure 6.1a executed on a two-processor platform. The worst-case execution time (WCET) of each task $t_i (i = 1, \ldots, 8)$ is $w_1 = w_2 = w_3 = w_4 = w_5 = w_6 = w_7 = w_8 = 1$, and 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.

Figures 6.2a, 6.2c and 6.2e show an initial schedule with balanced workloads across the two processors, its corresponding retimed schedule and its schedule after frequency scaling, respectively. Figures 6.2b, 6.2d and 6.2f show an initial schedule with unbalance 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 $E_{tot} = \sum_{i=1}^{|T|} f_i^3 e_i$ to compute the total energy, with E_{tot}, f_i, e_i being the total energy, operating frequency of task t_i and execution time of task t_i, respectively. Slack reclamation in Figure 6.2e shows that all the tasks can be assigned with an optimal frequency of 0.67 and approximately generating 3.612 mJ. On the other hand, Figure 6.2f 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 approximately generates 6.216 mJ. From this example, we can see that a workload-balanced initial schedule when combined with retiming can produce usable slack for all the processors, which can then be utilized for frequency scaling to reduce the total energy consumption. In other words, it generates a more energy-efficient schedule when compared to an unbalance initial schedule.

Next, we demonstrate that a retiming technique may produce a schedule with unnecessary memory usage requirement. 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

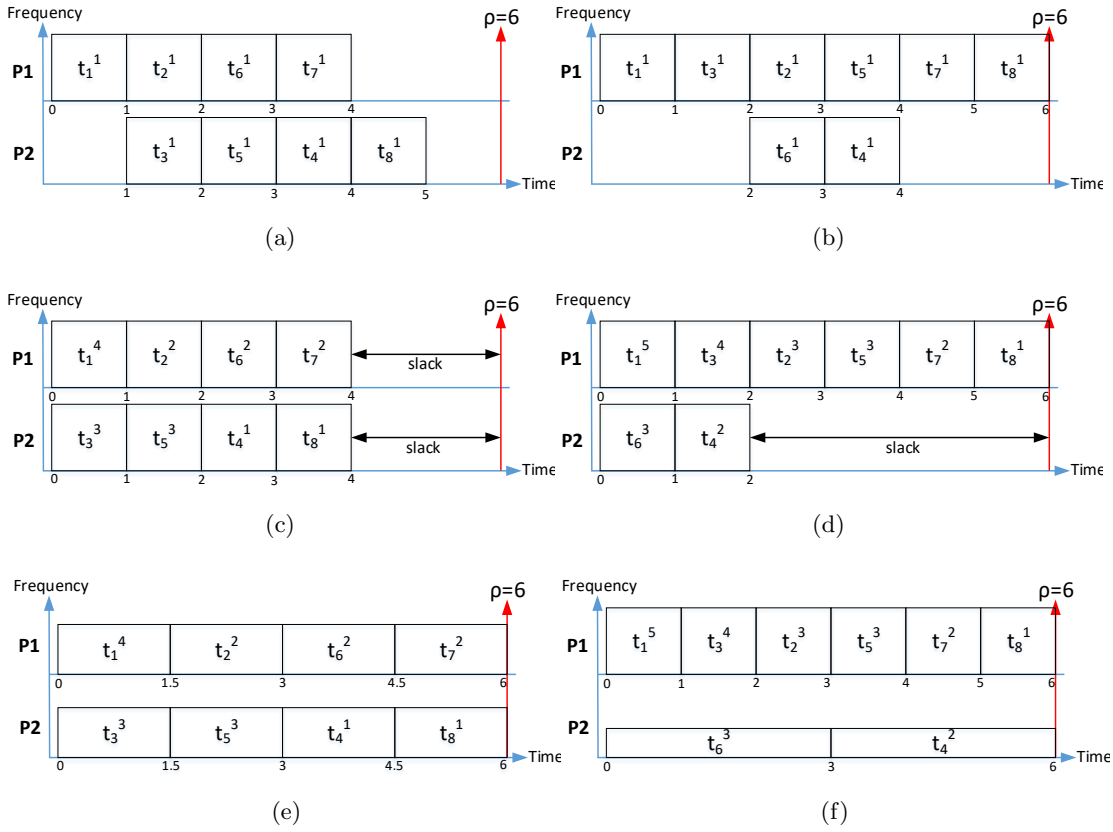


Figure 6.2: (a) An initial schedule with balanced workloads across all processors, (b) an initial schedule with unbalanced workloads across all processors. (c) A retimed schedule of Figure 6.2a, (d) a retimed schedule of Figure 6.2b. (e) Slack reclamation of schedule in Figure 6.2c, (f) Slack reclamation of schedule in Figure 6.2d

would create unnecessary extra memory usage. Assume that we attempt to achieve the retimed schedule i.e. the first steady period as in Figure 6.2c. Consider a first case 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 case which is our approach. We compute the retiming values of all the tasks by taking into account the task mapping, resulting in $R_1 = 3, R_2 = 1, R_3 = 2, R_4 = 0, R_5 = 2, R_6 = 1, R_7 = 1, R_8 = 0$. The maximum retiming value or the number of prologue required by

the first case is 4 and the second case is only 3. The maximum retiming value indicates that the first case requires more memory space to store data compared to the second case, albeit that both cases achieved the same objective schedule as in Figure 6.2c.

6.4 Scheduling Approach

Given a streaming application and a NoC-based MPSoC to execute the application, our primary objective is to minimize the total energy consumption of the system under memory capacity constraints. Our approach constructs an initial schedule under maximum frequencies such that the workload across all the processors is balanced, and 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 total energy and memory usage of the resultant schedule. If there is no memory capacity violation, our approach terminates. Otherwise, our approach iteratively fix the retiming values based on the initial schedule, rerun the NLP-based algorithm and the ILP-based algorithm to assign optimal discrete frequencies to all tasks and messages until the memory capacity 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.

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 is no memory spills i.e. memory violations, our heuristic terminates. Otherwise, repeat the following until memory capacity constraint is satisfied.

   (a) Fix the retiming values 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 capacity violations.

6.4.1 Computing Priorities

The priority of each task $t_i \in T$ and each edge $(t_i, t_j) \in E$ is its approximate successor-tree-consistent deadline d'_i and $d'_{i,j}$, respectively. A smaller deadline implies a higher priority. The approximate successor-tree-consistent deadlines (or modified deadlines) of all the tasks are computed in reverse topological order while the approximate successor-tree-consistent deadline 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 non-increasing order of their edge weights. For the tasks with the same edge weight, further sort them in 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.

6.4.2 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 follows.

- 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.

$$\mu_k = \sum_{t_i \in T_k}^{|T_k|} w_i/\rho \tag{6.7}$$

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' represent a set of scheduled tasks, a set of scheduled messages, respectively, and E' is a set of edges. 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' to resolve 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'.

6.4.3 Retiming

Given a set of scheduled tasks and messages under maximum frequencies, we attempt to reschedule certain instances of tasks and messages so that the amount of usable slack could be increased and utilized during frequency scaling. This could be done by transforming certain intra-period dependencies into inter-period dependencies. In other words, we re-

group 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 non-negative 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 the semantic correctness. If the retiming value of a parent node t_i is lesser than the retiming value of its child node t_j, this 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 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 $S = \{u_k : u_k$ 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 immediate 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 $R_i = max(R_i, R_j + 1)$.

ii. If u_i is assigned to the same processor (link) as u_j, update $R_i = R_j$.

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.

$$R_i = \begin{cases} max\{R_i, R_j + 1\}, & \text{if } u_j \text{ is a child node of } u_i \text{ and both are on different resources.} \\ R_j, & \text{if } u_j \text{ is a child node of } u_i \text{ and both are on the same resource.} \\ 0, & \text{if } u_i \text{ is a sink node.} \end{cases} \tag{6.8}$$

Based on the retiming value of each task and each message, we form a retimed graph $G'_r = (T \cup M', E'')$ to represent the new intra-period dependencies of G' after retiming. Firstly, we make a copy G'_r of G'. Then, we remove certain edges in G'_r. An edge (u_i, u_j) $u_i, u_j \in T \cup M'$ 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 i.e. 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' with these constraints.

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.

6.4.4 Discrete Frequency Selection

We have a set of scheduled tasks and messages, each of which has been assigned a retiming value and the corresponding retimed graph G'_r. Our objective is to assign an optimal discrete frequency to each task and each message under 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 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.

Computing Optimal Continuous Frequencies

We have $G'_r = (T \cup M', E'')$ where T, M' and E'' 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:

$$e_i = c_i / f_i \quad \forall t_i \in T \tag{6.9}$$

   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 $M_{i,j}$:

$$e_{i,j} = c_{i,j} / (\lambda f_{i,j}) \quad \forall M_{i,j} \in M' \tag{6.10}$$

   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:

$$s_i + e_i(e_{i,j}) \leq s_j \quad \forall (u_i, u_j) \in E'' \tag{6.11}$$

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:

$$v_{min} \leq v_{s_i} \leq v_{max} \quad \forall t_i \in T \tag{6.12}$$

$$f_{min} \leq f_i \leq f_{max} \quad \forall t_i \in T \tag{6.13}$$

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:

$$v^l_{min} \leq v_{s_{i,j}} \leq v^l_{max} \quad \forall M_{i,j} \in M' \tag{6.14}$$

$$f^l_{min} \leq f_{i,j} \leq f^l_{max} \quad \forall M_{i,j} \in M' \tag{6.15}$$

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:

$$s_i + e_i \leq d'_i \quad \forall t_i \in T \tag{6.16}$$

where d'_i is the modified deadline of t_i.

Furthermore, for each supply voltage and frequency pair for all the tasks and messages, constraint (6.6) in Section 6.2.2 is used.

The objective function is shown as follows:

$$\min\{ \underbrace{\sum_{t_i \in T} \varepsilon_i(v_{s_i}, f_i)}_{\text{computation energy}} + \underbrace{\sum_{M_{i,j} \in M'} \varepsilon_{i,j}(v_{s_{i,j}})}_{\text{communication energy}} \} \tag{6.17}$$

Since the objective function is convex, this convex NLP problem can be solved in polynomial time [87].

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_i^{opt} and $f_{i,j}^{opt}$ be the optimal frequencies of t_i and $M_{i,j}$, respectively, computed by the NLP-based algorithm. 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_i^l be the largest frequency of the processor where $t_i \in T$ is assigned such that f_i^l is less than f_i^{opt}, and f_i^{l+1} be the frequency at a higher level of that processor. Similarly, let $f_{i,j}^l$ be the largest frequency of a link that is less than f_i^{opt}, and $f_{i,j}^{l+1}$ be the frequency at a higher level of a link. Clearly, in an optimal schedule, the frequency for t_i must be either f_i^l or f_i^{l+1}, and the frequency for $M_{i,j}$ must be either $f_{i,j}^l$ or $f_{i,j}^{l+1}$.

Therefore, we introduce a binary decision variable x_i to choose between f_i^l or f_i^{l+1} for each

task $t_i \in T$ and $f_{i,j}^l$ or $f_{i,j}^{l+1}$ for each message $M_{i,j} \in M'$ as follows.

$$x_i = \begin{cases} 0, & \text{if } (v_{s_i}^l,\ f_i^l)\ ((v_{s_{i,j}}^l,\ f_{i,j}^l)) \text{ is used} \\ 1, & \text{if } (v_{s_i}^{l+1},\ f_i^{l+1})\ ((v_{s_{i,j}}^{l+1},\ f_{i,j}^{l+1})) \text{ is used} \end{cases} \tag{6.18}$$

where $v_{s_i}^l$ and $v_{s_i}^{l+1}$ are the corresponding supply voltages of f_i^l and f_i^{l+1}, respectively while $v_{s_{i,j}}^l$ and $v_{s_{i,j}}^{l+1}$ are the corresponding supply voltages of $f_{i,j}^l$ and $f_{i,j}^{l+1}$, respectively.

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

$$\varepsilon^{comp} = \sum_{t_i \in T} (1 - x_i)\varepsilon_i(v_{s_i}^l, f_i^l) + x_i\varepsilon_i(v_{s_i}^{l+1}, f_i^{l+1}) \tag{6.19}$$

where $\varepsilon_i(v_{s_i}^l, f_i^l)$ and $\varepsilon_i(v_{s_i}^{l+1}, f_i^{l+1})$ are the total computation energy consumptions of t_i at the frequency f_i^l and at the frequency f_i^{l+1}, respectively. Notice that both $\varepsilon_i(v_{s_i}^l, f_i^l)$ and $\varepsilon_i(v_{s_i}^{l+1}, f_i^{l+1})$ are constants.

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

$$\varepsilon^{comm} = \sum_{M_{i,j} \in M'} (1 - x_i)\varepsilon_{i,j}(v_{s_{i,j}}^l) + x_i\varepsilon_{i,j}(v_{s_{i,j}}^{l+1}) \tag{6.20}$$

where $v_{s_{i,j}}^l$ and $v_{s_{i,j}}^{l+1}$ are the corresponding supply voltages of $f_{i,j}^l$ and $f_{i,j}^{l+1}$, respectively.

The objective function is shown as follows.

$$\min\{\varepsilon^{comp} + \varepsilon^{comm}\} \tag{6.21}$$

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

$$e_i = (1 - x_i)\epsilon_i^l + x_i\epsilon_i^{l+1} \quad \forall t_i \in T \tag{6.22}$$

$$e_{i,j} = (1 - x_{i,j})\epsilon_{i,j}^l + x_{i,j}\epsilon_{i,j}^{l+1} \quad \forall M_{i,j} \in M' \tag{6.23}$$

where ϵ_i^l and ϵ_i^{l+1} are the execution times of t_i when using f_i^l and f_i^{l+1}, respectively. Similarly, $\epsilon_{i,j}^l$ and $\epsilon_{i,j}^{l+1}$ are the communication times of $M_{i,j}$ when using $f_{i,j}^l$ and $f_{i,j}^{l+1}$, respectively. Notice that ϵ_i^l, ϵ_i^{l+1}, $\epsilon_{i,j}^l$ and $\epsilon_{i,j}^{l+1}$ are constants.

The precedence constraint (6.11) for each edge and the deadline constraint (6.16) 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 follows.

Computing Memory Usage

Given a set of scheduled tasks and messages, each of which is assigned with an optimal discrete frequency, 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 [18]. Next, we describe each step. The first step works as follows:

- For each edge $(u_i, u_j) \in E'$ (u_i, $u_j \in T \cup M'$) 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 $\theta_l (l = 0, \cdots, R_s)$, i.e. within the prologue and the first steady period, do the following.
        (a) Add a lifetime segments 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 $li_k.s$ and $li_k.\Omega$ by appending the prologue with $R^{max}\rho$ time unit as in Equation (6.24) and Equation (6.25), respectively.

$$li_k.s = \begin{cases} (R^{max} - l)\rho + s_i, & \text{if } u_i \text{ is a message node} \\ (R^{max} - l)\rho + s_i + e_i, & \text{if } u_i \text{ is a task node} \end{cases} \tag{6.24}$$

$$li_k.\Omega = (R^{max} - l + \delta_i)\rho + s_j + e_j \qquad \forall u_j \tag{6.25}$$

(b) If $li_k.\Omega$ is greater than $(R^{max}+1)\rho$, set $li_i.\Omega$ 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 [18]. 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 $\Gamma_{j+1}(j = 0, 1, \cdots, |\Gamma| - 2)$, do the following.
     (a) Find a set of lifetime segments L_k^{j+1}, 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 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 which intersect within interval $[\Gamma_j, \Gamma_{j+1}]$ can be considered as a clique. A clique can be represented as an undirected graph where each lifetime segment is a node and all the nodes are connected between each other to form a complete graph.

6.4.5 Repair Approach

We employ an algorithm to fix the schedule in case of memory capacity violations. Our 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 parent or data producer task t_i: the total energy difference and memory requirement 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 that reduces the memory usage the most, but with a minimum increase in total energy consumption.

We compute the total energy difference metric as follows.

$$\Delta E_i = \varepsilon_{\sigma'} - \varepsilon_{\sigma} \tag{6.26}$$

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.

$$\Delta MEM_i = MEM_{\sigma'}^{max} - MEM_{\sigma}^{max} \tag{6.27}$$

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 Equation (6.26) and Equation (6.27) 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}.
    4. Construct a retimed graph G_r' based on R_{new} as in Section 6.4.3.
    5. Assign optimal discrete frequencies to all tasks and messages based on G_r' as in Section 6.4.4.

6. Compute the total energy and maximum memory usage of the new schedule. Then, compute the total energy difference ΔE_k, memory requirement difference ΔMEM_k and latency l_k of the new schedule as compared to the current schedule.
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.

Updating retiming values

As aforementioned, the retiming value of each task or each message has the following constraints.

- The retiming value is non-negative.
- The retiming value of a parent node must be greater than or equal to its children nodes.

In order to ensure the validity of the retiming value of each $t_i(M_{i,j})$ each time we update a task (message) retiming value, we design a function to recursively update the retiming values of all its successors $t_j(M_{i,j})$ of t_i accordingly.

1. Let $t_i(M_{i,j}) \in G'$ be the task (message) such that its current retiming value R_{cur_i} needs to be reduced by 1.
2. Make a copy of the current retiming values of all tasks and messages, R_{new} of R_{cur}.

3. If R_{cur_i} is equal to 0, terminates. Otherwise, update R_{new_i} as $R_{cur_i} - 1$.
4. Find a set S, of $t_i(M_{i,j})$ immediate successors such that their retiming values are equal to R_{cur_i}.
5. For each $t_j(M_{s,t}) \in S$, do the following.
   (a) Repeat steps (1) until (5).

6.5 Experimental Results

In order to evaluate our approaches, we compare them with two state-of-the-art approaches, RDAG+GeneS [5] and JCCTS [6] by using a set of real and synthetic benchmarks. We choose RDAG+GeneS [5] and JCCTS [6] because they are the latest approaches considering similar problem structures as ours. Firstly, these approaches consider similar system models as ours. They consider a periodic, dependent task model and homogeneous MPSoC with distributed memory. Secondly, they employ task-level software pipelining and DVFS to reduce the energy consumption of the system.

6.5.1 Experimental Setup

We use a set of 10 synthetic benchmarks obtained in [89] and two real-world benchmarks: automotive from the E3S benchmarks suite [90] and ATR [91]. ATR is widely used for mobile military systems and usually requires real time processing [86]. It is used to recognize objects based on data obtained from sensors. The ATR consists of 17 tasks to process one frame. The E3S benchmarks suite which provides the automotive benchmark was designed for use in automated system-level assignment, and scheduling research. These two benchmarks are modelled from real applications. Also, these applications are periodic in nature. These sets of benchmarks are widely used by the embedded systems community for task scheduling research.

Table 6.2 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 the total communication divided by the 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-by-2 mesh NoC, 3-by-3 mesh NoC and 4-by-4 mesh NoC as the target MPSoCs. We define each processor is based on the 0.07 μm technology in Table 5.3 with its parameters provided in [25], that follow the power model in Section 6.2.2. The accuracy of these technology parameters has been verified through SPICE simulations [25]. Also, they were widely used in the previous energy-aware task scheduling research [33, 80–83].

Each processor has five voltage and frequency levels. The minimum supply voltage v_{min} and maximum supply voltage v_{max} follows 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 Table 5.6 [3] [87].

Table 6.2: The characteristics of benchmarks.

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

We define our proposed approach as Ours. We implement Ours, RDAG+GeneS [5] and JCCTS [6] 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 caches.

6.5.2 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.

Firstly, we discuss on the impact of processing parallelism of each benchmark on the total energy consumption. Figure 6.3 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 indicate that in most scenarios, the total energy consumption decrease as the number of processors increases. This is due to our approach constructs initial schedules such that the total utilization of all the processors is minimized. Thus, a larger number of processors results in processing parallelism to increase and creates more static slack, resulting in lower frequencies for the tasks and messages.

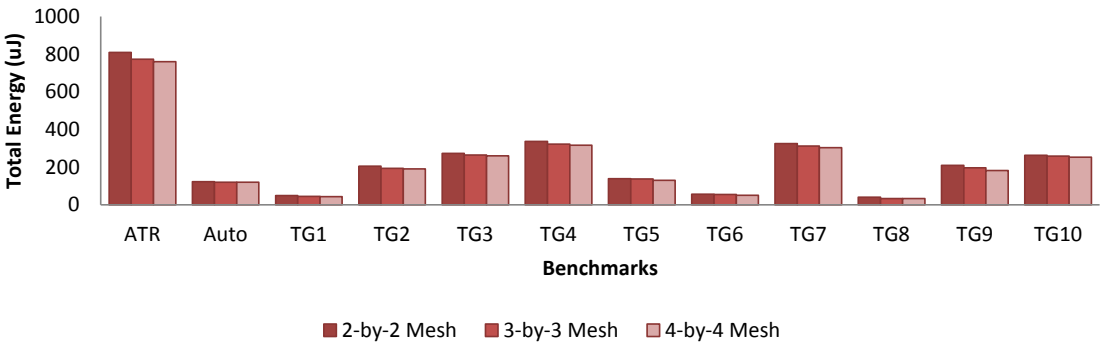


Figure 6.3: Total energy consumption of Ours for all benchmarks on three different NoC architectures.

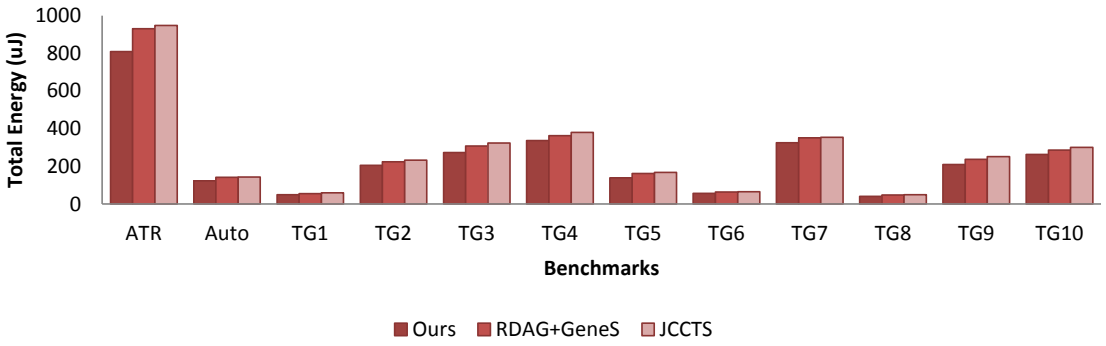
Secondly, we compare the total energy consumption of each benchmark using the Ours, RDAG+GeneS [5] and JCCTS [6]. Since Ours consider memory capacity constraints, we set the memory constraint of each scenario to be equal to the maximum memory usage of each scenario computed using RDAG+Genes. Figure 6.4 shows the total energy consumption (computation plus communication energy) of all the benchmarks using Ours, RDAG+GeneS [5] and JCCTS [6]. 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 indicate that Ours has the advantage of reducing the total energy consumption of all the benchmarks, when compared to the other approaches. Specifically, the largest improvement of 40.82% occurred at synthetic benchmark B48 on a 2-by-2 mesh NoC, the least improvement of 7.53% at synthetic benchmark B61 on a 4-by-4 mesh NoC, while the average improvement of all scenarios compared to RDAG+GeneS [5] is 17.31%.

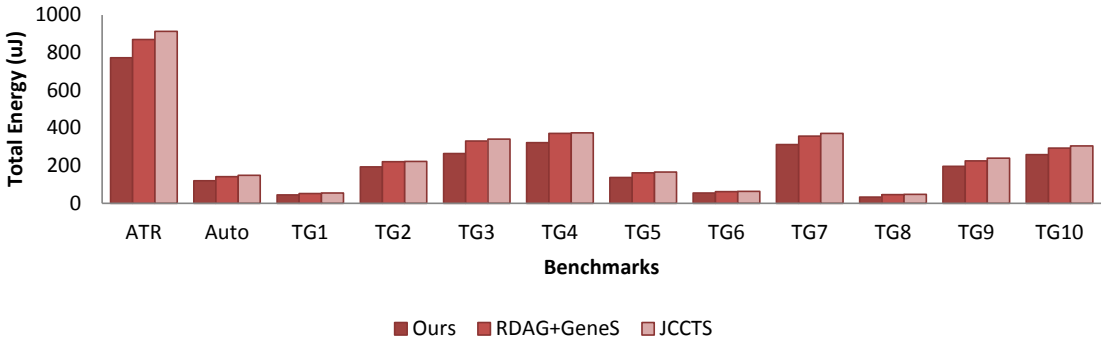
The primary reason of Ours has advantages compared to RDAG+GeneS, is that RDAG+Genes uses the Genetic Algorithm (GA) to schedule and assign discrete frequencies to all tasks. GA is based on natural selections and does not necessarily find the optimal solutions. On the contrary, Ours uses NLP-based algorithm and 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 compared to JCCTS [6] are 46.46%, 21.67% and 10.75%, respectively. The maximum improvement happens on synthetic benchmark B48 on a 2-by-2 mesh NoC and the minimum improvement occurred at synthetic benchmark B7 on a 4-by-4 mesh NoC.

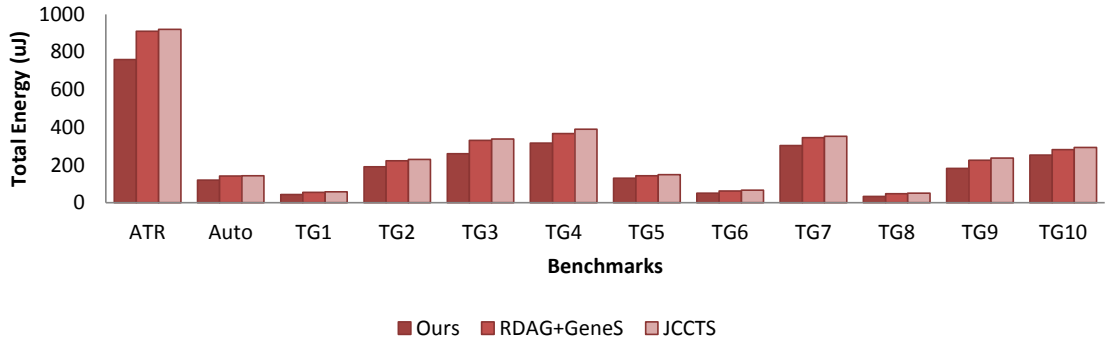
The primary reason for this is that, JCCTS attempts to minimize the prologue length while totally removing the inter-processor communication overhead. We observe that their approach might works to remove the communication overhead, but not in terms of total



(a)



(b)



(c)

Figure 6.4: Total energy consumption of Ours, RDAG+GeneS [5] and JCCTS [6] for all benchmarks on (a) 2-by-2 mesh NoC, (b) 3-by-3 mesh NoC, and (c) 4-by-4 mesh NoC.

energy. This is because in order to minimize the maximum retiming value or the prologue length, significant intra-period dependencies still exist. Consequently, these dependencies blocked the utilization of slack for frequency scaling and thus limit the opportunity for

energy reductions. On the contrary, in Ours, our retiming technique considers the task mapping and tries to increase and fully exploit the usable slack during frequency scaling.

Thirdly, 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.

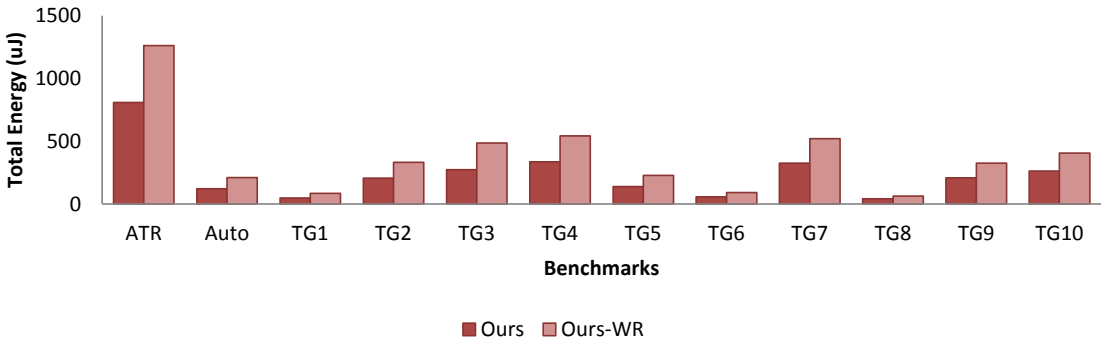
1. Construct an initial schedule as in Section 6.4.2.
2. Assign an optimal discrete frequency for each task and message as in Section 6.4.4.

Figures 6.5,6.6 and 6.7 show the total energy consumption and the corresponding maximum memory usages of all benchmarks using Ours and Ours-WR. For the Figures 6.5a,6.6a and 6.7a 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 Figures 6.5b,6.6b and 6.7b showing maximum memory usages, each vertical axis denotes the memory size requirements of each benchmark using a specific approach, and each horizontal axis denotes the benchmarks.

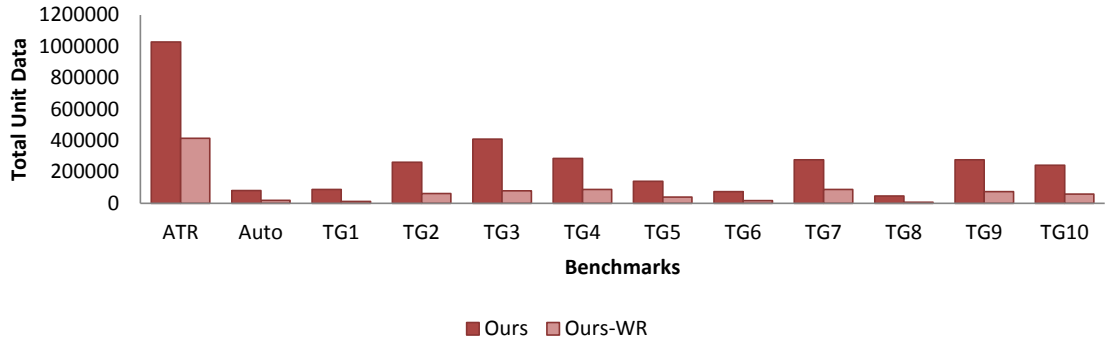
Overall, the simulation results indicate that retiming could effectively reduce the total energy consumption. Ours is advantageous in reducing the total energy consumption in all scenarios. However, Ours-WR constructs schedules that require the least memory usage. This is due to Ours-WR not implementing pipelining, requiring no extra memory overhead. We observe that the total energy consumption could be improved by up to 77% when using our retiming technique.

Algorithm Running Time

We show the running times to construct the schedules produced by Ours, RDAG+GeneS [5] and JCCTS [6]. Table 6.3 summarizes the average running times of Ours, RDAG+GeneS



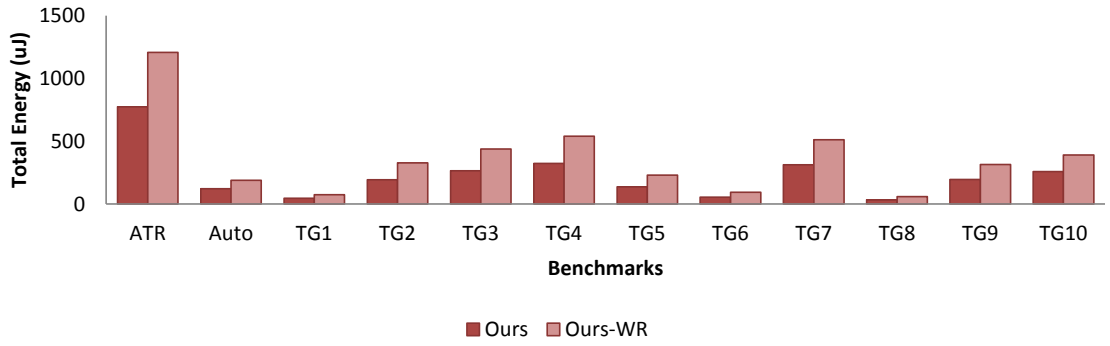
(a)



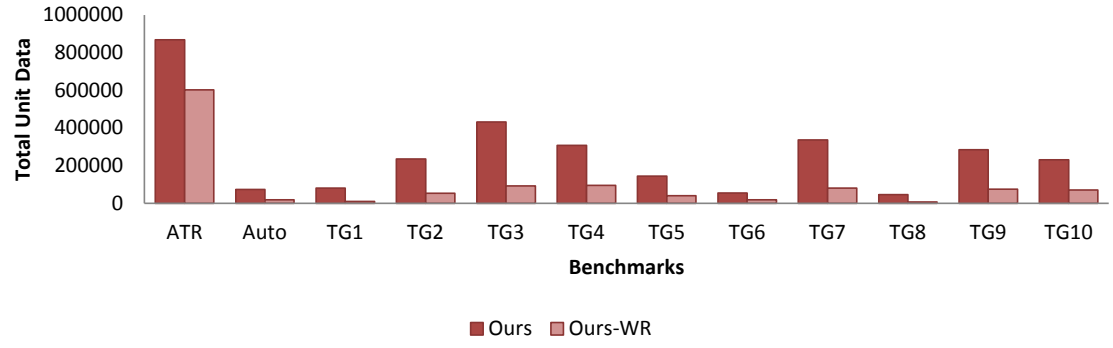
(b)

Figure 6.5: (a) Total energy consumption of Ours and Ours-WR, (b) memory usage of Ours and Ours-WR, for all benchmarks on 2-by-2 mesh NoC.

[5] and JCCTS [6] approaches on each NoC architecture. For all benchmarks, the average running time ratio of the RDAG+GeneS [5] approach compared to Ours is 97.97% while the average running time ratio of JCCTS [6] compared to Ours is 91.74%. As we can see, our approaches are much slower compared to the RDAG+GeneS [5] and JCCTS [6] approaches. We found repetitive NLP-based algorithm and ILP-based algorithm dominates the total running time of Ours. Although the run time is slower, our approach computes much better energy-efficient schedules than the RDAG+GeneS [5] and JCCTS [6] approaches do. Notice that our approaches target embedded systems. Therefore, the running time for constructing an offline schedule at the design stage should not be an issue.



(a)



(b)

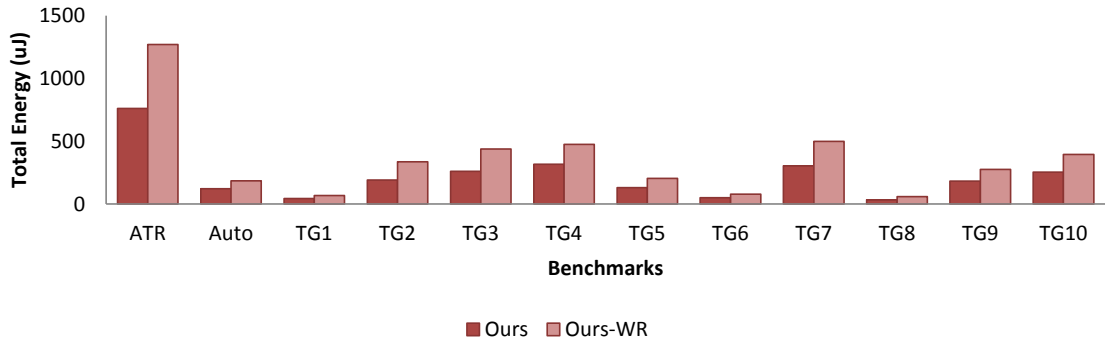
Figure 6.6: (a) Total energy consumption of Ours and Ours-WR, (b) memory usage of Ours and Ours-WR, for all benchmarks on 3-by-3 mesh NoC.

Table 6.3: Average running times of Ours, RDAG+GeneS [5] and JCCTS [6] on each architecture.

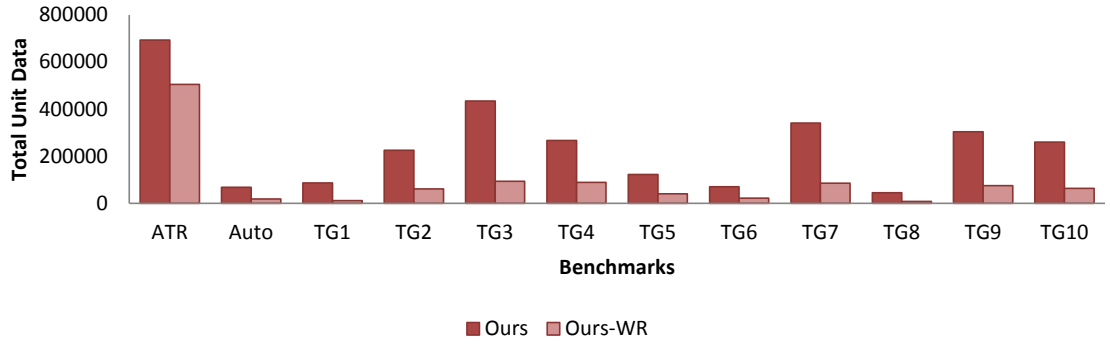
Architecture	Ours (s)	RDAG+GeneS (s)	JCCTS (s)
2-by-2	264.71	266.59	248.70
3-by-3	280.21	276.45	253.07
4-by-4	297.33	282.14	255.28

6.6 Summary

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



(a)



(b)

Figure 6.7: (a) Total energy consumption of Ours and Ours-WR, (b) memory usage of Ours and Ours-WR, for all benchmarks on 4-by-4 mesh NoC.

MPSoC with discrete frequencies such that the total energy consumption of all the tasks is minimized under memory capacity constraints. Our approach consists of 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, and the workload across all the processors is balanced, a retiming heuristic considering the task mapping 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 fix the schedule in the case of memory capacity violations. We have evaluated our approach and compared it with two state-of-the-art approaches, RDAG+GeneS [5] and

JCCTS [6]. Experimental results show that our approaches perform significantly better than the two approaches in terms of the total energy consumption while satisfying the memory capacity constraints.

Chapter 7

Conclusion and Future Work

This final chapter summarizes the contributions made in this thesis and discusses several open problems for future work.

7.1 Conclusion

Energy consumption is one of the critical design considerations for embedded systems. An energy-aware system increases the lifetime of the system, particularly for systems that rely on batteries. Moreover, it reduces the operating costs, reduces heat dissipation, thereby increasing system reliability. Dynamic Voltage and Frequency Scaling (DVFS) is a powerful technique used for reducing energy consumption.

Modern embedded processors, as well as the communication subsystems, are equipped with DVFS. It is a mechanism which aims to reduce the energy consumption by enabling the adjustment of the operating frequency and supply voltage according to the demands. Using the power model, the supply voltage is the main factor of power consumption and to work safely, its value is determined by the frequency at which the circuit is clocked. The supply voltage can be lowered if the frequency is lowered, and vice versa. Consequently,

DVFS affects the execution time of tasks. A task with a lower frequency and supply voltage extends its execution time. This is explained by the execution time of a task being inversely proportional to its operating frequency.

Real-time embedded applications may impose certain constraints, such as deadline and precedence constraints. There are consequences when deadlines are not met, the severity of which depends on the embedded application. Precedence constraints enforce the ordering of the application tasks. Unlike independent application tasks, a feasible task schedule needs to meet the deadlines of all tasks, and also needs to adhere to the precedence constraints. These constraints complicate both task scheduling and DVFS, because they significantly impact the producibility and usability of the slack for frequency scaling. Therefore, the problem becomes more challenging. Hence, DVFS should be properly conformed using a scheduler of the embedded operating system for efficient deployment.

In this thesis, we investigate three different problems. We study the energy-aware task scheduling for application tasks with precedence and deadline constraints on a homogeneous Multiprocessor System-on-Chip (MPSoC) assuming continuous frequencies, the energy-aware task scheduling for application tasks with precedence and deadline constraints on a heterogeneous NoC-based MPSoC considering discrete frequencies and the energy-aware task scheduling for periodic dependent application tasks on a homogeneous NoC-based MPSoC considering discrete frequencies.

In Chapter 4, we discuss the problem of scheduling a set of non-preemptible application tasks with precedence and individual deadline constraints on a homogeneous MPSoC assuming continuous frequencies such that the total processor energy consumption of all the tasks is minimized under two power models, specifically the total dynamic power and the total power. We present a unified two-phase scheduling approach to the problem. Firstly, it uses a list-based scheduling consisting of a novel priority scheme to construct an initial schedule under maximum frequencies. The priority of each task is its approximate successor-tree-consistent deadline. Secondly, it formulates the problem of selecting an op-

timal frequency and supply voltage for each task into a convex Non-Linear Programming (NLP) problem. Furthermore, Chapter 4 presents experimental results showing the effectiveness of our proposed approach in minimizing not only the total dynamic energy but also the total energy consumption.

In Chapter 5, we describe the problem of scheduling a set of non-preemptible tasks with precedence constraints and individual deadline constraints on a heterogeneous NoC-based MPSoC with discrete frequencies such that the total energy consumption of all the tasks is minimized. In addition, we explicitly consider communication contention as well as communication energy. We present two novel approaches to the problem. Both approaches consist of an iterative NLP-based algorithm for task mapping and scheduling. Then, one approach uses an ILP-based algorithm for assigning optimal discrete frequencies to each task and each message, while the other approach uses a polynomial-time heuristic for assigning discrete frequencies to each task and each message. Moreover, Chapter 5 presents experimental results showing our proposed approaches perform significantly better compared to state-of-the-art approaches, in terms of total energy consumption. Besides, the performance of our proposed approach using the polynomial-time heuristic is very close to that of our proposed approach using ILP-based algorithm in computing discrete frequencies for all the tasks and messages, with better running time.

In Chapter 6, we present 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 capacity constraints. We propose a novel approach that integrates task-level software pipelining with DVFS to address the problem. Initially, our approach constructs an initial schedule using maximum frequencies, such that the workload across all processors is balanced. Next, we employ a novel retiming technique to transform intra-period dependencies into inter-period dependencies considering the task mapping to enhance parallelism. Then, our approach uses an NLP-based algorithm and an

ILP-based algorithm to assign optimal discrete frequencies to all the tasks and messages. An iterative algorithm is employed to resolve memory capacity violations. Also, Chapter 6 shows extensive experimental results showing the effectiveness of our proposed approach, in terms of minimizing the total energy consumption while satisfying all the constraints.

7.2 Future Work

Although a lot of advances have been made in the area of energy-aware task scheduling, several open problems need to be solved in our future research.

The first open problem is minimizing the total energy consumption for multi-rate periodic dependent tasks on a heterogeneous NoC-based MPSoC. Our attempt would be on extending our approach using the task-level software pipelining with DVFS to address this problem. There are three significant challenges. Firstly, the processors across a heterogeneous platform vary in performance and energy profile. The heterogeneity complicates the task assignment and the use of pipelining to increase parallelism. To date, energy-aware approaches employing task-level software pipelining and DVFS work on homogeneous platforms. Furthermore, an embedded system may contain multiple applications, each of which having different periods. This kind of system is called multi-rate. The starting point in solving this problem shall be on constructing an initial schedule using maximum frequencies to ensure the schedulability of each task. It is sufficient to consider all tasks within one hyper-period, that is the least common multiple of all the application periods. A task with a shorter period should be executed more frequently than a task with longer period within the hyper-period. Consequently, each task has a different slowdown potential. Therefore, it would be interesting to find a way to efficiently assign and order all tasks to maximize the opportunities to be exploited for lowering operating frequencies during slack reclamation. Furthermore, inter-processor communication overhead, as well as network congestion, may complicate the problem further. To employ task-level

software pipelining in a heterogeneous platform, an efficient task-to-processor mapping is required. The mapping affects not only the execution performance of each task but also the inter-processor communication overhead as well as the memory capacity overhead, and therefore the total energy consumption. In this case, a clustering-based approach with the combination of task-level software pipelining may be helpful and should be explored. Furthermore, these challenges need to be solved in a unified way. We will work out an efficient approach combining task-level software pipelining with DVFS for multi-rate periodic dependent tasks on a heterogeneous NoC-based MPSoC.

The second open problem is online task scheduling on NoC-based MPSoCs. We will extend our study on the problem of minimizing the total energy consumption of a set of dependent tasks with deadline constraints for homogeneous as well as heterogeneous NoC-based MPSoCs to include energy-aware online scheduling. Without prior knowledge on the actual execution time (AET) during run-time, this problem becomes more difficult. A task may run shorter than its worst-case execution time (WCET), and dynamic slack exists that could be utilized for frequency scaling. The dynamic slack is computed as the difference between the WCET of a task and its AET. This slack can be claimed by subsequent tasks for further minimizing the total energy consumption. Another challenge is to design an algorithm that should have a low degree of complexity, as the scheduling decision needs to be made on-the-fly, based on current requirements.

The third open problem is energy-aware task scheduling on a NoC-based MPSoC with heterogeneous Voltage Frequency Islands (VFI). A number of commercial state-of-the-art multi-core processors, such as IBM Power 7 series and Intel Itanium i7 use VFI. This feature is an intermediate between global DVFS, providing one frequency for all the cores, and per-core DVFS, which provides individual frequency for each core. It is based on the Globally Asynchronous Locally Synchronous (GALS) design in which a chip is organized into a number of clusters or islands, each of which operates at its own supply voltage and clock frequency. All processor cores in an island share a common frequency. In

general, the cores in an island can be homogeneous or heterogeneous, and islands can have different types and number of cores i.e. heterogeneous. This feature provides more flexibility in managing the power and performance in multi-core processors. It is possible to achieve significant energy savings within a certain timing constraint. The key challenge to this problem is on assigning each task to the appropriate island and core, as well as selecting its frequency such that the total energy consumption consisting of computation and communication is minimized.

The last open problem is integrating DVFS with other techniques such as Dynamic Power Management (DPM). DPM is a technique used to minimize the static power by turning off the processor when it is inactive. However, it is only beneficial if the idle interval is longer than the break-even time since there are energy and time overheads when a processor switches between off-mode and on-mode. In general, DVFS and DPM counteract each other with respect to energy reduction and a trade-off between them plays a critical role in energy consumption reduction [92]. With DVFS, reducing the frequency of a task prolongs its execution time which shortens the idle intervals, thus limiting the potential to reduce static power. Alternatively, running a task at a higher frequency may produce longer idle intervals, providing a better chance to reduce the static power, but will increase the dynamic power and switching overhead. Therefore, the key question is on how to optimally integrate DVFS and DPM such that the total energy consumption is minimized.

Bibliography

[1] K. Li, "Power and performance management for parallel computations in clouds and data centers," *Journal of Computer and System Sciences*, vol. 82, no. 2, pp. 174–190, 2016.

[2] S. Su, Q. Huang, J. Li, X. Cheng, P. Xu, and K. Shuang, "Enhanced energy-efficient scheduling for parallel tasks using partial optimal slacking," *The Computer Journal*, vol. 58, no. 2, pp. 246–257, 2014.

[3] D. Li and J. Wu, "Energy-efficient contention-aware application mapping and scheduling on noc-based mpsocs," *Journal of Parallel and Distributed Computing*, vol. 96, pp. 1–11, 2016.

[4] J.-J. Han, M. Lin, D. Zhu, and L. T. Yang, "Contention-aware energy management scheme for noc-based multicore real-time systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 3, pp. 691–701, 2015.

[5] Y. Wang, H. Liu, D. Liu, Z. Qin, Z. Shao, and E. H.-M. Sha, "Overhead-aware energy optimization for real-time streaming applications on multiprocessor system-on-chip," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 16, no. 2, p. 14, 2011.

[6] Y. Wang, D. Liu, Z. Qin, and Z. Shao, "Optimally removing intercore communication overhead for streaming applications on mpsocs," *IEEE Transactions on Computers*, vol. 62, no. 2, pp. 336–350, 2013.

[7] M. Dirjish `https://www.sensorsmag.com/embedded/`.

[8] V. Danielito `https://dcvizcayno.wordpress.com/2015/08/28/`.

[9] W. Shen `http://mil-embedded.com/articles/`.

[10] Wikipedia `https://en.wikipedia.org/wiki/Mars_Exploration_Rover`.

[11] A. Pathania, A. E. Irimiea, A. Prakash, and T. Mitra, "Power-performance modelling of mobile gaming workloads on heterogeneous mpsocs," in *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*, pp. 1–6, IEEE, 2015.

[12] M. Donno, A. Ivaldi, L. Benini, and E. Macii, "Clock-tree power optimization based on rtl clock-gating," in *Proceedings of the 40th annual Design Automation Conference*, pp. 622–627, ACM, 2003.

[13] B. Hoefflinger, "Itrs: The international technology roadmap for semiconductors," in *Chips 2020*, pp. 161–174, Springer, 2011.

[14] X. Feng, R. Ge, and K. W. Cameron, "Power and energy profiling of scientific applications on distributed systems," in *Parallel and Distributed Processing Symposium*, pp. 34–34, 2005.

[15] L. Huang, F. Yuan, and Q. Xu, "Lifetime reliability-aware task allocation and scheduling for mpsoc platforms," in *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE'09.*, pp. 51–56, IEEE, 2009.

[16] G. Manimaran and C. S. R. Murthy, "A fault-tolerant dynamic scheduling algorithm for multiprocessor real-time systems and its analysis," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 11, pp. 1137–1152, 1998.

[17] J. D. Ullman, "Np-complete scheduling problems," *Journal of Computer and System Sciences*, vol. 10, no. 3, pp. 384–393, 1975.

[18] H. Jiang, C.-M. Li, and F. Manya, "An exact algorithm for the maximum weight clique problem in large graphs.," in *AAAI*, pp. 830–838, 2017.

[19] F. Angiolini, P. Meloni, S. M. Carta, L. Raffo, and L. Benini, "A layout-aware analysis of networks-on-chip and traditional interconnects for mpsocs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 3, pp. 421–434, 2007.

[20] U. Y. Ogras and R. Marculescu, *Modeling, analysis and optimization of network-on-chip communication architectures*, vol. 184. Springer Science & Business Media, 2013.

[21] O. Sinnen and L. A. Sousa, "Communication contention in task scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 6, pp. 503–515, 2005.

[22] É. Cota, A. de Morais Amory, and M. S. Lubaszewski, *Reliability, Availability and Serviceability of Networks-on-chip*. Springer Science & Business Media, 2011.

[23] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced cpu energy," in *Annual Symposium on Foundations of Computer Science*, pp. 374–382, 1995.

[24] R. Jejurikar and R. Gupta, "Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems," in *Proceedings of the 2004 international symposium on Low power electronics and design*, pp. 78–81, ACM, 2004.

[25] S. M. Martin, K. Flautner, T. Mudge, and D. Blaauw, "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, 2002.

[26] H. Topcuoglu, S. Hariri, and M.-y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[27] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," in *Low Power Electronics and Design, 1998. Proceedings. 1998 International Symposium on*, pp. 197–202, IEEE, 1998.

[28] D. Shin, J. Kim, and S. Lee, "Intra-task voltage scheduling for low-energy hard real-time applications," *IEEE Design & Test of Computers*, vol. 18, no. 2, pp. 20–30, 2001.

[29] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, no. 1, pp. 5–35, 1991.

[30] Y. Zhang, X. S. Hu, and D. Z. Chen, "Task scheduling and voltage selection for energy minimization," in *Proceedings of the 39th annual Design Automation Conference*, pp. 183–188, ACM, 2002.

[31] R. Mishra, N. Rastogi, D. Zhu, D. Mossé, and R. Melhem, "Energy aware scheduling for distributed real-time systems," in *Parallel and Distributed Processing Symposium*, 2003.

[32] L.-F. Leung, C.-Y. Tsui, and W.-H. Ki, "Minimizing energy consumption of multiple-processors-core systems with simultaneous task allocation, scheduling and voltage assignment," in *Design Automation Conference*, pp. 647–652, 2004.

[33] A. Andrei, M. Schmitz, P. Eles, Z. Peng, and B. M. Al-Hashimi, "Overhead-conscious voltage selection for dynamic and leakage energy reduction of time-constrained systems," *IEE Proceedings-Computers and Digital Techniques*, vol. 152, no. 1, pp. 28–38, 2005.

[34] H. Kimura, M. Sato, Y. Hotta, T. Boku, and D. Takahashi, "Emprical study on reducing energy of parallel programs using slack reclamation by dvfs in a power-scalable high performance cluster," in *IEEE International Conference on Cluster Computing*, pp. 1–10, IEEE, 2006.

[35] L. Wang, G. Von Laszewski, J. Dayal, and F. Wang, "Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with dvfs," in *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, pp. 368–377, IEEE, 2010.

[36] J. Luo and N. K. Jha, "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, 2000.

[37] R. Xu, R. Melhem, and D. Mosse, "Energy-aware scheduling for streaming applications on chip multiprocessors," in *RTSS*, pp. 25–38, 2007.

[38] R. Watanabe, M. Kondo, M. Imai, H. Nakamura, and T. Nanya, "Task scheduling under performance constraints for reducing the energy consumption of the gals multiprocessor soc," in *DATE*, pp. pp. 1–6, 2007.

[39] H. Liu, Z. Shao, M. Wang, and P. Chen, "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*, pp. 92–101, IEEE, 2008.

[40] J. Luo and N. K. Jha, "Power-efficient scheduling for heterogeneous distributed real-time embedded systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 6, pp. 1161–1170, 2007.

[41] M. Qiu, Z. Ming, J. Li, S. Liu, B. Wang, and Z. Lu, "Three-phase time-aware energy minimization with dvfs and unrolling for chip multiprocessors," *Journal of Systems Architecture*, vol. 58, no. 10, pp. 439–445, 2012.

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

[43] J. Hu and R. Marculescu, "Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints," in *Design, Automation and Test in Europe Conference and Exhibition*, vol. 1, pp. 234–239, IEEE, 2004.

[44] L. Yan, J. Luo, and N. K. Jha, "Joint dynamic voltage scaling and adaptive body biasing for heterogeneous distributed real-time embedded systems," *IEEE Transac-*

tions on Computer-Aided Design of Integrated Circuits and Systems, vol. 24, no. 7, pp. 1030–1041, 2005.

[45] V. Kianzad, S. S. Bhattacharyya, and G. Qu, "Casper: an integrated energy-driven approach for task graph scheduling on distributed embedded systems," in *16th IEEE International Conference on Application-Specific Systems, Architecture Processors*, pp. 191–197, 2005.

[46] S. Hua and G. Qu, "Power minimization techniques on distributed real-time systems by global and local slack management," in *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, pp. 830–835, ACM, 2005.

[47] M. T. Schmitz, B. M. Al-Hashimi, and P. Eles, "Iterative schedule optimization for voltage scalable distributed embedded systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 3, no. 1, pp. 182–217, 2004.

[48] M. T. Schmitz, B. M. Al-Hashimi, and P. Eles, "Energy-efficient mapping and scheduling for dvs enabled distributed embedded systems," in *Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings*, pp. 514–521, IEEE, 2002.

[49] B. Gorjiara, N. Bagherzadeh, and P. H. Chou, "Ultra-fast and efficient algorithm for energy optimization by gradient-based stochastic voltage and task scheduling," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 12, no. 4, p. 39, 2007.

[50] Y. Liu, B. Veeravalli, and S. Viswanathan, "Novel critical-path based low-energy scheduling algorithms for heterogeneous multiprocessor real-time embedded systems," in *International Conference on Parallel and Distributed Systems*, pp. 1–8, 2007.

[51] J. Luo and N. K. Jha, "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*, p. 719, IEEE Computer Society, 2002.

[52] P.-C. Chang, I.-W. Wu, J.-J. Shann, and C.-P. Chung, "Etahm: An energy-aware task allocation algorithm for heterogeneous multiprocessor," in *DAC*, pp. pp. 776–779, 2008.

[53] L. K. Goh, B. Veeravalli, and S. Viswanathan, "Design of fast and efficient energy-aware gradient-based scheduling algorithms heterogeneous embedded multiprocessor systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 1, pp. pp. 1–12, 2009.

[54] B. Gorjiara, N. Bagherzadeh, and P. Chou, "An efficient voltage scaling algorithm for complex socs with few number of voltage modes," in *Proceedings of the 2004 International Symposium on Low Power Electronics and Design*, pp. 381–386, ACM, 2004.

[55] P. Ghosh, A. Sen, and A. Hall, "Energy efficient application mapping to noc processing elements operating at multiple voltage levels," in *Proceedings of the 2009 3rd ACM/IEEE International Symposium on Networks-on-Chip*, pp. 80–85, IEEE Computer Society, 2009.

[56] J. Huang, C. Buckl, A. Raabe, and A. Knoll, "Energy-aware task allocation for network-on-chip based heterogeneous multiprocessor systems," in *19th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, pp. 447–454, IEEE, 2011.

[57] O. He, S. Dong, W. Jang, J. Bian, and D. Z. Pan, "Unism: Unified scheduling and mapping for general networks on chip," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 8, pp. 1496–1509, 2012.

[58] S. Murali and G. De Micheli, "Bandwidth-constrained mapping of cores onto noc architectures," in *Proceedings of the conference on Design, automation and test in Europe-Volume 2*, p. 20896, IEEE Computer Society, 2004.

[59] W. Jang and D. Z. Pan, "A3map: Architecture-aware analytic mapping for networks-on-chip," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 17, no. 3, p. 26, 2012.

[60] I. Pietri and R. Sakellariou, "Energy-aware workflow scheduling using frequency scaling," in *43rd International Conference on Parallel Processing Workshops (ICCPW)*, pp. 104–113, IEEE, 2014.

[61] Q. Huang, S. Su, J. Li, P. Xu, K. Shuang, and X. Huang, "Enhanced energy-efficient scheduling for parallel applications in cloud," in *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pp. 781–786, IEEE, 2012.

[62] X. Lin, Y. Wang, Q. Xie, and M. Pedram, "Task scheduling with dynamic voltage and frequency scaling for energy minimization in the mobile cloud computing environment," *IEEE Transactions on Services Computing*, vol. 8, no. 2, pp. 175–186, 2015.

[63] W. Zheng and S. Huang, "An adaptive deadline constrained energy-efficient scheduling heuristic for workflows in clouds," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 18, pp. 5590–5605, 2015.

[64] J. Singh, S. Betha, B. Mangipudi, and N. Auluck, "Contention aware energy efficient scheduling on heterogeneous multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 5, pp. 1251–1264, 2015.

[65] Z. Zong, X. Qin, X. Ruan, K. Bellam, M. Nijim, and M. Alghamdi, "Energy-efficient scheduling for parallel applications running on heterogeneous clusters," in *Parallel Processing, 2007. ICPP 2007. International Conference on*, pp. 19–19, IEEE, 2007.

[66] J. Mei and K. Li, "Energy-aware scheduling algorithm with duplication on heterogeneous computing systems," in *Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing*, pp. 122–129, IEEE Computer Society, 2012.

[67] Q. Zhao, C. Xiong, C. Yu, C. Zhang, and X. Zhao, "A new energy-aware task scheduling method for data-intensive applications in the cloud," *Journal of Network and Computer Applications*, vol. 59, pp. 14–27, 2016.

[68] Z. Tang, L. Qi, Z. Cheng, K. Li, S. U. Khan, and K. Li, "An energy-efficient task scheduling algorithm in dvfs-enabled cloud environment," *Journal of Grid Computing*, vol. 14, no. 1, pp. 55–74, 2016.

[69] G. Xie, G. Zeng, X. Xiao, R. Li, and K. Li, "Energy-efficient scheduling algorithms for real-time parallel applications on heterogeneous distributed embedded systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 12, pp. 3426–3442, 2017.

[70] M. T. Schmitz and B. M. Al-Hashimi, "Considering power variations of dvs processing elements for energy minimisation in distributed systems," in *Proceedings of the 14th international symposium on Systems synthesis*, pp. 250–255, ACM, 2001.

[71] J. Luo and N. K. Jha, "Power-profile driven variable voltage scaling for heterogeneous distributed real-time embedded systems," in *VLSI Design, 2003. Proceedings. 16th International Conference on*, pp. 369–375, IEEE, 2003.

[72] D. Shin and J. Kim, "Communication power optimization for network-on-chip architectures," *Journal of Low Power Electronics*, vol. 2, no. 2, pp. 165–176, 2006.

[73] J. Hu and R. Marculescu, "Exploiting the routing flexibility for energy/performance aware mapping of regular noc architectures," in *Design, Automation and Test in Europe Conference and Exhibition, 2003*, pp. 688–693, IEEE, 2003.

[74] Y.-K. Kwok and I. Ahmad, "Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 7, no. 5, pp. 506–521, 1996.

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

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

[77] A. K. Singh, A. Kumar, and T. Srikanthan, "Accelerating throughput-aware runtime mapping for heterogeneous mpsocs," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 18, no. 1, p. 9, 2013.

[78] D. Liu, J. Spasic, G. Chen, and T. Stefanov, "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*, pp. 1–10, IEEE, 2015.

[79] Y. C. Lee and A. Y. Zomaya, "Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling," in *9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID'09*, pp. 92–99, IEEE, 2009.

[80] R. Jejurikar, C. Pereira, and R. Gupta, "Leakage aware dynamic voltage scaling for real-time embedded systems," in *Proceedings of the 41st Annual Design Automation Conference*, pp. 275–280, ACM, 2004.

[81] E. Seo, J. Jeong, S. Park, and J. Lee, "Energy efficient scheduling of real-time tasks on multicore processors," *IEEE transactions on parallel and distributed systems*, vol. 19, no. 11, pp. 1540–1552, 2008.

[82] U. Y. Ogras, R. Marculescu, D. Marculescu, and E. G. Jung, "Design and management of voltage-frequency island partitioned networks-on-chip," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 3, pp. 330–341, 2009.

[83] G. Chen, K. Huang, and A. Knoll, "Energy optimization for real-time multiprocessor system-on-chip with optimal dvfs and dpm combination," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 3s, p. 111, 2014.

[84] T. Tobita and H. Kasahara, "A standard task graph set for fair evaluation of multiprocessor scheduling algorithms," *Journal of Scheduling*, vol. 5, no. 5, pp. 379–394, 2002.

[85] D. Zhu, R. Melhem, and B. R. Childers, "Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 7, pp. 686–700, 2003.

[86] J. A. Ratches, C. Walters, R. G. Buser, and B. Guenther, "Aided and automatic target recognition based upon sensory inputs from image forming systems," *IEEE transactions on pattern analysis and machine intelligence*, vol. 19, no. 9, pp. 1004–1019, 1997.

[87] A. Andrei, P. Eles, Z. Peng, M. T. Schmitz, and B. M. Al Hashimi, "Energy optimization of multiprocessor systems on chip by voltage selection," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 3, pp. 262–275, 2007.

[88] C. Marcon, T. Webber, and A. A. Susin, "Models of computation for noc mapping: Timing and energy saving awareness," *Microelectronics Journal*, vol. 60, pp. 129–143, 2017.

[89] M. Lombardi, M. Milano, M. Ruggiero, and L. Benini, "Stochastic allocation and scheduling for conditional task graphs in multi-processor systems-on-chip," *Journal of scheduling*, vol. 13, no. 4, pp. 315–345, 2010.

[90] R. Dick, "Embedded system synthesis benchmarks suite," *ziyang. eecs. umich. edu/dickrp/e3s*, 2002.

[91] J. Kang and S. Ranka, "Dynamic slack allocation algorithms for energy minimization on parallel machines," *Journal of Parallel and Distributed Computing*, vol. 70, no. 5, pp. 417–430, 2010.

[92] M. E. Gerards and J. Kuper, "Optimal dpm and dvfs for frame-based real-time systems," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 9, no. 4, p. 41, 2013.