

ENERGY-EFFICIENT OPTIMAL CONTROL FOR
REAL-TIME COMPUTING SYSTEMS

by
MASON THAMMAWICHAI

A THESIS SUBMITTED IN FULFILMENT OF REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY OF IMPERIAL COLLEGE LONDON

DEPARTMENT OF AERONAUTICS
IMPERIAL COLLEGE LONDON
2016

Declaration of Originality

I hereby confirm that I am the sole author of this thesis and the work described in it, and that any ideas or information derived from the published and unpublished work of others have been acknowledged through standard referencing practices of the discipline.

Mason Thammawichai
Imperial College London
22 March 2016

Copyright Declaration

The copyright of this thesis rests with the author and is made available under a Creative Commons Attribution Non-Commercial No Derivatives licence. Researchers are free to copy, distribute or transmit the thesis on the condition that they attribute it, that they do not use it for commercial purposes and that they do not alter, transform or build upon it. For any reuse or redistribution, researchers must make clear to others the licence terms of this work.

Abstract

Moving toward ubiquitous Cyber-Physical Systems - where computation, control and communication units are mutually interacting - this thesis aims to provide fundamental frameworks to address the problems arising from such a system, namely the real-time multiprocessor scheduling problem (RTMSP) and the multi-UAV topology control problem (MUTCP).

The RTMSP is concerned with how tasks can be scheduled on available computing resources such that no task misses a deadline. An optimization-based control method was used to solve the problem. Though it is quite natural to formulate the task assignment problem as a mixed-integer nonlinear program, the computation cost is high. By reformulating the scheduling problem as a problem of first determining a percentage of task execution time and then finding the task execution order, the computation complexity can be reduced. Simulation results illustrate that our methods are both feasibility optimal and energy optimal. The framework is then extended to solve a scheduling problem with uncertainty in task execution times by adopting a feedback approach.

The MUTCP is concerned with how a communication network topology can be determined such that the energy cost is minimized. An optimal control framework to construct a data aggregation network is proposed to optimally trade-off between communication and computation energy. The benefit of our network topology model is that it is a self-organized multi-hop hierarchical clustering network, which provides better performance in term of energy consumption, reliability and network scalability. In addition, our framework can be applied to both homogeneous and heterogeneous mobile sensor networks due to the generalization of the network model. Two multi-UAV information gathering applications, i.e. target tracking and area mapping, were chosen to test the proposed algorithm. Based on simulation results, our method can save up to 40% energy for a target tracking and 60% for an area mapping compared to the baseline approach.

To my dear family and friends.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Dr Eric C. Kerrigan for his guidance and support throughout the course of my PhD. I would like to thank him for giving me the opportunity to join his research group here at Imperial College London. Of course, I would like to thank him for exposing me to fascinating subjects such as control and mathematical optimization. Working with him for the past three years has been a pleasure. He has been an excellent mentor. Without his excellent teaching and valuable feedback, I would not have been able to come this far in my academic career. Most importantly, he has been a great source of knowledge, motivation and support when I needed them most.

I would like to thank my colleagues from Eric's research group for spending their valuable times with me. In particular, I would like to thank Paola Falugi, Andrea Suardi, Bing Feng Ng, Edo Abraham, Zhe Feng, Ge Ming, Hari Vemuri, Andrea Picciau, Luxin Zhang and Bulat Knusainov. My appreciation also extends to my project collaborator, Sujit P. Balityarasimhuni from Indraprastha Institute of Information Technology, for his patience, instruction and cooperation.

I would like to extend my gratitude to the Royal Thai Air Force for their funding support throughout my academic life. Special thanks to Group Caption Punpakdee Pattanakul, the Royal Thai Air Force Attache, for his kindness and affection. I am extremely grateful to my colleagues and superiors at the Royal Thai Air Force Academy for their understanding, passion and encouragement. In particular, I would like to thank my superiors, Gp Capt Preecha Wongsa and Gp Capt Sanga Srisupapreeda for their trust and supervision. My friends, Flt Lt Thanpanat Buaphiban, Wg Cdr Varagorn Poonjun, Wg Cdr Chanchai Sugsanguan, Flt Lt Primrata Jantarachotigul (WRTAF), Dr Panida Yongskulrote, Dr Kessirin Putichote and Ms Patoe Buangam, I thank you all for standing with me during difficult times.

I would like to also thank my Thai friends here for their company in my solitude. It was an extreme pleasure that we could meet and learn together. Without these people, lunch time would have been dull.

Finally, I express my deepest gratitude to my parents and my brother for their unceasing support and love.

Contents

Abstract	5
Acknowledgements	9
Contents	11
List of Figures	15
List of Symbols	17
Abbreviations	21
1 Introduction	23
1.1 Computing Systems as Cyber-Physical Systems	23
1.2 Why is Control Needed in Computing Systems?	24
1.3 Feedback-based Computing Systems	24
1.4 Challenges in Control of Computing Systems	25
1.5 Thesis Layout	26
1.6 Related Publications	28
2 Energy-Efficient Real-Time Multiprocessor Scheduling	29
2.1 Introduction	30
2.1.1 Real-Time Systems	31
2.1.2 Real-Time Task	33
2.1.3 Real-Time Scheduling	34
2.1.3.1 Real-Time Scheduling Problems	34
2.1.3.2 Real-Time Scheduling Algorithm Paradigms	34
2.1.4 Terminologies and Definitions	35
2.1.5 Hardware Platform	36
2.1.5.1 Multiprocessor Systems	36
2.1.5.2 Power and Energy Consumption Model	37
2.1.5.3 Non-convex Power-Speed Relation	39
2.1.6 Power Management Schemes	39

2.1.7	Related Work	41
2.2	Feasibility Problem Formulation	46
2.2.1	System Model	46
2.2.2	Task/Processor Assumptions	46
2.2.3	Scheduling as a Continuous Optimal Control Problem (COCP)	46
2.3	Solving the Scheduling Problem with Finite-dimensional Mathematical Optimization	48
2.3.1	Mixed-Integer Nonlinear Program (MINLP-DVFS)	48
2.3.2	Computationally Tractable Multiprocessor Scheduling Algorithms	50
2.3.2.1	Solving the Workload Partitioning as a Continuous Nonlinear Program (NLP-DVFS)	51
2.3.2.2	Solving the Workload Partitioning as a Continuous Linear Program (LP-DVFS)	52
2.3.2.3	Task Ordering Algorithms	52
2.4	Energy Optimality Problem Formulation	58
2.4.1	Optimal Speed Profile	59
2.5	Simulations for Tasksets with No Uncertainties	62
2.5.1	Comparison Algorithms	62
2.5.1.1	LLREF/RT-SVFS	63
2.5.1.2	GWA-SVFS/GWA-NoDVFS	63
2.5.1.3	GWA-DDiscrete	64
2.5.1.4	GWA-SDiscrete	65
2.5.2	Homogeneous Multiprocessor System Simulations	66
2.5.2.1	System, Processor and Task Models	66
2.5.2.2	Simulation Setup and Results	66
2.5.3	Heterogeneous Multiprocessor System Simulations	77
2.5.3.1	System, Processor and Task Models	77
2.5.3.2	Simulation Setup and Results	78
2.6	Feedback Scheduling	83
2.6.1	Feedback Task Model and Assumptions	84
2.6.2	Feedback Scheduler	84
2.6.3	Simulation and Results	85
2.7	Conclusions	89
3	Multi-UAV Network Topology Optimal Control for Information Gathering Applications	91
3.1	Introduction	92
3.1.1	A Multi-UAV System as a Mobile Wireless Sensor Network	93
3.1.1.1	A Mobile Computing Node Architecture	93
3.1.1.2	Communication Patterns in Wireless Sensor Network	93

3.1.1.3	Hierarchical Network Structure	94
3.1.2	Minimizing Energy Consumption in Wireless Sensor Networks	96
3.1.3	Related Work	99
3.2	Application Details	101
3.2.1	System Assumptions	101
3.2.2	UAV Role Assignment	101
3.2.3	Aggregation Network Topology	102
3.3	Dynamic Model with Constraints	102
3.3.1	Communication Model and Constraints	102
3.3.2	Energy Models	105
3.3.3	UAV Dynamic Constraints	106
3.3.4	State Update Equation	106
3.4	Optimal Control Problem	107
3.5	Applications	109
3.5.1	Target Tracking	109
3.5.1.1	Target and Sensor Models	109
3.5.1.2	Information Filter	109
3.5.1.3	Simulation Setup	110
3.5.1.4	Simulation Results	111
3.5.2	Area mapping	114
3.5.2.1	Simulation Setup	115
3.5.2.2	Simulation Results	117
3.6	Conclusions	120
4	Conclusions and Future Work	121
4.1	Summary of Thesis Achievements	121
4.1.1	Real-Time Multiprocessor Scheduling	121
4.1.2	Multi-UAV Network Topology Optimal Control	122
4.2	Future Work	123
4.2.1	Real-Time Multiprocessor Scheduling	123
4.2.2	Multi-UAV Network Topology Optimal Control	128
	References	131
	Appendices	147
A	Mean Absolute Percentage Error (MAPE)	147
B	Curve Fitting	147

List of Figures

1.1	Feedback-based computing systems (adapted from [1])	24
2.1	Real-Time system block diagram	32
2.2	Example of a non-increasing active energy function, but where the active power consumption is an increasing function.	38
2.3	Fluid schedule and a practical schedule [2]	42
2.4	T-L Planes [2]	43
2.5	Feasible schedule at time interval $[\tau_\mu, \tau_{\mu+1}]$ obtained by McNaughton's wrap around algorithm	54
2.6	A feasible task schedule according to Algorithm 2.	57
2.7	A time-varying speed profile is better than a constant speed profile if the power function is not convex.	60
2.8	Actual data versus fitted model	67
2.9	An example of a minimum taskset density plot for different taskset characteristics	69
2.10	Simulation results for scheduling real-time tasks with implicit deadlines on two homogeneous processors with continuous speed range	70
2.11	Simulation results for scheduling real-time tasks with implicit deadlines on two homogeneous processors with discrete speed levels	71
2.12	Simulation results for scheduling real-time tasks with constrained deadlines on two homogeneous processors with continuous speed range	72
2.13	Simulation results for scheduling real-time tasks with constrained deadlines on two homogeneous processors with discrete speed levels	73
2.14	Examples of optimal speed profiles obtained by solving NLP-DVFS with constrained deadline taskset in Table 2.5 with $D = 1.4$	74
2.15	Examples of optimal speed profiles obtained by solving LP-DVFS with constrained deadline taskset in Table 2.5 with $D = 1.4$	75
2.16	Energy saving percentage box-plots of executing constrained deadline tasksets on a homogeneous system composed of two practical PowerPC 405LP processors	76
2.17	Actual data versus fitted Model	78
2.18	Simulation results for scheduling real-time tasks with implicit deadlines	81
2.19	Simulation results for scheduling real-time tasks with constrained deadlines	82

2.20	Fluid scheduling model with uncertainty in task execution time	85
2.21	Feedback scheduling architecture	85
2.22	Simulation results for different minimum taskset density with $\gamma_i = 0.5, \forall i \in I$	87
2.23	Simulation results for different estimation factor γ with $D = 1.25$	88
3.1	The architecture of a mobile node (adapted from [3])	94
3.2	Traffic patterns in WSN (redrawn from [3])	95
3.3	Hierarchical network structure	96
3.4	Multilevel hierarchical clustering architecture	97
3.5	Energy saving techniques in WSNs (adapted from [4])	98
3.6	Information flow of an aggregation network topology	102
3.7	Aggregation network topology example	104
3.8	Simulation trajectory plot	112
3.9	Normalised total energy consumption for different channel bandwidths with respect to baseline scheme	112
3.10	Aggregator node assignments at different time steps for channel bandwidth $B = 7$ Kbps	113
3.11	The search area is decomposed into lanes and each UAV is assigned to one lane. Once the UAV completes one lane, then another lane is assigned.	115
3.12	The path is given by waypoint (150,0) and (150,300). The vector field of the vehicle at various locations is shown. $\tau = 20$ and $\chi = \pi/3$	116
3.13	Example scenarios with a three vehicle system to illustrate the common data type configurations.	117
3.14	The normalized total energy of the MINLP compared to the baseline strategy for different bandwidth constraints having $\zeta = 0.5$	119
3.15	The normalized total energy of the MINLP with reference to the baseline strategy for different overlap factors having a channel bandwidth of $B = 10$ Mbps.	119
4.1	Impact of operating frequency on idle state residency (reprint from [5])	124
4.2	Power consumption as a function of workload for various multicore processors (reprinted from [6])	125
4.3	Example plot of energy vs temperature for some benchmark taskset (reprinted from [7])	126
4.4	Energy vs task types (reprinted from [8])	127
4.5	Example of a DAG transforming to a parallel synchronous task model (reprinted from [9])	128

List of Symbols

Chapter 2

Variables:

T	Task
b	Task arrival Time
c	Task required number of execution cycles
d	Task deadline
$\mathcal{L}$	Hyperperiod
n	Total number of tasks
s	Operating speed
f	Operating frequency
κ	Total number of processor types
$\underline{x}$	Minimum execution times of a task
δ	Task density
D	Taskset density
C	System capacity
m	Total number of processors
P	Power
V	Voltage
α, β, λ	Constants
E	Energy
x	Remaining execution times of a task
L	Largest deadline of all tasks
a	Processor assignment variable
τ	Major grid time step
h	Time interval
ω	Percentage of task execution times
l	Total number of speed levels
(σ, η)	Workload interval assignment
y	Workload ratio
z	Speed level selection variable

$\underline{y}$	Actual execution times of a task
γ	Estimation factor
Subscripts/Superscripts:	
r	Processor type index
i	Task index
d	Dynamic
s	Static
dd	Supply
t	Threshold
k	Processor index
N	Total number of major grids
M	Total number of minor grids
μ	Major grid index
ν	Minor grid index
q	Speed level

Chapter 3

Variables:

N	Set of all UAVs (nodes)
n	Total number of nodes
C	Communication link matrix/vector
c	Communication link assignment
M	Set of all data types
m	Total number of data types
λ	Average data transmitting rate
$\bar{\lambda}$	Sensing rate
ϵ	Sufficiently small constant/energy constant
$ G $	Total number of sensors of a data type
a	Aggregator assignment
γ	Aggregator ratio
B	Communication bandwidth
h	Decision time interval length
E	Energy consumption
d	Distance between nodes
e	Energy state vector
ϕ	Inertial position vector
x	Position in x-axis
y	Position in y-axis

v/V	Speed/speed vector
ϕ/Φ	Heading angle/Heading angle vector
r	Distance/range
X	State of the system
u	Control input
π	Information contribution
H	Observation matrix
R	Measurement noise covariance matrix
F_0	State transition matrix
w_0	Process noise vector
Q_0	Noise covariance matrix
Z	Measurement vector
ν	Measurement noise vector
K	Distance-independent coefficient
Q	Information matrix
P	Covariance error matrix
$\hat{q}$	Information state vector
S	Set of sensor nodes
W	Width of a region
T	Length of a region
ζ	Overlap factor
N_ℓ	Total number of lanes
ℓ	Lane
ω	Waypoint
τ	Transition boundary
χ	Entry angle
Subscripts/Superscripts:	
i, j	UAV (node)
0	Source (target) node/initial state
$n + 1$	Sink node (base station)
z	Data type
c	Communication
s	Sensing
p	Processing
t	Transmitting/top
r	Receiving
β	Path loss exponent
$+$	Next state
k	Decision making round

κ	Lane index
b	Bottom
d	Desired heading angle

Abbreviations

CPS:	Cyber-Physical System
CPU:	Central Processing Unit
UAV:	Unmanned Aerial Vehicles
MINLP:	Mixed-Integer Nonlinear Program
NLP:	Nonlinear Program
LP:	Linear Program
ACPI:	Advanced Configuration and Power Interface
DVFS:	Dynamic Voltage and Frequency Scaling
DPM:	Dynamic Power Manangement
RTOS:	Real-Time Operating System
LCM:	Least Common Multiple
RM:	Rate Monotonic
EDF:	Earliest Deadline First
LLREF:	Largest Local Remaining Execution Time First
CMOS:	Complementary Metal-Oxide Semiconductor
Pfair:	Proportionate-fair
TL-Plane:	Time and Local Execution Time Plane
RT-SVFS:	Real-Time Static Voltage and Frequency Scaling
DP:	Deadline Partitioning
ILP:	Integer Linear Program
COCP:	Continuous Optimal Control Problem
GWA-SVFS:	Global Workload Allocation with Static Frequency Scaling
GWA-NoDVFS:	Global Workload Allocation with No Dynamic Frequency Scaling
GWA-SDiscrete:	Global Workload Allocation with Static Discrete Frequency Scaling
GWA-DDiscrete:	Global Workload Allocation with Dynamic Discrete Frequency Scaling
MAPE:	Mean Absolute Percentage Error
WSN:	Wireless Sensor Network
HITL:	Human-In-The-Loop
RF:	Radio Frequency
MWSN:	Mobile Wireless Sensor Network
AC/DC:	Analog to Digital Converter

MAC:	Medium Access Control
CSIP:	Collaborative Signal and Information Processing
SPIN:	Sensor Protocol for Information via Negotiation
CNS:	Central at Nearest Source
SPT:	Shortest Paths Tree
GIT:	Greedy Incremental Tree
CS:	Compressed Sensing
NL:	Network Lifetime
LEACH:	Low-Energy Adaptive Clustering Hierarchy
LEACH-C:	Low-Energy Adaptive Clustering Hierarchy Centralized
PEGASIS:	Power-Efficient Gathering in Sensor Information Systems
LA:	Local Aggregator
MA:	Master Aggregator
HARP:	Hierarchical Adaptive and Reliable Routing Protocol
AOI:	Area of Interest
PID:	Proportional Integral Derivative
MPC:	Model Predictive Control
DAC:	Directed Acyclic Graph

Chapter 1

Introduction

This chapter starts by introducing fundamental concepts and terminologies, as well as challenges, in the control of computing systems. This is followed by the thesis layout, and by a list of related publications.

1.1 Computing Systems as Cyber-Physical Systems

A cyber-physical system (CPS) is a system in which computation, networking, and physical processes are integrated and mutually interacting [10]. These systems involve the co-design of control and computing units. CPS is ubiquitous; ranging from personal electronic devices and smart home appliances, to large manufacturing applications or smart grids. By considering both control and computing together, substantial improvements in the performance and efficiency of the system can be obtained. In the past, computing systems have often been viewed as discrete numerical devices from a control perspective. However, this point of view does not apply to modern computers because their behaviour affects the physical process and/or control performance, and vice versa. For example, execution times of a piece of software vary from one computing platform to another due to differences in architecture, so, by being aware of a hardware implementation, one can decide on a more beneficial controller design. Specifically, physical properties of computers need to be incorporated when designing a controller. Therefore, one of the main contributions of this research is the formation of a fundamental framework that bridges the gap between control and computing by considering a computing system as a CPS.

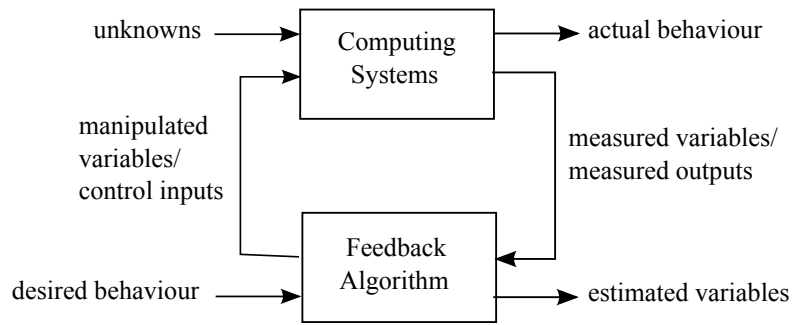


Figure 1.1: Feedback-based computing systems (adapted from [1])

1.2 Why is Control Needed in Computing Systems?

Many problems that arise from computing systems are typical control problems in some form or other [11]. In recent years, control has been applied to solve computing problems such as workload allocations (scheduling problems) [12], resource management [13–15], load balancing [16, 17], workload admission control [12, 18], power/energy management [19, 20], communication network congestion control [21], etc. Additionally, computing systems are associated with uncertainties such as hardware failure, unpredictable workload, lost/corrupted measurement signals, or changes in workload pattern, which is why feedback has been adopted to solve problems in computing systems. Problems in computing systems are often posed as regulation (or tracking) problems [22–25]. However, some problems are more naturally formulated as optimal control problems than as regulation problems. Examples include: the sleep-scheduling problem, which decides on the number of active cores; the problem of determining processor operating states; or even a generic scheduling problem, which makes decisions on which task will be executed on which processor and when.

1.3 Feedback-based Computing Systems

This section introduces control terminologies that will be used throughout the thesis. The essential elements of a typical feedback computing system are illustrated in Figure 1.1. Our abstraction of *computing systems* (often referred to as the plant in the control literature) includes not only a central processing unit (CPU), such as a single core processor, a multicore/multiprocessor, a mobile distributed computing system, or a network of computers, such as a server farm or data centre, but also other system components that need to be controlled, such as memory units, communication units, and software programs. *Unknowns*, such as unpredictable workloads, delay, and hardware failures bring uncertainties to the system. The *measured variables/outputs* are often used

to quantify the control performance. Examples of measured variables are power/energy consumption, user response time, computation time, CPU temperature and resource utilization. The feedback algorithm can then use the value of these measured variables to update a dynamic model of a computing system to obtain better values of the *estimated variables*. If there exists a difference between the actual behaviour and the desired behaviour of the system, the feedback algorithm uses the *manipulated variables/control inputs* as interfaces to adjust the impact of the system on the measured variables until the desired behaviour is sufficiently realised. Examples of manipulated variables are CPU operating frequency, changes in resource utilization or runtime parameters.

1.4 Challenges in Control of Computing Systems

This section identifies a few research challenges in applying control to computing systems.

- How does one model the dynamics of a computing system? In other words, the problem of establishing a relationship between control inputs and measurement outputs that might not appear to be governed by physical laws is difficult. This is why most of the work in the existing literature usually represents the computing dynamic with an approximated linear system model, which in fact might not be able to capture the real dynamic of the system. Moreover, since computing systems have a mixture of both discrete-dynamics, arising from the computation, and continuous-dynamics of the physical properties, newly developed modelling principles are needed to cope with these hybrid systems [1].
- Should one have time-driven or event-driven control? In particular the question of how often the state of the system should be updated is an important one because computations are associated with both energy and time. It might be better to re-compute control inputs only when necessary, since the state of a computing system only changes when there is a triggering event, or when a pre-defined condition is satisfied. However, for some systems it is preferable to correct the system behaviour as early as possible in order to achieve a certain performance. In some cases the cost of the feedback algorithm is not an issue.
- Control design for physical and/or virtual components? Typically, computing system resources such as CPU and memory consist of both physical and virtual components. Specifically, with multi-threading technology, a single physical component can appear to be two logical components from an operating system point of view. While the operating system sees the two logical components as independent units, they are in fact sharing the same physical resource. Up to now,

there is no work taking this factor into account when designing a controller for a computing system. Therefore, to be able to apply control to modern computing systems, the control algorithm might need to take this aspect of the systems into account as well.

- Centralised vs distributed System? All components of a computing system may or may not physically reside on the same chip. Therefore, control algorithms might have to consider the effects of the communication delay and loss on the system performance as well as the costs associated with network latency. The question of how to incorporate network communication into the system is another challenge.
- Control implementation at an operating system (kernel) level or a user/application level? An operating system acts as an interface between software and hardware. Since our target system is a computing system, it might make more sense to implement control at an operating system level in order to minimize delays, by transmitting control signals directly to the hardware layer rather than forwarding the signals from the user space. However, the infrastructure of a kernel is not suitable for complicated control computations. In particular, the ability to implement an optimization solver at operating system level will enhance the performance of the computing systems tremendously, due to more efficient control implementation.

1.5 Thesis Layout

Chapter 2: Energy-Efficient Real-Time Multiprocessor Scheduling

This chapter presents three novel mathematical optimization formulations that solve the same heterogeneous multiprocessor scheduling problem for a real-time taskset. Our formulations are based on a global scheduling scheme and a fluid model. The first formulation is a mixed-integer nonlinear program, since the scheduling problem is intuitively considered as an assignment problem. However, by changing the scheduling problem to first determine a task workload partition and then to find the execution order of all tasks, the computation time can be significantly reduced. Specifically, the workload partitioning problem can be formulated as a continuous nonlinear program for a system with continuous operating frequency, and as a continuous linear program for a practical system with a discrete speed level set. The task ordering problem can be solved by an algorithm, which has a time complexity linear in the total number of tasks. The work is evaluated against existing global energy/feasibility optimal workload allocation formulations. The results illustrate that our algorithms are both feasibility optimal and energy optimal for both implicit and constrained deadline tasksets. The benefit of our

formulation compared with existing work is that our algorithms can solve a more general class of scheduling problems due to incorporating a scheduling dynamic model in the formulations and allowing for a time-varying speed profile. Moreover, our algorithms can be applied to both online and offline scheduling schemes. For a homogeneous system, the results provided by our algorithm can achieve up to an 80% energy saving compared to an algorithm without a frequency scaling scheme and up to 70% energy saving compared to a constant frequency scaling scheme for some simulated tasksets. For a heterogeneous system, an energy saving up to 40% can be observed for some simulated taskset with constrained deadlines.

To cope with uncertainty in task execution times, we propose an optimal control framework that takes advantage of feeding back information upon finished tasks to solve a real-time multiprocessor scheduling problem with uncertainty in task execution times, with the objective of minimizing the total energy consumption. The simulation results illustrate that our feedback scheduling algorithm can save up to 40% energy compared to the open-loop algorithm for two practical processor models, i.e. PowerPC 405LP and XScale.

Chapter 3: Multi-Unmanned Aerial Vehicle Network Topology Optimal Control for Information Gathering Applications¹

Typical mobile agent networks, such as a system composed of Unmanned Aerial Vehicles (UAVs), are constrained by limited resources: energy, computing power, memory and communication bandwidth. In particular, limited energy affects elements of system performance directly, such as system lifetime. Moreover, it has been demonstrated experimentally, in the wireless sensor network literature, that the total energy consumption is often dominated by the communication cost, i.e. the computational and sensing energy are small compared to the communication energy consumption. For this reason, the lifetime of the network can be extended significantly by minimizing the communication distance as well as the amount of communication data, at the expense of increasing computational cost. In this work, we aim to achieve an optimal trade-off between communication energy and computational energy. Specifically, we propose a mixed-integer optimization formulation for a multi-hop hierarchical clustering-based self-organizing UAV network incorporating data aggregation, to obtain an energy-efficient information routing scheme. The proposed framework is tested on two applications, namely target tracking and area mapping. Based on simulation results, our method can save significant amounts of energy compared to the baseline strategy, where there is no data aggregation and clustering scheme.

¹This work is in collaboration with P.B. Sujit, Department of Electrical and Computer Engineering, Indraprastha Institute of Information Technology, India and João B. Sousa, Department of Electrical and Computer Engineering, University of Porto, Porto, Portugal.

1.6 Related Publications

Most of the work and results presented in this thesis are based on the contents of the following publications:

- M. Thammawichai and E. C. Kerrigan. “Energy-Efficient Scheduling for Homogeneous Multiprocessor Systems,” in IEEE Real-Time Systems Symp. (RTSS 2016), Submitted on 3 April 2016.
- M. Thammawichai and E. C. Kerrigan, “Energy-Efficient Real-Time Scheduling for Two-Type Heterogeneous Multiprocessors,” IEEE Trans. Parallel Distrib. Syst., Submitted on 31 March 2016.
- M. Thammawichai and E. C. Kerrigan. “Feedback Scheduling for Energy-Efficient Real-Time Homogeneous Multiprocessor Systems,” IEEE 55th Conf. on Decision and Control (CDC 2016), Submitted on 4 March 2016.
- M. Thammawichai, S.P. Baliyarasimhuni, E. C. Kerrigan and João B. Sousa, “Optimizing Communication and Computation for Multi-UAV Information Gathering Applications,” IEEE Trans. Mobile Comput., Submitted on 31 March 2016.

Chapter 2

Energy-Efficient Real-Time Multiprocessor Scheduling

In this chapter, three mathematical programming formulations are presented to solve the same real-time task scheduling problem on multiprocessor systems with dynamic voltage and frequency scaling capabilities. The main contributions are:

- A mixed-integer nonlinear program (MINLP) formulation that adopts the fluid model used in [2, 26–28] to represent a scheduling dynamic to solve a multiprocessor scheduling problem. The MINLP formulation relies on the optimal control of a given system's dynamic to globally solve for a valid schedule.
- By determining a percentage of task execution time, rather than task assignments, the same scheduling problem can be divided into two sub-problems, i.e. a workload partitioning problem and a task ordering problem. A nonlinear program (NLP) is proposed to solve the workload partitioning problem for a system with continuous operating speed and we propose a linear program (LP) for a practical system where a processor has a discrete operating speed set.
- We have produced a multiprocessor scheduling algorithm that is both feasibility optimal and energy optimal.
- Our formulations are capable of solving multiprocessor scheduling problems with any periodic taskset characteristics, i.e. implicit, constrained and arbitrary deadlines.
- Moreover, the proposed formulations can be also extended to a multi-core architecture, which only allows frequency to be changed at cluster-level, rather than at core-level as in a multiprocessor.
- Lastly, a feedback and optimal control framework is proposed to solve a real-time

scheduling problem with uncertainty in task execution times.

The chapter is organized as follows: Section 2.1 provides an introduction to necessary notations and fundamental knowledge on real-time systems. Section 2.2 defines our scheduling problem in detail, starting by introducing a system model in Section 2.2.1, followed by task and processor assumptions in Section 2.2.2. Section 2.2.3 presents multiprocessor scheduling as an infinite-dimensional optimal control problem. Details on solving the scheduling problem using finite-dimensional mathematical optimization is given in Section 2.3. Specifically, a mixed-integer nonlinear formulation with time discretization is proposed in Section 2.3.1. A continuous nonlinear program and a continuous linear program to solve a workload partitioning problem are presented in Section 2.3.2.1 and Section 2.3.2.2, respectively. Task ordering algorithms are presented in Section 2.3.2.3. Simulation setup and results for tasksets with no uncertainties are presented in Section 2.5. The feedback scheduling framework and simulation results are given in Section 2.6 and Section 2.6.3, respectively. Finally, conclusions are summarised in Section 2.7.

2.1 Introduction

Due to higher computational power demands in today’s computing systems, e.g. sensor networks, multi-robot systems, automotive electronics, avionics, satellites/space systems, medical applications, as well as personal electronic devices, multiprocessor architectures play a major role in modern computational systems. A statistical study indicates that computing devices including server farms, data centres, portable devices and desktops will consume more than 14% of the global electricity consumption by 2020 [29]. As the performance and speed of processors increase, the main challenges in the design of future high-performance computing systems are processor power consumption and heat dissipation. Moreover, these systems may need to operate under tight power and energy requirements while meeting real-time requirements, that is, the deadlines.

As specified by the Advanced Configuration and Power Interface (ACPI) [30] — an open industry standard for device configuration, power and thermal management — power usage of a device can be controlled by various methods. For example, by controlling CPU time in the idling power states, changing operating frequency in the performance states, or putting a CPU to sleep in throttling states when the CPU temperature is critically high.

Dynamic Voltage and Frequency Scaling (DVFS) technique has been widely used as an energy management scheme in real-time systems. Typically, a processor running at

a higher clock frequency consumes higher energy than a processor running at a lower clock frequency. Hence, a DVFS technique aims to reduce the power/energy consumption by dynamically allowing a processor to reduce its operating frequency/voltage to match its workload. Since timeliness is an important aspect for real-time systems, the main consideration in applying DVFS in the system is to ensure that the deadline constraint is not violated. Another common technique is called Dynamic Power Management (DPM). Generally, computing devices are designed with two types of states: the active state, which is a workload executing state, and the lower-power idle/sleep state. The latter can consist of more than one state, i.e. the deeper idle states, which save more energy, but have longer re-activating times. The DPM technique involves dynamically deciding when, and for how long, the unused units should be put into sleep/idle states to reduce the power/energy consumption of the system.

However, for this thesis, our focus is on the efficiency of the DVFS technique alone. To be able to apply the DVFS technique efficiently, both energy management and task scheduling have to be considered simultaneously, since they affect one another. For example, as the operating frequency is scaled, meeting the timing constraints of the real-time system will also depend on the order in which the tasks are executed. On the other hand, different task execution orders may result in different total energy consumptions.

To conclude, as the computational demands of real-time applications continue to increase, it is clear that an efficient energy management protocol is required. Furthermore, achieving the real-time constraints while effectively utilizing the available processing resources requires an efficient scheduling algorithm. With this motivation, this work is aimed at using an optimization-based approach to develop an algorithm for a multiprocessor real-time scheduling problem with the goal of minimizing energy consumption, while ensuring that both the execution cycle requirements and timeliness constraints of real-time tasks are satisfied.

2.1.1 Real-Time Systems

Real-time systems [31] are systems where performance depends on both the correctness of the computational results and the time in which these results are computed. For example, an autonomous flight control system needs to periodically measure and compute necessary data within a deadline in order to provide correct control of an aircraft. In order to guarantee the performance of a real-time system, time is clearly a crucial resource to be managed.

Figure 2.1 shows a typical real-time system block diagram composed of three different

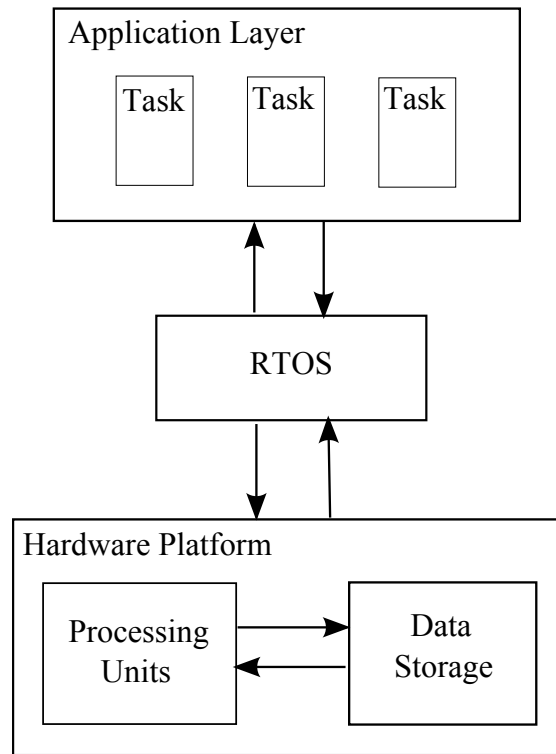


Figure 2.1: Real-Time system block diagram

layers, i.e. an application layer, a real-time operating system (RTOS), and a hardware platform. The application layer is a compilation of all tasks/programs that need to be executed. The real-time operating system, which is an interface between the application layer and the hardware platform, performs a scheduling decision that determines when a task will be executed and on which processor. Lastly, the hardware platform includes processors, memories, communication subsystems, etc.

A real-time system can be divided into three categories according to strictness of deadline: *hard*, *firm* and *soft* real-time systems. The consequence of not meeting a deadline in a hard real-time system is failure, whereas in a firm real-time system, or a soft real-time system, the value of a task drops to zero, or linearly decreases, respectively. Examples of hard real-time systems are primarily related to safety critical systems, e.g. aircraft/air traffic control systems, medical treatment systems, or nuclear reactor control, etc. In contrast, an audio-video decoder is a typical example of a firm real-time system since the cost of missing the deadlines of some tasks will only degrade the quality of service. Though, in this thesis, we only consider hard real-time systems, the scheduling algorithms for hard real-time systems can also be applied to firm and soft real-time systems due to the stricter constraint on the deadline.

2.1.2 Real-Time Task

A real-time task [31] is a computational task required in real-time applications under a timing constraint. It can be divided into two categories, i.e. periodic and aperiodic tasks. A periodic task is activated at a regular interval. It is normally assumed that the characteristics of a periodic task, such as its period or its computational workload, are known in advance. An aperiodic task is activated by a particular event. Typically, characteristics of an aperiodic task are not known until the task arrives in the system. If a minimum inter-arrival time — i.e. the minimum time between consecutive tasks — of an aperiodic task is known, then it is called a sporadic task.

Formally, a *real-time task* T_i is defined as a quadruple $T_i := (b_i, c_i, d_i, p_i)$; b_i is the task arrival time, c_i is the required number of processor cycles needed to complete the task, d_i is the task deadline relative to the arrival time and p_i is the task period.

A periodic task T_i is modelled as a triple $T_i := (c_i, d_i, p_i)$, while an aperiodic task is defined as a triple $T_i := (c_i, d_i, b_i)$. A periodic task is said to have an *implicit deadline* if its deadline is equal to its period, a *constrained deadline* if its deadline is not larger than its period, and an *arbitrary deadline* if its deadline can be less than, equal to, or greater than its period. Throughout the thesis, we will refer to a task as an aperiodic task model unless stated otherwise because a periodic task can be transformed into a collection of aperiodic tasks with appropriately defined arrival times and deadlines, i.e. the j^{th} instance of a periodic task T_i (often called job in the literature), where $j \geq 1$, arrives at time $(j - 1)p_i$, has the required execution cycles c_i and an absolute deadline at time $(j - 1)p_i + d_i$. Moreover, for a periodic taskset, we only need to find a valid schedule within hyperperiod $\mathcal{L}$, defined as the least common multiple (LCM) of all task periods, i.e. the total number of job instances of a periodic task T_i within hyperperiod $\mathcal{L}$ equals to $\mathcal{L}/p_i$.

A *sporadic task* is also characterized by a three-tuple as a periodic task, except that p_i is defined as the minimum separation time between successive jobs of the same task, i.e. the inter-arrival time of successive jobs is at least p_i time units.

A set composed of real-time tasks is called a *taskset*, i.e. $\{T_1, \dots, T_n\}$, where n is the total number of tasks in the taskset.

The two classes of tasksets describing an application are either (i) a taskset in which there are no uncertainties in the taskset characteristics, e.g. task execution times, the number of tasks in the system, etc., or (ii) a taskset associated with uncertainties, i.e. taskset characteristics can vary and tasks can join or leave the system at any time. For this thesis, we will first solve the scheduling problem with a taskset with deterministic characteristics, then the feedback scheme will be adapted to solve the

scheduling problem with uncertainty in task execution times. Note that our scheduling algorithms can be easily extended to cope with a sporadic taskset, which also exhibits non-deterministic task characteristics.

2.1.3 Real-Time Scheduling

2.1.3.1 Real-Time Scheduling Problems

Scheduling problems [32, 33] in a real-time system can be divided into sub-problems as follows:

- *The feasibility problem*: For a given specification on a taskset and constraints on a scheduling environment, determine whether there is a schedule for the taskset that guarantees all the tasks will meet their deadlines.
- *The allocation problem*: For a given specification on a taskset and constraints on a scheduling environment, determine on which processor tasks should be executed, and in which order, to obtain optimal performance.

For this work, we aim to develop an optimization-based algorithm to solve the above scheduling problems.

2.1.3.2 Real-Time Scheduling Algorithm Paradigms

A scheduling algorithm of a periodic real-time taskset can be classified under different priority schemes [32, 33] as follows:

- *Static priorities*: Each task in a system has a unique priority. Each job created by the same task has the same priority assignment. The Rate Monotonic (RM) [34] algorithm, which assigns a task priority inversely proportional to its period (i.e. a shorter period task will have a higher priority) is an example of this class.
- *Job-level dynamic priorities*: Jobs of the same task may have different priority assignments. However, a job's priority cannot be changed once it is assigned. The Earliest Deadline First (EDF) [35] algorithm, which assigns priority based on a job's deadline, is an example of this class.
- *Unrestricted dynamic priorities*: In this class, a job's priority may change at any time. A scheduling algorithm example of this class is the Largest Local Remaining Execution Time First (LLREF) [2] algorithm, which assigns priority based on a job's laxity, i.e. the difference between the job's remaining execution time and its deadline.

In addition to priority-based classification, a scheduling algorithm can also be classified according to the task migration¹ scheme as follows:

- *No migration (partitioned) scheduling*: Once a task is assigned to a processor, it cannot migrate, i.e. a job generated by a task can only be executed on a corresponding processor.
- *Restricted migration (task-level migration) scheduling*: Jobs of the same task can be assigned to a different processor, but each job can only be executed on a single processor.
- *Full migration (job-level migration/global) scheduling*: A job has no restriction on migration, but it cannot be executed on more than one processor at a time, i.e. parallel execution is not permitted.

There are several advantages of a partitioned scheduling scheme over a global scheduling scheme. First, in a partitioned scheduling paradigm, once a task is assigned to a specific processor, the multiprocessor scheduling problem is reduced to just a set of uniprocessor scheduling problems, which has been well studied. Second, by not allowing task to migrate during runtime, overheads such as context switches² or cache misses³ are minimized. However, a global scheduling scheme does have some advantages over a partitioned scheme. First, the global approach can provide an optimal schedule, while the partitioned scheme cannot. Second, in some systems, the cost of context switch overheads is lower than the performance cost, which makes global scheduling preferable. Lastly, for a system in which tasks can join or leave the system at any time, a partitioning scheme may need to be re-computed, resulting in computational overheads.

2.1.4 Terminologies and Definitions

This section provides basic terminologies and definitions used throughout the chapter. Note that when all processors in the system are the same, i.e. a homogeneous multiprocessor system, the appropriated superscript r will be dropped out.

Speed s^r : The operating speed s^r is defined as the ratio between the operating frequency f^r of processor type- r and the maximum system frequency f_{max} , i.e. $s^r := f^r / f_{max}$.

¹In computing, a task *migration* occurs when a task execution is suspended on one processor and continues on another processor, while the term *preemption* is used when the execution of a task on a processor is suspended in order to start executing another task.

²A context switch refers to the act of replacing the current executing task by another task, i.e. the state of the current task is saved, while the state of another task is restored and becomes an executing task.

³Cache miss occurs when the data requested for processing cannot be found in the cache memory.

$f_{max} := \max\{f_{max}^r \mid r \in R\}$, where $R := \{1, 2, \dots, \kappa\}$, f_{max}^r is the maximum operating frequency of processor of type- r and κ is the total number of processor types.

Minimum Execution Time⁴ $\underline{x}_i$: The minimum execution time $\underline{x}_i$ is the execution time of task T_i when executed at the maximum system frequency f_{max} , i.e. $\underline{x}_i := c_i / f_{max}$.

Task Density⁵ $\delta_i(s_i)$: For a periodic task, a task density $\delta_i(s_i)$ is defined as the ratio between the task execution time and the minimum of its deadline and its period, i.e. $\delta_i(s_i) := c_i / (s_i f_{max} \min\{d_i, p_i\})$, where s_i is the task execution speed.

Taskset Density $D(s_i)$: A taskset density $D(s_i)$ of a periodic taskset is defined as the summation of all task densities in the taskset, i.e. $D(s_i) := \sum_{i=1}^n \delta_i(s_i)$. The minimum taskset density D is given by $D := \sum_{i=1}^n \delta_i(1)$.

System Capacity C : The system capacity C is defined as $C := \sum_{r \in R} s_{max}^r m_r$, where s_{max}^r is the maximum speed of processor type- r , i.e. $s_{max}^r := f_{max}^r / f_{max}$, $f_{max}^r := \max f^r$, m_r is the total number of processors of type- r .

Feasibility Optimal: An algorithm is feasibility optimal if the algorithm is guaranteed to be able to construct a valid schedule such that no deadlines are missed, provided a schedule exists.

Energy Optimal: An algorithm is energy optimal when it is guaranteed to find a schedule that minimizes the energy, while meeting the deadlines, provided such a schedule exists.

Step Function: A function $f : X \rightarrow \mathbb{R}$ is a step (also called a piecewise constant) function, denoted $f \in \mathcal{PC}$, if there exists a finite partition $\{X_1, \dots, X_p\}$ of $X \subseteq \mathbb{R}$ and a set of real numbers $\{\phi_1, \dots, \phi_p\}$ such that $f(x) = \phi_i$ for all $x \in X_i, i \in \{1, \dots, p\}$.

2.1.5 Hardware Platform

2.1.5.1 Multiprocessor Systems

In this thesis, we consider two types of multiprocessor system:

- *Homogeneous Multiprocessor System*, in which all processors are identical, i.e. all processors have the same operating frequency set, and power consumption model,

⁴In the literature, this is often called ‘worst-case execution time’. However, in the case where the speed is allowed to vary, using the term ‘minimum execution time’ makes more sense, since the execution time increases as the speed is scaled down. For simplicity of exposition, we also assume no uncertainty, hence ‘worst-case’ is not applicable here. Extensions to uncertainty should be relatively straightforward, in which case $\underline{x}_i$ then becomes ‘minimum worst-case execution time’.

⁵When all tasks are assumed to have implicit deadlines, this is often called ‘task utilization’.

and the required execution cycles of the same task on all processors are the same.

- *Heterogeneous Multiprocessor System*, which is composed of more than one type of processors, i.e. all processor types have different operating frequency sets, power consumption models and the execution cycles required by the same task on different processor types are different.

A processor considered in the real-time literature can be either ideal or non-ideal (practical). An ideal processor operates in a continuous range of frequencies, while a practical processor can only operate at discrete levels of frequency. Typically, the speed of executing a task is assumed to be linear in frequency.

2.1.5.2 Power and Energy Consumption Model

A power consumption model can be expressed as a summation of dynamic power consumption P_d and static power consumption P_s . Dynamic power consumption is due to the charging and discharging of CMOS gates, while static power consumption is due to subthreshold leakage current and reverse bias junction current [36]. The dynamic power consumption of CMOS processors at a clock frequency $f = sf_{max}$ is given by

$$P_d(s) = C_{ef} V_{dd}^2 s f_{max}, \quad (2.1a)$$

where the constraint

$$s f_{max} \leq \zeta \frac{(V_{dd} - V_t)^2}{V_{dd}} \quad (2.1b)$$

has to be satisfied [36]. Here $C_{ef} > 0$ denotes the effective switch capacitance, V_{dd} is the supply voltage, V_t is the threshold voltage ($V_{dd} > V_t > 0$ V) and $\zeta > 0$ is a hardware-specific constant.

From (2.1b), it follows that if s increases, then the supply voltage V_{dd} may have to increase (and if V_{dd} decreases, so does s). In the literature, the total power consumption is often simply expressed as an increasing function of the form

$$P(s) := P_d(s) + P_s = \alpha s^\beta + P_s, \quad (2.2)$$

where $\alpha > 0$ and $\beta \geq 1$ are hardware-dependent constants, while the static power consumption P_s is assumed to be either constant or zero [37].

The energy consumption of executing and completing a task T_i at a constant speed s_i is given by

$$E(s_i) := \frac{c_i}{f_{max}} \frac{(P_d(s_i) + P_s)}{s_i} \quad (2.3a)$$

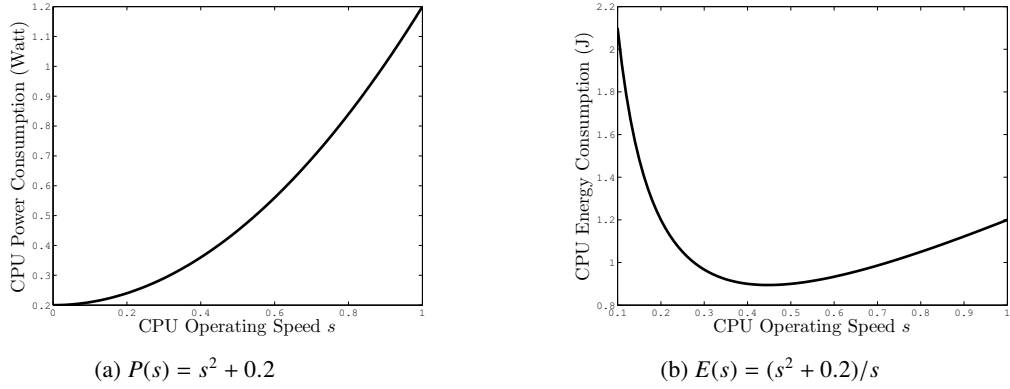


Figure 2.2: Example of a non-increasing active energy function, but where the active power consumption is an increasing function.

$$= \frac{x_i(P_d(s_i) + P_s)}{s_i}. \quad (2.3b)$$

In the literature, it is often assumed that E is an increasing function of the operating speed. However, because $s \mapsto 1/s$ is a decreasing function, it follows that the energy consumed might not be an increasing function if P_s is non-zero; Figure 2.2 gives an example of when the energy is non-monotonic, even if the power is an increasing function of clock frequency. This result implies the existence of a non-zero energy-efficient speed s_{eff} , i.e. the minimizer of (2.3) [38–40].

The total energy consumption of executing a real-time task T_i can be expressed as a summation of active energy consumption and idle energy consumption, i.e. $E = E_{active} + E_{idle}$, where E_{active} is the energy consumption when the processor is busy executing the task, and E_{idle} is the energy consumption when the processor is idle. The energy consumption of executing and completing a task T_i at a constant speed s_i is

$$E(s_i) = E_{active}(s_i) + E_{idle} \quad (2.4a)$$

$$= \frac{c_i}{f_{max}} \frac{(P_{active}(s_i) - P_{idle})}{s_i} + P_{idle}d_i \quad (2.4b)$$

$$= \frac{x_i(P_{active}(s_i) - P_{idle})}{s_i} + P_{idle}d_i, \quad (2.4c)$$

where $P_{active}(s_i) := P_{da}(s_i) + P_{sa}$ is the total power consumption in the active interval, $P_{idle} := P_{di} + P_{si}$ is the total power consumption during the idle period. $P_{da} > 0$ and $P_{sa} \geq 0$ are dynamic and static power consumption during the active period, respectively. Similarly, $P_{di} > 0$ and $P_{si} \geq 0$ are the dynamic and static power consumption during the idle period. P_{di} will be assumed to be a constant, since

the processor is executing a nop (no operation) instruction at the lowest frequency f_{min} during the idle interval. Thus, when compared with executing other instructions during the active interval, the dynamic power consumption during the idle interval is strictly less than the dynamic power consumption during the active interval, i.e. $P_{di} < P_{da}(f_{min}/f_{max})$. P_{sa} and P_{si} are also assumed to be constants where $P_{si} < P_{sa}$. Note that $P_{active}(s_i) - P_{idle}$ is strictly greater than zero.

2.1.5.3 Non-convex Power-Speed Relation

Traditionally, it is often assumed that the power consumption has a convex relationship with the operating speed described in Section 2.1.5.2. However, such an assumption might not be valid for current system architecture and chip process technology. For example, for a heterogeneous chip multiprocessor system, which is a promising architecture for future computing platforms [41], the relationship between power consumption and speed might not be convex due to its heterogeneity. Moreover, as chip sizes are scaled down, the leakage power consumption is expected to dominate over the dynamic power consumption [42]; the total power consumption can no longer be assumed to be a convex function of speed. Additionally, the non-convex relationship between the energy consumption and processor speed can be observed as a result of scaling a supply voltage [43]. Therefore, in this work, we will not make an assumption that the power/energy model is convex.

2.1.6 Power Management Schemes

There are two energy management techniques that have been commonly implemented in modern computing systems, both at hardware and software levels: DPM technique and DVFS scheme. Examples include Intel's SpeedStep technology, AMD's PowerNow! technology and the Linux Kernel Power Management Scheme.

Conceptually, DPM technique exploits the fact that the workload is nonuniform over the operating time. By dynamically predicting the workload with a certain degree of confidence, some unused system (and its components) can be put into a power-efficient state, an idle state, or turned off. Hence, the system power consumption is reduced. However, the drawback for this technique is that there is a penalty in terms of extra energy and/or latency in bringing the device back to an active state once it is in idle state. Therefore, the problem of managing the power/energy consumption under DPM technique is to make decisions on which components should be in which idle or sleep states, and when, as well as how long it should be in that state in order to minimize the total energy consumption of executing the workload. Though it is a good strategy

to turn off components of the system when there is no workload, “when to shutdown” the components is a difficult question. For example, the simple strategy of going into sleep state as soon as possible might not be the best option due to the existence of state transition costs, i.e. latency and energy associated with state entry/resume. Hence, it might be the case that the energy consumed during the state transition might be greater than the energy saved. This leads to the concept of energy break-even time [44], defined as the minimum time interval (including the state entry/resume latencies) that allows the system to achieve the required energy-saving. Particularly, the system should consider sleep-state transition only when the idle period is greater than the energy break-even time; otherwise it consumes higher energy by entering the sleep state than by staying in the active state.

On the other hand, the DVFS technique aims at dynamically determining the operating speed profile such that the total energy consumption is minimized, since the dynamic power consumption of a CMOS processor is a function of both operating frequency and supply voltage. The main challenge in applying this techniques is to guarantee the feasibility of the schedule as well as the timeliness constraint of a real-time task when the operating frequency is scaled. However most of the DVFS schemes currently implemented in modern computing systems are based on simple heuristic strategies. For example, the Linux Kernel supports DVFS through its CPUfreq subsystem, which provides an interface between hardware and software so that the system/user can dynamically change the CPU operating frequency through control drivers. The CPU operating frequency is adjusted based on different power management policies, called governors. The following governors are available within the CPUfreq subsystem [45]:

- *Performance*: The CPU frequency is statically set to the highest possible clock frequency.
- *Powersave*: The CPU frequency is statically set to the lowest possible clock frequency.
- *Ondemand*: The CPU frequency is dynamically adjusted according to the workload. The highest possible clock frequency is set when the CPU workload exceeds a certain threshold, normally at 80% of the current operating frequency. The frequency is then gradually decreased when the workload is below the threshold.
- *Conservative*: Like the Ondemand governor, the frequency is dynamically adjusted according to the workload, but the frequency is either increased or decreased by a set value, which is 5% of the highest frequency by default.

- *Userspace*: This governor allows user to set the CPU to a specific frequency allowed by the hardware.

Note that in this thesis only DVFS technique will be considered as the main power/energy reduction scheme, i.e. we aim to use a mathematical optimization to make decisions on the CPU operating speed profile such that the total energy consumption of executing the workload within the deadlines is minimized. Energy efficient scheduling algorithms for real-time applications on DVFS computing systems have been well studied, as surveyed in [46]. However, some assumptions associated with a real physical system are often ignored for simplicity. These assumptions are: continuous operating frequency range, zero idle power consumption and a cubic power consumption model. By assuming that the energy consumption model is a strictly increasing convex function of frequency, [47–49] show that the optimal strategy is to set the operating frequency to the lowest possible frequency such that the task will meet its deadlines. In contrast, a speed assignment problem within the constraints of a practical system is considered in [50] by assuming discrete frequency levels and using the measured power consumption of the system. They demonstrated that assuming zero idle power consumption can cause an abnormality, called inefficient frequency to arise in a practical system, i.e. operating at a higher frequency consumes less energy per cycle than operating at a lower frequency. In [51, 52] the DVFS technique is exploited to reduce the energy consumption of a parallel computing system by using a linear combination of discrete frequency levels. Specifically, in [51], the proposed scheme is to use a linear combination of the maximum and the minimum frequency levels, while in [52], a linear combination of two adjacent frequency levels, close to the determined continuous optimal frequency value, is selected. However, for this thesis, we will show (in section 2.4.1) that the optimal speed profile solution to a uniprocessor scheduling problem is a linear combination of at most two speed levels for any arbitrary power/energy functions of speed/frequency, and a constant for a convex power consumption model.

2.1.7 Related Work

There are two well-known feasibility optimal homogeneous multiprocessor scheduling algorithms: *Proportionate-fair* (Pfair) [26] and *Largest Local Remaining Execution Time First* (LLREF) [2]. Both Pfair and LLREF are global scheduling algorithms that allow full task migration and are based on a fluid scheduling model.

A fluid model, shown in Figure 2.3, is the ideal schedule path of a task T_i with the required number of execution cycles c_i and the deadline d_i . The remaining minimum execution time $x_i(t)$, where $x_i(0) = \underline{x}_i$, is represented by a straight line where the slope of the line is the execution speed of the task. However, a practical task execution path

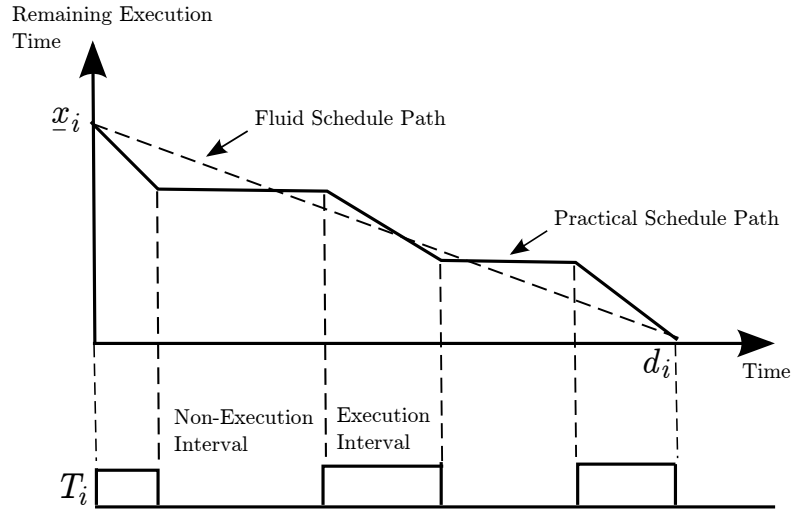


Figure 2.3: Fluid schedule and a practical schedule [2]

is nonlinear, since a task may be preempted by other tasks. The execution interval of a task is represented by a line with a negative slope and the non-execution interval is represented by a line with zero slope. By trying to track a fluid schedule path, the Pfair algorithm can construct an optimal quantum-based schedule, where a quantum is defined as any time interval. By introducing the notion of *fairness*, the algorithm ensures that no task is one or more quanta away from the task's fluid model at any time instant. However, the Pfair algorithm suffers from a significant run-time overhead, because tasks are split into several quantum length segments, incurring frequent algorithm invocations and task migrations.

To overcome this problem, a task needs to be split [2]. However, a task should be split/preempted at other scheduling events, rather than using a time quantum as in the Pfair algorithm. Therefore, the LLREF algorithm describes a fluid schedule without using a time quantum by introducing a Time and Local Execution Time Plane (T-L plane) concept. A T-L plane is a simple fluid schedule with time on the x-axis and the remaining execution time on the y-axis. When the T-L planes of n tasks are considered together, an overlapped isosceles triangle (TL^k) can be formed between two consecutive scheduling events, i.e. task deadlines, where k represents the k^{th} scheduling event. An example of T-L plane instances for three tasks with different deadlines is shown in Figure 2.4. The x-axis of the TL^k is time, while the y-axis represents the remaining execution time of the tasks x_i^k , also known as local remaining execution time. Within a T-L plane, a fluid path can move in two directions. If the task is selected for execution, the fluid path can move diagonally down, otherwise it moves horizontally, as in Figure 2.3.

With this representation, there are two time instants where the LLREF algorithm needs

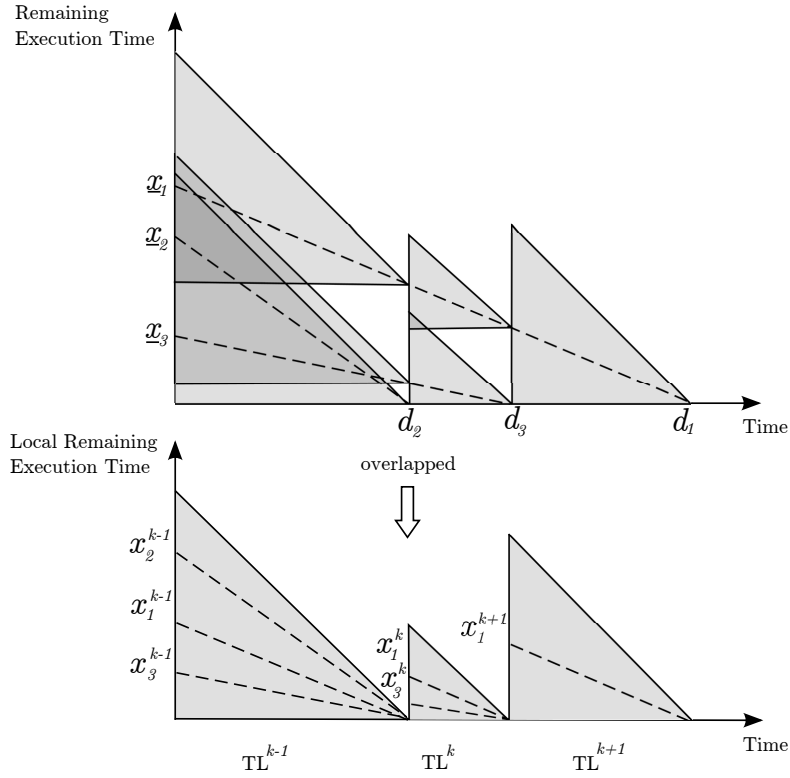


Figure 2.4: T-L Planes [2]

to make a decision. One occurs when the local remaining time of an executing task is zero, i.e. when a fluid path hits the bottom line, called a *bottom hitting event*, and it is better to select another task to run. The other event happens when a fluid path hits a diagonal line of a triangle, called a *ceiling hitting event*. This means that the task has no local laxity and the task needs to be selected immediately in order to finish the local execution time in time. Hence, at these two events, the LLREF algorithm is called, which selects the first m largest local remaining execution time tasks for execution, where m is the number of processors in a system, to obtain local feasibility.

By extending the LLREF algorithm, [27, 28] incorporated a DVFS scheme with a fluid model scheduling algorithm on a multiprocessor to obtain an energy efficient scheduling algorithm for real-time systems. The real-time static voltage and frequency scaling (RT-SVFS) algorithm [27] modified the T-L plane such that the slope of a fluid schedule can vary between 0 and 1, where 1 means that a task is executed at a maximum frequency. The overall algorithm is similar to [2], except that the algorithm needs to decide on the execution speed/frequency of each processor. Two open-loop algorithms are proposed in [27], i.e. Uniform RT-SVFS and Independent RT-SVFS. If all processors in the system cannot run at different speeds, then a uniform speed is chosen, i.e. $s^k := \max\{\delta_{max}, D/m\}$, where s^k is the executing speed of the k^{th} T-L plane and δ_{max} is the maximum task density in the taskset. In the case of independent processors, a task is

classified as heavy or light based on its density, i.e. a heavy task density is greater than 0.5. Each heavy task T_i is individually assigned to one of the processors with $s_i^k = \delta_i$, while all light tasks are executed on the rest of the processors with uniform speed, as explained earlier.

To improve the performance of the open-loop algorithm of [27], a closed-loop dynamic algorithm is proposed in [28]. The dynamic algorithm considers uncertainty in the task's execution time. Because the task execution times are based on a worst-case analysis, it is often the case that tasks will finish earlier than expected, causing processors to be idle during that period. Therefore, it is better to feed back this information and obtain a new schedule. By extending [27], an energy-efficient scheduling algorithm for sporadic real-time tasks was proposed in [53]. Since a sporadic real-time task's arrival time is unknown in advance, the algorithm is modified to cope with this uncertainty by adding an additional event, called an *arrival event*, on top of existing events.

The unified theory of the deadline partitioning technique and its feasibility optimal versions, called DP-FAIR, for periodic and sporadic tasks are given in [54]. Deadline Partitioning (DP) [54] is the technique that partitions time into intervals bounded by two successive task deadlines, after which each task is allocated the workload and is scheduled at each time interval. A simple optimal scheduling algorithm based on DP-FAIR, called DP-WRAP, was presented in [54]. The DP-WRAP algorithm partitioned time according to the DP technique and, at each time interval, the tasks are scheduled using McNaughton's wrap around algorithm [55]. The work of [54] was extended in [56, 57] by incorporating a DVFS scheme to reduce power consumption.

The algorithms that are based on the fairness notion [2, 27, 28, 54, 56, 57] are feasibility optimal, but they have seldom been applied in a real system, since they suffer from high scheduling overheads, i.e. task preemptions and migrations. Recently, two feasibility optimal algorithms that are not based on the notion of fairness have been proposed. One is the RUN algorithm [58], which uses a dualization technique to reduce the multiprocessor scheduling problem to a series of uniprocessor scheduling problems. The other is U-EDF [59], which generalizes the EDF algorithm to multiprocessors by reducing the problem to EDF on a uniprocessor.

Alternatively to the above methods, the multiprocessor scheduling problem can also be formulated as an optimization problem. However, since the problem is NP-hard, in general, an approximated polynomial-time heuristic method is often used. An example of these approaches can be found in [60, 61], which consider energy-aware multiprocessor scheduling with probabilistic task execution times. A partitioning approach is chosen, i.e. the tasks are partitioned among the set of processors, followed by computing the running frequency based on the task execution time probabilities.

Among all of the feasibility assignments, an optimal energy consumption assignment is chosen by solving a mathematical optimization problem, where the objective is to minimize some energy function. The constraints are in place to ensure that all tasks will meet their deadlines, and that only one processor is assigned to a task. In a partitioned scheduling algorithm, once a task is assigned to a specific processor, the multiprocessor scheduling problem is reduced to a set of uniprocessor scheduling problems, which has already been well studied. However, the partitioned scheduling may not provide a feasibility optimal schedule.

For heterogeneous multiprocessor systems, not only the different operating frequency sets among processors should be considered, but also the hardware architecture of the processors, since task execution time will be different for each processor type. In other words, the system has to be captured by two aspects: the difference in operating speed sets and the minimum execution cycles required by different tasks on different processor types.

With these aspects, fully-migration/global based scheduling algorithms, where tasks are allowed to migrate between different processor types, are not applicable in practice, since it will be difficult to identify how much computational work is executed on one processor type compared to another processor type due to differences in instruction sets, register formats, etc. Thus, most of the work related to heterogeneous multiprocessor scheduling are partitioned-based/non-preemptive task scheduling algorithms [62–68], i.e. tasks are partitioned onto one of the processor types and a homogeneous multiprocessor scheduling algorithm is used to find a valid schedule. With this scheme, the heterogeneous multiprocessor scheduling problem is reduced to a task partitioning problem, which can be formulated as an integer linear program (ILP). Examples of such work are [62, 66].

However, with the advent of ARM two-type heterogeneous multi-core architecture, such as the big.LITTLE architecture [69], that supports task migrations among different core types, a global scheduling algorithm is possible. Therefore, in this work, we are interested in solving a scheduling problem of this special architecture. The first energy-aware global scheduling framework for a two-type heterogeneous multi-core platform is presented in [70, 71], where an algorithm called Hetero-Split is proposed to solve a workload assignment and a Hetero-Wrap algorithm to solve a schedule generation problem. Their framework is similar to ours, except that we adopt a fluid model to represent a scheduling dynamic, our assigned operating frequency is time-varying and the CPU idle energy consumption is also considered.

2.2 Feasibility Problem Formulation

Though our objective is to minimize the total energy consumption, we will first consider a feasibility problem before presenting an optimality problem.

2.2.1 System Model

For this work, we consider a set of n real-time tasks that are to be partitioned on a two-type heterogeneous multiprocessor system composed of m_r processors of type- r , $r \in R$, $R = \{1, 2\}$. Note that the problem has been restricted to a two-type heterogeneous multiprocessor, rather than a generalised r -type heterogeneous multiprocessor. This is because there exists no known solution to the task ordering problem of a generalised heterogeneous multiprocessor system, which is a necessary condition to solve the relaxed scheduling problem proposed in Section 2.3.2. However, our proposed workload partitioning algorithms can be easily extended to a generalised r -type heterogeneous multiprocessor system. We will assume that the system supports task migration among processor types, e.g. sharing the same instruction set and having a special interconnection for data transfer between processor types. Note that c_i is the same for all processor types, since the instruction set is the same. Moreover, as a homogeneous multiprocessor system is a special case of a heterogeneous multiprocessor system, a homogeneous multiprocessor system problem formulation can be obtained by dropping the superscript/subscript r where appropriate. Therefore, for the rest of the thesis, we will consider a scheduling problem as a two-type heterogeneous multiprocessor scheduling problem unless stated otherwise.

2.2.2 Task/Processor Assumptions

All tasks do not share resources, do not have any precedence constraints and are ready to start at the beginning of the execution. A task can be preempted/migrated between different processor types at any time. The cost of preemption and migration is assumed to be negligible or included in the minimum task execution times. Processors of the same type are homogeneous, i.e. having the same set of operating frequencies and power consumptions. Each processor's voltage/speed can be adjusted individually.

2.2.3 Scheduling as a Continuous Optimal Control Problem (COCP)

Below, we will refer to the sets $I := \{1, \dots, n\}$, $K^r := \{1, \dots, m_r\}$ and $\Gamma := [0, L]$, where L is the largest deadline of all tasks. Note that $\forall i, \forall k, \forall r, \forall t$ are short-hand

notations for $\forall i \in I, \forall k \in K^r, \forall r \in R, \forall t \in \Gamma$, respectively. The scheduling problem can therefore be formulated as the following infinite-dimensional continuous-time feasibility control problem:

find $x_i(\cdot), a_{ik}^r(\cdot), s_k^r(\cdot), \forall i \in I, k \in K^r, r \in R$

subject to

$$x_i(b_i) = \underline{x}_i, \quad \forall i \quad (2.5a)$$

$$x_i(t) = 0, \quad \forall i, t \notin [b_i, b_i + d_i) \quad (2.5b)$$

$$\dot{x}_i(t) \geq - \sum_{r=1}^{\kappa} \sum_{k=1}^{m_r} a_{ik}^r(t) s_k^r(t), \quad \forall i, t, \quad \text{a.e.} \quad (2.5c)$$

$$\sum_{r=1}^{\kappa} \sum_{k=1}^{m_r} a_{ik}^r(t) \leq 1, \quad \forall i, t \quad (2.5d)$$

$$\sum_{i=1}^n a_{ik}^r(t) \leq 1, \quad \forall k, r, t \quad (2.5e)$$

$$s_k^r(t) \in S^r, \quad \forall k, r, t \quad (2.5f)$$

$$a_{ik}^r(t) \in \{0, 1\}, \quad \forall i, k, r, t \quad (2.5g)$$

$$a_{ik}^r(\cdot) \in \mathcal{PC}, s_k^r(\cdot) \in \mathcal{PC}, \quad \forall i, k, r, t \quad (2.5h)$$

where the state $x_i(t)$ is the remaining minimum execution time of task T_i at time t , the control input $s_k^r(t)$ is the execution speed of the k^{th} processor of type- r at time t and the control input $a_{ik}^r(t)$ is used to indicate the processor assignment of task T_i at time t , i.e. $a_{ik}^r(t) = 1$ if and only if task T_i is active on processor k of type- r . Notice that here we formulated the problem with speed selection at a core-level; a stricter assumption of a multi-core architecture, i.e. a cluster-level speed assignment, is straightforward. Particularly, by replacing a core-level speed assignment s_k^r with a cluster-level speed assignment s^r in the above formulation.

The initial conditions on the minimum execution time of all tasks and task deadline constraints are specified in (2.5a) and (2.5b), respectively. The fluid model of the scheduling dynamic is given by the differential constraint (2.5c). Constraint (2.5d) ensures that each task will be assigned to at most one non-idle processor at a time. Constraint (2.5e) guarantees that each non-idle processor will only be assigned to at most one task at a time. The speeds are constrained by (2.5f) to take on values from $S^r \subseteq [0, 1]$. Constraint (2.5g) emphasizes that task assignment variables are binary. Lastly, (2.5h) denotes that the control inputs should be step functions.

Fact 2.1. A solution to (2.5) where (2.5c) is satisfied with equality can be constructed from a solution to (2.5).

Proof. Let (a, s, x) be a feasible point to (2.5). Let $t_i := \min\{t \in [b_i, b_i + d_i] \mid x_i(t) \leq 0\}$, $\forall i$. Choose $(\tilde{a}, \tilde{s}, \tilde{x})$ such that (i) $\tilde{a}_{ik}^r(t)\tilde{s}_k^r(t) = a_{ik}^r(t)s_k^r(t)$, $\forall i, k, r, t \leq t_i$ and (ii) $\tilde{a}_{ik}^r(t)\tilde{s}_k^r(t) = 0$, $\forall i, k, r, t > t_i$. Choose $\tilde{x}_i(0) = \underline{x}_i$, $\forall i$ and $\dot{\tilde{x}}_i(t) = -\sum_{r=1}^K \sum_{k=1}^{m_r} \tilde{a}_{ik}^r(t)\tilde{s}_k^r(t)$, $\forall i, k, r, t$. It follows that $(\tilde{a}, \tilde{s}, \tilde{x})$ is a solution to (2.5) where (2.5c) is an equality. $\square$

2.3 Solving the Scheduling Problem with Finite-dimensional Mathematical Optimization

The original problem (2.5) will be discretized by introducing piecewise constant constraints on the control inputs s and a . Let $\mathcal{T} := \{\tau_0, \tau_1, \dots, \tau_N\}$, which we will refer to as the major grid, denote the set of discretization time steps corresponding to the distinct arrival times and deadlines of all tasks within L , where $0 = \tau_0 < \tau_1 < \tau_2 < \dots < \tau_N = L$.

2.3.1 Mixed-Integer Nonlinear Program (MINLP-DVFS)

The above scheduling problem, subject to piecewise constant constraints on the control inputs, can be naturally formulated as an MINLP, defined below. Since the context switches due to task preemption and migration can jeopardize the performance, a variable discretization time step [72] method is applied on a minor grid, so that the solution to our scheduling problem does not depend on the size of the discretization time step. Let $\{\tau_{\mu,0}, \dots, \tau_{\mu,M}\}$ denote the set of discretization time steps on a minor grid on the interval $[\tau_\mu, \tau_{\mu+1}]$ with $\tau_\mu = \tau_{\mu,0} \leq \dots \leq \tau_{\mu,M} = \tau_{\mu+1}$, so that $\{\tau_{\mu,1}, \dots, \tau_{\mu,M-1}\}$ is to be determined for all μ from solving an appropriately-defined optimization problem.

Let $\forall \mu$ and $\forall \nu$ be short notations for $\forall \mu \in U := \{0, 1, \dots, N-1\}$ and $\forall \nu \in V := \{0, 1, \dots, M-1\}$. Define the notation $[\mu, \nu] := (\tau_{\mu,\nu})$, $\forall \mu, \nu$. Denote the discretized state and input sequences as

$$x_i[\mu, \nu] := x_i(\tau_{\mu,\nu}), \quad \forall i, \mu, \nu \quad (2.6a)$$

$$s_k^r[\mu, \nu] := s_k^r(\tau_{\mu,\nu}), \quad \forall k, r, \mu, \nu \quad (2.6b)$$

$$a_{ik}^r[\mu, \nu] := a_{ik}^r(\tau_{\mu,\nu}), \quad \forall i, k, r, \mu, \nu \quad (2.6c)$$

Let $s_k^r(\cdot)$ and $a_{ik}^r(\cdot)$ be step functions inbetween time instances on a minor grid, i.e.

$$s_k^r(t) = s_k^r[\mu, \nu], \quad \forall t \in [\tau_{\mu, \nu}, \tau_{\mu, \nu+1}), \mu, \nu \quad (2.7a)$$

$$a_{ik}^r(t) = a_{ik}^r[\mu, \nu], \quad \forall t \in [\tau_{\mu, \nu}, \tau_{\mu, \nu+1}), \mu, \nu. \quad (2.7b)$$

Let Λ denote the set of all tasks within L , i.e. $\Lambda := \{T_i \mid i \in I\}$. Define a task arrival time mapping $\Phi_b : \Lambda \rightarrow U$ by $\Phi_b(T_i) := \mu$ such that $\tau_\mu = b_i$ for all $T_i \in \Lambda$ and a task deadline mapping $\Phi_d : \Lambda \rightarrow U \cup \{N\}$ by $\Phi_d(T_i) := \mu$ such that $\tau_\mu = b_i + d_i$ for all $T_i \in \Lambda$. Define $\mathcal{U}_i := \{\mu \in U \mid \Phi_b(T_i) \leq \mu < \Phi_d(T_i)\}$, $\forall i \in I$ and let $\forall \mu_i$ be short notation for $\forall \mu \in \mathcal{U}_i$.

By solving a first-order ODE with piecewise constant input, a solution of the scheduling dynamic (2.5c) has to satisfy the difference constraint

$$x_i[\mu, \nu + 1] \geq x_i[\mu, \nu] - h[\mu, \nu] \sum_{r=1}^{\kappa} \sum_{k=1}^{m_r} s_k^r[\mu, \nu] a_{ik}^r[\mu, \nu], \forall i, \mu_i, \nu, \quad (2.8a)$$

where $h[\mu, \nu] := \tau_{\mu, \nu+1} - \tau_{\mu, \nu}, \forall \mu, \nu$.

The discretization of the original problem (2.5) subject to piecewise constant constraints on the inputs (2.7) is therefore equivalent to the following finite-dimensional MINLP:

find $x_i[\cdot], a_{ik}^r[\cdot], s_k^r[\cdot], h[\cdot], \forall i \in I, k \in K^r, r \in R$

subject to (2.8a) and

$$x_i[\Phi_b(T_i), 0] = \underline{x}_i, \quad \forall i \quad (2.8b)$$

$$x_i[\mu, \nu] = 0, \quad \forall i, \mu \notin \mathcal{U}_i, \nu \quad (2.8c)$$

$$\sum_{r=1}^{\kappa} \sum_{k=1}^{m_r} a_{ik}^r[\mu, \nu] \leq 1, \quad \forall i, \mu, \nu \quad (2.8d)$$

$$\sum_{i=1}^n a_{ik}^r[\mu, \nu] \leq 1, \quad \forall k, r, \mu, \nu \quad (2.8e)$$

$$s_k^r[\mu, \nu] \in S^r, \quad \forall k, r, \mu, \nu \quad (2.8f)$$

$$a_{ik}^r[\mu, \nu] \in \{0, 1\}, \quad \forall i, k, r, \mu, \nu \quad (2.8g)$$

$$0 \leq h[\mu, \nu], \quad \forall \mu, \nu \quad (2.8h)$$

$$\sum_{\nu=0}^{M-1} h[\mu, \nu] \leq \tau_{\mu+1} - \tau_\mu, \quad \forall \mu \quad (2.8i)$$

where (2.8h)-(2.8i) enforce upper and lower bounds on discretization time steps.

Theorem 2.1. *Let the size of the minor grid $M \geq \max_r \{m_r\} - 1$. A solution to (2.5) exists if and only if a solution to (2.8) exists.*

Proof. Follows from the fact that if a solution exists to (2.5), then the Hetero-Wrap scheduling algorithm [71] can find a valid schedule with at most $m_r - 1$ migrations within the cluster- r . [71, Lemma 2].

Next, we will show that $(\tilde{a}[\cdot], \tilde{s}[\cdot], \tilde{x}[\cdot], \tilde{h}[\cdot])$, a solution to (2.8), can be constructed from $(a(\cdot), s(\cdot), x(\cdot))$, a solution to (2.5). Specifically, choose $\tilde{h}[\mu, \nu] = \tau_{\mu, \nu+1} - \tau_{\mu, \nu}$ as above and $\tilde{a}_{ik}^r[\mu, \nu] = a_{ik}^r(t)$ such that $a_{ik}^r(t)$ is a constant for all $t \in \tilde{h}[\mu, \nu]$ and

$$\tilde{h}[\mu, \nu] \tilde{a}_{ik}^r[\mu, \nu] = \int_{\tau_{\mu, \nu}}^{\tau_{\mu, \nu+1}} a_{ik}^r(t) dt, \quad \forall i, r, \mu, \nu. \quad (2.9)$$

Then (2.8a)-(2.8c) are satisfied with $\tilde{x}_i[\mu, \nu] = x_i(\tau_{\mu, \nu})$, $\forall i, \mu, \nu$. It follows from (2.5d), (2.5e) and (2.5g) that (2.8d), (2.8e) and (2.8g) are satisfied, respectively. (2.8f) is satisfied with $\tilde{s}_k^r[\mu, \nu] = s_k^r(\tau_{\mu, \nu})$.

Suppose now we have $(\tilde{a}[\cdot], \tilde{s}[\cdot], \tilde{x}[\cdot], \tilde{h}[\cdot])$, a solution to (2.8). We can choose $(a(\cdot), s(\cdot), x(\cdot))$ to be a solution to (2.5) if the inputs are the step functions $a_{ik}^r(t) = \tilde{a}_{ik}^r[\mu, \nu]$ and $s_k^r(t) = \tilde{s}_k^r[\mu, \nu]$ when $\tilde{h}[\mu, \nu] \leq t - \tau_{\mu, \nu} < \tilde{h}[\mu, \nu + 1]$, $\forall i, k, r, \mu, \nu$. It is simple to verify that (2.5) is satisfied by the above choice. $\square$

2.3.2 Computationally Tractable Multiprocessor Scheduling Algorithms

The time to compute a solution to problem (2.8) is impractical even with a small problem size. However, if we relax the binary constraints in (2.8g) so that the value of a can be interpreted as the percentage of a time interval during which the task is executed (this will be denoted as ω in later formulations), rather than the processor assignment, the problem can be reformulated as an NLP for a system with continuous operating speed and an LP for a system with discrete speed levels. The NLP and LP can be solved at a percentage of the time taken to solve the MINLP above. Particularly, the heterogeneous multiprocessor scheduling problem can be simplified into two steps:

STEP 1: Workload Partitioning

Determine the percentage of task execution times and execution speed within a time interval such that the feasibility constraints are satisfied.

STEP 2: Task Ordering

From the solution given in the workload partitioning step, find the execution order of all tasks within a time interval such that no task will be executed on more than one processor at a time.

2.3.2.1 Solving the Workload Partitioning as a Continuous Nonlinear Program (NLP-DVFS)

Since knowing the processor on which a task will be executed does not help in finding the task execution order, the corresponding processor assignment subscript k of the control variables ω and s is dropped to reduce the number of decision variables. Moreover, partitioning time using only a major grid (i.e. $M = 1$) is enough to guarantee a valid solution, i.e. the percentage of the task execution time within a major grid is equal to the sum of all percentages of task execution times in a minor grid. Since we only need a major grid, we define the notation $[\mu] := \tau_\mu$ and $h[\mu] := \tau_{\mu+1} - \tau_\mu$. Note that we make an assumption that $h[\mu] > 0, \forall \mu$. We also assume that the set of allowable speed levels S^r is a closed interval given by the lower bound s_{min}^r and upper bound s_{max}^r .

Consider now the following finite-dimensional NLP:

$$\text{find } x_i[\cdot], \omega_i^r[\cdot], s_i^r[\cdot], \forall i \in I, r \in R$$

subject to

$$x_i[\Phi_b(T_i)] = \underline{x}_i, \quad \forall i \quad (2.10a)$$

$$x_i[\mu] = 0, \quad \forall i, \mu \notin \mathcal{U}_i \quad (2.10b)$$

$$x_i[\mu + 1] \geq x_i[\mu, \nu] - h[\mu] \sum_{r=1}^K \omega_i^r[\mu] s_i^r[\mu], \quad \forall i, \mu \quad (2.10c)$$

$$\sum_{r=1}^K \omega_i^r[\mu] \leq 1, \quad \forall i, \mu \quad (2.10d)$$

$$\sum_{i=1}^n \omega_i^r[\mu] \leq m_r, \quad \forall r, \mu \quad (2.10e)$$

$$s_{min}^r \leq s_i^r[\mu] \leq s_{max}^r, \quad \forall i, r, \mu \quad (2.10f)$$

$$0 \leq \omega_i^r[\mu] \leq 1, \quad \forall i, r, \mu \quad (2.10g)$$

where $\omega_i^r[\mu]$ is defined as the percentage of the time interval $[\tau_\mu, \tau_{\mu+1}]$ for which task T_i is executing on a processor of type- r at speed $s_i^r[\mu]$. (2.10d) guarantees that a task will not run on more than one processor at a time. The constraint that the total workload at each time interval should be less than or equal to the system capacity is specified in (2.10e). Upper and lower bounds on task execution speed and percentage of task execution time are given in (2.10f) and (2.10g), respectively.

2.3.2.2 Solving the Workload Partitioning as a Continuous Linear Program (LP-DVFS)

The problem (2.10) can be further simplified to an LP if the set of speed levels S^r is finite, as is often the case for practical systems. We denote with s_q^r the execution speed at level $q \in Q^r := \{1, \dots, l_r\}$ of an r -type processor, where l_r is the total number of speed levels of an r -type processor. Let $\forall q$ be short-hand for $\forall q \in Q^r$.

Consider now the following finite-dimensional LP:

find $x_i[\cdot], \omega_{iq}^r[\cdot], \forall i \in I, q \in Q^r, r \in R$

subject to

$$x_i[\Phi_b(T_i)] = \underline{x}_i, \quad \forall i \quad (2.11a)$$

$$x_i[\mu] = 0, \quad \forall i, \mu \notin \mathcal{U}_i \quad (2.11b)$$

$$x_i[\mu + 1] \geq x_i[\mu] - h[\mu] \sum_{r=1}^{\kappa} \sum_{q=1}^{l_r} \omega_{iq}^r[\mu] s_q^r, \quad \forall i, \mu \quad (2.11c)$$

$$\sum_{r=1}^{\kappa} \sum_{q=1}^{l_r} \omega_{iq}^r[\mu] \leq 1, \quad \forall i, \mu \quad (2.11d)$$

$$\sum_{i=1}^n \sum_{q=1}^{l_r} \omega_{iq}^r[\mu] \leq m_r, \quad \forall r, \mu \quad (2.11e)$$

$$0 \leq \omega_{iq}^r[\mu] \leq 1, \quad \forall i, q, r, \mu \quad (2.11f)$$

where $\omega_{iq}^r[\mu]$ is the percentage of the time interval $[\tau_\mu, \tau_{\mu+1}]$ for which task T_i is executing on a processor of type- r at a speed level q . Note that all constraints are similar to (2.10), but the speed levels are fixed.

Theorem 2.2. *A solution to (2.10) can be constructed from a solution to (2.11), and vice versa, if the discrete speed set S^r is any finite subset of the closed interval $[s_{min}^r, s_{max}^r]$ with s_{min}^r and s_{max}^r in S^r for all r .*

Proof. Let $(\tilde{x}, \tilde{\omega}, \tilde{s})$ denote a solution to (2.10) and (x, ω) a solution to (2.11). The result follows by noting that one can choose $\lambda_q^r[\mu] \in [0, 1]$ such that $\sum_q \lambda_q^r[\mu] s_q^r[\mu] = \tilde{s}_i^r[\mu]$, $\omega_{iq}^r[\mu] = \lambda_q^r[\mu] \tilde{\omega}_i^r[\mu]$ and $\sum_q \lambda_q^r[\mu] = 1$, $\forall i, q, r, \mu$ are satisfied. $\square$

2.3.2.3 Task Ordering Algorithms

This section discusses how to find a valid schedule in the task ordering step for each time interval $[\tau_\mu, \tau_{\mu+1}]$. For homogeneous multiprocessor systems, the execution order of all active tasks within each time interval given a solution to (2.10) or (2.11), can be

Algorithm 1 McNaughton's wrap around algorithm [55]

```

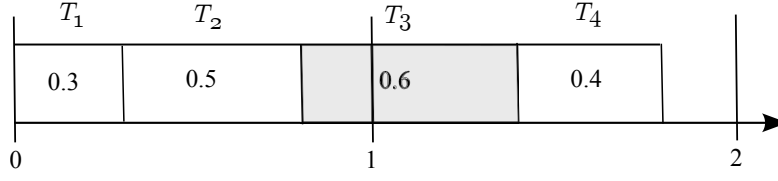
1: INPUTS:  $\omega_i, \forall i$ 
2:  $\sigma_{ik} \leftarrow 0, \eta_{ik} \leftarrow 0, \forall i, k$ 
3: for all  $i$  do
4:   if  $i = 1$  then
5:      $\sigma_{i1} \leftarrow 0, \eta_{i1} \leftarrow \omega_i$ 
6:      $k \leftarrow 1$ 
7:   else
8:     if  $\eta_{(i-1)k} + \omega_i \leq k$  then
9:        $\sigma_{ik} \leftarrow \eta_{(i-1)k}$ 
10:       $\eta_{ik} \leftarrow \sigma_{ik} + \omega_i$ 
11:    else
12:       $\sigma_{ik} \leftarrow \eta_{(i-1)k}$ 
13:       $\eta_{i(k+1)} \leftarrow \omega_i - (\eta_{ik} - \sigma_{ik})$ 
14:       $k \leftarrow k + 1$ 
15:    end if
16:  end if
17: end for
18: RETURN:  $(\sigma_{ik}, \eta_{ik}) \in [0, 1]^2, \forall i, k$ 

```

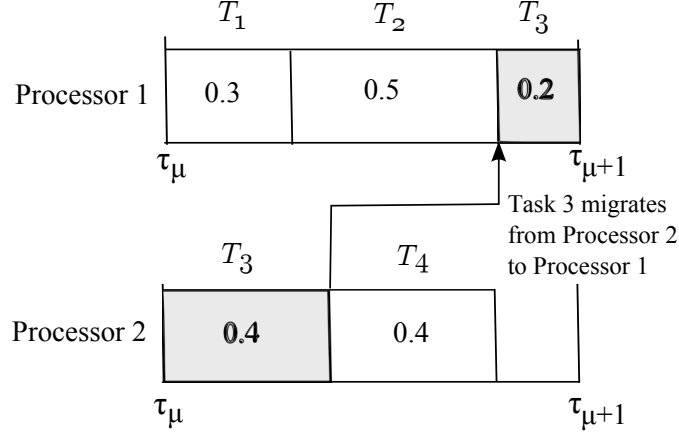
obtained by McNaughton's wrap around algorithm [55] shown in Algorithm 1. In the case of a practical system (2.11), we define $\omega_i := \sum_q \omega_{iq}$ as the total execution workload of task T_i and $\omega_{ik} := \eta_{ik} - \sigma_{ik}$ as the partitioned execution times of task T_i on processor k , where (σ_{ik}, η_{ik}) is a workload interval assignment of task i on processor k . Note that for each task, ω_{iq} should be executed next to each other in order to minimize the number of migrations and preemptions. Specifically, the workload partitions (percentages of execution times of the tasks ω_i) are aligned along the number line starting at zero in any arbitrary order, then they are split into chunks of length equal to 1. Each chunk is assigned to one processor. There will be at most $m - 1$ tasks that needs to be split/assigned to two processors at each time interval [54, Thm 6].

Consider a simple taskset composed of four tasks to be scheduled on two homogeneous processors. Suppose the percentages of execution times within the time interval $[\tau_{\mu, \nu}, \tau_{\mu, \nu+1}]$ obtained by solving (2.10) are $\omega_1 = 0.3$, $\omega_2 = 0.5$, $\omega_3 = 0.6$ and $\omega_4 = 0.4$. Figure 2.5 shows a feasible schedule of the taskset using McNaughton's wrap around algorithm.

For a heterogeneous multiprocessor system, one might also think of using McNaughton's wrap around algorithm to find a valid schedule for each processor within the processor type. However, McNaughton's wrap around algorithm only guarantees that a task will not be executed at the same time within the cluster. There exists a possibility that a task will be assigned to more than one processor type (cluster) at the same time.



(a) Task execution workloads are aligned along a number line.



(b) Each chunk of length 1 are assigned to each processor.

Figure 2.5: Feasible schedule at time interval $[\tau_\mu, \tau_{\mu+1}]$ obtained by McNaughton's wrap around algorithm

To avoid a parallel execution on any two clusters, we can adopt the Hetero-Wrap algorithm proposed in [71] to solve a task ordering problem of a two-type heterogeneous multiprocessor platform. The algorithm takes the workload partitioning solution to STEP 1 as its inputs and returns $(\sigma_{ik}^r, \eta_{ik}^r) \in [0, 1]^2, \forall i, k, r$, which is a task-to-processor interval assignment on each cluster. Once again, for a solution to problem (2.11), we define the total execution workload of a task on cluster- r $\omega_i^r := \sum_q \omega_{iq}^r$ and assume that the percentage of execution times of each task at all frequency levels ω_{iq}^r will be grouped together in order to minimize the number of migrations and preemptions. Details on the Hetero-Wrap algorithm are given in Algorithm 2. Specifically, the algorithm classified the tasks into four subsets: (i) IM_a a set of migrating tasks with $\sum_r \omega_i^r = 1$, (ii) IM_b a set of migrating tasks with $\sum_r \omega_i^r < 1$, (iii) CP_1 a set of partitioned tasks on cluster of type-1, and (iv) CP_2 a set of partitioned tasks on cluster of type-2. Then the algorithm employs the following simple rules:

- For a type-1 cluster, tasks are scheduled in the order of IM_a , IM_b and CP_1 using McNaughton's wrap around algorithm. That is, a slot along the number line is allocated, starting at zero, with the length equal to m_1 and the task is aligned with its assigned workload on empty slots of the cluster in the specified order starting from left to right.

Algorithm 2 Hetero-Wrap algorithm [71]

```

1: INPUT:  $\omega_i^r, m_r, \forall i, r$ 
2:  $\sigma_{ik}^r \leftarrow 0, \eta_{ik}^r \leftarrow 1, \forall i, k, r$ 
3:  $p_1 \leftarrow 0, p_2 \leftarrow m_2$ 
4:  $k_1 \leftarrow 1, k_2 \leftarrow m_2$ 
5: for  $r = 1, 2$  do
6:   if  $r = 1$  then
7:     for  $i \in \{IM_a, IM_b, CP_1\}$  do
8:       if  $p_1 = 0$  then
9:          $\eta_{ik_1}^r \leftarrow w_i^r$ 
10:         $p_1 \leftarrow \eta_{ik_1}^r$ 
11:       else
12:         if  $p_1 + w_i^r \leq k_1$  then
13:            $\sigma_{ik_1}^r \leftarrow p_1 - (k_1 - 1)$ 
14:            $\eta_{ik_1}^r \leftarrow p_1 + w_i^r - (k_1 - 1)$ 
15:            $p_1 \leftarrow p_1 + w_i^r$ 
16:         else
17:            $\sigma_{ik_1}^r \leftarrow p_1 - (k_1 - 1)$ 
18:            $\eta_{i(k_1+1)}^r \leftarrow p_1 + w_i^r - k_1, k_1 \leftarrow k_1 + 1$ 
19:         end if
20:       end if
21:     end for
22:   else
23:     for  $i \in \{IM_a, IM_b, CP_2\}$  do
24:       if  $p_2 = m_2$  then
25:          $\sigma_{ik_2}^r \leftarrow 1 - \omega_i^r, p_2 \leftarrow \sigma_{ik_2}^r$ 
26:       else
27:         if  $p_2 - \omega_i^r \geq k_2 - 1$  then
28:            $\sigma_{ik_2}^r \leftarrow p_2 - \omega_i^r - (k_2 - 1)$ 
29:            $\eta_{ik_2}^r \leftarrow p_2 - (k_2 - 1)$ 
30:            $p_2 \leftarrow p_2 - \omega_i^r$ 
31:         else
32:            $\eta_{ik_2}^r \leftarrow p_2 - (k_2 - 1)$ 
33:            $k_2 \leftarrow k_2 - 1$ 
34:            $\sigma_{ik_2}^r \leftarrow p_2 - \omega_i^r - k_2 + 1$ 
35:         end if
36:       end if
37:     end for
38:   end if
39: end for
40: RETURN:  $(\sigma_{ik}^r, \eta_{ik}^r) \in [0, 1]^2, \forall i, k, r$ 

```

- For a type-2 cluster, in the same manner, tasks are scheduled using McNaughton's wrap around algorithm, but in the order of IM_a , IM_b and CP_2 starting from right to left. Note that the order of tasks in IM_a has to be consistent with the order in a type-1 cluster.

However, the algorithm requires a feasible solution to (2.10) or (2.11), in which IM_b has at most one task, which we will call an inter-cluster migrating task. From Theorem 2.2, we can always transform a solution to (2.10) into a solution to (2.11). Therefore, we only need to show that there exists a solution to (2.11) with at most one inter-cluster migrating task that lies on the vertex of the feasible region by the following facts and lemma.

Fact 2.2. *Among all the solutions to an LP, at least one solution lies at a vertex of the feasible region. In other words, at least one solution is a basic solution.*

Proof. The Fundamental Theorem of Linear Programming, which states that if a feasible solution exists, then a basic feasible solution exists [73, p.38]. $\square$

Fact 2.3. *A feasible solution to an LP that is not a basic solution can always be converted into a basic solution.*

Proof. This follows from the Fundamental Theorem of Linear Programming [73, p.38]. $\square$

Fact 2.4. [74, Fact 2] *Consider a linear program $\{\chi \in \mathbb{R}^n \mid A\chi \leq b\}$ for some $A \in \mathbb{R}^{(m+n) \times n}$, $b \in \mathbb{R}^{(m+n)}$. Suppose that n constraints are nonnegative constraints on each variable, i.e. $\chi_i \geq 0$, $\forall i \in \{1, 2, \dots, n\}$ and the rest are m linearly independent constraints. If $m < n$, then a basic solution will have at most m non-zero values.*

Proof. A unique basic solution can be identified by any $n+m$ linearly independent active constraints. Since there are n nonnegative constraints and $m < n$, a basic solution will have at most m non-zero values. $\square$

Lemma 2.1. *For a solution to (2.11) that lies on the vertex of the feasible region, there will be at most one inter-cluster partitioning task.*

Proof. The number of variables ω subjected to nonnegative constraint (2.11f) at each time interval of (2.11) is $n(\sum_r l_r)$. The number of variables ω subjected to a set of necessary and sufficient feasibility constraints (2.11d)-(2.11e) is $n + 2$. Note that we do not count the number of variables in (2.11c) because (2.11c) and (2.11d) are linearly dependent constraints for a given value of $\xi_i[\mu] := (x_i[\mu] - x_i[\mu + 1])/h[\mu]$. If we

Table 2.1: Execution Workload Partition Example

	T_1	T_2	T_3	T_4	T_5
ω_i^1	0.3	0.6	0.2	0.5	0
ω_i^2	0.7	0.4	0.4	0	0.5

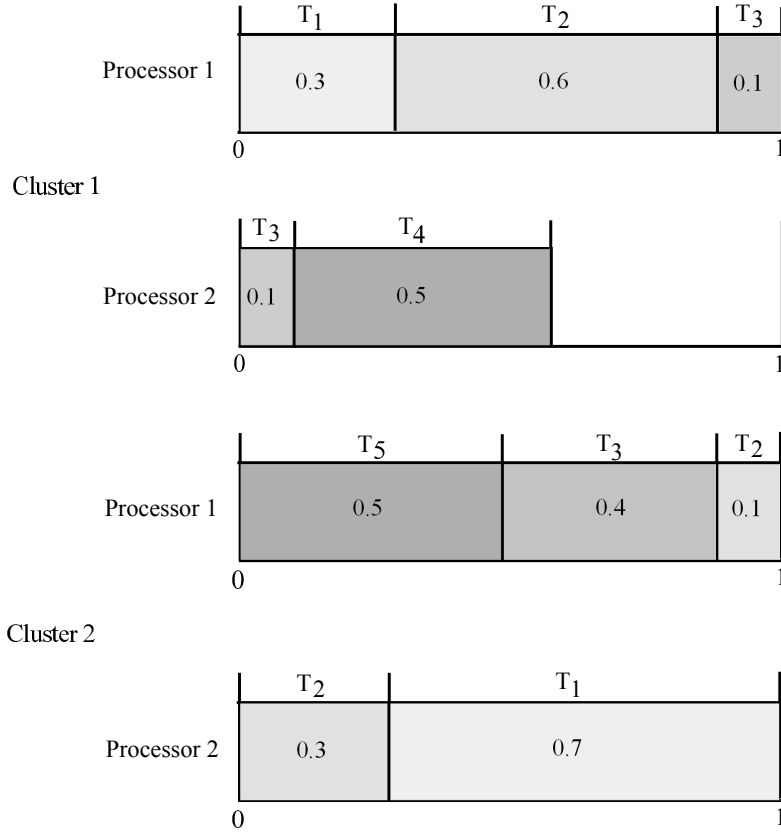


Figure 2.6: A feasible task schedule according to Algorithm 2.

assume that $n \geq 2$ and each processor type has at least one speed level, then it follows from Fact 2.4 that the number of non-zero values of variable ω , a solution to (2.11) at the vertex of the feasible region, is at most $n + 2$. Let γ be the number of tasks assigned to two processor types. Therefore, there are $2\gamma + (n - \gamma)$ entries of variable ω that are non-zero. This implies that $\gamma < 2$, i.e. the number of inter-cluster partitioning tasks is at most one. $\square$

To illustrate how Algorithm 2 works, consider a simple taskset in which the percentage of execution workload partition at time interval $[\tau_{\mu,\nu}, \tau_{\mu,\nu+1}]$ for each task is as shown in Table 2.1. A feasible schedule obtained by Algorithm 2 is shown in Figure 2.6. For this example, $m_1 = m_2 = 2$, $IM_a = \{T_1, T_2\}$, $IM_b = \{T_3\}$, $CP_1 = \{T_4\}$ and $CP_2 = \{T_5\}$.

Theorem 2.3. *If a solution to (2.5) exists, then a solution to (2.10)/(2.11) exists. Furthermore, at least one valid schedule satisfying (2.5) can be constructed from a solution to problem (2.10)/(2.11) and the output from Algorithm 2.*

Proof. The existence of a valid schedule is proven in [71, Thm 3]. It follows from Facts 2.2–2.4 and Lemma 2.1 that one can compute a solution with at most one inter-cluster partitioning task. Given a solution to (2.10)/(2.11) and the output from Algorithm 2 for all intervals, choose a to be a step function such that $a_{ik}^r(t) = 1$ when $\sigma_{ik}^r[\mu, \nu]h[\mu, \nu] \leq t - \tau_{\mu, \nu} < \eta_{ik}^r[\mu, \nu]h[\mu, \nu + 1]$ and $a_{ik}^r(t) = 0$ otherwise, $\forall i, k, r, \mu, \nu$. Specifically, one can verify that the following condition holds

$$h[\mu, \nu]\omega_i^r[\mu] = \int_{\tau_{\mu, \nu}}^{\tau_{\mu, \nu+1}} \sum_k a_{ik}^r(t) dt, \quad \forall i, r, \mu, \nu. \quad (2.12)$$

Then it is straightforward to show that (2.5) is satisfied. $\square$

Note that, although, we need to solve the same multiprocessor scheduling problem with two steps in this section, the computation times to solve (2.10) or (2.11) is extremely fast compared with problem (2.5), i.e. even for a small problem size, the times to compute a solution of (2.8) can be up to an hour, while (2.10) or (2.11) can be solved in milliseconds using a general-purpose desktop PC with off-the-shelf optimization solvers. Furthermore, the time complexity of Algorithms 2 is $\mathcal{O}(n)$ [71].

2.4 Energy Optimality Problem Formulation

The scheduling problem with the objective to minimize the total energy consumption of executing the taskset on a two-type heterogeneous multiprocessor system can be formulated as the following optimal control problems:

I) COCP:

$$\begin{aligned} & \underset{\substack{x_i(\cdot), a_{ik}^r(\cdot), s_k^r(\cdot), \\ \forall i \in I, k \in K^r, r \in R}}{\text{minimize}} & \sum_{r, k, i} \int_0^L \ell^r(a_{ik}^r(t), s_k^r(t)) dt & (2.13) \\ & \text{subject to (2.5).} \end{aligned}$$

II) MINLP-DVFS:

$$\begin{aligned} & \underset{\substack{x_i[\cdot], a_{ik}^r[\cdot], s_k^r[\cdot], h[\cdot], \\ \forall i \in I, k \in K^r, r \in R}}{\text{minimize}} & \sum_{r, \mu, \nu, k, i} h[\mu, \nu] \ell^r(a_{ik}^r[\cdot], s_k^r[\cdot]) & (2.14) \\ & \text{subject to (2.8).} \end{aligned}$$

III) NLP-DVFS:

$$\begin{aligned} & \underset{\substack{x_i[\cdot], \omega_i^r[\cdot], s_i^r[\cdot], \\ \forall i \in I, r \in R}}{\text{minimize}} & \sum_{r, \mu, v, i} h[\mu] \ell^r(\omega_i^r[\cdot], s_i^r[\cdot]) \\ & \text{subject to (2.10).} \end{aligned} \quad (2.15)$$

IV) LP-DVFS :

$$\begin{aligned} & \underset{\substack{x_i[\cdot], \omega_{iq}^r[\cdot], \\ \forall i \in I, q \in Q^r, r \in R}}{\text{minimize}} & \sum_{r, \mu, v, i, q} h[\mu] \ell^r(\omega_{iq}^r[\cdot], s_q^r) \\ & \text{subject to (2.11).} \end{aligned} \quad (2.16)$$

where $\ell^r(a, s) := a(P_{active}^r(s) - P_{idle}^r)$. Note that (2.16) is an LP, since the cost is linear in the decision variables.

2.4.1 Optimal Speed Profile

In this section, we present a proof of a general speed selection solution to a uniprocessor scheduling problem with real-time taskset. With this observation about an optimal speed profile, we can formulate an algorithm that is able to solve a broader class of scheduling problems, i.e. more general and robust to various taskset characteristics and power consumption models.

Consider the following simple example, illustrated in Figure 2.7, where the power consumption model $P(\cdot)$ is a concave function of speed. Assume that s_2 is the lowest possible constant speed at which a task T_i can be finished on time, i.e. $\underline{x}_i = s_2 d_i$. The energy consumed is $E(s_2) = P(s_2) d_i$ and the average power consumption $P(s_2) =: \bar{P}_c$. Let $\lambda \in [0, 1]$ be a constant such that $s_2 = \lambda s_1 + (1 - \lambda) s_3$, $s_1 < s_2 < s_3$. Suppose $s(\cdot)$ is a time-varying speed profile such that $s(t) = s_1, \forall t \in [0, t_1]$ and $s(t) = s_3, \forall t \in [t_1, d_i]$. We can choose t_1 such that $\underline{x}_i = s_1 t_1 + s_3 (d_i - t_1)$. The energy used in this case is $E(s_1, s_3) = t_1 P(s_1) + (d_i - t_1) P(s_3)$. If we let $\lambda = t_1 / d_i$, then the average power consumption $E(s_1, s_3) / d_i =: \bar{P}_{tv} = (t_1 / d_i) P(s_1) + (1 - (t_1 / d_i)) P(s_3) = \lambda P(s_1) + (1 - \lambda) P(s_3)$. Since $P(\cdot)$ is concave, $P(s_2) \geq \lambda P(s_1) + (1 - \lambda) P(s_3) = \bar{P}_{tv}$. This result implies that a time-varying speed profile is better than a constant speed profile when the power consumption is concave. Notably, the result can be generalised to the case where the power model is non-convex, non-concave, and discrete speed set.

Theorem 2.4. *Let a piecewise constant speed trajectory s^* be given that maps every time instant in a closed interval $[t_0, t_f]$ to the domain S of a power function $P : S \rightarrow \mathbb{R}$. There exists a piecewise constant speed trajectory s with at most one switch such that the amount of computations done and the energy consumed is the same as using s^* , i.e.*

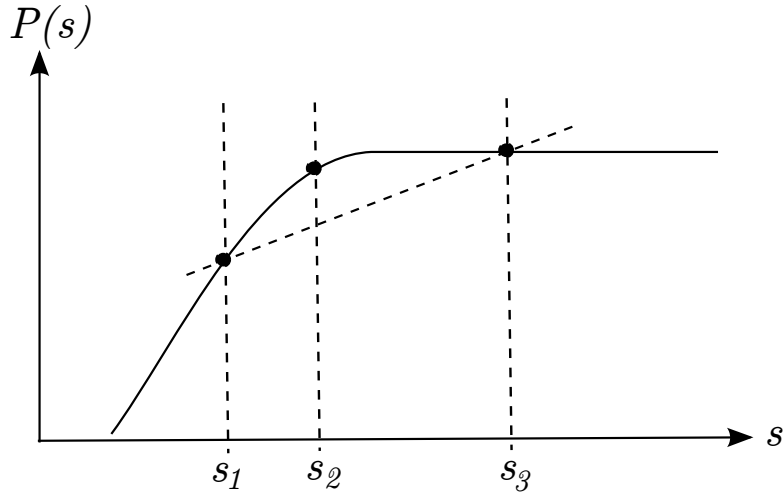


Figure 2.7: A time-varying speed profile is better than a constant speed profile if the power function is not convex.

s is of the form

$$s(t) := \begin{cases} \check{s} & \forall t \in [t_0, (t_f - t_0)\lambda + t_0) \\ \hat{s} & \forall t \in [t_0 + (t_f - t_0)\lambda, t_f] \end{cases}$$

where $\hat{s}, \check{s} \in S$, $\lambda \in [0, 1]$, such that the total amount of computations

$$c := \int_{t_0}^{t_f} s^*(t) dt = \int_{t_0}^{t_f} s(t) dt$$

and energy consumed

$$E := \int_{t_0}^{t_f} P(s^*(t)) dt = \int_{t_0}^{t_f} P(s(t)) dt.$$

Proof. Let $\{\mathcal{T}_1, \dots, \mathcal{T}_p\}$ be a partition of $[t_0, t_f]$ and range $s^* =: \{s_1, \dots, s_p\} \subseteq S$ such that $s^*(t) = s_i$ for all $t \in \mathcal{T}_i$, $i \in \mathcal{J} := \{1, \dots, p\}$. Define $\Delta_i := \int_{\mathcal{T}_i} dt$ as the size of the set $\mathcal{T}_i$ and $\lambda_i := \Delta_i / (t_f - t_0)$, $\forall i \in \mathcal{J}$.

It follows that $c = \sum_i s_i \Delta_i$ and $E = \sum_i P(s_i) \Delta_i$. Hence, the average speed $\bar{s} := c / (t_f - t_0) = \sum_i \lambda_i s_i$ and average power $\bar{P} := E / (t_f - t_0) = \sum_i \lambda_i P(s_i)$.

Note that $\sum_i \lambda_i = 1$. This implies that $(\bar{s}, \bar{P})$ is in the convex hull of the finite set $G := \{(s_i, P(s_i)) \in S \times \mathbb{R} \mid i \in \mathcal{J}\}$ with $\text{vert}(\text{conv } G) \subseteq G$. Hence, there exists a $\lambda \in [0, 1]$ and two points $\hat{s}$ and $\check{s}$ in S with $(\hat{s}, P(\hat{s})) \in \text{vert}(\text{conv } G)$ and $(\check{s}, P(\check{s})) \in \text{vert}(\text{conv } G)$ such that $\bar{s} = \lambda \check{s} + (1 - \lambda) \hat{s}$ and $\bar{P} = \lambda P(\check{s}) + (1 - \lambda) P(\hat{s})$. If s is defined as above with these values of λ , $\hat{s}$ and $\check{s}$, then one can verify that $\int_{t_0}^{t_f} s(t) dt = (t_f - t_0) \bar{s}$ and $\int_{t_0}^{t_f} P(s(t)) dt = (t_f - t_0) \bar{P}$. $\square$

The following result has already been observed in [75, Prop. 1] and [76, Cor. 1].

Corollary 2.1. *Given a processor with a convex power model and required workload within a time interval, there exists a constant optimal speed profile if the set of speed levels S is a closed interval.*

Proof. This is a special case of Theorem 2.4 and can be proven easily using Jensen's inequality. $\square$

Corollary 2.2. *An optimal speed profile to (2.13) can be constructed by switching between no more than two non-zero speed levels within each time interval defined by two consecutive time steps of the major grid $\mathcal{T}$.*

Proof. The overall optimal speed profile can be obtained by connecting an optimal time-varying speed profile proven in Theorem 2.4 for each partitioned time interval. Specifically, the generalised optimal speed profile is a step function. $\square$

The result of the above Theorem and Corollaries can be applied directly to scheduling algorithms that adopt the DP technique such as, LLREF, DP-WRAP, as well as our algorithms in Section 2.3. Consider the problem of determining the optimal speeds at each time interval defined by two consecutive task deadlines. By subdividing time into such intervals, we can easily determine the optimal speed profile of four uniprocessor scheduling paradigms classified by power consumption and taskset models, i.e. (i) a convex power consumption model with implicit deadline taskset, (ii) a convex power consumption model with constrained deadline taskset, (iii) a non-convex power consumption model with implicit deadline taskset and (iv) a non-convex power consumption model with constrained deadline taskset. Specifically, if the taskset has an implicit deadline, then the required workloads (taskset density) are equal for all time intervals; the optimal speed profiles of all schedule intervals are the same as well. Therefore, the optimal speed profile is a constant for (i) (Cor. 2.1) and a combination of two speeds for (iii) (Cor. 2.4). However, for a constrained deadline taskset, the required workload varies from interval to interval, but is constant within the interval. Hence, even if the power function is (ii) convex or (iv) non-convex, the optimal speed profile is a (time-varying) piecewise constant function. In other words, for generality, a time-varying speed profile with two speed levels at each partitioned time interval is guaranteed to provide an energy optimal solution.

Theorem 2.5. *Consider the optimization problems (2.13)–(2.16). An optimal speed profile for (2.13) can be constructed using any of the following methods:*

- *Compute a solution to (2.14) with the lower bound on M at least twice the bound*

Algorithm 3 LLREF (adopted from [2])

```

1: INPUT:  $\{T_1, \dots, T_n\}, \zeta, m$ 
2: sortByLLREF( $\zeta$ )
3:  $\{T_1, \dots, T_m\} = \text{SelectTasks}(\zeta)$ 
4: RETURN:  $\{T_1, \dots, T_m\}$ 

```

in Theorem 2.1.

- If the active power function P_{active} is convex and the speed level sets are closed intervals, compute a solution to (2.15). If there is more than one inter-cluster partitioning task, then the (finite) range of the optimal speed profile should be used to define and compute a solution to (2.16) with at most one inter-cluster partitioning task. This process is concluded with Algorithm 2.
- If the speed level sets are finite, compute a solution to (2.16) with at most one inter-cluster partitioning task, followed with Algorithm 2.

Proof. Follows from the choices of selecting a and s as in the proofs of Theorem 2.1 and Theorem 2.3. The cost of all problems are then equal. $\square$

2.5 Simulations for Tasksets with No Uncertainties

2.5.1 Comparison Algorithms

This section provides details on periodic taskset scheduling algorithms that will be compared with our algorithms, i.e. (i) LLREF [2] that represents the feasibility-optimal strategy with no DVFS scheme, (ii) RT-SVFS [27] that is the LLREF algorithm incorporated with a DVFS scheme, (iii) GWA-SVFS that represents a global energy/feasibility-optimal workload allocation with static frequency scaling scheme at a core-level, (iv) GWA-NoDVFS that is a global scheduling approach without frequency scaling scheme, (v) GWA-DDiscrete and (vi) GWA-SDiscrete that represent global energy/feasibility-optimal workload allocation with dynamic and static discrete frequency scaling schemes, respectively. Note that GWA-SVFS, GWA-NoDVFS, GWA-DDiscrete, and GWA-SDiscrete are based on the mathematical optimization formulation proposed in [70, 77], but adapted to our framework. Specifically, the GWA-SVFS is based on a static frequency scaling with a global scheduling scheme, while [77] is a partitioning-based formulation, and [70] is a generalized formulation for a two-type heterogeneous multi-core system. GP-DDiscrete and GWA-SDiscrete are extensions to practical systems.

Algorithm 4 Uniform RT-SVFS (adopted from [27])

1: **INPUT:** $\{\delta_1, \dots, \delta_n\}, D, m$
 2: $s^k = \max\{\delta_{max}, D/m\}$
 3: **RETURN:** s^k

2.5.1.1 LLREF/RT-SVFS

Details of LLREF were previously given in Section 2.1.7. The LLREF algorithm is briefly described in Algorithm 3, where ζ is an active-task queue. Specifically, at every scheduling event, i.e. bottom hitting event and ceiling hitting event, of each k^{th} T-L plane, Algorithm 3 is invoked to select tasks for execution. That is, the tasks are sorted in the order of largest local remaining execution time (line 1) and the largest m active tasks are chosen for execution (line 2). Note that since LLREF does not support DVFS, tasks are executed at the maximum speed. On the other hand, the RT-SVFS incorporates the DVFS scheme into T-L plane concept and allows the processor speed to vary between 0 and 1. The uniform execution speed decision is shown in Algorithm 4, where s^k is the execution speed of all processors at the k^{th} T-L plane.

2.5.1.2 GWA-SVFS/GWA-NoDVFS

GWA-SVFS/GWA-NoDVFS: Given m_r processors of type- r and n periodic tasks, determine a constant operating speed for each processor s_k^r and the workload ratio y_{ik}^r for all tasks within hyperperiod $\mathcal{L}$ that solves:

$$\underset{\substack{s_k^r, y_{ik}^r, \\ i \in I, k \in K^r, r \in R}}{\text{minimize}} \quad \sum_{r,i,k} \mathcal{L}_i \ell^r(\delta_{ik}^r(s_k^r), s_k^r) \quad (2.17a)$$

subject to

$$\sum_{r=1}^{\kappa} \sum_{k=1}^{m_r} y_{ik}^r = 1, \quad \forall i \quad (2.17b)$$

$$\sum_{r=1}^{\kappa} \sum_{k=1}^{m_r} \delta_{ik}^r(s_k^r) \leq 1, \quad \forall i \quad (2.17c)$$

$$\sum_{i=1}^n \delta_{ik}^r(s_k^r) \leq 1, \quad \forall k, r \quad (2.17d)$$

$$0 \leq y_{ik}^r \leq 1, \quad \forall i, k, r \quad (2.17e)$$

$$s_{min}^r \leq s_k^r \leq s_{max}^r, \quad \forall k, r \quad \text{(GWA-SVFS)} \quad (2.17f)$$

$$s_k^r = s_{max}^r, \quad \forall k, r \quad \text{(GWA-NoDVFS)} \quad (2.17g)$$

where y_{ik}^r is the ratio of the workload of task T_i on processor k of type- r , $\delta_{ik}^r(s_k^r)$ is

the task density on processor k type- r defined as $\delta_{ik}^r(s_k^r) := y_{ik}^r c_i / (s_k^r f_{max} \min\{d_i, p_i\})$ and $\mathcal{L}_i := \mathcal{L} \min\{d_i, p_i\} / p_i$. (2.17b) ensures that all tasks will be allocated the amount of required execution time. The constraint that a task will not be executed on more than one processor at the same time is specified in (2.17c). (2.17d) asserts that the assigned workload will not exceed processor type capacity. Upper and lower bounds on the workload ratio of a task are given in (2.17e). The difference between GWA-SVFS and GWA-NoDVFS lies in restricting a core-level operating speed s_k^r to be either a continuous variable (2.17f) or fixed at the maximum value (2.17g).

2.5.1.3 GWA-DDiscrete

GWA-DDiscrete: Given m_r processors of type- r and n periodic tasks, determine a percentage of the task workload y_{iq}^r at a specific speed level for all tasks within hyperperiod $\mathcal{L}$ that solves:

$$\begin{aligned} & \underset{\substack{y_{iq}^r \\ i \in I, q \in Q^r, r \in R}}{\text{minimize}} \quad \sum_{r,i,q} \mathcal{L}_i \ell^r(\delta_{iq}^r(y_{iq}^r), s_q^r) \end{aligned} \quad (2.18a)$$

subject to

$$\sum_{r=1}^{\kappa} \sum_{q=1}^{l_r} y_{iq}^r = 1, \quad \forall i \quad (2.18b)$$

$$\sum_{r=1}^{\kappa} \sum_{q=1}^{l_r} \delta_{iq}^r(y_{iq}^r) \leq 1, \quad \forall i \quad (2.18c)$$

$$\sum_{i=1}^n \sum_{q=1}^{l_r} \delta_{iq}^r(y_{iq}^r) \leq m_r, \quad \forall r \quad (2.18d)$$

$$0 \leq y_{iq}^r \leq 1, \quad \forall i, q, r \quad (2.18e)$$

where y_{iq}^r is the percentage of workload of task T_i on processor type- r at speed level q , $\delta_{iq}^r(y_{iq}^r)$ is the task density on processor type- r at speed level q , i.e. $\delta_{iq}^r(y_{iq}^r) := y_{iq}^r c_i / (s_q^r f_{max} \min\{d_i, p_i\})$. Constraint (2.18b) guarantees that the total execution workload of a task is allocated. Constraint (2.18c) ensures that a task will be executed only on one processor at a time. Constraint (2.18d) ensures that each processor type workload capacity is not violated. Constraint (2.18e) provides upper and lower bounds on a percentage of task workload at specific speed level.

2.5.1.4 GWA-SDiscrete

GWA-SDiscrete: Given m_r processors of type- r and n periodic tasks, determine a percentage of task workload y_{ikq}^r at a specific speed level and a processor speed level selection z_{kq}^r for all tasks within hyperperiod $\mathcal{L}$ that solves:

$$\underset{y_{ikq}^r, z_{kq}^r}{\text{minimize}} \quad \sum_{r,i,q} \mathcal{L}_i \ell^r(\delta_{ikq}^r(y_{ikq}^r) z_{kq}^r, s_q^r) \quad (2.19a)$$

subject to

$$\sum_{r=1}^{\kappa} \sum_{q=1}^{l_r} \sum_{k=1}^{m_r} y_{ikq}^r z_{kq}^r = 1, \quad \forall i \quad (2.19b)$$

$$\sum_{q=1}^{l_r} z_{kq}^r = 1, \quad \forall k, r \quad (2.19c)$$

$$\sum_{r=1}^{\kappa} \sum_{q=1}^{l_r} \delta_{ikq}^r(y_{ikq}^r) z_{kq}^r \leq 1, \quad \forall i \quad (2.19d)$$

$$\sum_{i=1}^n \sum_{q=1}^{l_r} \delta_{ikq}^r(y_{ikq}^r) z_{kq}^r \leq 1, \quad \forall k, r \quad (2.19e)$$

$$0 \leq y_{ikq}^r \leq 1, \quad \forall i, k, q, r \quad (2.19f)$$

$$z_{kq}^r \in \{0, 1\}, \quad \forall k, q, r \quad (2.19g)$$

where y_{ikq}^r is the workload partition of task T_i of processor k of an r -type at speed level q , z_{kq}^r is a speed level selection variable for processor k of an r -type, i.e. $z_{kq}^r = 1$ if a speed level q of an r -type processor is selected and $z_{kq}^r = 0$ otherwise. Constraint (2.19b), (2.19d)–(2.19f) are the same as the GWA-DDiscrete. Constraint (2.19c) ensures that only one speed level is selected. Constraint (2.19g) emphasises that the speed level selection variable is a binary.

Note that GWA-SVFS and GWA-NoDVFS are NLPs, GWA-DDiscrete is an LP and GWA-SDiscrete is a MINLP. Moreover, the formulation of GWA-DDiscrete allows a processor to run with a combination of constant discrete speed levels, while GWA-SVFS and GWA-SDiscrete only allow a constant execution speed for each processor. Moreover, when $d_i = p_i$ as in the case of an implicit deadline taskset, $\mathcal{L}_i = \mathcal{L}$, $\forall i$ in all of the above formulations. For all simulations, all MINLP formulations were modelled using ZIMPL [78] and solved with SCIP [79], while LP and NLP formulations were solved with SoPlex [80] and IPOPT [81], respectively.

Table 2.2: Commercial Processor Details for Simulation

Processor type	XScale [84]					PowerPC 405LP [85]			
Frequency (MHz)	150	400	600	800	1000	33	100	266	333
Speed	0.15	0.4	0.6	0.8	1.0	0.1	0.3	0.8	1.0
Voltage (V)	0.75	1.0	1.3	1.6	1.8	1.0	1.0	1.8	1.9
Active Power (mW)	80	170	400	900	1600	19	72	600	750
Idle Power (mW)	40 [50]					12			

2.5.2 Homogeneous Multiprocessor System Simulations

2.5.2.1 System, Processor and Task Models

The performance of solving the above optimization problems is evaluated on two models of practical commercial processors, namely an XScale and a PowerPC 405LP. The power consumption details of the two commercial processors, which have also been used in [50, 82, 83], are given in Table 2.2. The active power consumption models of the XScale and PowerPC 405LP shown in Table 2.3 were obtained by a polynomial curve fitting to the generic form (2.2) (details are given in the appendix B). The plots of the actual data versus the fitted models are shown in Fig. 2.8.

2.5.2.2 Simulation Setup and Results

For a simulation with implicit deadline taskset, we consider where three independent periodic real-time tasks need to be scheduled onto two homogeneous processors. The total energy consumption of each taskset in Table 2.4 was evaluated. Note that the first parameter of a task is $\underline{x}_i$; c_i can be obtained by multiplying $\underline{x}_i$ by f_{max} . Additionally, since the task period is the same as the deadline for an implicit deadline taskset, the last parameter of the task model is dropped. For the MINLP-DVFS, NLP-DVFS and LP-DVFS implementations, the major grid discretization step $N = 4$ since all tasksets have four distinct deadlines. For the MINLP-DVFS, we chose the minor grid discretization step $M = 3$, since there are at most three jobs in each major grid, which implies that there will be at most three context switches within each minor grid (Theorem 2.1).

Similarly, for a simulation with constrained deadline taskset, we consider the case where four independent periodic real-time tasks are to be scheduled onto two processors. Details of the tasksets are shown in Table 2.5. The major grid discretization step $N := 2$ and the minor grid discretization step $M := 4$.

Simulation results for implicit deadline tasksets are shown in Figure 2.10 and 2.11. The minimum taskset density D is represented on the horizontal axis. The vertical

Table 2.3: Processor Power Consumption Fitted model

Processor type	Active Power Model	MAPE ⁶
XScale	$P_{active}(s) := 1524.92s^{3.0269} + 75.1092$	1.1236
PowerPC 405LP	$P_{active}(s) := 736.87s^{2.0990} + 13.1333$	5.2323

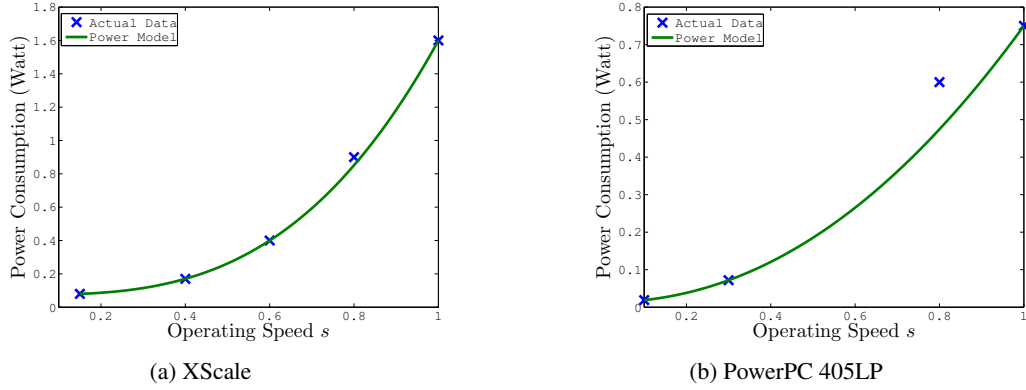


Figure 2.8: Actual data versus fitted model

axis represents the total energy consumption normalised by the energy used by GWA-NoDVFS, i.e. less than 1 means that an algorithm does better than GWA-NoDVFS. For a system with continuous speed range (Fig. 2.10), the results from solving MINLP-DVFS, NLP-DVFS and GWA-SVFS are the same. Though our MINLP-DVFS and NLP-DVFS formulations allow the task executing speed to vary with time, the obtained solutions are the same as solving GWA-SVFS, where processor speed is constant. Specifically, the result suggests that an optimal speed profile is a constant rather than time-varying for this system. However, it is not true for a system with discrete speed levels (Fig. 2.11). As can be observed, LP-DVFS is energy optimal, while GWA-SDiscrete is not.

Note that, for implicit deadline tasksets, the number of tasks as well as the task parameters do not have any effects on the total energy consumption if the densities of the tasksets are the same because our algorithms are feasibility optimal; the energy consumption will be the same for every taskset with the same density due to the equal amount of execution workload.

Figures 2.12 and 2.13 show the simulation results for constrained deadline tasksets. It can be seen from the plots that for a time-varying workload (taskset density) such as a constrained deadline taskset, our results from solving MINLP-DVFS, NLP-DVFS and LP-DVFS are energy optimal, while GWA-SVFS, GWA-SDiscrete and GWA-NoDVFS are not. This is because our formulations are incorporated with time, which provides benefits on solving a scheduling problem with both time-varying workload

⁶See Appendix A.

Table 2.4: Implicit Deadline Simulation Tasksets

D	Taskset	D	Taskset
0.2	(0.5,5),(0.5,10),(1,20)	1.2	(2,5),(6,10),(4,20)
0.4	(1,5),(1,10),(2,20)	1.4	(3,5),(4,10),(8,20)
0.6	(1,5),(2,10),(4,20)	1.6	(3,5),(4,10),(12,20)
0.8	(1,5),(4,10),(4,20)	1.8	(2,5),(8,10),(12,20)
1.0	(1,5),(4,10),(8,20)	2.0	(4,5),(6,10),(12,20)

Table 2.5: Constrained Deadline Simulation Tasksets

D	Taskset
0.4	(0.75,5,10),(0.75,5,10),(0.5,10,10),(0.5,10,10)
0.6	(1,5,10),(1,5,10),(1,10,10),(1,10,10)
0.8	(1.5,5,10),(1.5,5,10),(1,10,10),(1,10,10)
1.0	(2,5,10),(2,5,10),(1,10,10),(1,10,10)
1.2	(2.5,5,10),(2.5,5,10),(1,10,10),(1,10,10)
1.4	(3,5,10),(3,5,10),(1,10,10),(1,10,10)
1.6	(3,5,10),(3,5,10),(2,10,10),(2,10,10)
1.8	(3.5,5,10),(3.5,5,10),(2,10,10),(2,10,10)
2.0	(4,5,10),(4,5,10),(2,10,10),(2,10,10)

(constrained deadline taskset) and constant workload (implicit deadline taskset). To illustrate this, consider an implicit deadline taskset $\{(4, 5, 5), (6, 10, 10), (12, 20, 20)\}$ and a constrained deadline taskset $\{(4, 5, 10), (6, 10, 20), (12, 20, 40)\}$. The minimum taskset density plots of the two tasksets are shown in Figure 2.9. As can be seen in the figure that the minimum taskset density of an implicit deadline taskset is a constant, while it is a piecewise constant for a constrained deadline taskset. Note that for an algorithm that does not consider the taskset density as a function of time will only see the minimum taskset density as a constant for both implicit and constrained deadline tasksets (a constant equals to 2.0 for this example), therefore, the algorithm will always overestimate the system workload. Specifically, the algorithm cannot take advantage of reducing the executing speed when the system workload density is indeed less than the predicted value. For example, consider Figure 2.9b, during the time interval $[5, 20]$, where the minimum taskset density is less than 2.0, our proposed algorithms will execute the tasks at a lower speed, resulting in a lower energy consumption.

In general, compared to the constant speed profile, the time-varying speed profile saving increases as the minimum taskset density increases. It can also be noticed that the percentage saving of the Xscale is nonlinear compared to that of the PowerPC's. This is because the power consumption model of the Xscale is a cubic, while the PowerPC's power consumption model is a quadratic. Examples of optimal speed profiles obtained by solving NLP-DVFS and LP-DVFS for constrained deadlines taskset in Table 2.5 with $D = 1.4$ are shown in Figures 2.14 and 2.15, respectively. These results suggest

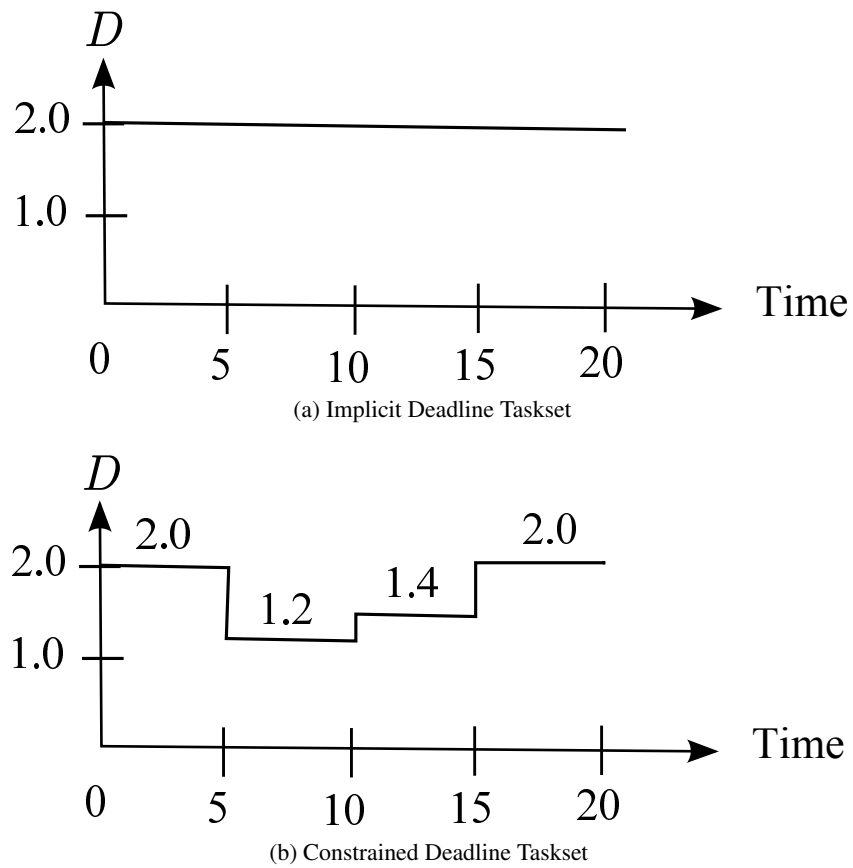


Figure 2.9: An example of a minimum taskset density plot for different taskset characteristics

that an optimal speed profile for a constrained deadline taskset is time-varying with at most one speed level change at each partitioned time interval, which complies with Corollary 2.2.

However, it has to be mentioned that the energy saving percentage varies with the taskset, which implies that the number shown on the plots can be varied, but the significant outcomes stay the same. An example of how large the variability is illustrated in Figure 2.16, where 10 constrained deadline tasksets are randomly generated for each taskset density and are scheduled on a homogeneous system composed of two practical PowerPC 405LP processors.

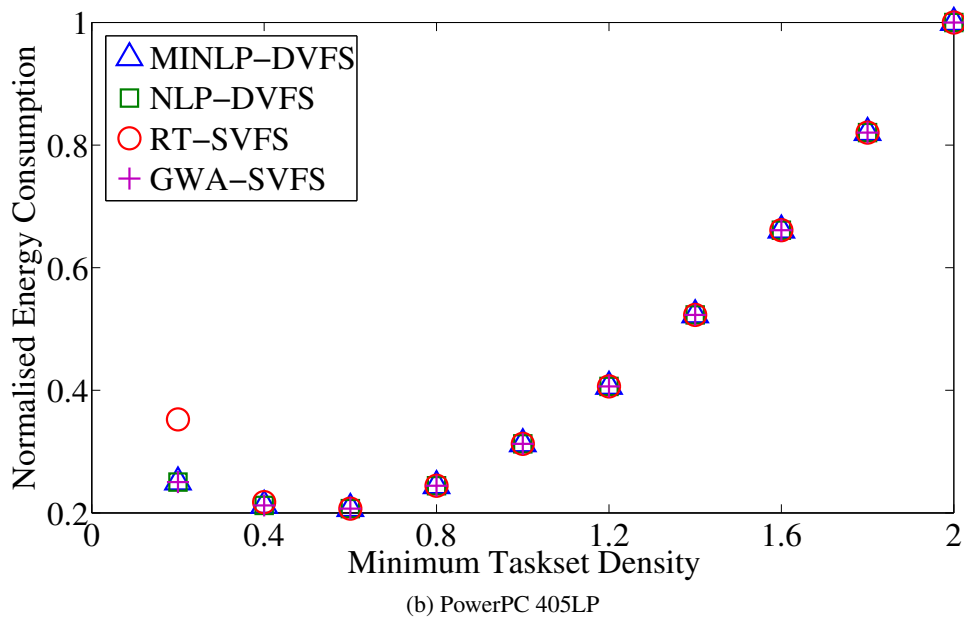
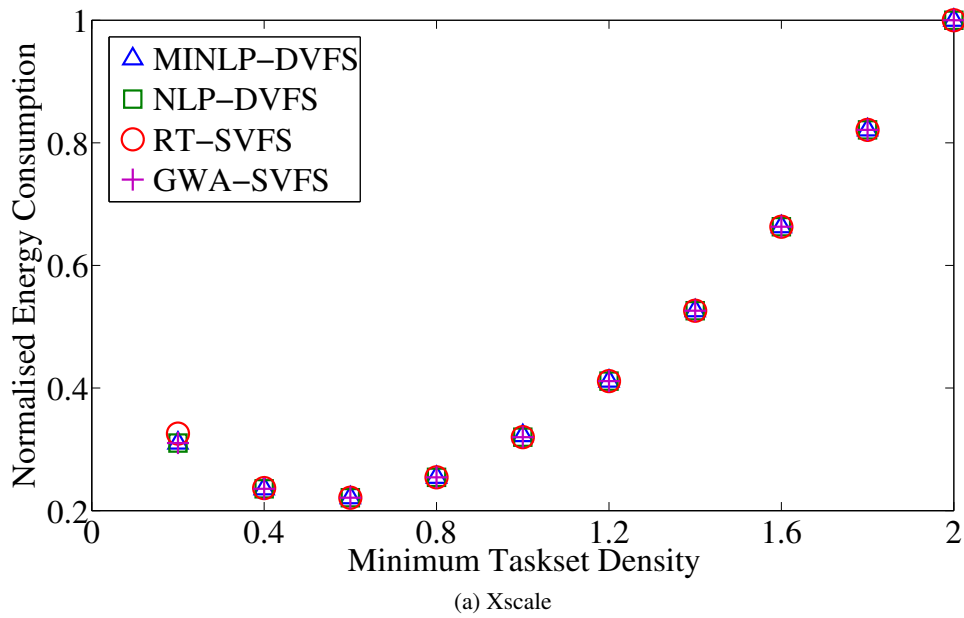
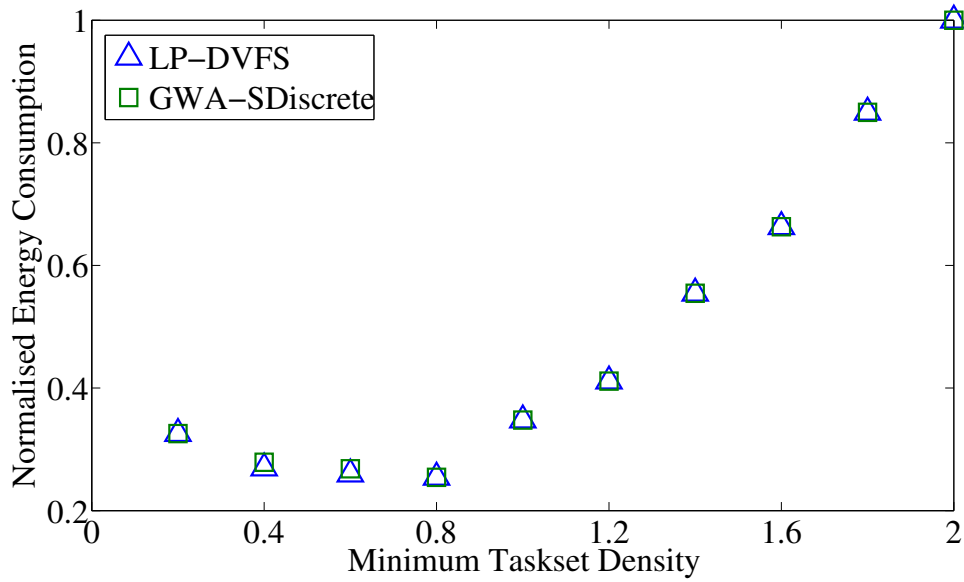
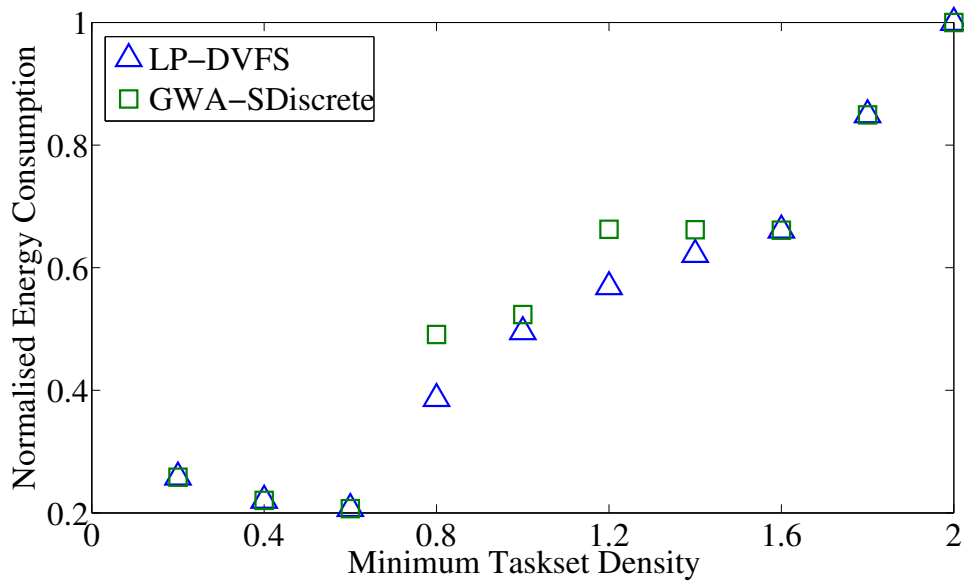


Figure 2.10: Simulation results for scheduling real-time tasks with implicit deadlines on two homogeneous processors with continuous speed range



(a) Xscale



(b) PowerPC 405LP

Figure 2.11: Simulation results for scheduling real-time tasks with implicit deadlines on two homogeneous processors with discrete speed levels

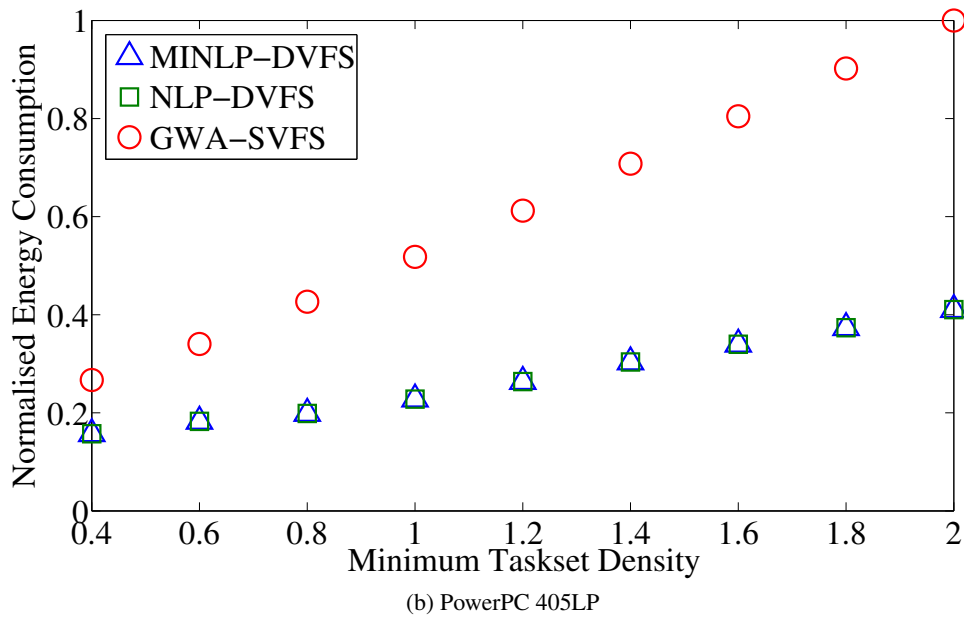
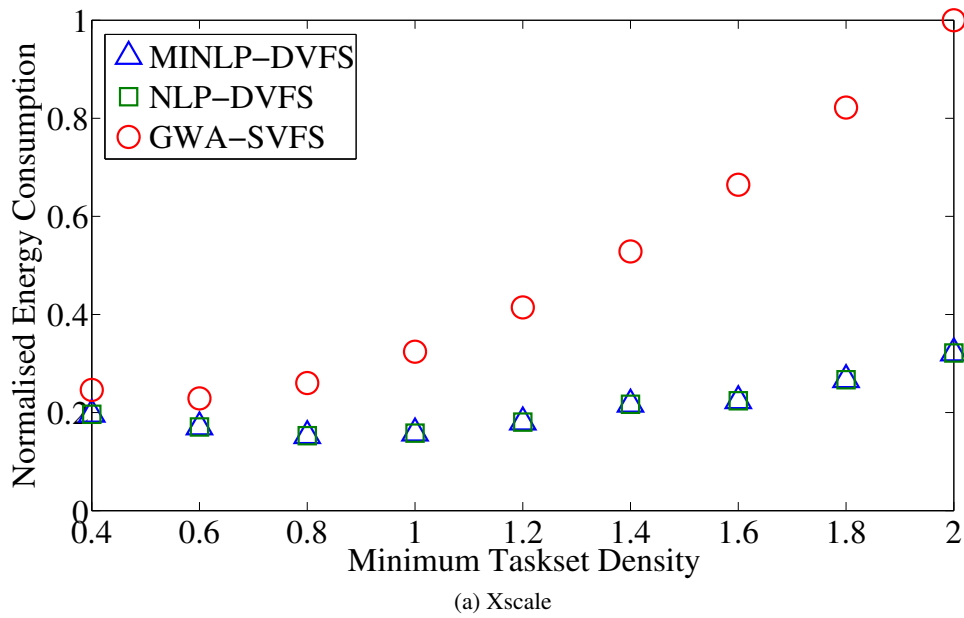


Figure 2.12: Simulation results for scheduling real-time tasks with constrained deadlines on two homogeneous processors with continuous speed range

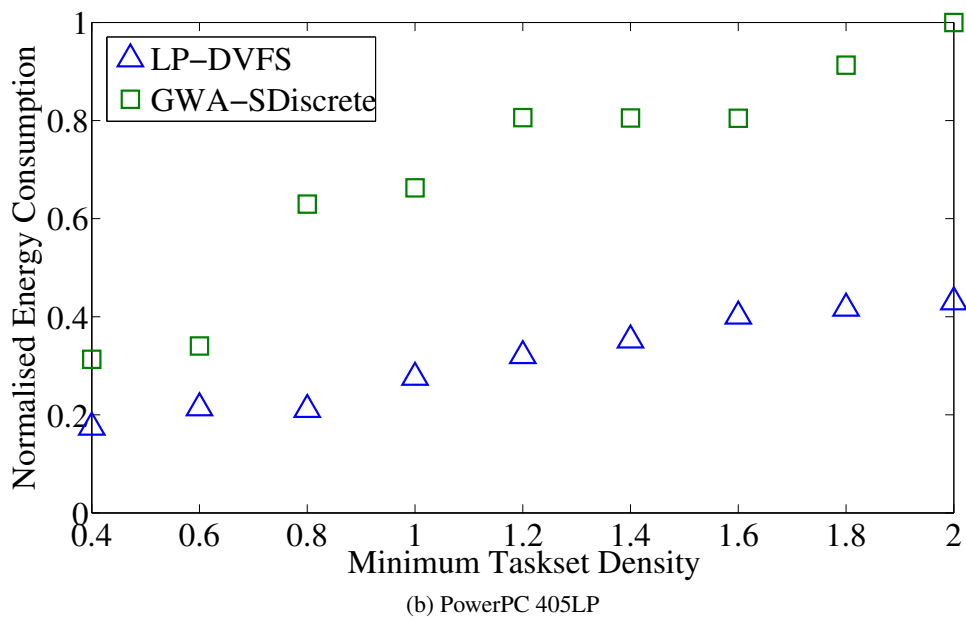
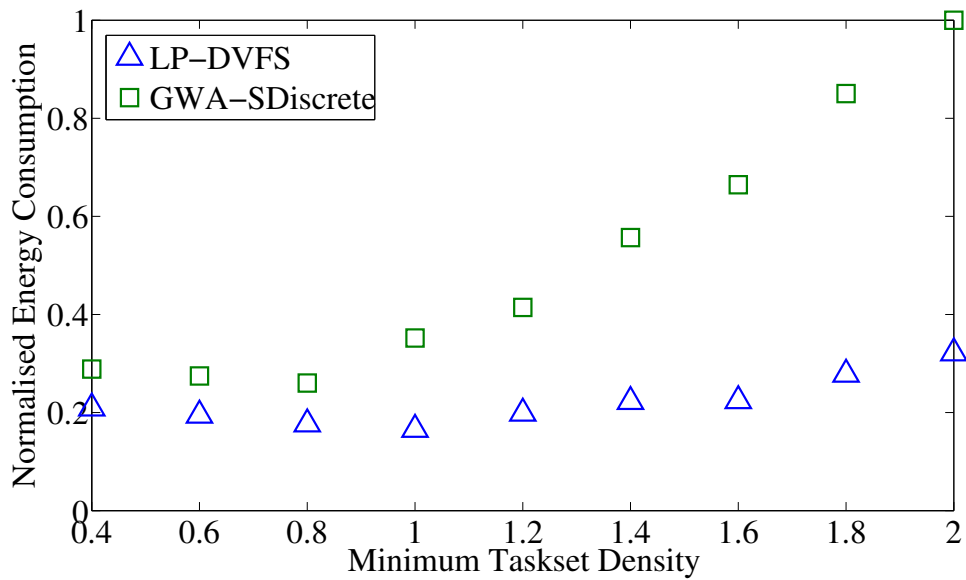


Figure 2.13: Simulation results for scheduling real-time tasks with constrained deadlines on two homogeneous processors with discrete speed levels

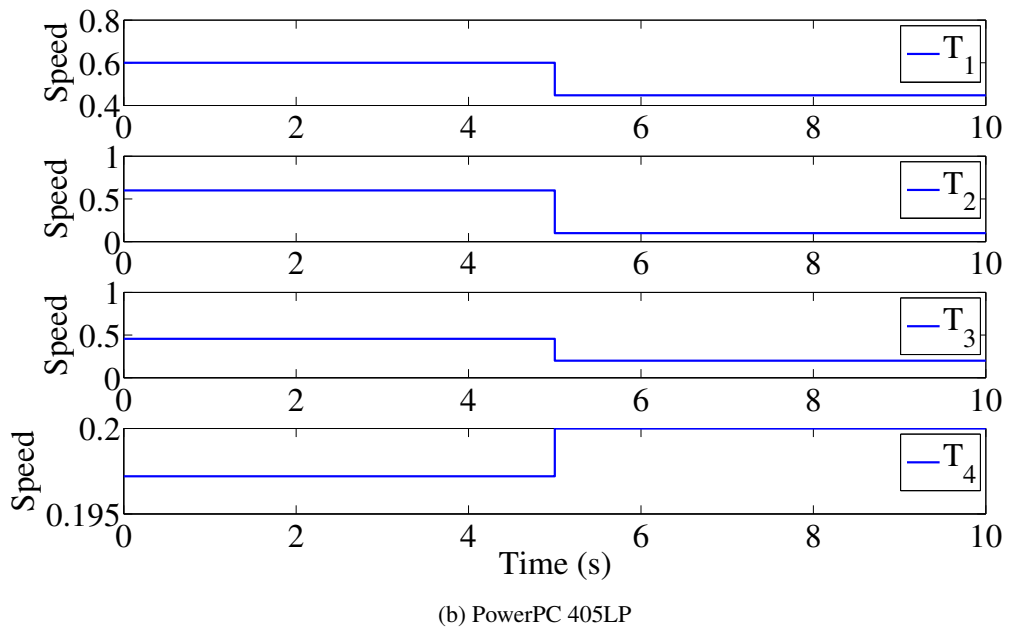
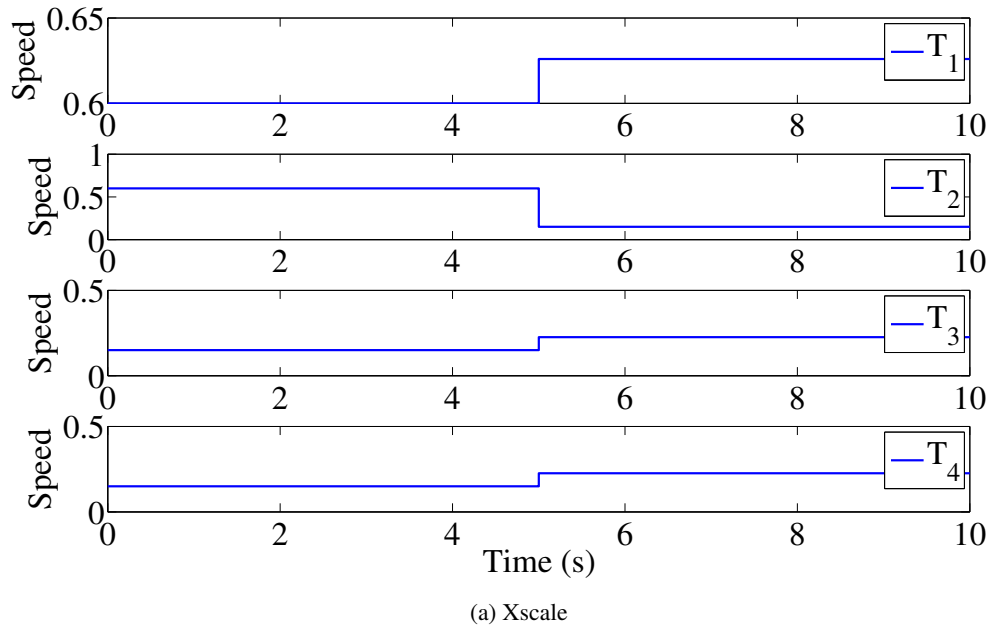


Figure 2.14: Examples of optimal speed profiles obtained by solving NLP-DVFS with constrained deadline taskset in Table 2.5 with $D = 1.4$

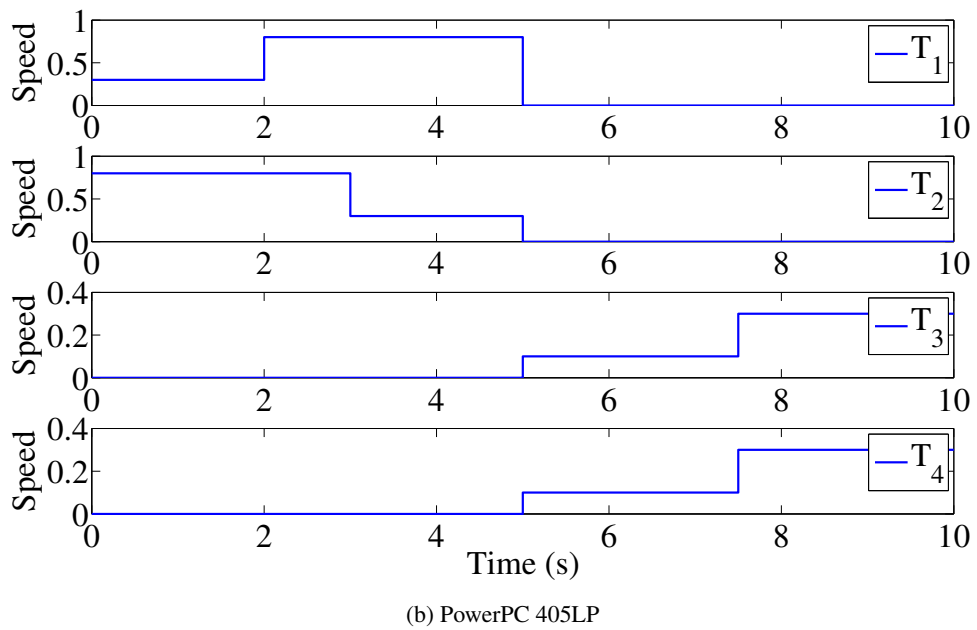
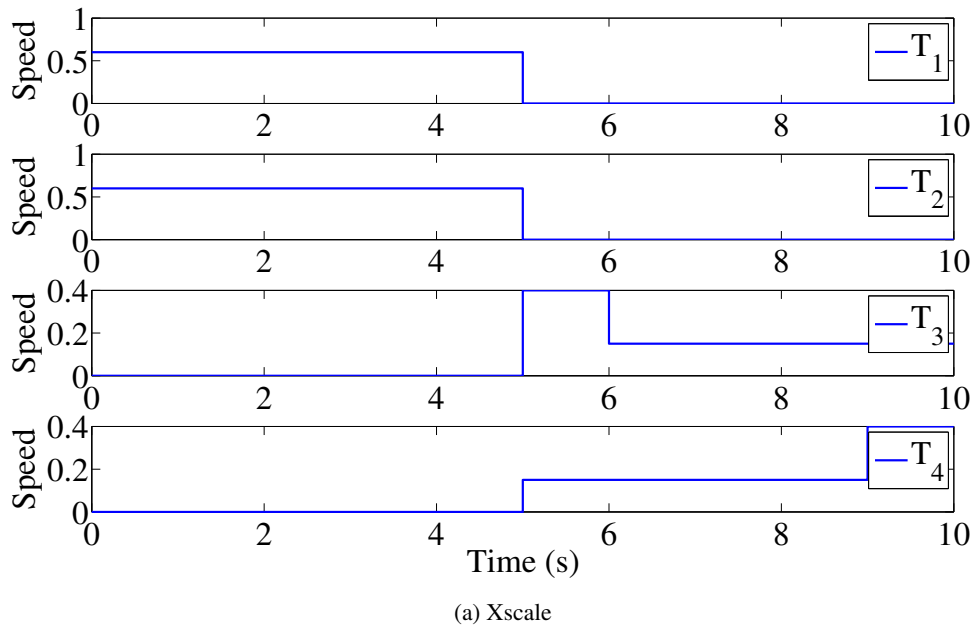


Figure 2.15: Examples of optimal speed profiles obtained by solving LP-DVFS with constrained deadline taskset in Table 2.5 with $D = 1.4$

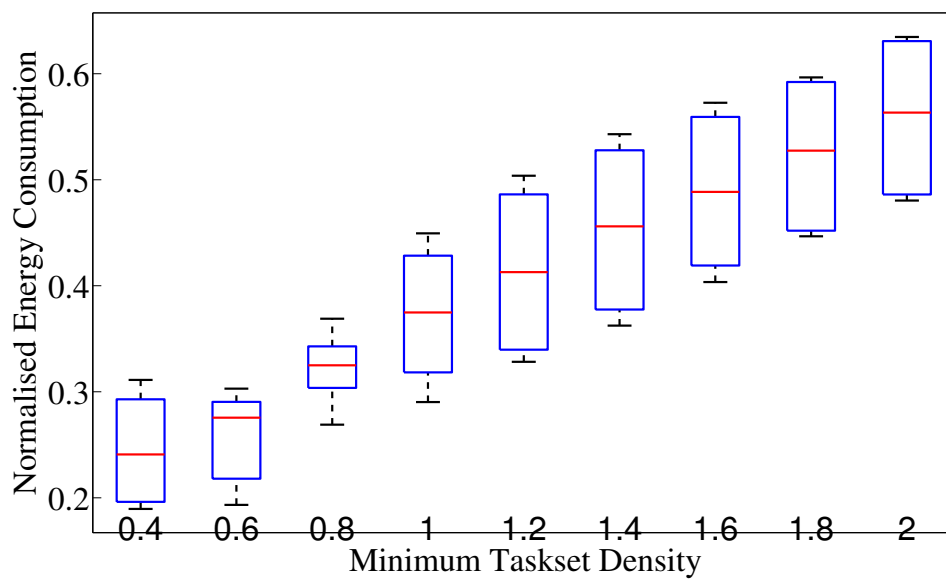


Figure 2.16: Energy saving percentage box-plots of executing constrained deadline tasksets on a homogeneous system composed of two practical PowerPC 405LP processors

Table 2.6: ARM Cortex-A15 (big) Processor Details [70]

Voltage (V)	0.93	0.96	1.0	1.04	1.08	1.1	1.15	1.2	1.23
Freq. (MHz)	800	900	1000	1100	1200	1300	1400	1500	1600
Speed	0.5	0.5625	0.625	0.6875	0.75	0.8125	0.875	0.9375	1.0
Power (mW)	327	392	472	562	661	742	874	1,019	1,142

Table 2.7: ARM Cortex-A7 (LITTLE) Processor Details [70]

Voltage (V)	0.9	0.94	1.01	1.09	1.2
Freq. (MHz)	250	300	400	500	600
Speed	0.1563	0.1875	0.25	0.3125	0.375
Power (mW)	32	42	64	92	134

2.5.3 Heterogeneous Multiprocessor System Simulations

2.5.3.1 System, Processor and Task Models

The energy efficiency of solving the above optimization problems is evaluated on a system composed of two different processor types, i.e. the ARM big.LITTLE architecture, where a big core provides faster execution times, but consumes more energy than a LITTLE core. The details of the ARM Cortex-A15 (big) and Cortex-A7 (LITTLE) core, which have been validated in [70], are given in Tables 2.6 and 2.7. The active power consumption models, obtained by a polynomial curve fitting to the generic form (2.2), are shown in Table 2.8. The plots of the actual data versus the fitted models are shown in Fig. 2.17. The idle power consumption was not reported, thus we will assume this to be a constant strictly less than the lowest active power consumption, i.e. $P_{idle} = 70$ mW for the big core and $P_{idle} = 12$ mW for the LITTLE core.

To illustrate that our formulations are able to solve a broader class of multiprocessor scheduling problem than other optimal algorithms reported in the literature, we consider periodic taskset models with both implicit and constrained deadlines. However, a more general taskset model such as an arbitrary deadline taskset, where the deadline could be greater than the period, a sporadic taskset model, where the inter-arrival time of successive tasks is at least p_i time units, and an aperiodic taskset can be solved by our algorithms as well.

To guarantee the existence of a valid schedule, the minimum taskset density has to be less than or equal to the system capacity. Moreover, a periodic task needs to be able to be executed on any processor type. Specifically, a minimum task density should be less than or equal to the lowest capacity of all processor types, i.e. $\delta_i(1) \leq 0.375$ for this particular architecture.

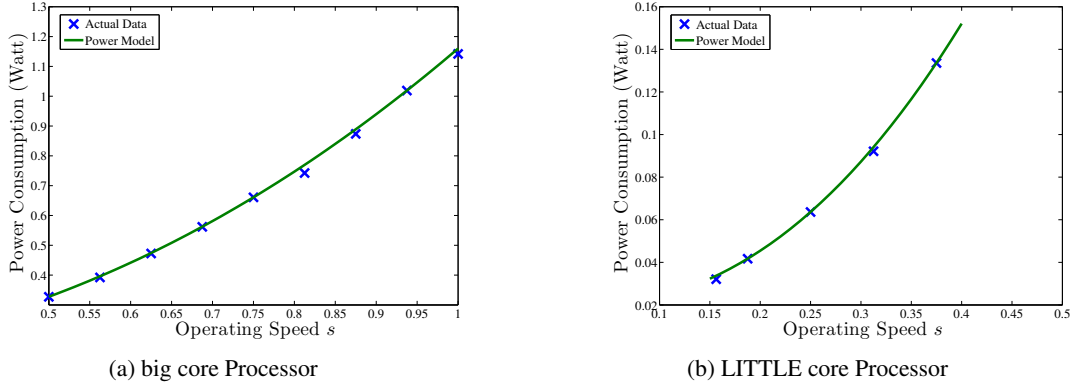


Figure 2.17: Actual data versus fitted Model

Table 2.8: ARM Processor Power Consumption Models

Processor	Active Power Model	MAPE
big	$P_{active}(s) := 1063.9s^{2.2} + 95.9075$	0.9283
LITTLE	$P_{active}(s) := 1103.17s^{2.3034} + 18.3549$	1.4131

2.5.3.2 Simulation Setup and Results

For simplicity and without loss of generality, consider the case where independent real-time tasks are to be executed on two-type processor architectures, for which the details are given in Section 2.5.3.1. For an implicit deadline taskset, we consider the system to be composed of two big cores and six LITTLE cores, which has a system capacity of $4.25 = (2 + 2.25)$. The total energy consumption of each taskset with a minimum taskset density varying from 0.5 to system capacity, with a step of 0.25, given in Table 2.9, are evaluated. Again, the first parameter of a task is $\underline{x}_i$, not c_i . Figure 2.18a shows simulation results for scheduling a real-time taskset with implicit deadlines on an ideal system. The minimum taskset density D is represented on the horizontal axis. The vertical axis is the total energy consumption normalised by GWA-NoDVFS.

The three algorithms with a DVFS scheme, i.e. MINLP-DVFS, NLP-DVFS, and GWA-SVFS, produce the same optimal energy consumption, though both of our algorithms allow the operating speed to vary with time compared with a constant frequency scaling scheme, used by GWA-SVFS. The simulation results again suggest that the optimal speed is a constant, rather than time-varying, for an implicit deadline taskset that has a constant workload over time. This result complies with Corollary 2.1. Moreover, the little core, which only has 37.5% computing power compared with the big core, and consumes considerably less power even when running at full speed, will be selected by the optimizer before considering the big core. This is why we can see two upward parabolic curves in the figures, where the first one corresponds to the case where only

Table 2.9: Simulation Implicit Deadline Tasksets

D	Taskset
0.50	(1,5),(1,10),(4,20)
0.75	(1,5),(1,10),(5,20),(4,20)
1.00	(1,5),(1,10),(7,20),(7,20)
1.25	(1,5),(1,10),(6,20),(6,20),(7,20)
1.50	(1,5),(3,10),(6,20),(7,20),(7,20)
1.75	(1,5),(2,10),(6,20),(7,20),(7,20),(7,20)
2.00	(1,5),(3,10),(7,20),(7,20),(7,20),(7,20),(2,20)
2.25	(1,5),(3.5,10),(3.5,10),(6,20),(7,20),(7,20),(7,20)
2.50	(1,5),(3,10),(3,10),(3,10),(7,20),(7,20),(7,20),(7,20)
2.75	(1,5),(3.5,10),(3.5,10),(3,10),(7,20),(7,20),(7,20),(7,20),(3,20)
3.00	(1,5),(3,10),(3,10),(3,10),(3,10),(7,20),(7,20),(7,20),(7,20),(4,20)
3.25	(1,5),(3.5,10),(3.5,10),(3.5,10),(3.5,10),(7,20),(7,20),(7,20),(7,20),(5,20)
3.50	(1,5),(3.5,10),(3.5,10),(3,10),(3,10),(7,20),(7,20),(7,20),(7,20),(7,20),(5,20)
3.75	(1,5),(3.5,10),(3.5,10),(3,10),(3,10),(7.5,20),(7.5,20),(7.5,20),(7.5,20),(7.5,20),(7.5,20)
4.00	(1,5),(3.5,10),(3.5,10),(3,10),(3,10),(7.5,20),(7.5,20),(7.5,20),(7.5,20),(7.5,20),(7.5,20),(5,20)
4.25	(1,5),(3.75,10),(3.75,10),(3.75,10),(3.75,10),(7.5,20),(7.5,20),(7.5,20),(7.5,20),(7.5,20),(7.5,20),(6,20)

Table 2.10: Simulation Constrained Deadline Tasksets

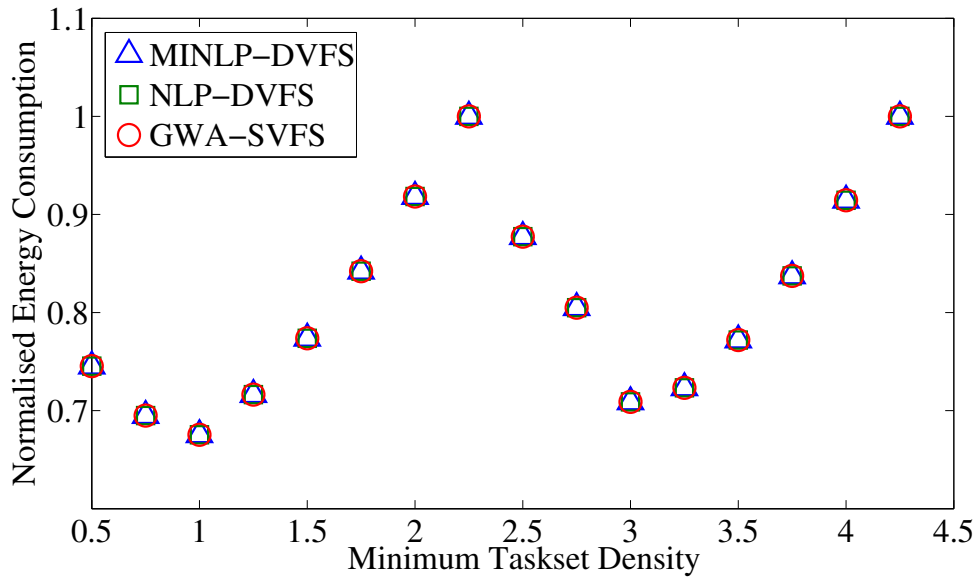
D	Taskset
0.250	(0.9375,5,10),(0.625,10,10)
0.375	(1.5625,5,10),(0.625,10,10)
0.500	(1.875,5,10),(1.25,10,10)
0.625	(1.875,5,10),(1.5,10),(0.5,10,10)
0.750	(1.875,5,10),(1.625,5,10),(0.5,10,10)
0.875	(1.875,5,40),(1.75,5,40),(6,40,40)
1.000	(1.875,5,40),(1.875,5,40),(0.5,5,40),(6,40,40)
1.125	(1.875,5,40),(1.5,5,40),(1.3125,5,40),(6,40,40),(1.5,40,40)
1.25	(1.875,5,40),(1.875,5,40),(1.5625,5,40),(6,40,40),(1.5,40,40)
1.375	(1.875,5,40),(1.875,5,40),(1.875,5,40),(6,40,40),(4,40,40)

little cores in the system are selected, while both core types are selected in the second, which happens when the minimum taskset density is larger than the little-core cluster's capacity.

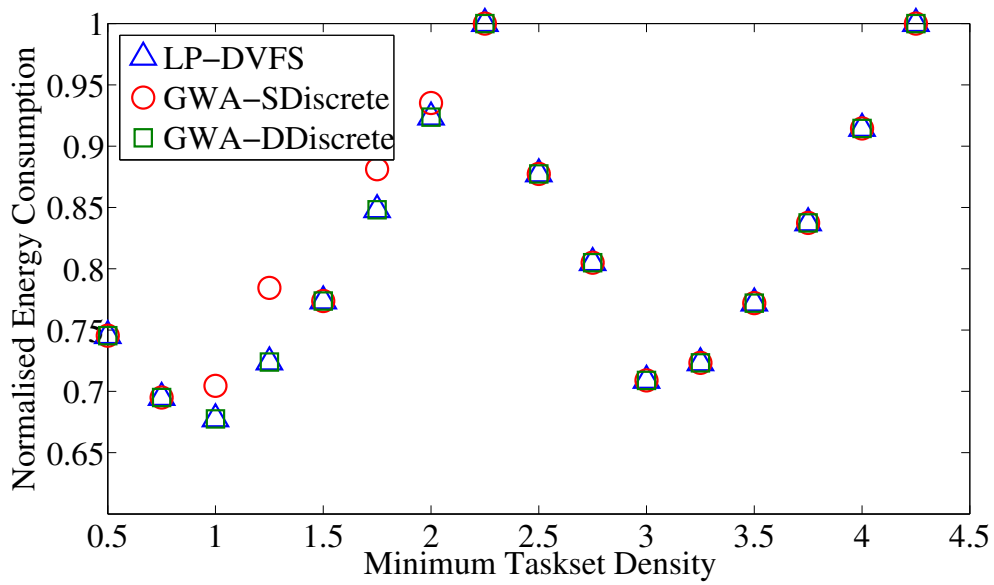
However, for a practical system, where a processor has discrete speed levels, the constant speed assignment is not an optimal strategy. As can be observed in Figure 2.18b, the LP-DVFS and GWA-DDiscrete are energy optimal, while the GWA-SDiscrete is not. The results imply that to obtain an energy optimal schedule, a combination of discrete speed levels is necessary.

For real-time taskset with constrained deadlines, we consider a system with one big core

and one LITTLE core, i.e. a system capacity of $1.375=(1+0.375)$. The simulation results of executing each taskset, listed in Table 2.10, are shown in Figures 2.19, where the total energy consumption normalised by GWA-NoDVFS is on the vertical axis. It can be seen from the plots that for a taskset with a piecewise constant and time-varying workload, i.e. constrained deadlines, GWA-SVFS, GWA-DDiscrete and GWA-SDiscrete cannot provide an optimal energy consumption, while our algorithms are optimal. This is because time is incorporated in our formulations, which provides benefits on solving a scheduling problem with time-varying workload as well as constant workload. Lastly, it has to be mentioned that the energy saving percentage varies with the taskset, which implies that the number on the plots shown here can be varied, but the significant outcomes stay the same.

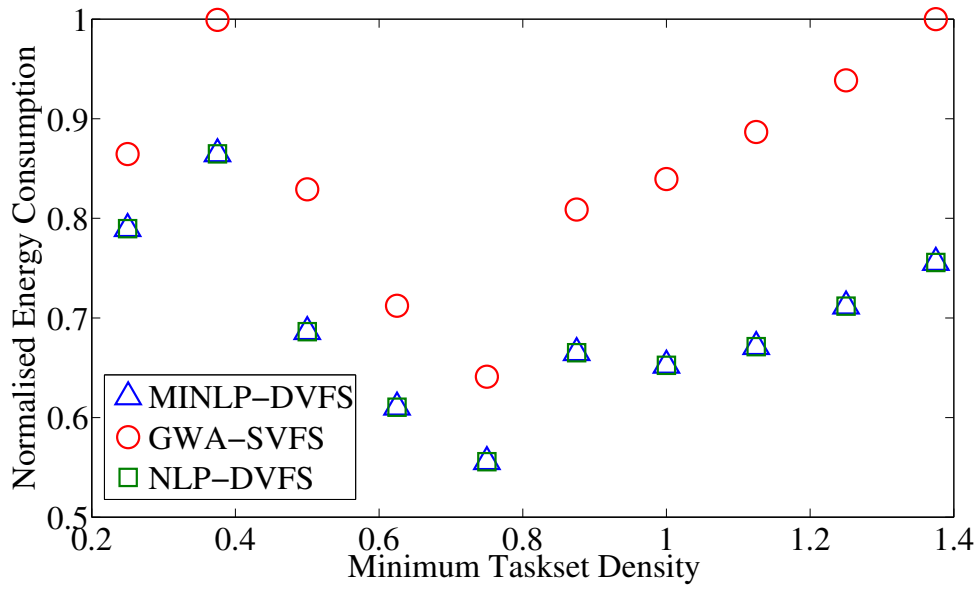


(a) A system with continuous speed range

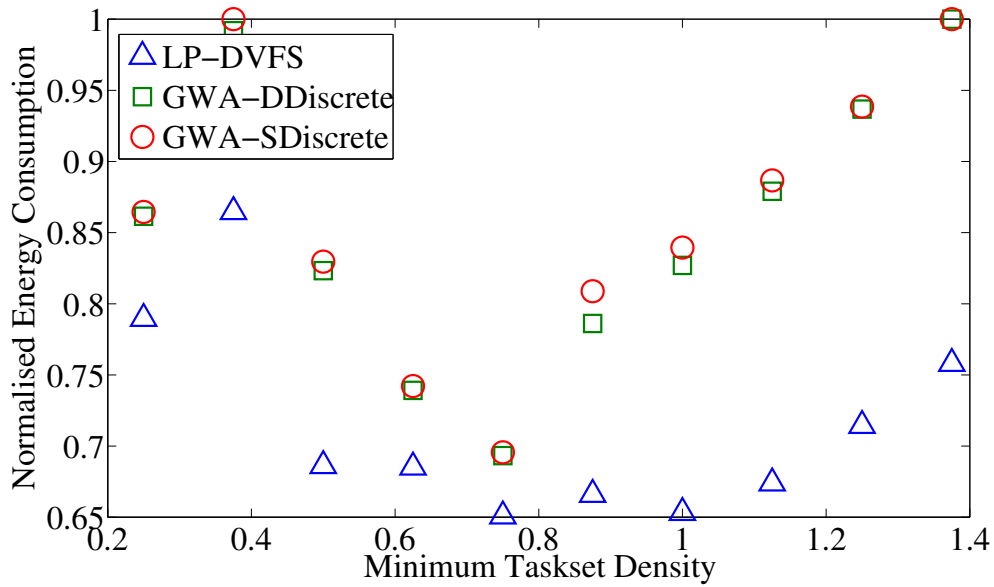


(b) A system with discrete speed levels

Figure 2.18: Simulation results for scheduling real-time tasks with implicit deadlines



(a) A system with continuous speed range



(b) A system with discrete speed levels

Figure 2.19: Simulation results for scheduling real-time tasks with constrained deadlines

2.6 Feedback Scheduling

Though a lot of work has been proposed to solve real-time scheduling problems, most of them are based on the assumption that the computational task parameters, e.g. the task's execution time, period and deadline, do not change. In other words, they are open-loop controllers. Though an open-loop scheduler can provide good performance in a predictive environment, the performance can be degraded in an unpredictable environment, where there are uncertainties in task parameters. Specifically, the actual execution time of the task can vary by as much as 87% of measured worst-case execution times [86]. Since it is often the case that the task parameters are based on the worst-case, it follows that the system workload is overestimated, resulting in higher energy consumption due to non-optimal solutions. Therefore, in this section, we aim to apply feedback methods from control theory to address a scheduling problem subjected to time-varying workload uncertainty.

Only a few works have adopted feedback methods from control theory to cope with a dynamically changing environment for real-time scheduling. For example, [24] proposed an energy-aware feedback scheduling architecture for soft real-time tasks for a uniprocessor. A proportional controller is used to adjust the workload utilization through a variable voltage optimization unit. Specifically, the controlled variable is the energy savings ratio and the manipulated variable is the worst-case utilization.

Similarly, [23] proposed a feedback method for estimating task execution times to improve the system performance, i.e. the number of tasks that meet the deadlines and the number of tasks that are admitted to the system. That is, the estimated execution time is calculated at each decision time interval based on the measurements of the task deadline miss ratios and task rejection ratios.

In [25], a feedback method was developed for a uniprocessor hard real-time scheduling problem with DVFS to cope with varying execution time tasksets. In the same manner, the actual execution time of the task is fed back to a PID controller to adjust the estimated execution time of the task, as well as the execution frequency.

A two-level power optimization control on a multi-core real-time systems was proposed in [87]. At the core-level, the utilization of each CPU is monitored and a DVFS scheme is implemented in response to uncertainties in task execution times in order to obtain a desired utilization. To further reduce power consumption, task reassignment and idle core shutdown schemes were employed at the processor level.

All of the work in this area only consider feedback of real-time scheduling as regulation problems. However, our work will consider real-time multiprocessor scheduling as a constrained optimal control problem [88], which can be combined with a feedback

scheme to handle uncertainties in an unpredictable scheduling environment, as is done in model predictive control [89]. Our proposed scheme would also be known as a slack reclamation scheme in the real-time scheduling literature, in which the slack time due to early completion of a task is exploited to reduce energy consumption by decreasing the operating speed of the remaining tasks in the system [90, 91].

2.6.1 Feedback Task Model and Assumptions

A task T_i is assumed to be aperiodic. The estimated minimum execution time $\underline{x}_i$ is the estimated execution time of the task T_i when executed at the maximum clock frequency f_{max} , i.e. $\underline{x}_i := c_i / f_{max}$. The minimum task density δ_i is defined as the ratio between the task minimum execution time and deadline, i.e. $\delta_i := \underline{x}_i / d_i$. The actual minimum execution time of the task $\underline{y}_i$ is the actual execution time when the task is executed at clock frequency f_{max} , i.e. $\underline{y}_i := \gamma_i \underline{x}_i$, where $0 < \gamma_i \leq 1$ is the estimation factor. Note that the actual execution time of the task is not known until the task has finished. We will assume that the tasks can be preempted at any time. Moreover, task migration is allowed. There is no delay with task preemption or migration, since we assume that the delay is added to the estimated task execution times or that the delay is negligible. Lastly, it will also be assumed that tasks do not have any resource or precedence constraints, i.e. the task is ready to start upon its arrival time.

2.6.2 Feedback Scheduler

As can be seen, our open-loop workload partitioning algorithms are based on the estimated minimum execution time $\underline{x}_i$. The task will often finish earlier than expected, i.e. the actual minimum execution time $\underline{y}_i$ is often less than $\underline{x}_i$. Consider Figure 2.20, which illustrates a fluid path of executing a task T_i . Our open-loop algorithm follows a different path from the one that we really want to follow, i.e. the dotted line, due to uncertainty in task execution times. In other words, the open-loop algorithm can provide a solution that is overestimating the system workload, leading to higher energy consumption, due to the fact that the system operates at an unnecessarily higher speed. Therefore, it is better to feed back information whenever (i) a task finishes or (ii) a new task arrives at the system, in order to recalculate a new control action to respond to the changing workload.

The overall architecture of our feedback scheduling system is given in Figure 2.21, where the scheduler is called at two scheduling events. One occurs when a task finishes its required executing workload/cycles on one of the processors and the other when a new task arrives. The scheduler is composed of two sub-units, i.e. a workload

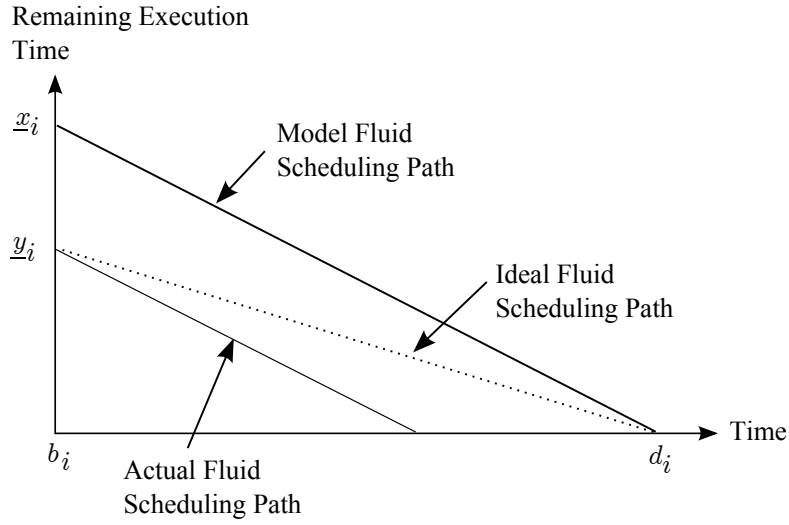


Figure 2.20: Fluid scheduling model with uncertainty in task execution time

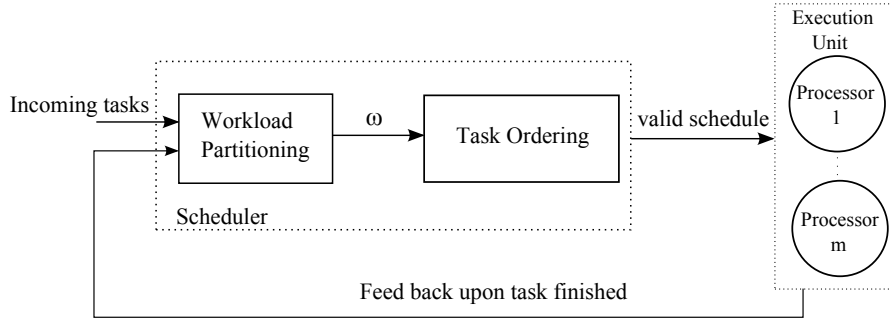


Figure 2.21: Feedback scheduling architecture

partitioning unit and a task ordering unit. By solving (2.10)/(2.11), the workload partitioning unit provides control input ω to the task ordering unit, which then uses Algorithm 1/Algorithm 2 to produce a valid schedule to the execution unit.

2.6.3 Simulation and Results

To evaluate the performance of our feedback scheme, we consider a set of aperiodic tasks to be scheduled on a PowerPC 405LP and an XScale details given in Table 2.2. The energy consumed by executing each taskset shown in Table 2.11 on two different homogeneous systems composed of two processors of the same type were evaluated. The simulation was simulated on MATLAB [92], the LP (2.11) was modelled using OPTI TOOLBOX [93] and solved with SoPlex [80].

For this simulation, we only consider the scheduling event when a task finishes. In other words, we assume that there are no new tasks arriving at the system during the simulation. Three algorithms are compared: (i) Feedback LP-DVFS, which is our

Table 2.11: Feedback Simulation Tasksets

D	Taskset
0.50	(1,5,0),(2,10,0),(1.5,15,0)
0.75	(1,5,0), (3.5,10,0), (3,15,0)
1.00	(2,5,0), (4,10,0), (3,15,0)
1.25	(1,5,0), (6.5,10,0), (6,15,0)
1.50	(2,5,0), (7,10,0), (6,15,0)
1.75	(3,5,0), (7.5,10,0), (6,15,0)
2.00	(4,5,0), (6,10,0), (9,15,0)

LP-DVFS + McNaughton’s wrap around algorithm (Algorithm 1), (ii) Open-Loop LP-DVFS, which is our LP-DVFS without feedback information on finishing tasks, and (iii) No mismatch/Ideal, which is our LP-DVFS with the actual minimum task execution times equal to the estimated, i.e. $\underline{x}_i = \underline{y}_i$. Though McNaughton’s wrap around algorithm does not require the task execution order, for simplicity, our implementation assigns execution priority based on a task’s deadline, i.e. earliest deadline will be executed first.

Figure 2.22 shows results from executing the tasksets in Table 2.11 on two homogeneous multiprocessor systems, composed of two of each processor type, with the estimation factor $\gamma_i = 0.5$, $\forall i \in I$. Note that, the first parameter of a task given in Table 2.11 is $\underline{x}_i$. The vertical axis is the total energy consumption normalised by the Open-Loop LP-DVFS algorithm. It can be seen that for a system composed of PowerPCs, the feedback scheme can save energy up to about 40% compared to an open-loop scheme. However, for a system with XScale processors, the feedback scheme starts to perform better than the open-loop scheme only when the density of the system is half-full. Moreover, the saving percentage of the XScale system is less than that of the PowerPC’s. This is due to the differences in the distribution of speed levels of the two processor types, i.e. the XScale processor has more evenly distributed speed levels than that of the PowerPC; therefore, the optimizer can select the operating speed level that is closer to the optimal continuous speed value.

The results from varying the estimation factor of the taskset with $D = 1.25$ are shown in Figure 2.23. Note that, for this simulation, the estimation factors of all tasks are the same. For a PowerPC system, the energy saving is high when the estimation factor is low. In addition, the difference between the energy consumed by the feedback strategy and the ideal decreases as the estimation factor increases. On the other hand, for an XScale system, the maximum energy saving does not occur when the estimation factor is the lowest, but rather occurs at $\gamma = 0.5$. Furthermore, the energy consumption difference between the feedback and optimal/ideal is larger than that of the PowerPC’s.

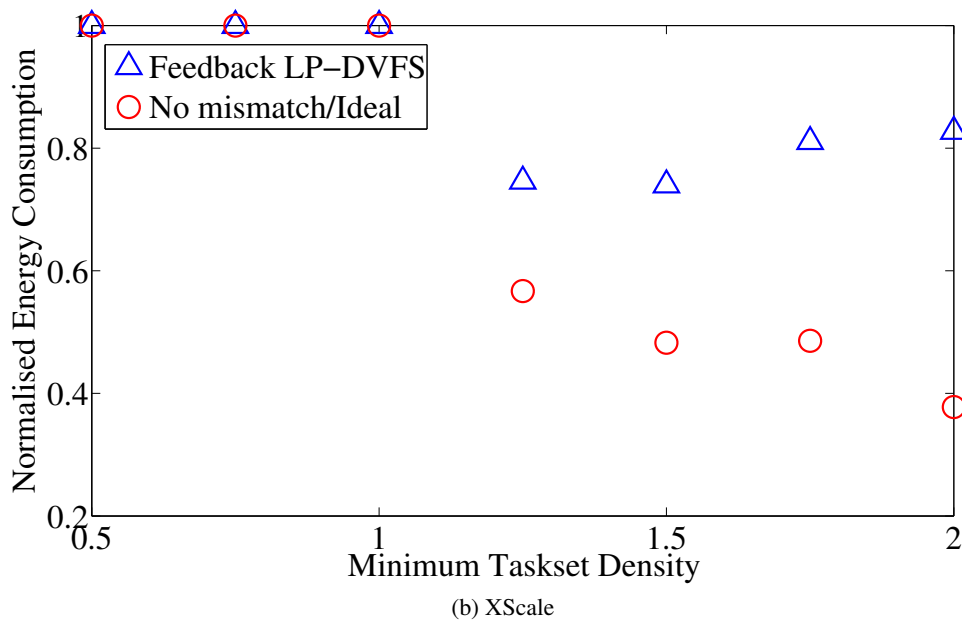
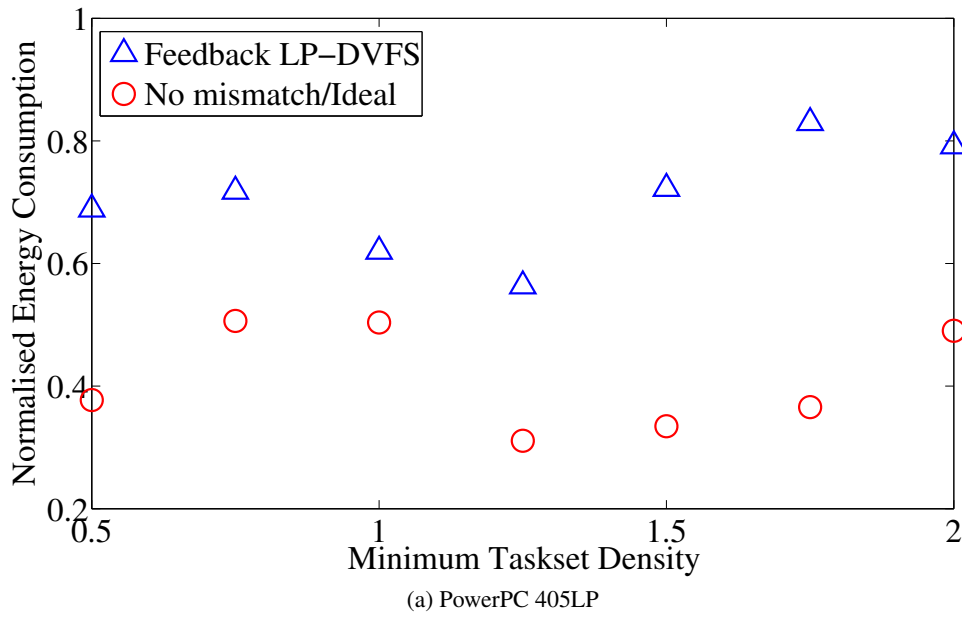


Figure 2.22: Simulation results for different minimum taskset density with $\gamma_i = 0.5$, $\forall i \in I$.

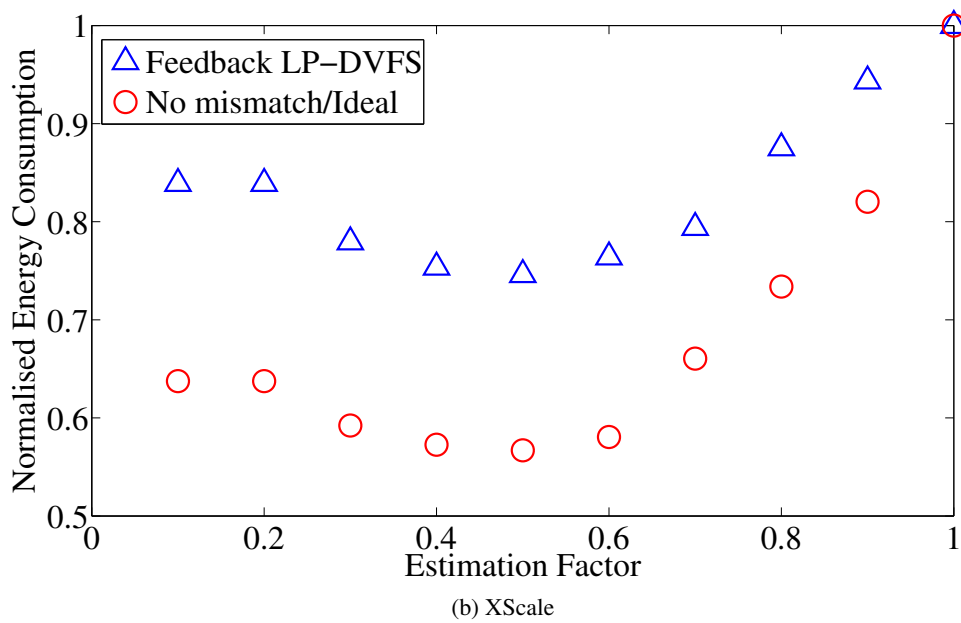
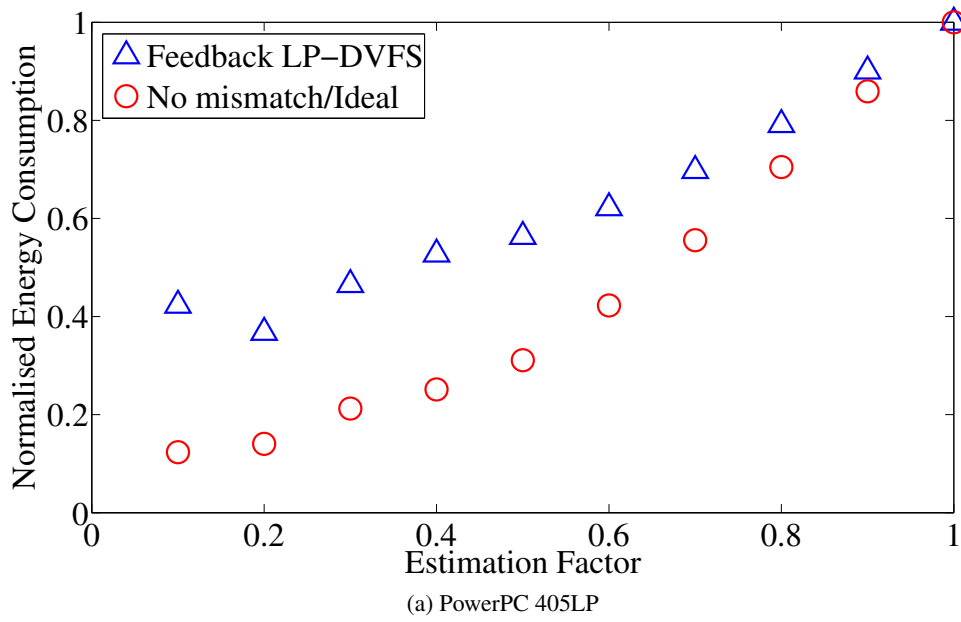


Figure 2.23: Simulation results for different estimation factor γ with $D = 1.25$.

2.7 Conclusions

This chapter presented three novel mathematical optimization formulations, i.e. an MINLP, NLP and LP, to solve the same multiprocessor scheduling problem with the objective of minimizing the total energy consumption. We have shown that the multiprocessor scheduling problem is computationally tractable by first solving the workload partitioning problem and then the task ordering problem. The simulation results illustrate that our algorithms are both feasibility optimal and energy optimal when compared with existing global energy/feasibility optimal workload allocation algorithms. Moreover, we have shown via proof and simulation that a constant frequency scaling scheme is enough to guarantee optimal energy consumption for an ideal system with a constant workload and a convex power function, while this is not true in the case of a time-varying workload or a non-convex power function. For a practical system with discrete speed levels, a time-varying speed assignment is necessary to obtain an optimal energy consumption in general.

A feedback method from control theory was adopted to solve a multiprocessor scheduling problem with uncertainty in task execution times. We have shown that our proposed closed-loop optimal control scheduling algorithm performs better than the open-loop algorithm in terms of energy efficiency. Simulation results suggest that the difference between closed-loop and open-loop performance can be reduced by having more refined distribution of operating speed levels.

Chapter 3

Multi-UAV Network Topology Optimal Control for Information Gathering Applications

This chapter proposes an optimal control problem for mobile agent systems to trade-off between the communication and computation energy. In particular, we present an MINLP formulation for a multi-hop hierarchical cluster-based self-organizing UAV network to attain an energy-efficient reporting mechanism. For this work, we adopt a hierarchical cluster-based data aggregation technique from the wireless sensor network (WSN) field, but the topology of the network and the number of aggregators are dynamically decided. The main contributions of this work are:

- A general MINLP optimization framework that dynamically (i) selects the optimal number of agents to communicate to the base station and perform on-board data aggregation as well as (ii) determines the communication network topology of the multi-UAV system.
- Propose a multiple data-type data aggregation network model, which can be applied to a heterogeneous mobile computing system.
- Address two information gathering applications, namely target tracking and area mapping.
- Compare the performance of the MINLP approach with a baseline approach, where there is no data aggregation and clustering scheme.

The chapter is organized as follows: Section 3.1 provides basic knowledge on WSNs, terminologies and related work. In Section 3.2, the application details are introduced. Details on system dynamic models and constraints are given in Section 3.3. The optimal

control problem formulation is presented in Section 3.4. The simulation details and results of applying the proposed optimal control framework to the two applications are presented in Section 3.5. We conclude in Section 3.6.

3.1 Introduction

Inexpensive mobile agents, such as UAVs, are useful for several remote monitoring applications such as agriculture [94], geology [95], ecology [96] and forestry [97]. The viability of UAVs for scientific and non-military applications are due to reduced cost of the UAVs, low sensor cost, and ease in handling. Typically, these applications are large-scale and the mission time can be shortened by introducing multiple UAVs.

Central to these applications is the necessity of having a human-in-the-loop (HITL) capability that increases situational awareness and operator autonomy to modify missions dynamically. For HITL, UAVs have to gather and disseminate information periodically to the operator who may be located at a distance (base station) from the operational arena. Typical information required at the base station is aerial footage [98], which is a communication intensive operation consuming considerable energy. Unfortunately, low-cost UAVs have limited flight time due to battery/fuel capacity. Hence, there is a need to find different mechanisms by which flight time endurance can be increased. One way is to use gliders that take advantage of the up-drafts to soar for long endurance [99]. However, due to varying UAV height during soaring it is very difficult to maintain a good resolution of the terrain, which is necessary for mapping or surveillance applications. Instead, we propose to optimize the energy consumed by various units in a given aircraft to increase the flight time and hence the UAV team mission time.

For many applications [94, 97], it is necessary that a UAV must fly at a constant speed and maintain a prescribed height. Under these conditions, the major energy consumption units are propulsion, sensing, computation and communication. On average, the power consumed during flight is approximately constant. The sensing and the computational units also consume constant power. However, the energy expended by the communication depends on (i) the amount of data to be transmitted, (ii) the distance between a vehicle and the base station and (iii) the number of vehicles transmitting data to the base station. Moreover, the communication cost is far greater than the sensing and computational energy. For example, a typical sensor node consumes 1 nJ-1 μ J per sample, roughly 1 pJ per instruction for computation, while communicating via radio frequency (RF) is at the cost of 100 nJ-50 μ J per bit [100]. Hence, it is better for the UAVs to co-operate with each other to minimize the team communication energy by

performing computation on-board such that the amount of data to be transmitted is minimized. That is, optimally selecting (a) which vehicles should be the computing nodes and (b) determining how many vehicles are required to communicate with the base station. In this thesis, we propose an MINLP that determines an optimal solution to (a) and (b).

3.1.1 A Multi-UAV System as a Mobile Wireless Sensor Network

A UAV fleet can also be thought of as a mobile wireless sensor network (MWSN), where each UAV can be represented as a sensor node in the network. Therefore, in this section, we will employ fundamental terminologies and concepts widely used in the WSN area.

3.1.1.1 A Mobile Computing Node Architecture

For this thesis, a UAV will be modelled as a mobile computing node, which is composed of three primary modules: a sensor module, a processing module and a wireless communication module, where interactions between modules are shown in Figure 3.1. The detailed description on each module is given below:

- *Sensor module*: The main activities of this module include sensing operation, analogue to digital conversion (AC/DC), and signal modulation.
- *Processing module*: The processing module is responsible for data processing, sensor controlling, and the communication protocol.
- *Wireless communication module*: The wireless communication module is used for data transmitting and receiving. We will assume that there exists a medium access control (MAC) protocol, which allows a UAV to communicate with other UAVs and the base station within a transmission range.

3.1.1.2 Communication Patterns in Wireless Sensor Network

In response to different applications, WSNs offer a wide range of information routing patterns. In general, the two communication patterns in WSNs are either single-hop or multi-hop. In a single-hop network, a sensor node is able to communicate with any other node or base station directly. For a multi-hop network, the communication between two nodes is routed by a sequence of hops through a chain of neighbouring nodes. Depending on how the data is routed and whether the in-network data processed is

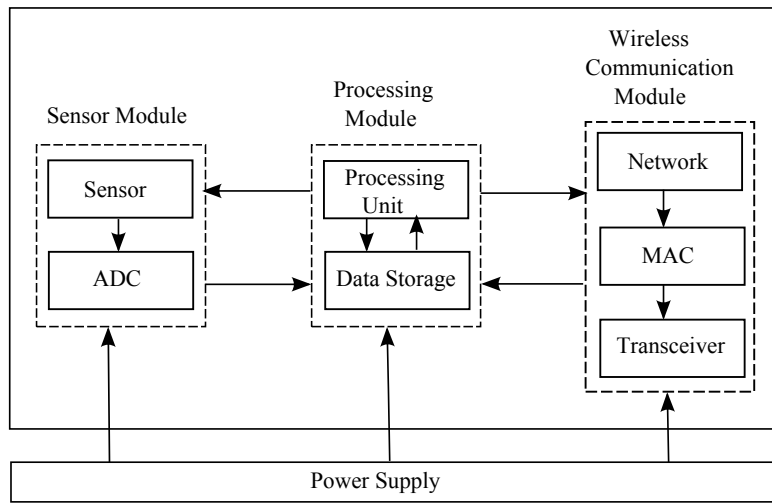


Figure 3.1: The architecture of a mobile node (adapted from [3])

supported, a multi-hop communication pattern can be further divided into the followings (see Figure 3.2) [3]:

- *Local communication*: Typically, this pattern is used for the status broadcasting of the node to its neighbours as well as directly transmitting the data between two nodes.
- *Point-to-point routing*: For this pattern, the data packet can be sent from any node to any node. Wireless LAN is an example of this communication pattern.
- *Convergence*: This pattern, which is typically used in data collection applications, routes the data packets from multiple source nodes to a single destination node.
- *Aggregation*: The raw data packets are processed within the relay node, then the aggregated data is routed to the destination node.
- *Divergence*: This pattern is often used to send the command from base station to the nodes in the network.

3.1.1.3 Hierarchical Network Structure

The structure of the network can be classified according to the connection among the nodes: A *flat network* where all nodes have the same level of connections and a *hierarchical/clustering network*, which deploys different hierarchies of connections [3]. Based on the information routing pattern within the cluster, the hierarchical network can be further classified as single-hop cluster or multi-hop cluster, as shown in Figure 3.3(a) and Figure 3.3(b), respectively. Furthermore, depending on the number of hierarchical levels of clusters, the network structure can be categorized as single-level or multi-level

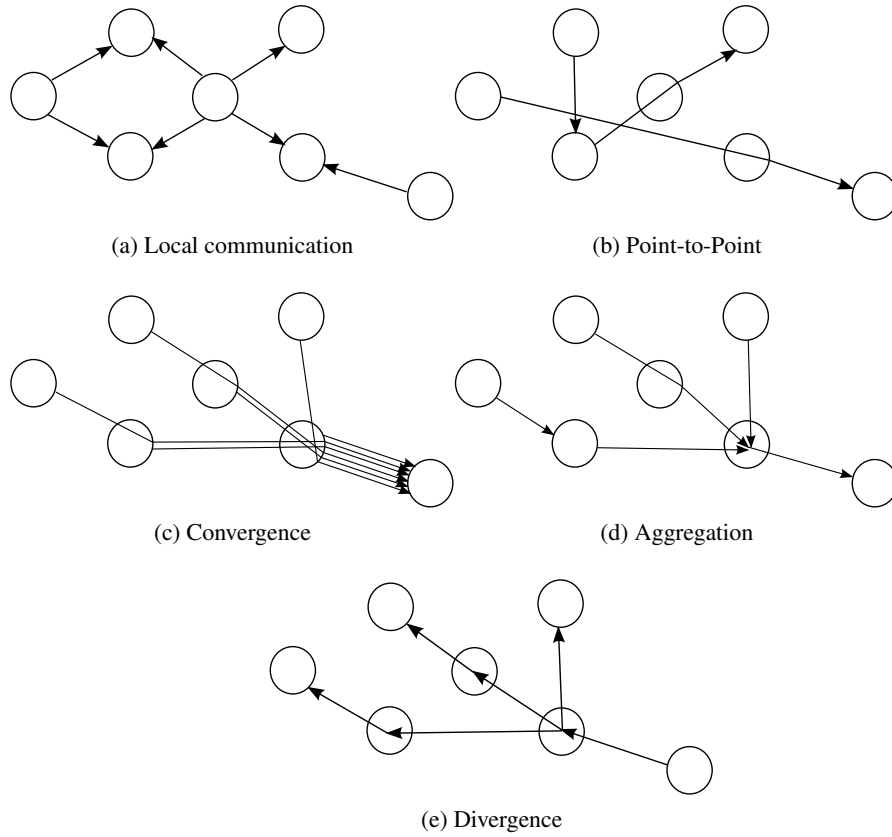


Figure 3.2: Traffic patterns in WSN (redrawn from [3])

clustering. Figure 3.4 illustrates a multi-level hierarchical clustering architecture.

For this work, we are focusing on the hierarchical network structure because it can provide better performances in term of energy consumption, stability and scalability. Since the required power for transmission has an exponential approximate relationship with the transmission distance, by dividing the system into clusters the total energy consumption for communication can be significantly reduced due to the long-distance one-hop communication being replaced by the short-distance multi-hop information routing.

The challenges in designing a clustering-based algorithm rely on the cluster-head selections and cluster organization. There are a few simple heuristic strategies for selecting the cluster-head, for example, choosing the node with the higher residual energy or the nearest node to the base station. However, for this work, the mobile nodes are dynamically self-organized into clusters in which data aggregating process are assumed to be done within the cluster-head node. Additionally, the cluster-heads are selected by solving the optimization algorithm with the objective of minimizing the communication and computation energy.

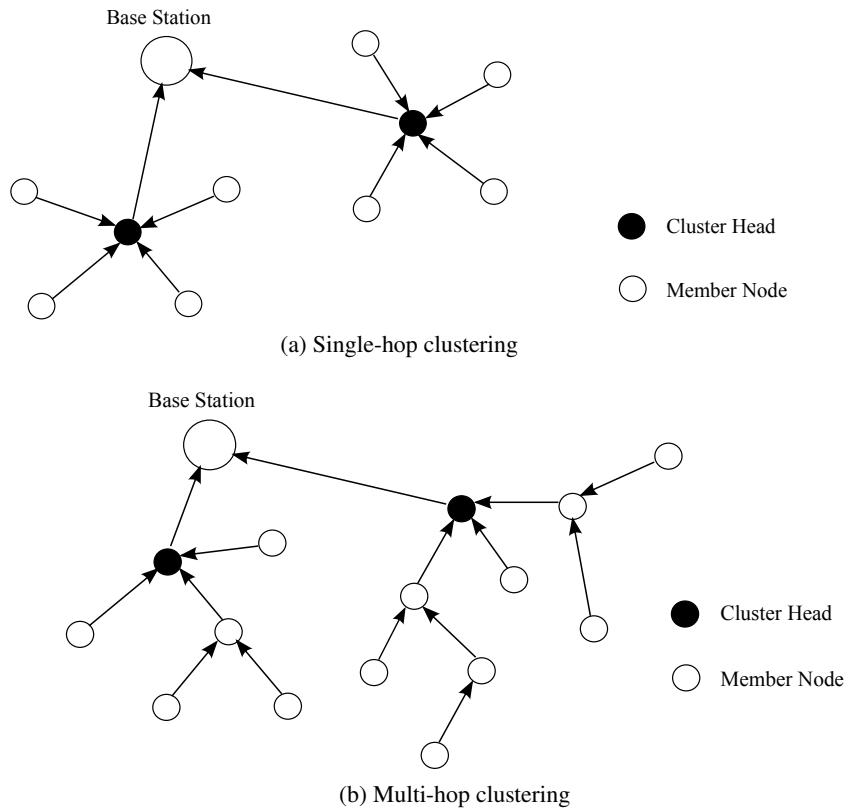


Figure 3.3: Hierarchical network structure

3.1.2 Minimizing Energy Consumption in Wireless Sensor Networks

Due to limited physical size of the sensor node, effectively managing available resources is one of the main challenges in designing wireless sensor nodes. Specifically, the WSN design and implementation are constrained by the following resources: energy, computation power, data storage and communication bandwidth. Wireless sensor nodes are equipped with battery energy supply as well as limited data storage; hence their computational powers are restricted. Moreover, the bandwidth in wireless communication is often low as well. Above all, efficient energy use can significantly extend the lifetime of the network. Figure 3.5 summarizes the energy-saving techniques that have been utilized in the literature for each subsystem of a wireless sensor node. It is well-known that the computation energy is insignificant compared to communication energy, i.e. the energy consumption of communicating 1 bit over the wireless medium could be 1000-10000 times greater than that of processing of the same bit [101]. Therefore, most existing works in the literature mainly target reduction in communication energy, i.e. minimizing the amount of communication data as well as communication distance. Typical approaches to attack the in-network data processing are collaborative signal and information processing (CSIP), and data aggregation. CSIP refers to an emerging interdisciplinary research area that focuses on developing

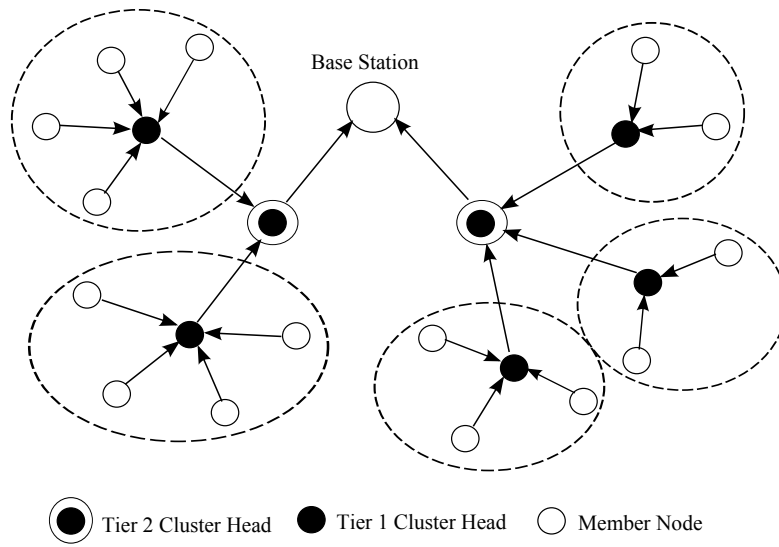


Figure 3.4: Multilevel hierarchical clustering architecture

new methods and algorithms to provide solutions to challenges in information and data processing in WSNs, such as optimizing signal processing and communication, data fusion, and asynchronous distributed data processing [102]. For example, [101] proposed an information-based scheme to solve a target tracking problem in WSNs to illustrate the key issues in CSIP, that is, how to dynamically determine what information needs to be sensed, which node should perform a sensing, how often the node must be communicated, and to whom the node should be communicated with. Another example is the work of [103], where mobile agents are used to carry the raw data from the sensors to the processing units.

Similarly, data aggregation/fusion techniques aim to reduce the amount of communication data in terms of the number and/or the length of information packets through eliminating the redundancy in neighbouring nodes. The main component in data aggregation is to determine how the information should be aggregated. The most straightforward data aggregation scheme eliminates the duplicate information from multiple sources. An example is a scheme called sensor protocols for information via negotiation (SPIN), which is proposed in [104]. A SPIN algorithm introduces the negotiations between the nodes in order to eliminate the transmission of redundant/unnecessary information within the network. A simple minimizing or maximizing function could be adopted as an aggregation function as well. In [105], three heuristic data aggregation schemes for a multiple source and single sink nodes system are proposed, i.e. Centre at Nearest Source (CNS), Shortest Paths Tree (SPT), and Greedy Incremental Tree (GIT). For the CNS scheme, the nearest source to sink node acts as the aggregator. For SPT, the information from the source is routed using the shortest path. Whenever there is overlapping between the paths, the information is combined to form the aggregation tree. Lastly, for GIT, the

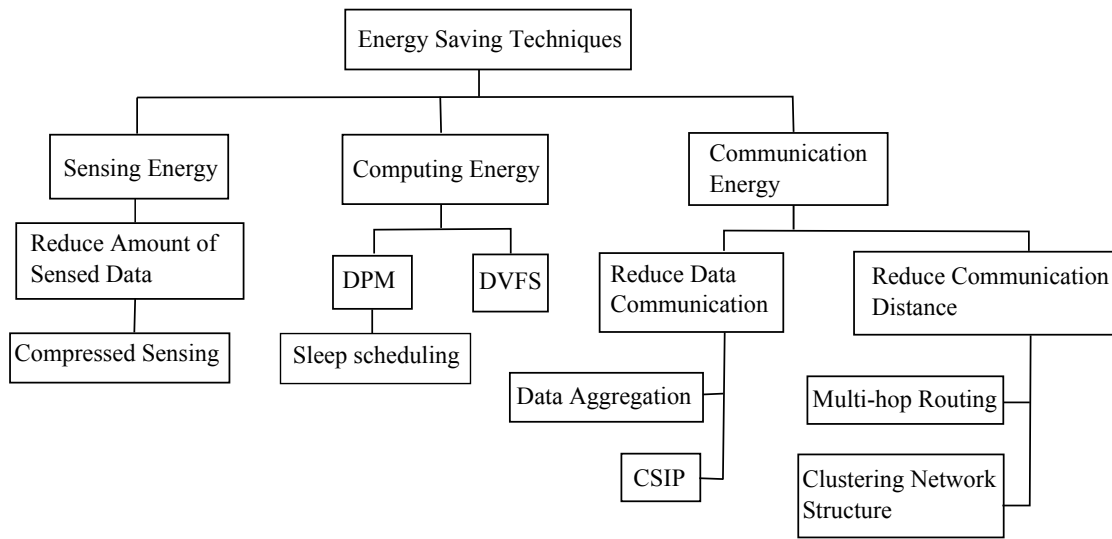


Figure 3.5: Energy saving techniques in WSNs (adapted from [4])

aggregation tree is sequentially built, i.e. at each step, the closest node to the current tree is aggregated. Moreover, data aggregation is often combined with the network clustering scheme [106–108] where the cluster-head node also acts as an aggregator. Noticeably, the above energy minimization techniques can be incorporated within a multi-hop routing scheme and/or hierarchical network architecture.

As for computing units, the aforementioned techniques such as DPM and DVFS can also be adopted in the context of WSNs. For instance, since the workload in WSN applications is typically fluctuating, the sensor nodes or certain computing components of the nodes need not be active at all times. Therefore, by switching the nodes to lower power consumption states while maintaining the performance and functionality, significant amounts of energy could be saved. The problem of determining which nodes/components should be active or inactive is known as the sleep (or duty cycles) scheduling problem in WSNs. However, we will not provide any further details on this topic as it is not the focus of this thesis. Likewise, our work on multiprocessor scheduling algorithms with a DVFS scheme, which is presented in detail in Chapter 2, is also applicable to WSNs.

It can be seen that all the above data aggregation techniques aim to reduce the number of bits needed to represent the sensed data, not the amount of sensed data, because it is assumed that the sensing energy is far less than the communication energy. However, this is not true for energy hungry sensors such as, gas sensors. Therefore, in order to reduce the sensing energy, a compressed sensing (CS) technique, which exploits the rate of information in the signal, was proposed in [109] to minimize the signal redundancy during the sampling process, hence the sensing energy is reduced.

Lastly, it has to be mentioned that an optimal energy management scheme cannot be obtained by applying only one particular technique, since they are highly dependent on each other. For example, incorporating the data aggregation and clustering scheme might reduce the communication energy, but it will increase the price of computational energy in return because a more complex algorithm will be required. Therefore, for this work, we are interested in determining the trade-offs between these techniques such that an optimal energy efficient management scheme is achieved.

3.1.3 Related Work

Similar to our Multi-UAV information gathering problem, the goal of a WSN is to maximize network lifetime while delivering raw data to the sink (base station) [110]. In order to maximize the lifetime of a network, data aggregation techniques have been proposed for WSNs where some computations are performed within the node to reduce the communication cost. It has been shown that by using a sensor node as a communication relay/aggregator, an energy-efficient communication strategy can be obtained [111, 112]. Data correlations between different sensor nodes can be exploited to minimize the number of sensors sending the data to the base station [113]. A compressed sensing technique to reduce the data volume to be transmitted was proposed in [114].

Hierarchical Network Routing is also one of the techniques in prolonging a network lifetime. For this approach, the nodes are grouped into clusters and the cluster-head for each group is selected based on various election algorithms [3]. The cluster-head is responsible for aggregation, compression and forwarding data to the base station. For example, in the Low-Energy Adaptive Clustering Hierarchy (LEACH) protocol proposed in [106], a stochastic scheme is used to determine whether a node will become a cluster-head in each decision making round, i.e. the probability that a node will become a cluster-head is $1/P$, where P is the desired percentage of cluster-heads. The Low-Energy Adaptive Clustering Hierarchy Centralized (LEACH-C) protocol [115], which is an improvement of LEACH, uses global information from the network to determine an optimal number of cluster-heads via a centralized control at the base station. A chain-based protocol, called Power-Efficient Gathering in Sensor Information Systems (PEGASIS), where the nodes are only allowed to communicate with nearby nodes and take turns to transmit data to the base station, was proposed in [116]. A hierarchical data aggregation technique where sensor nodes were grouped into clusters was proposed in [108]. A local aggregator (LA) for each cluster was selected, then a set of master aggregators (MAs) were selected based on LAs. To select MAs, an integer program is solved such that the total communication energy is minimized

while performing minimum aggregation computations, such as finding an average or a maximum.

A Hierarchical Adaptive and Reliable Routing Protocol (HARP), which is based on a self-organized hierarchical tree structure, was proposed in [117]. The difference between HARP and LEACH is that HARP allows multi-hop routing patterns for both inter-cluster and intra-cluster connections, while LEACH keeps a simple single-hop architecture. Furthermore, a simple k-means clustering algorithm [118], where the cluster-head nodes are selected such that the distance from each member node to its nearest cluster-head is minimized, can be used to partition nodes into clusters. A disadvantage of k-means clustering-based algorithms is that the number of clusters has to be pre-determined, hence the number of clusters might not be optimal.

Another approach is to have a mobile sensing node collect data from the nodes to reduce the communication overload [119–122]. Since the UAVs are mobile, using another UAV to collect data from the surveying UAVs is not an ideal approach. However, similar to WSN data aggregation, the UAVs can perform computations on board to produce concise data and periodically transmit to the base station, as in [123] for an image processing application. Data transmission to the base station can be performed either directly or through a UAV relay network [124, 125].

A UAV with sensing capabilities can be applied to perform target tracking due to its adaptability, scalability and better performance than a static wireless sensor network. However, most of the work on UAV target tracking applications focuses only on target tracking accuracy, and neglects communication and computation energy consumption [126–129]. Target tracking algorithms are based on target state estimation. By combining multiple sensor readings, which originated from different moments in time and at different distances from the UAVs, a more accurate state estimate can be obtained [130]. Precisely, the tracking objective is to maximize the information contribution [130, 131] from each node. In general, it has been shown that the measurement obtained from the most distant node does not contribute much to the target tracking accuracy. Therefore, it would be energy-efficient to select only a subset of the UAVs to be tracking nodes. The problem of deciding a subset of tracking sensor nodes could be formulated as an MINLP as in [132], where the observation covariance depends on the distance, i.e. the further away from the target, the less accurate the measurement.

Energy-efficiency in target tracking applications has been extensively studied in the WSN area; see [133] for an up-to-date survey on the topic. In the WSN literature, two classes of methods have been used to achieve energy-efficiency in collaborative WSN-based target tracking applications: a sensing-related method and a communication-

related method. The sensing-related method uses information-driven [134, 135] and data filtering techniques [136] to determine which sensor should sense the information and to which sensor the data should be sent in order to maximize the information value, i.e. target tracking accuracy. The communication-related approach aims at optimizing routing and aggregation techniques [137] as well as network self-organization [138–140] to extend the network lifetime.

UAV mapping applications have primarily focussed on how to use camera vision for 3D mapping of the field [95], terrain vegetation classification [94], infectious disease landscape [141], and road condition assessment [142]. The issue of optimizing energy to enhance mission time is not considered. By using the formulation developed in this paper, the applications can achieve better coverage.

3.2 Application Details

We are looking at the scenario where a team of n UAVs is given a mission to either pursue a single target or survey an area of interest (AOI) and needs to periodically send the data back to the base station.

3.2.1 System Assumptions

We will assume that at each decision making time interval, each UAV (node) $i \in N := \{1, \dots, n\}$ has the same capability of sensing, data aggregation and communication functions, where n is the total number of UAVs in the fleet. A UAV can reach any other UAV using one-hop communication. A sensing UAV periodically senses a target/AOI, i.e. information (a data packet) is generated at a constant rate. The energy for propulsion is separate from communication and sensing and will not be considered. The information can be of different types, therefore our model can be thought of as either a single source or multiple sources with different data types. For simplicity, we will consider a system with only one base station to report the data to. Note that an extension to multiple sink nodes (base stations) is relatively straightforward.

3.2.2 UAV Role Assignment

Following the works of [111] and [143], we will assume that the UAVs can be assigned to one or more of the following roles at each time interval: (i) a sensor, which observes the target/AOI (called node 0) and produces the data that will be relayed to the base station (called node $n + 1$), (ii) a relay, which simply passes its own data to the next

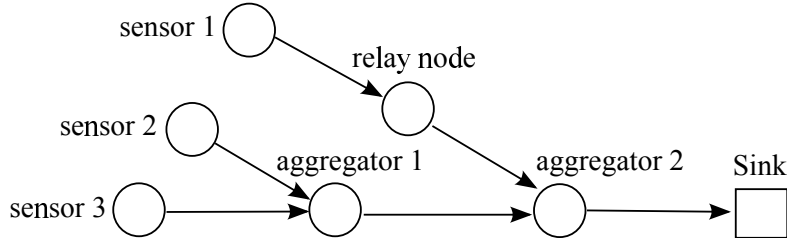


Figure 3.6: Information flow of an aggregation network topology

level node without any processing, or (iii) an aggregator, which receives one or more data from other nodes, then aggregates the received data with its own data to produce a single data package and sends the aggregated data to the next level node.

3.2.3 Aggregation Network Topology

For our targeted applications the obtained information can be highly correlated, hence, using an in-network data aggregation scheme will not only minimize the data redundancy, but also increase energy-efficiency in data routing. Figure 3.6 illustrates the information flow of an aggregation network topology. In particular, the data obtained from the source (target/AOI) can be processed within the aggregator or passed along the relay node and routed to the sink node (base station). Note that, in this work, the network topology is dynamic, i.e. the roles of the UAVs are decided at each decision making time interval.

3.3 Dynamic Model with Constraints

3.3.1 Communication Model and Constraints

Let $C := [c_{ijz}]$ denote a communication link matrix, i.e. $c_{ijz} = 1$ if node i transmits data of type z to node j for $i, j \in N^+ := N \cup \{0, n+1\}$, $z \in M := \{1, \dots, m\}$. Note that $c_{0iz} = 1$ if node i is a sensor of data type z and $c_{i(n+1)z} = 1$ if node i sends data type z to the base station. The communication link matrix C is subject to

$$c_{ijz} \in \{0, 1\}, \quad \forall i \in N^+, j \in N^+, z \in M \quad (3.1)$$

$$\sum_{j=1}^n c_{0jz} \geq 1, \quad \forall z \in M \quad (3.2)$$

$$\sum_{i=1}^n c_{i(n+1)z} \geq 1, \quad \forall z \in M \quad (3.3)$$

$$\sum_{j=1}^{n+1} c_{ijz} \leq 1, \quad \forall i \in N, z \in M \quad (3.4)$$

$$c_{iiz} = 0, \quad \forall i \in N^+, z \in M \quad (3.5)$$

where (3.2)–(3.3) guarantee that for each information type there is at least one communication link from a source to a node and there must be at least one communication link between a node and the base station. Constraint (3.4) enforces that there is only one communication link of each data type out of a node. Constraint (3.5) prevents self communication.

Let $\lambda_{ijz} \geq 0$ denote the average rate (packets per second) at which data of type z is transmitted from node i to node j . Note that λ_{0jz} represents the sensing rate of data type z , assumed to be a constant equal to $\bar{\lambda}_z$ packets per time interval. Following the definition of the communication link matrix C , λ_{ijz} needs to satisfy:

$$\lambda_{ijz} = 0 \Rightarrow c_{ijz} = 0, \quad \forall i \in N^+, j \in N^+, z \in M, \quad (3.6a)$$

$$\lambda_{ijz} > 0 \Rightarrow c_{ijz} = 1, \quad \forall i \in N^+, j \in N^+, z \in M. \quad (3.6b)$$

Constraint (3.6) says that if there is data flow between two nodes, then the link assignment should be active. The constraint (3.6) can be implemented as the following inequality constraints:

$$\epsilon c_{ijz} \leq \lambda_{ijz} \leq |G_z| \bar{\lambda}_z c_{ijz}, \quad \forall i \in N^+, j \in N^+, z \in M, \quad (3.7)$$

where ϵ is a sufficiently small positive number and $|G_z|$ is the total number of sensors of data type z . In other words, suppose $\lambda_{ijz} \neq 0$, then (3.7) is true if and only if $c_{ijz} = 1$. Suppose $\lambda_{ijz} = 0$, then (3.7) is true if and only if $c_{ijz} = 0$.

Denote $a_{iz} \in \{0, 1\}, \forall i \in N, z \in M$ as the data type aggregator assignment, where by definition

$$a_{iz} = 1 \iff \sum_{j=0}^n c_{jiz} > 1, \quad \forall i \in N, z \in M. \quad (3.8)$$

In other words, if there is more than one packet of the same data type transmitted to a node, then the node will act as an aggregator. Constraint (3.8) can be written as a set of linear inequalities as follows:

$$(1 - n)a_{iz} + \sum_{j=0}^n c_{jiz} \leq 1, \quad \forall i \in N, z \in M, \quad (3.9a)$$

$$(1 + \epsilon)a_{iz} - \sum_{j=0}^n c_{jiz} \leq 0, \quad \forall i \in N, z \in M, \quad (3.9b)$$

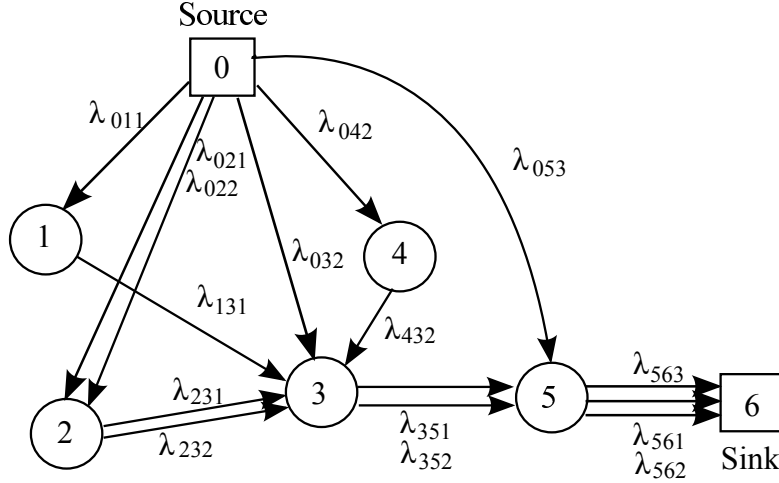


Figure 3.7: Aggregation network topology example

where ϵ is a sufficiently small positive number. To guarantee a feasible communication link, the data flow within the node needs to be conserved, i.e. the incoming data equals the aggregated outgoing data:

$$\sum_{z=1}^m \sum_{j=1}^{n+1} c_{ijz} \lambda_{ijz} = \sum_{z=1}^m \sum_{j=0}^n c_{jiz} \lambda_{jiz} (1 + (\gamma_z - 1) a_{iz}), \forall i \in N, z \in M, \quad (3.10)$$

where $0 \leq \gamma_z \leq 1$ is the aggregation ratio of data type z . Observe that when $\gamma_z = 1$, then there is no data aggregation/processing.

Since the nodes are communicating via wireless network, the channel bandwidth is shared among the nodes. This implies that communication between two nodes restrains available bandwidth to other neighbour nodes. Therefore, bandwidth limitation should be considered in our formulation as well, i.e. all communication data (number of transmitting/receiving bits) should not be greater than the channel bandwidth limitation. Specifically, the bandwidth constraints can be formulated as

$$\sum_{z=1}^m \sum_{j=1}^{n+1} c_{ijz} \lambda_{ijz} L + \sum_{z=1}^m \sum_{j=1}^n c_{jiz} \lambda_{jiz} L \leq Bh, \forall i \in N, \quad (3.11)$$

where B is the channel bandwidth (bits per second), h is the decision time interval and L is the packet length (number of bits per packet).

Finally, we will use an example scenario to show the information flow topology that can be achieved from our model. Consider Figure 3.7 where the system is composed of five UAVs that are given a mission to retrieve three different types of information. Nodes 1, 2 and 4 are sensor nodes, node 3 is both a sensor and an aggregator, while node 5 is a sensor as well as a relay node. The correlated data obtained from node 1 (λ_{131}) and

node 2 (λ_{231}) are processed within node 3. At the same time, the data obtained from nodes 2 (λ_{232}), 0 (λ_{032}) and 4 (λ_{432}) are also processed within node 3. Specifically, from (3.10), the outgoing data flow after the aggregation within node 3: $\lambda_{351} = (\lambda_{131} + \lambda_{231})\gamma_1$ and $\lambda_{352} = (\lambda_{032} + \lambda_{232} + \lambda_{432})\gamma_2$. Both processed data streams/packets are relayed to node 5, which are transmitted to the base station. Note that node 5 acts as a relay node because the data received from node 3 and its own data are of different types.

3.3.2 Energy Models

We will adopt an energy consumption model, which has been commonly used in the wireless sensor network literature [144–146]. The total energy in most multi-UAV applications is composed of three terms. The first term is the sensing energy E^s , which is the energy used to sense a target/AOI. We will assume that the energy to sense one bit of information is a constant equal to ϵ_s J. The sensing energy consumed by node i within the time interval is

$$E_i^s(c_{0iz}) := \epsilon_s L \sum_{z=1}^m \bar{\lambda}_z c_{0iz}, \quad \forall i \in N. \quad (3.12)$$

The second term is the aggregation energy E^p , which is the energy to do data processing. The energy to process one bit of information is also assumed to be a constant equal to ϵ_p J. The aggregation energy consumed by node i within the time interval is

$$E_i^p(c_{jiz}, \lambda_{jiz}, a_{iz}) := \epsilon_p L \sum_{z=1}^m \bar{\lambda}_z c_{0iz} a_{iz} + \epsilon_p L \sum_{z=1}^m \sum_{j=1}^n c_{jiz} \lambda_{jiz} a_{iz}, \quad \forall i \in N. \quad (3.13)$$

The last energy term is the communication cost, which is composed of two parts: the transmitting energy E^t and the receiving energy E^r . The transmitting energy depends on the distance between the nodes d_{ij} , i.e. $E^t(d_{ij}) := \epsilon_t + \epsilon_{rf} d_{ij}^\beta$, where $\beta \geq 2$ is the path loss exponent, ϵ_t (J/bit) and ϵ_{rf} (J/bit/m $^\beta$) are constants. The energy of receiving one bit of information is assumed to be a constant equal to ϵ_r J. The receiving energy consumed by node i within the time interval is

$$E_i^r(c_{jiz}, \lambda_{jiz}) := \epsilon_r L \sum_{z=1}^m \sum_{j=1}^n c_{jiz} \lambda_{jiz}, \quad \forall i \in N. \quad (3.14)$$

The transmitting energy consumed by node i within the time interval is

$$E_i^t(c_{ijz}, \lambda_{ijz}, d_{ij}) := \sum_{z=1}^m \sum_{j=1}^{n+1} (\epsilon_t + \epsilon_{rf} d_{ij}^\beta) c_{ijz} \lambda_{ijz} L, \quad \forall i \in N. \quad (3.15)$$

The total energy used by node i for sensing a target/AOI, processing information and communication during the time interval is denoted by

$$E_i := E_i^s + E_i^p + E_i^r + E_i^t, \forall i \in N. \quad (3.16)$$

Let e_i be the energy stored in the i^{th} UAV at time t , then the remaining energy e_i^+ at time $t + h$ is given by

$$e_i^+ := e_i - E_i \geq 0, \forall i \in N. \quad (3.17)$$

3.3.3 UAV Dynamic Constraints

The two-dimensional UAV kinetic model is given by:

$$\begin{bmatrix} \dot{x}_i \\ \dot{y}_i \end{bmatrix} = f(\varphi_i, v_i, \psi_i) = \begin{bmatrix} v_i \cos \psi_i \\ v_i \sin \psi_i \end{bmatrix}, \forall i \in N, \quad (3.18)$$

where $\varphi_i = [x_i \ y_i]^T$ is the inertial position, v_i is the speed and ψ_i is the heading of the i^{th} UAV. We will assume that UAVs fly at a constant speed and heading in the interval $[t, t + h]$ and are subject to the following constraint:

$$v_{\min} \leq v_i \leq v_{\max}, \forall i \in N, \quad (3.19)$$

where $v_{\min}$ and $v_{\max}$ are lower and upper bounds on speed.

Moreover, since we assume that the UAVs are in one-hop communication range and to avoid collision among UAVs at each time interval, the following constraints are necessary:

$$r_c > d_{ij} \geq r_{safe}, \forall i \neq j, (i, j) \in N \times N, \quad (3.20)$$

where r_c is a sufficiently large positive number defined as a communication range limit, d_{ij} is the distance between two nodes and r_{safe} is the safety distance.

3.3.4 State Update Equation

Let k denote the k^{th} decision making round at time interval $[t_k, t_{k+1}]$, i.e. $t_{k+1} - t_k = h$. The state X_i and the control input u_i for the i^{th} UAV are defined as

$$X_i := (e_i, \varphi_i), \quad \forall i \in N, \quad (3.21)$$

$$u_{ijz} := (c_{0iz}, c_{ijz}, \lambda_{ijz}, a_{iz}, v_i, \psi_i), \quad \forall i \in N, z \in M, j \in N \cup \{n + 1\}, \quad (3.22)$$

where $X := (X_1, X_2, \dots, X_n)$ is the state of the overall system. The components of the overall system control input u are all u_{ijz} , $i \in N$, $j \in N \cup \{n+1\}$, $z \in M$.

Obviously, all the variables in the previous sections can be considered as a function of k . Let $X(k)$ denote the state of the overall system and $u(k)$ denote the system control input at time t_k . The overall system state update equation is given by

$$X(k+1) = \phi(X(k), u(k), k), \quad \forall k, \quad (3.23)$$

where ϕ can be derived from (3.17) and (3.18).

3.4 Optimal Control Problem

We formulate the optimal control problem to determine the roles of the UAVs as an MINLP. We apply this formulation to a multi-UAV target tracking application and a multi-UAV mapping application. The MINLP is solved at each time instant t_k .

D) Target Tracking: Though our main objective is to minimize the total energy consumed by all nodes in the system (3.16), for the target tracking application the target tracking accuracy should be considered as well. Particularly, this can be incorporated as a constraint that guarantees a minimum information contribution $\pi_{\min}$ requirement as

$$\pi := \sum_{z=1}^m \sum_{i=1}^n c_{0iz} \operatorname{tr}\{H_i(t)^T \log(R_i^{-1}(t))H_i(t)\} \geq \pi_{\min}, \quad (3.24)$$

where π is the information contribution, $H_i(t)$ is the observation model and $R_i(t)$ is the measurement noise covariance. Note that our definition of information contribution is slightly different from the one defined and used in [130–132]. Specifically, we took the natural logarithm of the inverse of $R_i(t)$ to reduce the decay rate of information contribution in order to match with the target tracking application using mobile agents, i.e. the useful information can be obtained within a reasonable distance between the sensor and the target.

The sensing range limit can be implemented as the following constraint:

$$c_{0jz}(d_{0j}^2 - r_s^2) \leq 0, \quad \forall j \in N, z \in M, \quad (3.25)$$

where d_{0j} is the distance between the node and the target and r_s is the maximum sensing range. Constraint (3.25) states that if a node is a sensor, then the distance between the node and the target has to be less than the maximum sensing range. Note that the square of the distance is chosen for an easier implementation.

The multi-UAV target tracking problem can be formulated as the following optimal control problem: Given n UAVs, a target and a base station, determine a role for each UAV, a communication network link and a UAV trajectory that solves

$$\begin{aligned} & \underset{u}{\text{minimize}} && \sum_{i=1}^n E_i \\ & \text{subject to} && (3.1)–(3.5), (3.7), (3.9)–(3.20), \\ & && (3.24) \text{ and } (3.25) \end{aligned}$$

II) Area Mapping: Given n UAVs, an AOI, a base station and a UAV trajectory, determine a role for each UAV and a communication network link that solves

$$\begin{aligned} & \underset{u}{\text{minimize}} && \sum_{i=1}^n E_i \\ & \text{subject to} && (3.1)–(3.5), (3.7), (3.9)–(3.17) \text{ and} \\ & && v_i = V_i, \forall i \in N, & (3.26a) \\ & && \psi_i = \Psi_i^d, \forall i \in N, & (3.26b) \\ & && c_{0iz} = C_i, \forall i \in N, & (3.26c) \end{aligned}$$

where V_i is the constant speed of the vehicle and Ψ_i^d is the desired heading angle of the path. C_i is a pre-determined data type sensor assignment vector. Also, note that for an area surveying/mapping application, the UAV dynamic constraints described in Section 3.3.3 are not included because we assume that the trajectory of each UAV and the collision avoidance among UAVs are decided by a path planning controller.

Note that our optimal control problems can be thought of as a network flow problem incorporating with data aggregation scheme. A typical network flow problem is a network optimization problem with the objective of maximizing the total amount of flow from the source node to the sink node and is subjected to two constraints. The first is a flow capacity constraint on communication edge, i.e. the flow on each edge cannot exceed the maximum flow capacity. The second is the network flow conservative constraint, which says that an incoming flow is equal to an outgoing flow. Though, a typical network flow can be formulated as an LP, our optimal control problems cannot due to the data aggregation scheme. Specifically, by allowing data aggregation within the node, the flow conservative constraint is violated. Hence, the data aggregation assignment integer variables cannot be relaxed, resulting in MINLP formulations as shown above.

3.5 Applications

This section provides simulation results to illustrate the correctness and effectiveness of our framework in trading off between communication and computation energy consumptions in multi-UAV applications. For validation purposes, multiple UAV single-target tracking and area mapping applications are chosen as our demonstration examples. All simulations were simulated on MATLAB [92] and the MINLP was modelled using OPTI TOOLBOX [93] and solved with SCIP [79].

3.5.1 Target Tracking

3.5.1.1 Target and Sensor Models

For a target tracking application, we will follow the work of [132] to set up the optimization problem to make a decision on a subset of the UAVs to be sensor nodes. The motion of a target will be modelled as a linear discrete-time Markov process:

$$X_0(t+1) = F_0(t)X_0(t) + w_0(t), \quad (3.27)$$

where $X_0(t)$ is the state vector of a target, $F_0(t)$ is the state transition matrix and $w_0(t)$ is the process noise assumed to be zero mean Gaussian noise with covariance $Q_0(t)$.

The measurement equation of a sensor is

$$Z_i(t) = H_i(t)X_0(t) + v_i(t), \quad (3.28)$$

where $v_i(t)$ is the measurement noise assumed to be zero mean Gaussian with covariance $R_i(t)$. We will assume that the measurement noise covariance is a function of the distance between a sensor and a target, i.e. $R_i(t) := K(t)d_{0i}^\beta(t)$, where $K(t)$ is a distance-independent coefficient, and $d_{0i}(t)$ is the distance from a sensor to a target. Moreover, we will also assume that the measurement noise covariances are uncorrelated between any two nodes.

3.5.1.2 Information Filter

For multi-sensor data fusion, we use an information filter [132, 147], which is an inverse covariance form of the Kalman filter. Let $\hat{X}_0(t|t)$ and $\hat{X}_0(t+1|t)$ denote the target estimated state vector and target predicted state vector, respectively. Define the information matrix $Q(t|t) := P^{-1}(t|t)$ and $Q(t+1|t) := P^{-1}(t+1|t)$, the information state vector $\hat{q}(t|t) := P^{-1}(t|t)\hat{X}_0(t|t)$ and $\hat{q}(t+1|t) := P^{-1}(t+1|t)\hat{X}_0(t+1|t)$, where $P(t|t)$ and

$P(t+1|t)$ are the covariances of the estimation error $X_0(t|t) - \hat{X}_0(t|t)$ and the prediction error $X_0(t+1|t) - \hat{X}_0(t+1|t)$. The prediction and estimation steps are

Estimation:

$$\hat{q}(t|t) = \hat{q}(t|t-1) + H_i^T(t)R_i^{-1}(t)Z_i(t), \quad (3.29)$$

$$Q(t|t) = Q(t|t-1) + H_i^T(t)R_i^{-1}(t)H_i(t), \quad (3.30)$$

Prediction:

$$\hat{q}(t+1|t) = Q(t+1|t)F_0(t+1)Q^{-1}(t|t)\hat{q}(t|t), \quad (3.31)$$

$$Q(t+1|t) = (F_0(t+1)Q^{-1}(t|t)F_0^T(t+1) + Q_0(t+1))^{-1}. \quad (3.32)$$

For multi-sensor data fusion, i.e more than one node tracking the target, (3.29) and (3.30) are replaced, respectively by

$$\hat{q}(t|t) = \hat{q}(t|t-1) + \sum_{i \in S} H_i^T(t)R_i^{-1}(t)Z_i(t), \quad (3.33)$$

$$Q(t|t) = Q(t|t-1) + \sum_{i \in S} H_i^T(t)R_i^{-1}(t)H_i(t), \quad (3.34)$$

where S is a set of sensor nodes.

3.5.1.3 Simulation Setup

For simplicity, we consider a small UAV network, i.e. $n = 3$, which is deployed to track a single target in a two-dimensional area and needs to periodically report the target state back to the base station. Note that we consider the single target state as one data type. The base station is at (0,0). The initial positions of the UAVs are at positions (0,100), (100,0), and (100,100). The target initial position is (20,20). The target state vector $X_0(t)$ in (3.27) is composed of the target positions in the x and y axes, and velocities in the x and y axes, denoted as v_x and v_y , respectively. The parameters corresponding to the target state (3.27), measurement equations (3.28) and information filter are [132]:

$$F_0(t) = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad Q_0(t) = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 0.04 & 0 \\ 0 & 0 & 0 & 0.04 \end{pmatrix}, \quad \forall t$$

$$H_i(t) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}, \quad K(t) = \begin{pmatrix} 1 \times 10^{-6} & 0 \\ 0 & 1 \times 10^{-6} \end{pmatrix}, \quad \forall t$$

$$\hat{q}(1|0) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad Q(1|0) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

For all simulations, we let the target velocities be $v_x = 10$ m/s $v_y = 15$ m/s. The UAV parameters [148] are $v_{\min} = 10$ m/s, $v_{\max} = 30$ m/s, the initial UAV energy budget is 10 J, the communication range $r_c = 500$ m, the sensing range $r_s = 200$ m, the safety distance $r_{safe} = 50$ m, the decision time interval h is 1 s. The energy parameters [111] are $\epsilon_s = 50$ nJ/bit, $\epsilon_p = 10$ nJ/bit, $\epsilon_r = 135$ nJ/bit, $\epsilon_t = 45$ nJ/bit, $\epsilon_{rf} = 0.1$ nJ/bit/m², $\gamma_z = 0.7$, $\beta = 2$, $L = 1024$ bits/packet, $\bar{\lambda}_z = 5$ packets/time interval and $\pi_{min} = 6$.

3.5.1.4 Simulation Results

We compare the results obtained from the MINLP with a baseline strategy where all sensor nodes individually communicate with the base station using a single-hop communication protocol. The UAVs and the target trajectories are shown in Figure 3.8. The comparison is performed in terms of energy consumed per decision time interval $[t, t + h]$ between the MINLP and the baseline strategy. The vertical axis in Figure 3.9 represents the system energy consumption per decision time $[t, t + h]$ normalized by the baseline scheme. Similar to the observations in [149], which studies the impact of bandwidth constraints of the energy consumption of WSN, our simulation also suggests that the channel bandwidth constraint has an effect on the energy consumption of the system. This is due to the restriction on the information flow pattern. Specifically, when the bandwidth is limited below the threshold value of 5 Kbps (not shown on the plot), the MINLP algorithm cannot find a solution that is better than the baseline strategy, hence no energy saving can be obtained. However, when the channel bandwidth is above the threshold, the MINLP can provide an optimal strategy that can save energy consumption up to 40% compared to the baseline strategy, as shown in Figure 3.9. However, the energy saving improvement cannot be observed with an increase in $B > 6$ Kbps. Figure 3.10 shows the aggregator role assignments of each UAV at each time instance of the simulation, where 1 refers to an active role.

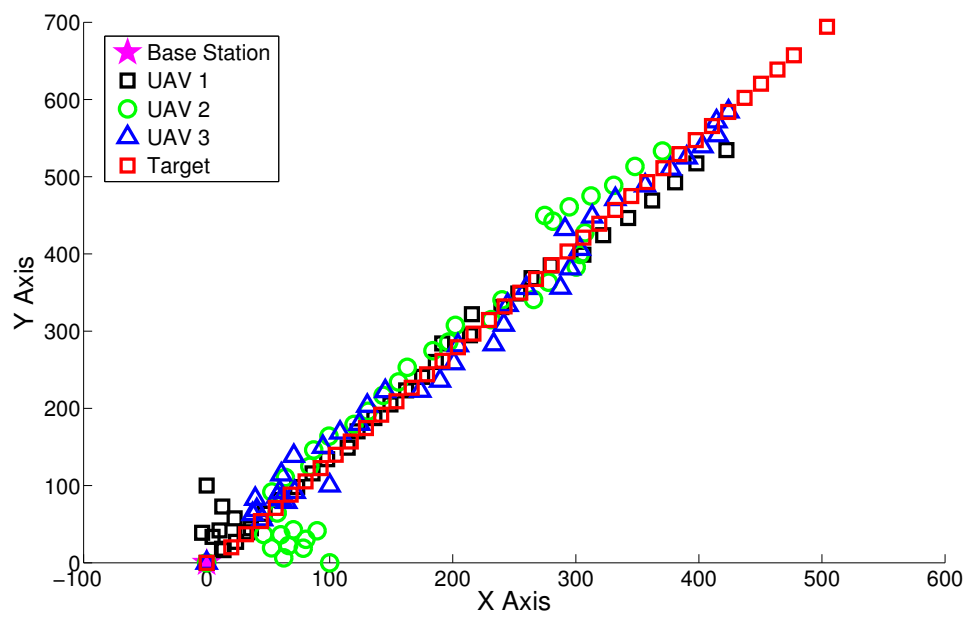


Figure 3.8: Simulation trajectory plot

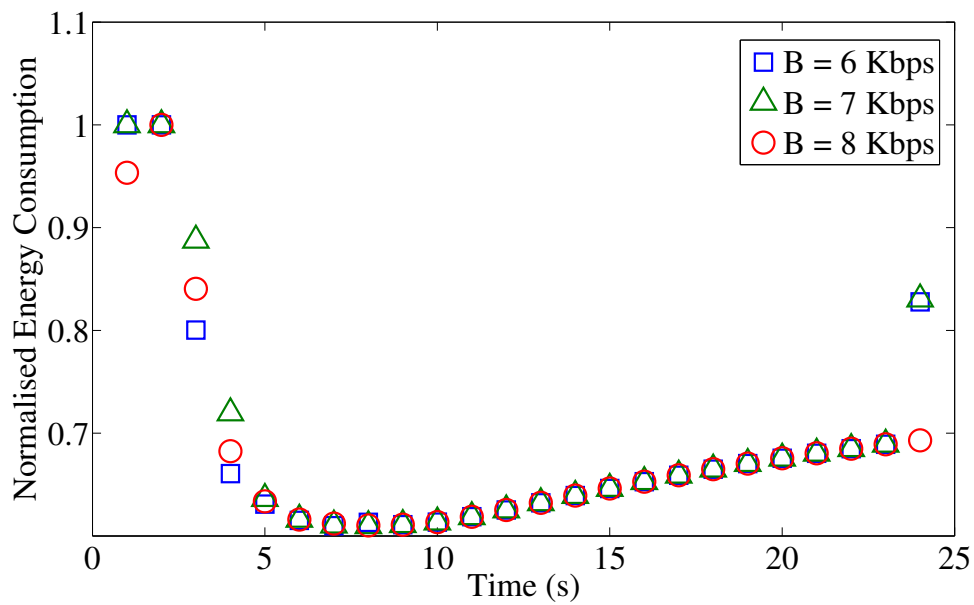


Figure 3.9: Normalised total energy consumption for different channel bandwidths with respect to baseline scheme

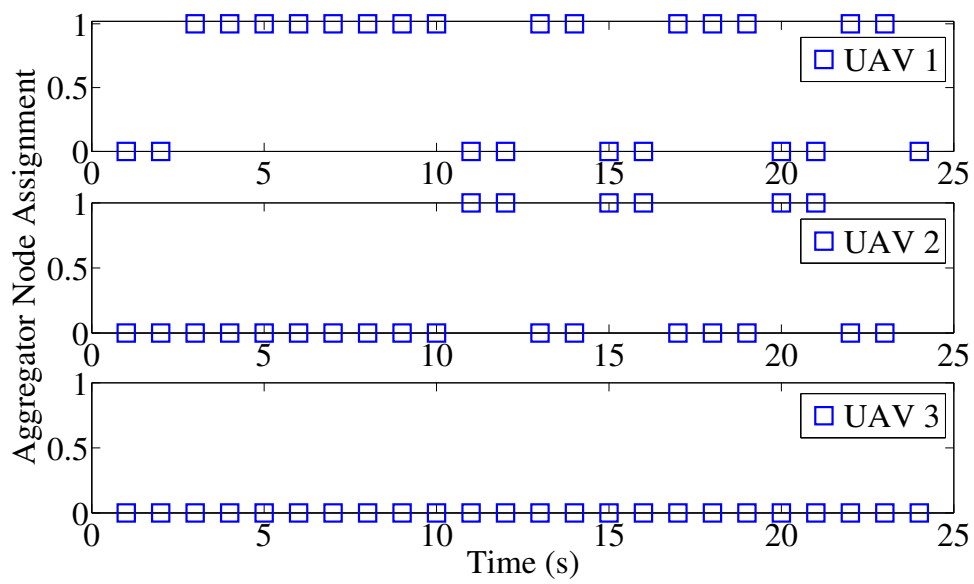


Figure 3.10: Aggregator node assignments at different time steps for channel bandwidth $B = 7$ Kbps

3.5.2 Area mapping

A team of n UAVs are to be deployed to survey a rectangular region with a length of T meters and a width of W meters using cameras. The vehicles are subject to communication, sensing and energy constraints. Each UAV has a sensing range of r_s meters determined by the camera resolution and altitude. Typically, mapping applications are performed using a lawn-mowing pattern and hence we split the rectangular region into lanes of width ζr_s , where $0 < \zeta \leq 1$ is the overlap factor. $\zeta = 1$ implies the distance between the lanes is r_s and there is no overlap of sensing regions between the aerial survey of UAVs, while $0 < \zeta < 1$ implies there is some overlap of the sensor footprint between two adjacent lanes. In terms of area coverage $\zeta = 1$ is the best strategy. However, for mapping purposes, there must be at least 50% overlap between two lanes to create good mosaics [150]. We assume a linear relationship between the overlap factor and the data aggregation ratio, i.e. $\zeta = \gamma$, which means that the higher the overlapping area, the higher the data reduction after data processing. Note that here we assume that the overlap factor is a constant and the same for all nodes, therefore the subscript z of γ notation is dropped. The number of lanes are $N_\ell := \lceil \frac{T}{2\zeta r_s} \rceil + 1$ and each lane is denoted by $\ell_\kappa, \kappa = 1, \dots, N_\ell$. The vehicles use way-point navigation for the survey and hence each lane ℓ_κ is represented by two way-points $\ell_\kappa = (\omega_\kappa^b, \omega_\kappa^t)$, where, $\omega_\kappa^b = (x_\kappa^b, y_\kappa^b), \omega_\kappa^t = (x_\kappa^t, y_\kappa^t)$ as shown in Figure 3.11. Lane ℓ_κ can be accurately tracked using any accurate path following algorithm [151].

The time taken by the UAV team to survey the complete region depends on the number of UAVs deployed; when $n = 1$, the lower bound on the mission time is WTN_ℓ seconds. Initially, UAV i is given a lane $\ell_i, i \in N$ in terms of the waypoints $\ell_i = (\omega_i^b, \omega_i^t)$. Once the vehicle reaches ω_i^t , the lane $\ell_{i+n} = (\omega_{i+n}^t, \omega_{i+n}^b)$ is assigned. However, we can see that UAV i was assigned the waypoint sequence (ω_i^b, ω_i^t) for the first lane while $(\omega_{i+n}^t, \omega_{i+n}^b)$ was assigned the next lane. If we assigned $(\omega_{i+n}^b, \omega_{i+n}^t)$, then the vehicle has to travel from ω_i^t to ω_{i+n}^b , which is unproductive travel, since the vehicle expends fuel without surveying any of the region. Hence, we assign the UAV with an alternating sequence of waypoints.

The desired heading angle ψ_i^d is determined as

$$\psi_i^d = \begin{cases} \arctan(y_\kappa^b - y_\kappa^t, x_\kappa^t - x_\kappa^b) & \text{if } \ell_\kappa = (\omega_i^b, \omega_i^t) \\ \arctan(y_\kappa^b - y_\kappa^t, x_\kappa^b - x_\kappa^t) & \text{if } \ell_\kappa = (\omega_i^t, \omega_i^b). \end{cases} \quad (3.35)$$

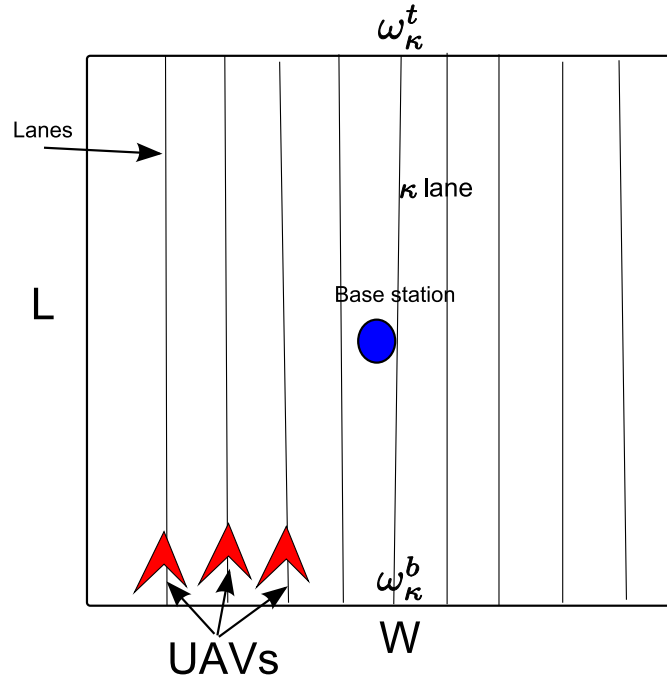


Figure 3.11: The search area is decomposed into lanes and each UAV is assigned to one lane. Once the UAV completes one lane, then another lane is assigned.

3.5.2.1 Simulation Setup

We consider a region of $3000\text{ m} \times 3000\text{ m}$ and the base station is located in the middle at $(1500, 1500)$. The sensing range of the vehicles $r_s = 100\text{ m}$, the communication range $r_c = 500\text{ m}$ and the speed of the vehicles is 10 m/s . We assume three vehicles are deployed to perform the mapping. The parameters used in the simulation are $\epsilon_s = 50\text{ nJ/bit}$, $\epsilon_p = 10\text{ nJ/bit}$, $\epsilon_r = 135\text{ nJ/bit}$, $\epsilon_t = 45\text{ nJ/bit}$, $\epsilon_{rf} = 0.1\text{ nJ/bit/m}^2$, $\beta = 2$, $L = 1280 \times 720\text{ bits/packet}$ and $\bar{\lambda}_z = 5\text{ packets/time interval}$. Each UAV communicates to the base station every $h = 5\text{ seconds}$. The vector field based path following algorithm [152] is selected as the UAV path planning controller. The vector field based path following approach uses a two-fold strategy. When the vehicle is far away from the desired path, the algorithm directs the vehicle towards the path until the vehicle is τ meters from the path as shown in Figure 3.12, where the parameter τ is the transition boundary between moving towards the path and following the path. The vehicle then transits into following the desired path with an entry angle of χ . The effects of τ and χ are well studied in [152] and [121]. For all simulations, we use $\tau = 20\text{ meters}$ and $\chi = \pi/3\text{ rad}$.

For the mapping application, the values for c_{0iz} depend on the distance between the nodes. That is, if the distance is greater than twice that of the sensing range r_s , then we will assume that the sensing data are not related and cannot be aggregated. In other words, the data are of different types. In order to illustrate how c_{0iz} values are

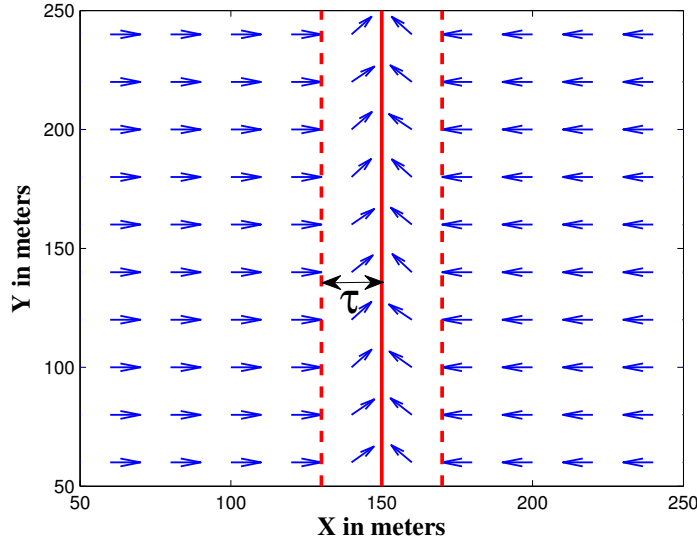


Figure 3.12: The path is given by waypoint (150,0) and (150,300). The vector field of the vehicle at various locations is shown. $\tau = 20$ and $\chi = \pi/3$.

determined at each decision interval, consider a three vehicle system in Figure 3.13a where a distance between node i and node j $d_{ij} > 2r_s$. For this scenario, there is no common data type between the nodes due to no overlap of the sensed regions, i.e. $z \in \{0, 1, 2\}$. Therefore, the values of $c_{010} = 1, c_{011} = 0, c_{012} = 0, c_{020} = 0, c_{021} = 1, c_{022} = 0, c_{030} = 0, c_{031} = 0$ and $c_{032} = 1$, which implies that none of the nodes have common data type.

Now consider the scenario as shown in Figure 3.13b, where nodes 1 and 2 have a common data type $z = 0$ and node 3 is distant from nodes 1 and 2. Therefore, in this case, we have $c_{010} = 1, c_{011} = 0, c_{020} = 1, c_{021} = 0, c_{030} = 0$ and $c_{031} = 1$. Similar to this scenario, if node 2 and 3 have a common data type $z = 1$, while node 1 is distant from nodes 2 and 3, then $c_{010} = 1, c_{011} = 0, c_{020} = 0, c_{021} = 1, c_{030} = 0$ and $c_{031} = 1$.

Another scenario is where one of the nodes may have two common data types, as shown in Figure 3.13c in which node 2 shares data with node 1 and node 3, but node 1 and node 3 are far from each other and do not have common data. In this case, we set $c_{010} = 1, c_{011} = 0, c_{020} = 1, c_{021} = 1, c_{030} = 0$ and $c_{031} = 1$. The last scenario is where all nodes are within $2r_s$ distance of each other as shown in Figure 3.13d. In this case, $c_{010} = 1, c_{020} = 1$ and $c_{030} = 1$. Thus, depending on the node positions and overlap regions, the values c_{0iz} are pre-determined at the beginning of each decision interval. For our simulations, we consider the scenarios shown in Figure 3.13c for $\zeta < 0.75$, and Figure 3.13d for $\zeta > 0.75$.

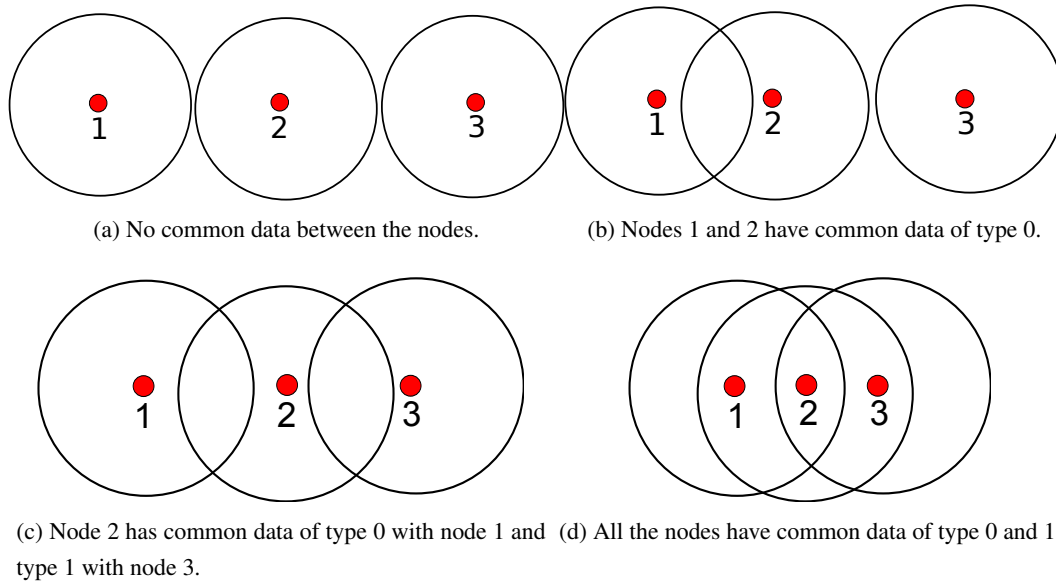


Figure 3.13: Example scenarios with a three vehicle system to illustrate the common data type configurations.

3.5.2.2 Simulation Results

The bandwidth allocated to communicate with the base station plays a key role in determining the computing nodes. Figure 3.14 shows the total energy consumption of the MINLP normalised to the baseline strategy for every $h = 5$ seconds with an overlap factor $\zeta = 0.5$. When the available bandwidth is less than 6 Mbps (not shown on the plot), the nodes communicate directly to the base station. Hence, we do not show this effect. However, when we increase the bandwidth, data aggregation behaviours can be observed. As shown in Figure 3.14, the energy saving is close to 20% for most of the decision cycles (for $B = 6$ Mbps). With further increase in bandwidth to $B = 10$ Mbps, we can see that there is further increase in energy saving of 35%. However, with increase in bandwidth beyond 10 Mbps no further improvement in energy saving was observed. As expected, the energy reduction is due to co-operation among the agents, i.e. when the bandwidth is sufficiently large, more energy-efficient feasible information flow patterns are allowed.

In the mapping application, the overlap factor ζ plays a key role in determining the amount of information that needs to be transmitted by the aggregator node to the base station. When ζ increases, the agents are close to each other with high overlap. Therefore, during the mosaic operation, the resultant image size will be smaller compared to the sum of individual images. In order to validate this hypothesis, we carried out experiments with different overlap factors $\zeta = 0.3, 0.5, 0.7$ and 0.9 for the

same bandwidth of 10 Mbps. In Figure 3.15, we can see the effect of ζ for a given bandwidth. Specifically, the energy saving increases as ζ increases. For example, when $\zeta = 0.9$, we can achieve savings of up to 60% compared to the baseline strategy.

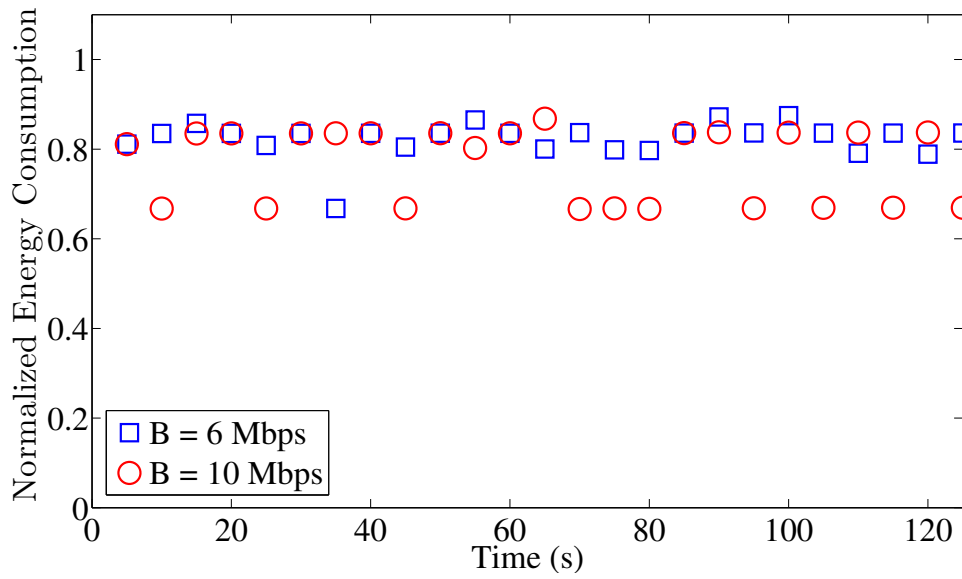


Figure 3.14: The normalized total energy of the MINLP compared to the baseline strategy for different bandwidth constraints having $\zeta = 0.5$.

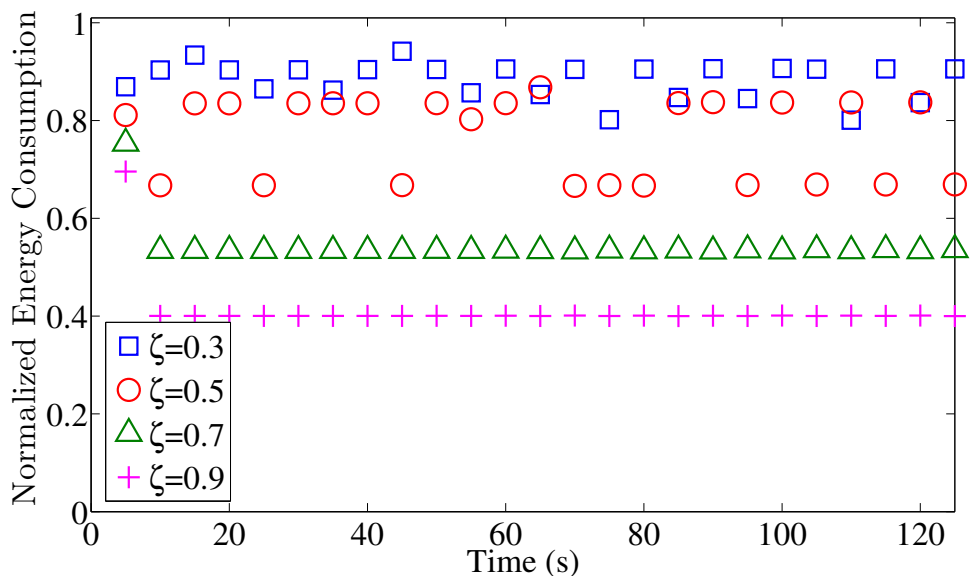


Figure 3.15: The normalized total energy of the MINLP with reference to the baseline strategy for different overlap factors having a channel bandwidth of $B = 10$ Mbps.

3.6 Conclusions

Cooperation between mobile computing agents enables them to optimize the computation and communication energy consumption, thereby increasing the system lifetime. We have devised an MINLP formulation that shows lower energy consumption by incorporating data aggregation and clustering schemes. The MINLP formulation is generic and we utilized this generality by validation on two data gathering applications, namely target tracking and mapping. We have studied the effect of different parameters on MINLP decision-making. Simulation results show that the channel bandwidth has a direct impact on the energy-saving scheme, i.e. sufficient bandwidth is necessary for implementation of an intelligent information routing scheme.

The proposed MINLP formulation can be further extended to optimize the energy consumption of various units. One potential direction is to make a decision on when to communicate to the base station. Currently, we assume that the decision interval is fixed. However, depending on the amount of data, channel bandwidth and the transceiver energy properties, the decision cycle can be dynamically selected to optimize the overall energy consumption.

Chapter 4

Conclusions and Future Work

4.1 Summary of Thesis Achievements

This section provides a summary of the main contributions presented in this thesis.

4.1.1 Real-Time Multiprocessor Scheduling

- The formulation of a real-time multiprocessor scheduling as an infinite-dimensional continuous-time optimal control problem was presented.
- An MINLP formulation framework was proposed to solve a hard real-time multiprocessor scheduling problem with DVFS capabilities. Two sub-problems were proposed to solve the same scheduling problem, i.e. by partitioning the task workload and then finding the task execution order. A computationally tractable NLP and LP are proposed to solve the workload partitioning problem for an ideal system and a practical system, respectively.
- We provide an optimal speed profile solution to a uniprocessor scheduling problem with real-time taskset. That is, (i) for any given workload within the time interval, the optimal speed profile composes of at most two speed levels. (ii) For a processor with a convex power consumption model, the optimal speed profile is a constant. (iii) In general, the time-varying speed profile is better than a constant speed profile if the power function is not convex.
- We have produced a multiprocessor scheduling algorithm that is both feasibility optimal and energy optimal when compared with both existing heuristic algorithms and global energy/feasibility optimal workload allocation algorithms.

- Simulation results illustrate the possibility of an energy saving of up to 70% for a homogeneous system and 40% for a heterogeneous multiprocessor system with arbitrary constrained deadline tasksets, using our algorithms.
- Our formulations are capable of solving a multiprocessor scheduling problem with any periodic taskset characteristics, i.e. an implicit, a constrained and an arbitrary deadline taskset. In other words, our formulations can solve a more general class of scheduling problems compared to existing work, due to the incorporation of a scheduling dynamic as well as the time-varying speed profile.
- Though we made an assumption that processor frequencies can be individually adjusted on our formulations, a stricter assumption of a multicore architecture, where a frequency change is only allowed at a cluster-level, can be easily incorporated into our formulations.
- The proposed algorithms can be applied to both an online scheduling scheme, where the characteristics of the taskset is not known until the time of execution, and an offline scheduling scheme, where the taskset information is known a priori.
- We proposed an optimal control framework that takes advantage of feeding back information upon finished tasks to solve a real-time multiprocessor scheduling problem with uncertainty in task execution times.
- Although our proposed multiprocessor scheduling framework was designed for a real-time system, it can also be applied at a higher level of abstraction such as a data centre, a server farm or cloud computing as well as other generic scheduling problems, such as multi-stage multi-machine problems.

4.1.2 Multi-UAV Network Topology Optimal Control

- A general MINLP optimization framework for a multi-UAV network to optimally trade-off between the communication and the computational energy was presented. That is, to dynamically determine: (i) the optimal number of agents to communicate to the base station, (ii) the role of each UAV: a sensor, a relay or an aggregator, (iii) the communication links among the UAVs to obtain an energy-efficient information routing network with data aggregation.
- Our data aggregation network model exploits three benefit characteristics: (i) a self-organizing network, which means that the topology of the network is dynamically decided at each decision time interval, resulting in more flexible and reliable network, (ii) a multi-hop network, which exploits the shorter communication distance to prolong the lifetime of the network and (iii) a

hierarchical clustering network, which can provide a better performance in terms of energy consumption, reliability as well as scalability.

- A generalised data aggregation network model that allows multiple flows of more than one data type within the network. In other words, our network model can be applied to a heterogeneous mobile computing system, where only the same data types are allowed to be aggregated/processed, i.e. a system with more than one sensor type.
- Two information gathering applications, namely target tracking and area mapping are addressed by our proposed optimal control framework to illustrate both the correctness and the effectiveness in trading off communication and computation energy.
- Simulation results show an energy saving of up to 40% for target tracking and 60% for area mapping when comparing the performance of our MINLP formulation with a baseline approach, where there is no data aggregation and clustering scheme.

4.2 Future Work

4.2.1 Real-Time Multiprocessor Scheduling

- **Joint Optimization of DVFS and DPM:**

DVFS and DPM techniques have been widely used as an energy management scheme in modern computing systems. However, most existing literature only considers these techniques separately. However, in order to achieve an optimal energy management scheme at a system level, it is necessary to integrate both DVFS and DPM into a single framework, because the two techniques counteract each other. In particular, DVFS scheme executes at a lower frequency, results in less energy consumption, but a longer execution time. In other words, this limits the possibility of setting the system to a lower-power state. On the other hand, the processor might have to operate at a higher frequency in order to gain a longer idle period for applying the DPM technique, obviously at the cost of higher CPU dynamic energy consumption and state transition energy. Figure 4.1 shows the impact of CPU frequency on the time for remaining in an idle state (S0i1), i.e. the idle/sleep time increases as the CPU frequency increases. There are few works [5, 153, 154] that consider the two techniques together in order to study the trade-off. With the scheduling frameworks proposed in this thesis it is possible to incorporate DPM, i.e. in addition to making decisions on the operating speed,

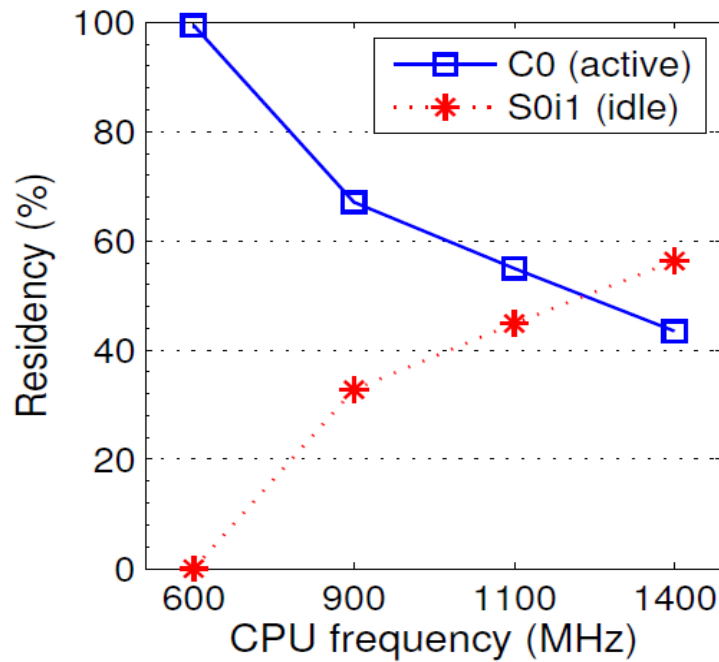


Figure 4.1: Impact of operating frequency on idle state residency (reprint from [5])

task assignment and task execution order. We also need to decide on when and how long the processor should be in active/idle state in order to minimize the total energy consumption.

- **Leakage Power-Aware/Thermal-Aware Scheduling:**

As chips become smaller, static power consumption, which is mainly from leakage power, cannot be neglected, because as the frequency decreases the ratio of static power consumption to total power consumption increases. The static power consumption can be approximated as a linear function of speed. Therefore, reducing the number of active cores during low workloads may result in lower total energy consumption, leading to a problem of deciding how many cores will be used for different workloads. The idea of shutting down unnecessary cores is a common practice in today's multicore processor, as can be seen from Figure 4.2 where the system under low workload should not utilize all of its cores. For example, the power consumption of using 2 cores under a task load less than 0.8 is significantly lower than using 8 cores. Thus, the problem of deciding which core to shut down or how many cores are needed is also interesting to investigate since even in a large data centre, it is unlikely that all of its processors will be needed concurrently.

Though this thesis has focused on minimizing the total energy consumption of a system, the problem of how to manage the heat generated by a system is also

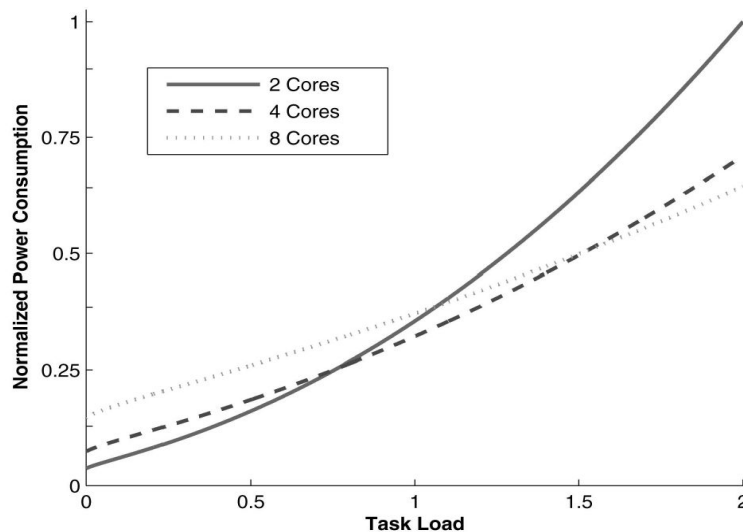


Figure 4.2: Power consumption as a function of workload for various multicore processors (reprinted from [6])

one of the main concerns of computing system design today. This is especially true for deep sub-micron technologies, since the temperature of the chip has an impact on system performance and reliability as well as on packaging/cooling costs. Despite the fact that reducing energy/power consumption may reduce system temperature, it has been shown in [7] that the solution provided by solving for energy optimization does not always result in an optimal thermal solution, as shown in Figure 4.3. In the figure, energy consumption increases as chip temperature decreases, since there is a nonlinear relationship between the leakage power, the temperature and the supply voltage [36]. The problem of determining this trade-off can be incorporated into our energy optimization formulation either as a constraint on maximum peak temperature or as a joint multi-objective function.

- **Generalised Workload Models:**

In this thesis, we only consider a CPU intensive task model. However, the power consumption varies with different types of workload, i.e. CPU intensive, memory intensive, I/O intensive tasks or a mixture of tasks [8]. Figure 4.4 illustrates that with the same CPU utilization workload, different types of task, e.g. floating point operations, integer operations, memory cycle intensive operations, consume different amounts of energy. Thus, this aspect can also be incorporated into the task model as well as into the total power consumption model.

- **Intra-task Parallelism/ Data-dependency Task Model**

Many real-time applications can be modelled as a finite number of independent

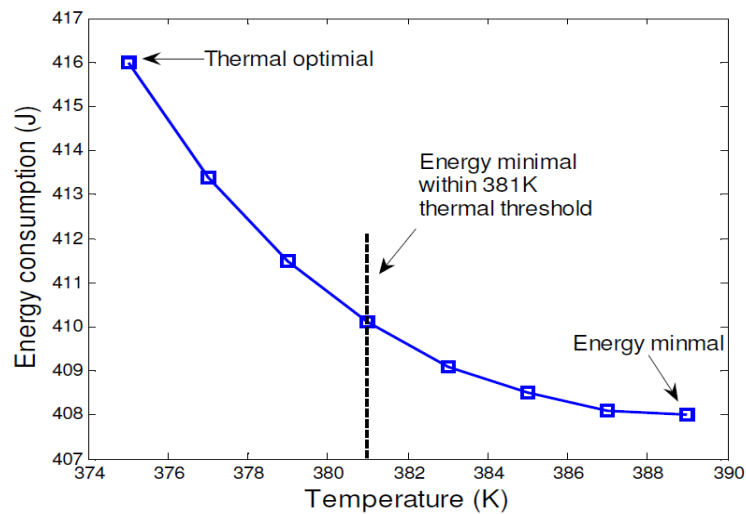


Figure 4.3: Example plot of energy vs temperature for some benchmark taskset (reprinted from [7])

sequential task models, i.e. where intra-task parallelism is not allowed, as defined in our thesis. However it is not possible to exploit the full potential of a multiprocessor/multicore architecture where parallelism is a main contribution. Therefore, in recent years, there has been an effort in the real-time community to propose a suitable model for intra-parallelism tasks as well as to develop scheduling for such tasks. For example, a parallel task can be modelled as a synchronous task (Figure 4.5b), which consists of a sequence of segments [9]. In particular, each segment has an arbitrary number of parallel execution workloads (often called threads) that will be synchronised at the end of the segment and a segment cannot start executing unless the previous segment has finished. To address the parallel implicit-deadline task scheduling problem, [9] proposed a task decomposition algorithm that decomposes each implicit deadline thread of the parallel synchronous task at each segment into a set of sequential constrained deadline tasks, i.e. by assigning each thread with an intermediate deadline (less than or equal to the task deadline). Following this, a conventional scheduling algorithm, such as a global earliest deadline first scheduling algorithm or a partitioned deadline monotonic scheduling algorithm, is adopted to find a valid schedule of the transformed constrained deadline taskset.

A more generalised task model can be represented as a directed acyclic graph (DAG), shown in Figure 4.5a, where a vertex represents a sequential job and a directed edge corresponds to a precedence constraint, i.e. if there is an edge from vertex 1 to vertex 2, then the job associated with vertex 2 only begins when the job associated with vertex 1 has finished. Jobs (vertices) that are not connected

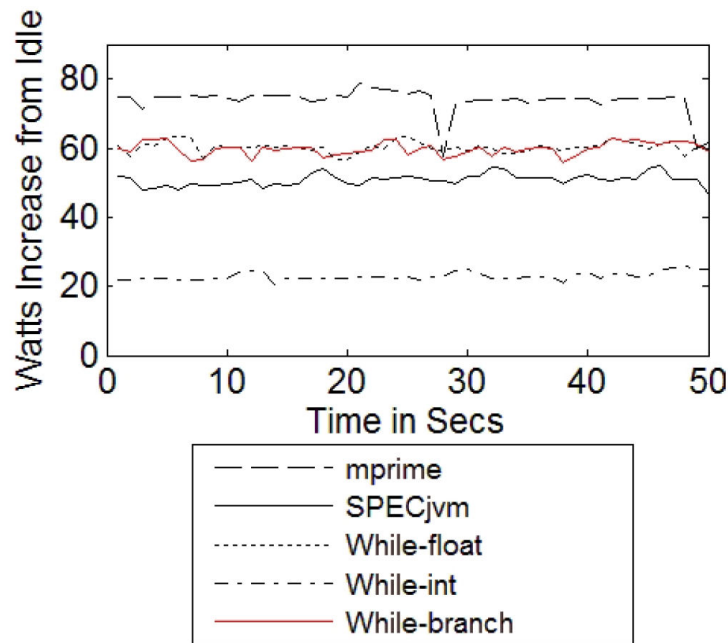


Figure 4.4: Energy vs task types (reprinted from [8])

may be executed in parallel. It has been illustrated in [9, 155] that the DAG can be transformed to a synchronous task (see Figure 4.5 as an example). Therefore, the approach described earlier can also be extended to the DAG task model. Likewise, our scheduling algorithms can also be applied after the task decomposition step to obtain an optimal scheduling. However, the question of how to optimally decompose parallel tasks has not yet been answered.

- **Communication-Aware Scheduling for Distributed Systems**

For a multiprocessor system where multiple processors/cores are adjacently connected or even on the same chip, we can assume that the intra- and inter-task communications, as well as the communication energy, can be neglected, which is a typical assumption in the real-time literature, including ours. However, for distributed computing systems such as cloud computing, wireless sensor networks or mobile computing networks, the actual communication tasks need to be considered in the model as well, because where the task is scheduled has an impact on the communication energy consumption. For example, if the same task is split and assigned to different distributed computing nodes, then depending on the distance between the processing nodes, the communication energy can vary. In other words, communication decisions among tasks can have a significant effect on task scheduling decisions and vice versa. Thus, the computation scheduling problem and the communication scheduling problem have to be solved simultaneously in order to obtain an optimal trade-off [156]. It is simple to model

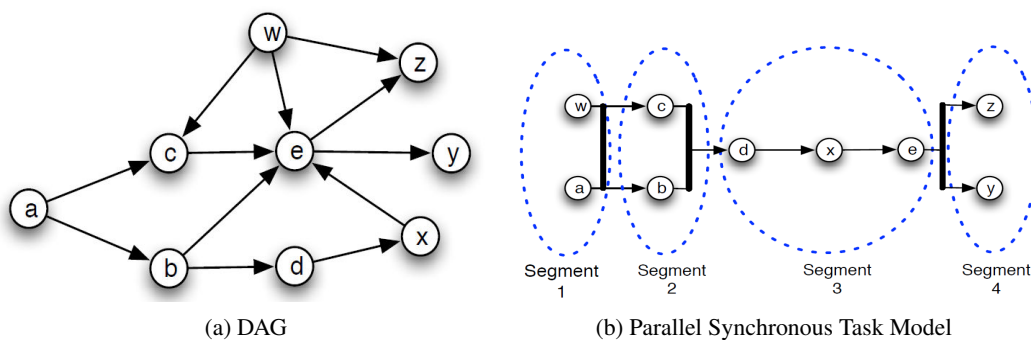


Figure 4.5: Example of a DAG transforming to a parallel synchronous task model (reprinted from [9])

an inter-task communication, i.e. by assigning a communication constraint and cost on the edge of the DAG task model or explicitly adding communication-task vertices to a typical DAG model [145, 157]. However, how best to model the intra-task communication, which only exists when there is an inter-task assignment, is difficult. Therefore, how to incorporate a communication scheduling model into our formulations is quite a challenge.

4.2.2 Multi-UAV Network Topology Optimal Control

- **Practical Communication Network Model**

For the work presented in this thesis, perfect communication among the mobile computing nodes is assumed. In particular, (i) the communication channels are available at all times, (ii) the channels are strongly connected, meaning that any UAV can communicate with any other UAV using one-hop communication and (iii) the mobile node is able to finish transmitting its data within the decision time interval. However, these assumptions might not hold in a practical environment. Therefore, an explicit communication protocol, as well as how to schedule the communication tasks, has to be clearly addressed and incorporated into the proposed framework.

- **Energy Balancing for Network lifetime Extension**

Besides energy minimization, maximizing the lifetime of the network is the main consideration when deploying the sensor network for practical applications. There are many interpretations of the network lifetime. For this discussion, we will define the lifetime of the network as the maximum time until the first node in the network runs out of its battery/energy. For this work, the main objective is to minimize the total energy consumption of the network. However, the network lifetime can be reduced as the total energy consumption is minimized [158]. For

example, to reduce the communication energy, the data is extensively routed via the node closest to the base station, resulting in this node being the first node to exhaust its energy budget due to over-utilization. In other words, the optimal solution to an energy minimization problem is not necessarily an optimal solution to a network lifetime maximization. Balancing energy consumption among nodes at each time interval is often used as the solution to prolonging the network lifetime [159–161]. For instance, two objective functions are adopted in [160] for multi-objective programming of a network lifetime extension problem formulation, i.e. the sum of all variances of energy consumption of each node and the sum of energy of consumption of each node. Likewise, these objective functions could be incorporated into our proposed framework, but the problem of determining a suitable objective function for data aggregation mobile computing systems is a subject of further study.

- **Centralised vs Decentralised Control**

For this work, the proposed frameworks are designed with a centralized control assumption, i.e. it is assumed that either the base station or one of the UAVs is selected as a centralised administrator. However, a centralised control might not be the best implementation option for an energy efficient information routing network of a large number of distributed mobile computing nodes, because of the communication overhead due to global information demands. Therefore, for a large WSN, nodes can be grouped into clusters with a small number of member nodes to construct a hierarchical network of clusters, then our algorithm can be applied at an intra-cluster level as well as an inter-cluster level to obtain an energy-efficient routing scheme. Nevertheless, the problem of how to optimally select cluster-heads and partition the network also needs to be considered.

- **Hybrid Network Topology Control**

Besides the clustering scheme, there are two other techniques that have been used in WSN to minimize the energy consumption and maximize the life time of the network, i.e. power adjustment and power mode approaches [162]. The power adjustment approach allows a node to adjust its transmission power to make decisions on the receiving node in order to minimize the reduced energy in transmitting. On the other hand, the power mode approach saves the energy consumption by selecting different modes of operation of a sensor, i.e. coordinating the sleep, idle, transmit and receive scheduling of the nodes. Therefore, by combining different approaches, a more energy-efficient management scheme can be obtained. For example, a combined framework of a clustering approach and a power adjusting approach was proposed in [163] to obtain a trade-off between energy cost and number of communication hops. Specifically, the framework

applied a cluster-based algorithm to determine the number of cluster heads and used the power control-based algorithm to determine the connections among nodes. Moreover, by incorporating the power mode approach, the number of nodes participating in the information routing scheme at each interval can be determined to save more energy. In brief, for our proposed framework, which only exploits the clustering scheme, additional energy savings can be achieved by integrating power adjusting and power mode approaches.

References

- [1] E. C. Kerrigan, “Feedback and time are essential for the optimal control of computing systems,” in *Nonlinear Model Predictive Control, 2015. 5th IFAC conf.*, Sep. 2015, pp. 380–387.
- [2] H. Cho, B. Ravindran, and E. Jensen, “An optimal real-time scheduling algorithm for multiprocessors,” in *Real-Time Systems Symp., 2006. RTSS '06. 27th IEEE Int.*, Dec. 2006, pp. 101–110.
- [3] N. Pantazis, S. Nikolidakis, and D. Vergados, “Energy-efficient routing protocols in wireless sensor networks: A survey,” *Communications Surveys Tutorials, IEEE*, vol. 15, no. 2, pp. 551–591, Feb. 2013.
- [4] Q. Wang and H. Hassanein, “A comparative study of energy-efficient (E^2) protocols for wireless sensor networks,” in *Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems*, M. Ilyas and I. Mahgoub, Eds. Boca Raton, FL, USA: CRC Press, Inc., 2004, ch. 18, pp. 1–22.
- [5] A. Min *et al.*, “Joint optimization of DVFS and low-power sleep-state selection for mobile platforms,” in *Communications (ICC), 2014 IEEE Int. Conf.*, Jun. 2014, pp. 3541–3546.
- [6] E. Seo *et al.*, “Energy efficient scheduling of real-time tasks on multicore processors,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 11, pp. 1540–1552, Nov. 2008.
- [7] Y. Liu *et al.*, “Thermal vs energy optimization for DVFS-enabled processors in embedded systems,” in *Quality Electronic Design, 2007. ISQED '07. 8th Int. Symp.*, Mar. 2007, pp. 204–209.
- [8] A. Jaianthilal, Y. Jiang, and S. Mishra, “Modeling CPU energy consumption for energy efficient scheduling,” in *Proc. 1st Workshop on Green Computing (GCM '10)*, Nov. 2010, pp. 10–15.

- [9] A. Saifullah *et al.*, “Multi-core real-time scheduling for generalized parallel task models,” in *Real-Time Systems Symp. (RTSS), 2011 IEEE 32nd*, Nov. 2011, pp. 217–226.
- [10] W. Wolf, “Cyber-physical systems,” in *Computer*, vol. 42, no. 3. Los Alamitos, CA, USA: IEEE-CS, Mar. 2009, pp. 88–89.
- [11] J. Hellerstein, S. Singhal, and Q. Wang, “Research challenges in control engineering of computing systems,” *IEEE Trans Netw. Service Manag.*, vol. 6, no. 4, pp. 206–211, Dec. 2009.
- [12] R. Urgaonkar *et al.*, “Dynamic resource allocation and power management in virtualized data centers,” in *Network Operations and Management Symp. (NOMS), 2010 IEEE*, Apr. 2010, pp. 479–486.
- [13] Z. Wang, X. Zhu, and S. Singhal, “Utilization and SLO-based control for dynamic sizing of resource partitions,” in *Ambient Networks*, ser. Lecture Notes in Computer Science, J. Schönwälder and J. Serrat, Eds. Springer Berlin Heidelberg, 2005, vol. 3775, pp. 133–144.
- [14] X. Zhu, Z. Wang, and S. Singhal, “Utility-driven workload management using nested control design,” in *American Control Conf.*, Jun. 2006, pp. 6033–6038.
- [15] M. Kjaer, M. Kihl, and A. Robertsson, “Resource allocation and disturbance rejection in web servers using SLAs and virtualized servers,” *IEEE Trans. Netw. Service Manag.*, vol. 6, no. 4, pp. 226–239, Dec. 2009.
- [16] Y. Diao *et al.*, “Incorporating cost of control into the design of a load balancing controller,” in *Real-Time and Embedded Technology and Applications Symp., 2004. Proc. RTAS 2004 10th IEEE*, May 2004, pp. 376–385.
- [17] Z. Tang *et al.*, “Resource-constrained load balancing controller for a parallel database,” *IEEE Trans. Control Syst. Technol.*, vol. 16, no. 4, pp. 834–840, Jul. 2008.
- [18] P. Bhoj, S. Ramanathan, and S. Singhal, “Web2K: Bringing QoS to Web Servers,” May 2000. [Online]. Available: <http://www.hpl.hp.com/techreports/2000/HPL-2000-61.pdf>
- [19] M. Wang *et al.*, “Adaptive performance control of computing systems via distributed cooperative control: Application to power management in computing clusters,” in *Autonomic Computing, 2006. ICAC '06 IEEE Int. Conf.*, Jun. 2006, pp. 165–174.

- [20] D. Kusic *et al.*, “Power and performance management of virtualized computing environments via lookahead control,” in *Autonomic Computing, 2008. ICAC '08. Int. Conf.*, Jun. 2008, pp. 3–12.
- [21] D. Cavendish, M. Gerla, and S. Mascolo, “A control theoretical approach to congestion control in packet networks,” *IEEE/ACM Trans. Netw.*, vol. 12, no. 5, pp. 893–906, Oct. 2004.
- [22] C. Lu *et al.*, “Feedback control real-time scheduling: Framework, modeling, and algorithms*,” *Real-Time Syst.*, vol. 23, no. 1/2, pp. 85–126, Jul. 2002.
- [23] D. R. Sahoo *et al.*, “Feedback control for real-time scheduling,” in *Proc. American Controls Conf.*, May 2002, pp. 1254–1259.
- [24] A. Soria-Lopez, P. Mejia-Alvarez, and J. Cornejo, “Feedback scheduling of power-aware soft real-time tasks,” in *Computer Science, 2005. ENC 2005. 6th Mexican Int. Conf.*, Sep. 2005, pp. 266–273.
- [25] Y. Zhu and F. Mueller, “Feedback EDF scheduling of real-time tasks exploiting dynamic voltage scaling,” *Real-Time Syst.*, vol. 31, no. 1-3, pp. 33–63, Dec. 2005.
- [26] S. K. Baruah *et al.*, “Proportionate progress: A notion of fairness in resource allocation,” in *Proc. 25th Annu. ACM Symp. Theory of Computing (STOC '93)*, 1993, pp. 345–354.
- [27] K. Funaoka, S. Kato, and N. Yamasaki, “Energy-efficient optimal real-time scheduling on multiprocessors,” in *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE Int. Symp.*, May 2008, pp. 23–30.
- [28] K. Funaoka *et al.*, “Dynamic voltage and frequency scaling for optimal real-time scheduling on multiprocessors,” in *Industrial Embedded Systems, 2008. SIES 2008 Int. Symp.*, Jun. 2008, pp. 27–33.
- [29] W. Vereecken *et al.*, “Overall ICT footprint and green communication technologies,” in *Communications, Control and Signal Processing (ISCCSP), 2010 4th Int. Symp.*, Mar. 2010, pp. 1–6.
- [30] Hewlett-Packard, Intel, Microsoft, Phoenix, and Toshiba, “*Advanced Configuration and Power Interface Specification (ACPI)*,” 2010. [Online]. Available: <http://www.acpi.info/DOWNLOADS/ACPIspec50.pdf>
- [31] C. S. R. Murthy and G. Manimaran, *Resource management in real-time systems and networks*. Cambridge, MA, USA: The MIT Press, 2001.

- [32] R. I. Davis and A. Burns, "A survey of hard real-time scheduling algorithms and schedulability analysis techniques for multiprocessor systems," Department of Computer Science, University of York, Tech. Rep., Nov. 2009.
- [33] J. Leung, L. Kelly, and J. H. Anderson, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Boca Raton, FL, USA: CRC Press, Inc., 2004.
- [34] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: exact characterization and average case behavior," in *Real Time Systems Symp., 1989, Proc.*, Dec. 1989, pp. 166–171.
- [35] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *JACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.
- [36] J. M. Rabaey, A. P. Chandrakasan, and B. Nikolic, *Digital integrated circuits : a design perspective*, 2nd ed., ser. Prentice Hall electronics and VLSI series. Pearson Education, Jan. 2003.
- [37] D. Li and J. Wu, *Energy-aware Scheduling on Multiprocessor Platforms*, ser. Springer Briefs in Computer Science. Springer New York, 2013.
- [38] A. Miyoshi *et al.*, "Critical power slope: Understanding the runtime effects of frequency scaling," in *Supercomputing (ICS '02), Proc. 16th Int. Conf.*, Jun. 2002, pp. 35–44.
- [39] J.-J. Chen, H.-R. Hsu, and T.-W. Kuo, "Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems," in *Real-Time and Embedded Technology and Applications Symp., 2006. Proc. 12th IEEE*, Apr. 2006, pp. 408–417.
- [40] H. Aydin, V. Devadas, and D. Zhu, "System-level energy management for periodic real-time tasks," in *Real-Time Systems Symp., 2006. RTSS '06. 27th IEEE Int.*, Dec. 2006, pp. 313–322.
- [41] R. Kumar *et al.*, "Heterogeneous chip multiprocessors," *Computer*, vol. 38, no. 11, pp. 32–38, Nov. 2005.
- [42] D. Duarte *et al.*, "Impact of scaling on the effectiveness of dynamic power reduction schemes," in *Computer Design: VLSI in Computers and Processors, 2002. Proc. 2002 IEEE Int. Conf.*, 2002, pp. 382–387.
- [43] B. Zhai *et al.*, "Theoretical and practical limits of dynamic voltage scaling," in *Design Automation Conf., 2004. Proc. 41st*, Jul. 2004, pp. 868–873.

- [44] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 8, no. 3, pp. 299–316, Jun. 2000.
- [45] D. Brodowski, "CPUFreq Governors," 2010. [Online]. Available: <http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.33.1.tar.bz2>
- [46] J.-J. Chen and C.-F. Kuo, "Energy-efficient scheduling for real-time systems on dynamic voltage scaling (DVS) platforms," in *Embedded and Real-Time Computing Systems and Applications, 2007. RTCSA 2007. 13th IEEE Int. Conf.*, Aug. 2007, pp. 28–38.
- [47] H. Aydin *et al.*, "Determining optimal processor speeds for periodic real-time tasks with different power characteristics," in *Real-Time Systems, 13th Euromicro Conf., 2001.*, Jun. 2001, pp. 225–232.
- [48] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," *SIGOPS Oper. Syst. Rev.*, vol. 35, no. 5, pp. 89–102, Oct. 2001.
- [49] J. R. Lorch and A. J. Smith, "Improving dynamic voltage scaling algorithms with PACE," in *Measurement and Modeling of Computer Systems (SIGMETRICS '01), Proc. 2001 ACM SIGMETRICS Int. Conf.*, 2001, pp. 50–61.
- [50] R. Xu *et al.*, "Practical PACE for embedded systems," in *Embedded Software (EMSOFT'04), Proc. 4th ACM Int. Conf.*, Sep. 2004, pp. 54–63.
- [51] N. Rizvandi *et al.*, "Linear combinations of DVFS-enabled processor frequencies to modify the energy-aware scheduling algorithms," in *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM Int. Conf.*, May 2010, pp. 388–397.
- [52] N. B. Rizvandi, J. Taheri, and A. Y. Zomaya, "Some observations on optimal frequency selection in DVFS-based energy consumption minimization," *J. Parallel Distrib. Comput.*, vol. 71, no. 8, pp. 1154 – 1164, Aug. 2011.
- [53] D.-S. Zhang *et al.*, "An energy-efficient scheduling algorithm for sporadic real-time tasks in multiprocessor systems," in *High Performance Computing and Communications (HPCC), 2011 IEEE 13th Int. Conf.*, Sep. 2011, pp. 187–194.
- [54] G. Levin *et al.*, "DP-FAIR: A simple model for understanding optimal multiprocessor scheduling," in *Real-Time Systems (ECRTS), 2010 22nd Euromicro Conf.*, Jul. 2010, pp. 3–13.
- [55] R. McNaughton, "Scheduling with deadlines and loss function," *Management Science*, vol. 6(1), pp. 1–12, October 1959.

- [56] S. Funk *et al.*, “A global optimal scheduling algorithm for multiprocessor low-power platforms,” in *Real-Time and Network Systems (RTNS '12), Proc. 20th Int. Conf.*, Nov. 2012, pp. 71–80.
- [57] F. Wu, S. Jin, and Y. Wang, “A simple model for the energy-efficient optimal real-time multiprocessor scheduling,” in *Computer Science and Automation Engineering (CSAE), 2012 IEEE Int. Conf.*, vol. 3, May 2012, pp. 18–21.
- [58] P. Regnier *et al.*, “RUN: Optimal multiprocessor real-time scheduling via reduction to uniprocessor,” in *Real-Time Systems Symp. (RTSS), 2011 IEEE 32nd*, Nov. 2011, pp. 104–115.
- [59] G. Nelissen *et al.*, “U-EDF: An unfair but optimal multiprocessor scheduling algorithm for sporadic tasks,” in *Real-Time Systems (ECRTS), 2012 24th Euromicro Conf.*, Jul. 2012, pp. 13–23.
- [60] J.-J. Chen *et al.*, “Approximation algorithms for multiprocessor energy-efficient scheduling of periodic real-time tasks with uncertain task execution time,” in *Real-Time and Embedded Technology and Applications Symp., 2008. RTAS'08. IEEE*, Apr. 2008, pp. 13–23.
- [61] C. Xian, Y.-H. Lu, and Z. Li, “Energy-aware scheduling for real-time multiprocessor systems with uncertain task execution time,” in *Design Automation Conf., 2007. DAC '07. 44th ACM/IEEE*, Jun. 2007, pp. 664–669.
- [62] Y. Yu and V. Prasanna, “Power-aware resource allocation for independent tasks in heterogeneous real-time systems,” in *Parallel and Distributed Systems, 2002. Proc. 9th Int. Conf.*, Dec. 2002, pp. 341–348.
- [63] 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 Conf., 2004. Proc. ASP-DAC 2004. Asia and South Pacific*, Jan. 2004, pp. 647–652.
- [64] C.-Y. Yang *et al.*, “An approximation scheme for energy-efficient scheduling of real-time tasks in heterogeneous multiprocessor systems,” in *Design, Automation Test in Europe Conf. Exhibition, 2009. DATE '09*, Apr. 2009, pp. 694–699.
- [65] L. K. Goh, B. Veeravalli, and S. Viswanathan, “Design of fast and efficient energy-aware gradient-based scheduling algorithms heterogeneous embedded multiprocessor systems,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 1, pp. 1–12, Jan. 2009.

- [66] J.-J. Chen, A. Schranzhofer, and L. Thiele, “Energy minimization for periodic real-time tasks on heterogeneous processing units,” in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE Int. Symp.*, May 2009, pp. 1–12.
- [67] D. Li and J. Wu, “Energy-aware scheduling for frame-based tasks on heterogeneous multiprocessor platforms,” in *Parallel Processing (ICPP), 2012 41st Int. Conf.*, Sep. 2012, pp. 430–439.
- [68] M. Awan and S. Petters, “Energy-aware partitioning of tasks onto a heterogeneous multi-core platform,” in *Real-Time and Embedded Technology and Applications Symp. (RTAS’13), IEEE 19th*, Apr. 2013, pp. 205–214.
- [69] ARM, “*big.LITTLE Technology: The Future of Mobile*,” 2013. [Online]. Available: http://www.arm.com/files/pdf/big_LITTLE_Technology_the_Futue_of_Mobile.pdf
- [70] H. S. Chwa *et al.*, “*Energy and Feasibility Optimal Global Scheduling Framework on big.LITTLE platforms*,” Department of Computer Science, KAIST and Department of Computer Science and Engineering, Sungkyunkwan University, Republic of Korea, Tech. Rep., 2014. [Online]. Available: https://cs.kaist.ac.kr/upload_files/report/1407392146.pdf
- [71] H. S. Chwa, J. Seo, J. Lee, and I. Shin, “Optimal real-time scheduling on two-type heterogeneous multicore platforms,” in *Real-Time Systems Symp., 2015 IEEE*, Dec. 2015, pp. 119–129.
- [72] M. Gerdtz, “A variable time transformation method for mixed-integer optimal control problems,” *Optim. Control Appl. Methods*, vol. 27, no. 3, pp. 169–182, May/Jun. 2006.
- [73] R. J. Vanderbei, *Linear programming : foundations and extensions*, ser. International series in operations research & management science. Boston, Dordrecht, London: Kluwer Academic, 2001.
- [74] S. Baruah, “Task partitioning upon heterogeneous multiprocessor platforms,” in *Real-Time and Embedded Technology and Applications Symposium, 2004. Proc. RTAS 2004. 10th IEEE*, May 2004, pp. 536–543.
- [75] H. Aydin *et al.*, “Power-aware scheduling for periodic real-time tasks,” *IEEE Trans. Comput.*, vol. 53, no. 5, pp. 584–600, May 2004.
- [76] M. Gerards *et al.*, “Analytic clock frequency selection for global DVFS,” in *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro Int. Conf.*, Feb. 2014, pp. 512–519.

- [77] H. Aydin and Q. Yang, “Energy-aware partitioning for multiprocessor real-time systems,” in *Parallel and Distributed Processing Symp., 2003. Proc. Int.*, Apr. 2003, pp. 9 pp.–.
- [78] T. Koch, “Rapid mathematical prototyping,” Ph.D. dissertation, Technische Universität Berlin, 2004.
- [79] T. Achterberg, “SCIP: Solving constraint integer programs,” *Math. Program. Comput.*, vol. 1, no. 1, pp. 1–41, Jul. 2009.
- [80] R. Wunderling, “Paralleler und objektorientierter Simplex-Algorithmus,” Ph.D. dissertation, Technische Universität Berlin, 1996.
- [81] A. Wächter and L. T. Biegler, “On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming,” *Math. Program.*, vol. 106, no. 1, pp. 25–57, 2006.
- [82] G. Zeng *et al.*, “Practical energy-aware scheduling for real-time multiprocessor systems,” in *Embedded and Real-Time Computing Systems and Applications, 2009. RTCSA '09. 15th IEEE Int. Conf.*, Aug. 2009, pp. 383–392.
- [83] H.-C. Wang *et al.*, “Energy-efficient tasks scheduling algorithm for real-time multiprocessor embedded systems,” *J. Supercomput.*, vol. 62, no. 2, pp. 967–988, Nov. 2012.
- [84] Intel XScale Microarchitecture: Benchmarks, 2005. [Online]. Available: <http://web.archive.org/web/20050326232506/developer.intel.com/design/intelxscale/benchmarks.htm>
- [85] C. Rusu *et al.*, “Energy-efficient policies for request-driven soft real-time systems,” in *Real-Time Systems, 2004. ECRTS 2004. Proc. 16th Euromicro Conf.*, Jun. 2004, pp. 175–183.
- [86] J. Wegener and F. Mueller, “A comparison of static analysis and evolutionary testing for the verification of timing constraints,” *Real-Time Syst.*, vol. 21, no. 3, pp. 241–268, 2001.
- [87] X. Fu and X. Wang, “Utilization-controlled task consolidation for power optimization in multi-core real-time systems,” in *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2011 IEEE 17th Int. Conf.*, vol. 1, Aug. 2011, pp. 73–82.
- [88] M. Thammawichai and E. C. Kerrigan, “Energy-efficient scheduling for homogeneous multiprocessor systems,” arXiv:1510.05567v2 [cs.OS], 2015.

- [89] D. Q. Mayne, “Model predictive control: Recent developments and future promise,” *Automatica*, vol. 50, no. 12, pp. 2967–2986, 2014.
- [90] D. Zhu, R. Melhem, and B. Childers, “Scheduling with dynamic voltage/speed adjustment using slack reclamation in multi-processor real-time systems,” in *Real-Time Systems Symp., 2001. (RTSS 2001). Proc. 22nd IEEE*, Dec. 2001, pp. 84–94.
- [91] J.-J. Che, C.-Y. Yang, and T.-W. Kuo, “Slack reclamation for real-time task scheduling over dynamic voltage scaling multiprocessors,” in *Sensor Networks, Ubiquitous, and Trustworthy Computing, 2006. IEEE Int. Conf.*, vol. 1, Jun. 2006, pp. 8 pp.–.
- [92] MATLAB, *version 7.14.0.739 (R2012a)*. Natick, Massachusetts: The MathWorks Inc., 2012.
- [93] J. Currie and D. I. Wilson, “OPTI: Lowering the Barrier Between Open Source Optimizers and the Industrial MATLAB User,” in *Foundations of Computer-Aided Process Operations*, N. Sahinidis and J. Pinto, Eds., Savannah, Georgia, USA, 8–11 January 2012.
- [94] M. Bryson *et al.*, “Airborne vision-based mapping and classification of large farmland environments,” *J. Field Robot.*, vol. 27, no. 5, pp. 632–655, 2010.
- [95] F. Nex and F. Remondino, “UAV for 3D mapping applications: a review,” *Applied Geomatics*, vol. 6, no. 1, pp. 1–15, 2014.
- [96] K. Anderson and K. J. Gaston, “Lightweight unmanned aerial vehicles will revolutionize spatial ecology,” *Front. Ecol. Environ.*, vol. 11, no. 3, pp. 138–146, 2013.
- [97] L. Wallace *et al.*, “Development of a UAV-LiDAR system with application to forest inventory,” *Remote Sensing*, vol. 4, no. 6, pp. 1519–1543, 2012.
- [98] G. Zhou, “Near real-time orthorectification and mosaic of small UAV video flow for time-critical event response,” *IEEE Trans Geosci. Remote Sens.*, vol. 47, no. 3, pp. 739–747, Mar. 2009.
- [99] M. J. Allen, “Autonomous soaring for improved endurance of a small uninhabited air vehicle,” in *Proc. 43rd aerospace sciences meeting, AIAA*, Jan. 2005.
- [100] L. Doherty *et al.*, “Energy and performance considerations for smart dust,” *Int. J. Parallel Distrib. Syst. Netw.*, vol. 4, no. 3, pp. 121–133, 2001.

- [101] F. Zhao *et al.*, “Collaborative signal and information processing: an information-directed approach,” *Proc. IEEE*, vol. 91, no. 8, pp. 1199–1209, Aug. 2003.
- [102] S. Kumar, F. Zhao, and D. Shepherd, “Collaborative signal and information processing in microsensor networks,” *IEEE Signal Processing Mag.*, vol. 19, no. 2, pp. 13–14, Mar. 2002.
- [103] H. Qi, Y. Xu, and X. Wang, “Mobile-agent-based collaborative signal and information processing in sensor networks,” *Proc. IEEE*, vol. 91, no. 8, pp. 1172–1183, Aug. 2003.
- [104] J. Kulik, W. Heinzelman, and H. Balakrishnan, “Negotiation-based protocols for disseminating information in wireless sensor networks,” *Wirel. Netw.*, vol. 8, no. 2/3, pp. 169–185, Mar. 2002.
- [105] B. Krishnamachari, D. Estrin, and S. Wicker, “The impact of data aggregation in wireless sensor networks,” in *Distributed Computing Systems Workshops, 2002. Proc. 22nd Int. Conf.*, Jul. 2002, pp. 575–578.
- [106] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, “Energy-efficient communication protocol for wireless microsensor networks,” in *System Sciences, 2000. Proc. 33rd Annu. Hawaii Int. Conf.*, vol. 2, Jan. 2000, p. 10 pp.
- [107] K. Dasgupta, K. Kalpakis, and P. Namjoshi, “An efficient clustering-based heuristic for data gathering and aggregation in sensor networks,” in *Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE*, vol. 3, Mar. 2003, pp. 1948–1953.
- [108] J. Al-Karaki, R. Ul-Mustafa, and A. Kamal, “Data aggregation in wireless sensor networks - exact and approximate algorithms,” in *High Performance Switching and Routing, 2004. HPSR. Workshop*, Apr. 2004, pp. 241–245.
- [109] M. A. Razzaque and S. Dobson, “Energy-efficient sensing in wireless sensor networks using compressed sensing,” *Sensors*, vol. 14, no. 2, pp. 2822–2859, 2014.
- [110] Y. Chen and Q. Zhao, “On the lifetime of wireless sensor networks,” *IEEE Commun. Lett.*, vol. 9, no. 11, pp. 976–978, Nov. 2005.
- [111] M. Bhardwaj and A. Chandrakasan, “Bounding the lifetime of sensor networks via optimal role assignments,” *INFOCOM 2002. 21st Annu. Joint Conf. IEEE Computer and Communications Societies. Proc. IEEE*, vol. 3, pp. 1587–1596, Jun. 2002.

- [112] Y.-F. Wen and F.-S. Lin, “Energy-efficient data aggregation routing and duty-cycle scheduling in cluster-based sensor networks,” *Consumer Communications and Networking Conf.*, 2007. *CCNC 2007. 4th IEEE*, pp. 95–99, Jan. 2007.
- [113] H. Gupta *et al.*, “Efficient gathering of correlated data in sensor networks,” *ACM Trans. Sensor Network*, vol. 4, no. 1, pp. 4:1–4:31, Feb. 2008.
- [114] L. Xiang, J. Luo, and C. Rosenberg, “Compressed data aggregation: Energy-efficient and high-fidelity data collection,” *IEEE/ACM Trans. Netw.*, vol. 21, no. 6, pp. 1722–1735, Dec. 2013.
- [115] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, “An application-specific protocol architecture for wireless microsensor networks,” *IEEE Trans. Wireless Commun.*, vol. 1, no. 4, pp. 660–670, Oct. 2002.
- [116] S. Lindsey and C. Raghavendra, “PEGASIS: Power-efficient gathering in sensor information systems,” in *Aerospace Conf. Proc.*, 2002. *IEEE*, vol. 3, Mar. 2002, pp. 1125–1130.
- [117] F. Atero *et al.*, “A low energy and adaptive architecture for efficient routing and robust mobility management in wireless sensor networks,” in *Distributed Computing Systems Workshops (ICDCSW), 2011 31st Int. Conf.*, Jun. 2011, pp. 172–181.
- [118] T. Kanungo *et al.*, “An efficient k-means clustering algorithm: analysis and implementation,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 7, pp. 881–892, Jul. 2002.
- [119] O. Tekdas *et al.*, “Using mobile robots to harvest data from sensor fields,” *IEEE Trans. Wireless Commun.*, vol. 16, no. 1, pp. 22–28, Feb. 2009.
- [120] R. Sugihara and R. Gupta, “Improving the data delivery latency in sensor networks with controlled mobility,” in *Distributed Computing in Sensor Systems*, ser. Lecture Notes in Computer Science, S. Nikolettseas, B. Chlebus, D. Johnson, and B. Krishnamachari, Eds. Springer Berlin Heidelberg, 2008, vol. 5067, pp. 386–399.
- [121] P. Sujit, D. Lucani, and J. Sousa, “Bridging cooperative sensing and route planning of autonomous vehicles,” *IEEE J. Sel. Areas Commun.*, vol. 30, no. 5, pp. 912–922, Jun. 2012.
- [122] D.-T. Ho *et al.*, “Cluster-based communication topology selection and UAV path planning in wireless sensor networks,” in *Unmanned Aircraft Systems (ICUAS), 2013 Int. Conf.*, May 2013, pp. 59–68.

- [123] H.-Y. Shum and L.-W. He, "Rendering with concentric mosaics," in *Computer Graphics and Interactive Techniques (SIGGRAPH '99), Proc. 26th Annu. Conf.*, 1999, pp. 299–306.
- [124] P. Zhan, K. Yu, and A. Swindlehurst, "Wireless relay communications using an unmanned aerial vehicle," in *Signal Processing Advances in Wireless Communications, 2006. SPAWC '06. IEEE 7th Workshop*, Jul. 2006, pp. 1–5.
- [125] S. Ponda *et al.*, "Distributed planning strategies to ensure network connectivity for dynamic heterogeneous teams," *IEEE J. Sel. Areas Commun.*, vol. 30, no. 5, pp. 861–869, Jun. 2012.
- [126] D. Pack, G. York, and R. Fierro, "Information-based cooperative control for multiple unmanned aerial vehicles," in *Networking, Sensing and Control, 2006. ICNSC '06. Proc. 2006 IEEE Int. Conf.*, Dec. 2006, pp. 446–450.
- [127] S. Quintero *et al.*, "Optimal UAV coordination for target tracking using dynamic programming," in *Decision and Control (CDC), 2010 49th IEEE Conf.*, Dec. 2010, pp. 4541–4546.
- [128] L. Wang *et al.*, "Active sensing based cooperative target tracking using UAVs in an urban area," in *Advanced Computer Control (ICACC), 2010 2nd Int. Conf.*, vol. 2, Mar. 2010, pp. 486–491.
- [129] N. Adurthi and P. Singla, "Information driven optimal sensor control for efficient target localization and tracking," in *American Control Conf. (ACC), 2014*, Jun. 2014, pp. 610–615.
- [130] M. Kalandros *et al.*, "Tutorial on multisensor management and fusion algorithms for target tracking," *American Control Conf., 2004. Proc. 2004*, vol. 5, pp. 4734–4748, Jun. 2004.
- [131] M. Kalandros, "Covariance control for multisensor systems," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 38, no. 4, pp. 1138–1157, Oct. 2002.
- [132] Q. Ling, Y. Fu, and Z. Tian, "Localized sensor management for multi-target tracking in wireless sensor networks," *Inform. Fusion*, vol. 12, pp. 194–201, 2011.
- [133] O. Demigha, W.-K. Hidouci, and T. Ahmed, "On energy efficiency in collaborative target tracking in wireless sensor network: A review," *IEEE Commun. Surv. Tut.*, vol. 15, no. 3, pp. 1210–1222, Mar. 2013.
- [134] F. Zhao, J. Shin, and J. Reich, "Information-driven dynamic sensor collaboration," *IEEE Signal Processing Mag.*, vol. 19, no. 2, pp. 61–72, Mar. 2002.

- [135] R. Olfati-Saber, "Distributed tracking for mobile sensor networks with information-driven mobility," in *American Control Conf., 2007. ACC '07*, Jul. 2007, pp. 4606–4612.
- [136] J. Lin *et al.*, "Energy-efficient distributed adaptive multisensor scheduling for target tracking in wireless sensor networks," *IEEE Trans. Instrum. Meas.*, vol. 58, no. 6, pp. 1886–1896, Jun. 2009.
- [137] R. Rajagopalan and P. Varshney, "Data-aggregation techniques in sensor networks: A survey," *IEEE Commun. Surv. Tut.*, vol. 8, no. 4, pp. 48–63, Apr. 2006.
- [138] Y. Hamouda and C. Phillips, "Metadata-based adaptive sampling for energy-efficient collaborative target tracking in wireless sensor networks," in *Computer and Information Technology (CIT), 2010 IEEE 10th Int. Conf.*, Jun. 2010, pp. 313–320.
- [139] F. Delicato *et al.*, "An efficient heuristic for selecting active nodes in wireless sensor networks," *Comput. Netw.*, vol. 50, no. 18, pp. 3701 – 3720, 2006.
- [140] Z. Wang *et al.*, "A novel mobility management scheme for target tracking in cluster-based sensor networks," in *Distributed Computing in Sensor Systems*, ser. Lecture Notes in Computer Science, R. Rajaraman, T. Moscibroda, A. Dunkels, and A. Scaglione, Eds. Springer Berlin Heidelberg, 2010, vol. 6131, pp. 172–186.
- [141] K. M. Fornace, C. J. Drakeley, T. William, F. Espino, and J. Cox, "Mapping infectious disease landscapes: unmanned aerial vehicles and epidemiology," *Trends Parasitol.*, vol. 30, no. 11, pp. 514–519, 2014.
- [142] C. Zhang, "An UAV-based photogrammetric mapping system for road condition assessment," *Int. Arch. Photogramm. Remote Sens. Spatial Inform. Sci.*, vol. 37, pp. 627–632, 2008.
- [143] M. Patel, S. Venkatesan, and D. Weiner, "Role assignment for data aggregation in wireless sensor networks," *Advanced Information Networking and Applications Workshops, 2007, AINAW '07. 21st Int. Conf.*, vol. 2, pp. 390–395, May 2007.
- [144] J. Zhu and S. Papavassiliou, "On the energy-efficient organization and the lifetime of multi-hop sensor networks," *IEEE Commun. Lett.*, vol. 7, no. 11, pp. 537–539, Nov. 2003.
- [145] Y. Tian, J. Boangoat, E. Ekici, and F. Ozguner, "Real-time task mapping and scheduling for collaborative in-network processing in DVS-enabled wireless

- sensor networks,” *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th Int.*, p. 10 pp., Apr. 2006.
- [146] B. Zeng, J. Wei, and H. Liu, “Research of optimal task scheduling for distributed real-time embedded systems,” *Embedded Software and Systems, 2008. ICESS '08. Int. Conf.*, pp. 77–84, Jul. 2008.
 - [147] T. Vercauteren and X. Wang, “Decentralized sigma-point information filters for target tracking in collaborative sensor networks,” *IEEE Trans. Signal Process.*, vol. 53, no. 8, pp. 2997–3009, Aug. 2005.
 - [148] S. Kim *et al.*, “Coordinated trajectory planning for efficient communication relay using multiple UAVs,” *Control Eng. Pract.*, vol. 29, no. 0, pp. 42–49, 2014.
 - [149] H. Cotuk *et al.*, “The impact of bandwidth constraints on the energy consumption of wireless sensor networks,” in *Wireless Communications and Networking Conf. (WCNC), 2014 IEEE*, April 2014, pp. 2787–2792.
 - [150] B. P. Hudzietz and S. Saripalli, “An experimental evaluation of 3D terrain mapping with an autonomous helicopter,” in *Unmanned Aerial Vehicle in Geomatics (UAV-g), Int. Conf.*, Sep. 2011, pp. 137–142.
 - [151] P. Sujit, S. Saripalli, and J. Borges Sousa, “Unmanned aerial vehicle path following: A survey and analysis of algorithms for fixed-wing unmanned aerial vehicless,” *IEEE Control Syst.*, vol. 34, no. 1, pp. 42–59, 2014.
 - [152] D. Nelson *et al.*, “Vector field path following for miniature air vehicles,” *IEEE Trans. Robot.*, vol. 23, no. 3, pp. 519–529, 2007.
 - [153] V. Devadas and H. Aydin, “On the interplay of voltage/frequency scaling and device power management for frame-based real-time embedded applications,” *IEEE Trans. Comput.*, vol. 61, no. 1, pp. 31–44, Jan. 2012.
 - [154] M. E. T. Gerards and J. Kuper, “Optimal DPM and DVFS for frame-based real-time systems,” *ACM Trans. Archit. Code Optim.*, vol. 9, no. 4, pp. 41:1–41:23, Jan. 2013.
 - [155] A. Saifullah, D. Ferry, J. Li, K. Agrawal, C. Lu, and C. Gill, “Parallel real-time scheduling of DAGs,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 12, pp. 3242–3252, Dec. 2014.
 - [156] G. Varatkar and R. Marculescu, “Communication-aware task scheduling and voltage selection for total systems energy minimization,” in *Computer Aided Design, 2003. ICCAD-2003. Int. Conf.*, Nov. 2003, pp. 510–517.

- [157] D. Kliazovich *et al.*, “CA-DAG: Modeling communication-aware applications for scheduling in cloud computing,” *J. Grid Comput.*, pp. 1–17, 2015.
- [158] S. Sardellitti, S. Barbarossa, and A. Swami, “Optimal topology control and power allocation for minimum energy consumption in consensus networks,” *IEEE Trans. Signal Process.*, vol. 60, no. 1, pp. 383–399, Jan. 2012.
- [159] C. Efthymiou, S. Nikolettseas, and J. Rolim, “Energy balanced data propagation in wireless sensor networks,” in *Parallel and Distributed Processing Symp., 2004. Proc. 18th Int.*, Apr. 2004, pp. 225–232.
- [160] X. Liu, F.-M. Shao, and P. Xin, “Energy balance optimization for prolonging the lifetime of wireless sensor network,” in *Multimedia Technology (ICMT), 2010 Int. Conf.*, Oct. 2010, pp. 1–4.
- [161] N. Jin, K. Chen, and T. Gu, “Energy balanced data collection in wireless sensor networks,” in *Network Protocols (ICNP), 2012 20th IEEE Int. Conf.*, Oct. 2012, pp. 1–10.
- [162] A. Aziz *et al.*, “A survey on distributed topology control techniques for extending the lifetime of battery powered wireless sensor networks,” *IEEE Commun. Surv. Tut.*, vol. 15, no. 1, pp. 121–144, Jan. 2013.
- [163] Z. Mir, K.-W. Lim, and Y.-B. Ko, “On the interplay between clustering and power control in multihop wireless networks,” in *Computer Systems and Applications (AICCSA), 2014 IEEE/ACS 11th Int. Conf.*, Nov. 2014, pp. 310–317.

Appendices

A Mean Absolute Percentage Error (MAPE)

$$\text{MAPE} := \frac{1}{k} \sum_{i=1}^k \frac{|F(z(i)) - y(i)|}{|y(i)|} \times 100, \quad (\text{A.1})$$

where $\frac{|F(z(i)) - y(i)|}{|y(i)|}$ is the magnitude of the relative error in the i^{th} measurement, $z \mapsto F(z)$ is the estimated function, z is the input data, y is the actual data and k is the total number of fitted points.

B Curve Fitting

The following shows the curve fitting formulation to obtain the active power consumption function with the objective of minimizing an MAPE:

$$\underset{\alpha, \beta, P_s}{\text{minimize}} \quad 100 \times \frac{1}{\gamma} \sum_{i=1}^{\gamma} |\alpha s_i^{\beta} + P_s - P_{\text{active}}(s_i)| / P_{\text{active}}(s_i)$$

subject to

$$P_s \geq 0 \quad (\text{B.1a})$$

$$\alpha \geq 0 \quad (\text{B.1b})$$

$$\beta \geq 1 \quad (\text{B.1c})$$

where α , β and P_s are parameters to be fitted, γ is the total number of speed levels of a processor. The speed level of a processor s_i and the active power consumption of a processor $P_{\text{active}}(s_i)$ are measurement data.