DESIGN AND PERFORMANCE EVALUATION OF ENERGY-AWARE DVS-BASED SCHEDULING STRATEGIES FOR HARD REAL-TIME EMBEDDED MULTIPROCESSOR SYSTEMS

GOH LEE KEE

(B.Eng.(Hons.), NUS,

M.Eng., NUS)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF ELECTRICAL AND

COMPUTER ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2012

Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Goh Lee Kee

13 August 2012

Acknowledgements

I would like to thank my supervisor, Dr. Bharadwaj Veeravalli, for all the advices, suggestions and recommendations he has given me during the course of my research. I would also like to thank him for the patience and understanding that he has shown for the delays in my research progress due to my work commitments.

I would also like to show my appreciation to my employer, Institute for Infocomm Research (I^2R), and Agency for Science, Technology and Research (A*STAR), for their support in allowing me to pursue my Ph.D. degree on a part-time basis.

Last but not least, my thanks goes out to all my friends and colleagues working in the project under the Embedded & Hybrid Systems II (EHS-II) initiative of A*STAR, without which this dissertation would not have materialized. Specifically, I would like to thank Dr. Sivakumar Viswanathan for his guidance and feedback in the project, as well as Dr. Liu Yanhong and Mr. Sivanesan Kailash Prabhu for their help and cooperation during the course of the project.

Contents

Summary

Hard real-time applications have strict deadline requirements. Violation of these deadline requirements usually results in catastrophic failure of the system and cannot be tolerated. At the same time, when these applications are implemented on portable embedded devices, efficient energy management is essential to ensure a long operating lifetime of the system. This thesis evaluates the various energy-aware static scheduling strategies in the literature and proposes an efficient, energy gradient-based approach to generate these schedules by considering task mapping, task scheduling and voltage scaling in an integrated way. In addition, the thesis also proposes a few strategies to reduce the energy consumption further during runtime when the tasks do not require their worst-case execution cycles to complete. Last but not least, the thesis addresses the scenario where each processing

element has its own energy source. In this case, traditional methods of minimizing the total energy consumption do not necessary increase the system lifetime. A method is proposed to balance the energy consumption among the processing elements to improve the lifetime of the system. All the proposed strategies are compared against existing strategies in the literature through extensive simulation experiments to evaluate their performances.

List of Tables

List of Figures

Chapter 1

Introduction

As the demand for high-performance embedded systems increases, we observe an increasing number of systems incorporating multiple homogeneous or heterogeneous processing units on their platforms. An example of one such system is the software-defined radio (SDR) where it may consist of a general-purpose processor (GPP) for control, as well as a digital signal processor (DSP) and/or a field-programmable array (FPGA) for signal processing. There are also processors currently in the market that contain homogeneous or heterogeneous cores, such as the OMAP processors [1] by Texas Instruments. With the use of multiple processing elements in embedded systems, it is a challenge to efficiently manage the energy consumption of these systems in order to maximize their battery life. Modern day processors utilize dynamic voltage scaling (DVS) [33, 34, 44–47, 49, 50, 54, 63] to reduce the energy consumption. This technique lowers the supply voltage and operational

frequency during runtime at the expense of a longer execution time. By carefully scheduling the tasks to execute at different voltage levels, an optimized schedule with minimum energy consumption can be obtained without compromising the performance.

Hard real-time applications have strict deadline requirements and any deadline misses may lead to total system failures. For example, a nuclear control and monitoring system needs to respond to meltdown conditions in a timely manner to prevent catastrophic impacts. An airbag control system on a vehicle needs to inflate the airbag rapidly upon a vehicle collision to minimize the impact suffered by the passengers. In the medical and healthcare industry [6, 11], there are also applications that not only require hard real-time performance, but also low energy consumption as well. For example, an implantable pacemaker [37] needs to monitor and regulate the patient's heat beat and at the same time, it needs to consume as little energy as possible to prolong its battery life and reduce the occurrence of battery replacements. A wearable defibrillator [14, 51] runs on batteries and continuously monitors the patient's heart. When the patient suffers a cardiac arrest, the wearable defibrillator automatically sends a treatment shock to restore normal heart rhythm. A wearable fall pre-impact detection system [7, 17, 19, 20] for the elderly uses signals from accelerometers and gyroscopes worn on the body of the elderly to detect the onset of a fall. When the system detects a fall, it needs to

quickly inflate a hip cushion to prevent hip-related fractures.

In order to guarantee that the deadline constraints will not be violated while minimizing the total energy consumption on a multiprocessor system, static energy-aware scheduling algorithms are usually used to generate static, energy-optimized schedules in advance. These static scheduling algorithms usually use the worst-case execution times (WCETs) of the tasks to try to map the tasks to the processing elements and schedule them in such a way so that the total energy consumption is minimized. In this way, the deadline constraints will still be met in the worst-case scenario while the energy consumption is minimized as much as possible. During runtime, tasks may not require their WCETs to complete, resulting in slacks being generated. A slack is defined as the period of time that is unused by a task when it completes its execution earlier than in the worst-case scenario. To reduce the energy consumption further, dynamic scheduling algorithms are then employed during runtime to reclaim these unused slacks and use them to reduce the execution speeds and energy consumption of subsequent tasks while ensuring that the deadline constraints are still met.

1.1 Scope of Research Work

In this thesis, we shall look into the design of fast and efficient static and dynamic energy-aware scheduling algorithms for maximizing the lifetime of an embedded multiprocessor system using DVS-based techniques. Specifically, we design our algorithms to cater for both homogeneous and heterogeneous multiprocessor systems. Our design will focus on scheduling dependent tasks with precedence relationships as represented by a task precedence graph. A task precedence graph is a directed, acyclic graph (DAG) where nodes represent tasks and edges between the nodes represent the communication between the tasks. The directions on the edges represent the order in which the tasks must be executed while the weights on the edges represent the time required to communicate a result from one task to another if they are placed on different processors. Besides maximizing the lifetime of the system, the scheduling algorithms are also designed to ensure that the deadlines of the tasks are not violated. We design different algorithms for the scenario where the multiprocessor cores share the same energy source, as well as for the scenario where each core has its own energy source.

1.2 Research Contributions

The research contributions for this thesis are as follows:

1. We propose the Energy Gradient-based Multiprocessor Scheduling (EGMS)

algorithm [16, 22] for scheduling task precedence graphs in an embedded multiprocessor system having processing elements with DVS capabilities and sharing a single energy source. Unlike most static energy-aware scheduling algorithms that consider task ordering and voltage scaling separately from task mapping, our algorithms consider them in an integrated way. EGMS uses the concept of energy gradient to select tasks to be mapped onto new processors and voltage levels. We extend EGMS by introducing intratask voltage scaling using a Linear Programming (LP) formulation. The resulting algorithm, EGMS with Intra-task Voltage scaling (EGMSIV), is able to reduce the total energy consumption further.

2. We propose a method to improve the performance of static energy-aware scheduling algorithms using Potential Slack for Dynamic Scheduling Considerations (PSDSC). By applying PSDSC to static energy-aware scheduling algorithms, the generated static schedules will take into consideration the dynamic reclamation of unused slacks during runtime and try to optimize the average energy consumption of the application. We use the concept of potential slack to estimate the dynamic execution speeds and energy consumption of the tasks so that the average energy consumption can be minimized. At the same time, we ensure that all the tasks will still be able to meet their

deadline requirements even if they require their WCETs to execute. In addition, we also propose the Average-based Aggressive Dynamic Scheduling (AADS) algorithm that tries to aggressively lower the execution speeds of the tasks during runtime to reduce the energy consumption further.

3. We propose the Energy-Balanced Task Scheduling (EBTS) algorithm [18] which is a static scheduling algorithm for a multiprocessor system where each processing element has its own energy source. Specifically, we consider scheduling the tasks onto a cluster of heterogeneous sensor nodes connected by a single-hop wireless network so as to maximize the lifetime of the sensor network. In our algorithm, we assign the tasks to the sensor nodes so as to minimize the energy consumption of the tasks on each sensor node while keeping the energy consumption as balanced as possible. We also extend the algorithm to generate a second schedule. The algorithm, EBTS with Dual Schedule (EBTS-DS), improves the lifetime of the network further when the second generated schedule is used together with the original schedule.

Through rigorous simulations, the performance of all the proposed algorithms are compared to existing approaches presented in the literature. The results demonstrate that the proposed algorithms are capable of obtaining more energy-efficient schedules.

1.3 Organization of thesis

The thesis is organized as follows:

1. Chapter 1: The current chapter that defines the scope and summarizes the contributions of the research work that has been conducted.

2. Chapter 2: The chapter introduces the energy and power model used in this thesis. The task and system models will also be described in this chapter.

3. Chapter 3: Related work on energy-aware scheduling will be presented in this chapter.

4. Chapter 4: A thorough description of the proposed EGMS and EGMSIV algorithms for generating energy-efficient static schedules will be presented in this chapter.

5. Chapter 5: The proposed PSDSC and AADS algorithms for generating dynamic energy-efficient schedules will be presented in this chapter.

6. Chapter 6: The chapter describes the EBTS and EBTS-DS algorithms for scheduling processing nodes with individual energy sources.

7. Chapter 7: This chapter presents the conclusion of the thesis.

Chapter 2

Preliminaries

In this chapter, the basic power, task and system models shall be described.

2.1 Power Model

The total power consumed in a digital CMOS circuit [69] consists of three portions and is given by (2.1), where P_{dyn} denotes the dynamic power consumption, P_{static} the static power consumption and P_{sc} the short-circuit power consumption.

$$P_{total} = P_{dyn} + P_{static} + P_{sc} \tag{2.1}$$

The dynamic power dissipation P_{dyn} is given by (2.2), where C_{ef} denotes the effective load capacitance, V_{dd} the supply voltage and f the processor frequency. Reducing V_{dd} lowers the power consumption but increases the circuit delay. This circuit delay is given by (2.3), where T_D denotes the circuit delay, k a proportionality constant, V_T the threshold voltage and α the velocity saturation index. V_T and α are properties of the CMOS circuit and are constant for a particular circuit. Most literatures [44, 46, 49, 52, 54, 63] use the value $\alpha = 2$. The time taken to execute the task is given by (2.4), where t denotes the execution time of the task and n_c the number of execution cycles required to execute the task. The total dynamic energy dissipation is therefore given by (2.5).

$$P_{dyn} = C_{ef} \cdot V_{dd}^2 \cdot f \tag{2.2}$$

$$T_D = k\frac{V_{dd}}{(V_{dd} - V_T)^\alpha} \tag{2.3}$$

$$t = \frac{n_c}{f} \tag{2.4}$$

$$E_{dyn} = C_{ef} \cdot V_{dd}^2 \cdot n_c \tag{2.5}$$

From the above equations, we see that when there is a reduction in the supply voltage, the dynamic energy savings increase quadratically. DVS exploits this feature to reduce the dynamic energy consumption of the processor at the expense of longer execution times for the tasks.

The static power dissipation P_{static} is given by (2.6), where I_{subn} denotes the sub-threshold leakage current, V_{bs} the body bias voltage and I_j the reverse bias junction current.

$$P_{static} = V_{dd} \cdot I_{subn} + |V_{bs}| \cdot I_j \tag{2.6}$$

From the equation above, we observe that when the supply voltage is reduced, the static power consumption is also reduced. However, at very low voltage levels, the execution times for the tasks will be so long that the static energy consumption will start to increase instead.

The short-circuit power is only consumed during signal transitions and is generally negligible in practice [48].

2.2 Multiprocessor Systems with a Single Energy Source

2.2.1 System Model

The system consists of a set of N_p heterogeneous processors, $\{PE_1, PE_2, \ldots, PE_{N_p}\}$, connected to a single bus. Each processor is equipped with DVS functionality. The available discrete voltage levels of PE_j are given by $V(j,k),\ k = 1, 2, \cdots, N(j)$, where $N(j)$ denotes the total number of discrete voltage levels of

PE_j. Without loss of generality, we let $N(1) = N(2) = ... = N(N_p) = N_v$ for simplicity. The power consumption and processor frequency of PE_j at voltage level $V(j,k)$ are given by $P(j,k)$, and $f(j,k)$ respectively. The power consumption of the bus is denoted by P_b.

2.2.2 Task Model

We consider a hard real-time application that is run periodically. Let P be the period of the application. An instance of the application will be activated at time iP and it must be completed before the next instance is activated at time $(i+1)P$, where $i = 0, 1, 2, \ldots$ (i.e. the deadline d is equal to P for every execution instance of the application). The application is represented by a directed acyclic graph (DAG) which consists of a set of N_t dependent tasks $\{T_1, T_2, \ldots, T_{N_t}\}$ that are related by some precedence constraints. If a task T_i and its predecessor T_p are executed on different processing elements, a communication time of $C(p,i)$ is incurred. The worst-case and average-case number of execution cycles (WCEC and ACEC respectively) required to run T_i to completion is given by c_i^{wc} and c_i^{ac} respectively. On the other hand, the worst-case and average-case time taken to execute T_i vary depending on the processor voltage levels. Suppose T_i is executed on PE_j at the voltage level $V(j,k)$, the worst-case execution time and energy consumption needed to execute T_i in this case are denoted by $t^{wc}(i,j,k)$ and $e^{wc}(i,j,k)$

respectively, where

$$t^{wc}(i,j,k) = \frac{c_i^{wc}}{f(j,k)} \tag{2.7}$$

Similarly, the corresponding average-case execution time and energy consumption of T_i are denoted by $t^{ac}(i,j,k)$ and $e^{ac}(i,j,k)$ respectively, where

$$t^{ac}(i,j,k) = \frac{c_i^{ac}}{f(j,k)} \tag{2.8}$$

Finally, we define Γ_i as the ratio of ACEC to WCEC of T_i:

$$\Gamma_i = \frac{c_i^{ac}}{c_i^{wc}} \tag{2.9}$$

2.2.3 Problem Formulation

For multiprocessor systems with single energy source, our objective is to find a static schedule for the tasks in the task precedence graph on the heterogeneous processors at particular voltage levels such that the total energy consumption is minimized while the task precedence constraints are observed and all the tasks meet their deadline requirements. Therefore, we seek to minimize the total energy

consumption E of the system:

$$E = P_b \cdot t_c + \sum_{i=1}^{N_t} \sum_{j=1}^{N_p} \sum_{k=1}^{N_v} (x(i,j,k) \cdot e^{wc}(i,j,k)) \tag{2.10}$$

where t_c denotes the total duration of time for which the bus is used to transfer data. For the scenario without intra-task voltage scaling, we define $x(i,j,k)$ as follows:

$$x(i,j,k) = \begin{cases} 1 & \text{if } T_i \text{ is scheduled on } PE_j \text{ at} \\ & \text{the voltage level } V(j,k) \\ 0 & \text{otherwise} \end{cases} \tag{2.11}$$

On the other hand, when intra-task voltage scaling is used, $x(i,j,k)$ will denote the fraction of T_i that is scheduled on PE_j at the voltage level $V(j,k)$.

2.3 Multiprocessor Systems with Distributed Energy Sources

2.3.1 System Model

We consider a WSN that consists of a set of N_p heterogeneous sensor nodes with DVS functionality, $\{PE_1, PE_2, \ldots, PE_{N_p}\}$, connected by a single-hop wireless network with K communication channels. The computational speed of PE_i at voltage level V_j are given by S_{ij}. The time cost and energy consumption for

transmitting one unit of data between two sensor nodes PE_i and PE_j is denoted by τ_{ij} and ξ_{ij} respectively. It is assumed that the time and energy cost of wireless transmission is the same at both the sender and the receiver and no techniques such as modulation scaling [58] are used for energy-latency tradeoffs of communication activities. It is also assumed that negligible power is consumed by the sensor nodes and the radios when they are idle.

2.3.2 Task Model

We consider an application that is run periodically in the sensor network with period P. The application is represented by a DAG $G = (T, E)$, which consists of a set of N_t dependent tasks $\{T_1, T_2, \ldots, T_{N_t}\}$ connected by a set of ϱ edges $\{E_1, E_2, \ldots, E_\varrho\}$. Each edge E_i from T_j to T_k has a weight C_i, which represents the number of units of data to be transmitted from T_j to T_k. The source tasks in G (i.e. tasks with no incoming edges) are used for measuring or collecting data from the environment and so they have to be assigned to different sensor nodes. The time and energy cost of executing T_i on PE_j at the voltage level V_k are denoted by t_{ijk} and ϵ_{ijk} respectively. Let $\theta(T_i)$ denotes the sensor node to which T_i is assigned. The energy consumption of PE_i in one period of the application π_i is given by:

$$\pi_i = \sum_{j=1}^{N_t} \sum_{k=1}^{N_v} (x_{jk} \cdot \epsilon_{jik}) + \sum_{j=1}^{\varrho} (y_j \cdot C_j \cdot \xi_{\theta(T_a)\theta(T_b)}) \tag{2.12}$$

where T_a and T_b are connected by the edge E_j and x_{jk} and y_j are defined as follows:

$$x_{jk} = \begin{cases} 1 & \text{if } T_j \text{ is scheduled on } PE_i \text{ at} \\ & \text{the voltage level } V_k \\ 0 & \text{otherwise} \end{cases} \tag{2.13}$$

$$y_j = \begin{cases} 1 & \text{if either } T_a \text{ or } T_b \text{ (but not} \\ & \text{both) is scheduled on } PE_i \\ 0 & \text{otherwise} \end{cases} \tag{2.14}$$

2.3.3 Problem Formulation

Unlike multiprocessors systems with single energy source, minimizing the total energy consumption of the WSN does not necessarily increases the lifetime of the system. Let R_i be the remaining energy of PE_i. We define the norm-energy η_i [33] of PE_i as its energy consumption in one period normalized by its remaining energy:

$$\eta_i = \frac{\pi_i}{R_i} \tag{2.15}$$

The lifetime of the whole sensor network L is therefore determined by the sensor

node with the largest norm-energy. Hence, our objective is to maximize L:

$$L = \lfloor \frac{1}{max_i(\eta_i)} \rfloor \tag{2.16}$$

Chapter 3

Literature Review

In this chapter, some of the most recent and commonly used energy-aware scheduling strategies and their workings will be described in a brief style for the purpose of continuity. For a more detailed analysis of these strategies, the reader may refer to their respective references.

3.1 Heuristic Approach to Energy-aware Multiprocessor Scheduling

In this thesis, our objective is to schedule a task precedence graph on a heterogeneous multiprocessor system while maximizing the lifetime of the system using DVS techniques and ensuring the deadline constraints are met. The problem is

formulated in a way such that it also covers energy-aware scheduling on both homogeneous multiprocessor systems and uniprocessor systems. The problem of energy-aware scheduling on homogeneous multiprocessor systems [4, 8, 12, 24, 28, 35] is a subset of energy-aware scheduling on heterogeneous multiprocessor systems in which each task requires the same amount of computation time to execute on all the processors. The problem of energy-aware scheduling for uniprocessor systems [29, 30, 38, 41, 55, 56] is a subset of energy-aware scheduling on homogeneous multiprocessor systems in which the number of processors is one. The problem of energy-aware scheduling in heterogeneous multiprocessor systems is NP-hard [53, 77]. As such, it requires a computation time that is of the order of at least superpolynomial to the input size. When the uncertain execution times are considered during runtime, the problem becomes even harder. Due to the nature of NP-hard problems, it is impractical to obtain an optimal solution even for moderately sized problem. Instead, heuristic algorithms are usually used to solve these types of problems. While there is no proof that heuristic algorithms always produce good results, most heuristic algorithms are able to obtain reasonably good solutions in many cases using a much shorter computation time [27, 66, 75].

Metaheuristic approaches [10, 15, 43] is a class of heuristic algorithms that uses memory and learning to fine-tune candidate solutions in search of the best solution. Some popular metaheuristic approaches include tabu search [71, 72, 74],

simulated annealing [42, 76], particle swarm optimization [13, 65] and genetic algorithms [67, 70, 73]. Tabu search and simulated annealing are single solution-based search heuristics. This type of approach focus on modifying and improving a single candidate solution using local search strategies. For example, in simulated annealing, a single candidate solution is used to search for better candidate solutions among its neighbourhood using the idea of physical annealing of solids to attain minimum internal energy states. In each iteration of the algorithm, the current candidate solution has a certain probability of being replaced by one of its neighbouring candidate solution, which may not necessarily be better than the current candidate solution. This ensure that the search will not be trapped in a local optimal. The process terminates after a certain number of iterations has been reached. In tabu search, the immediate neighbours of a candidate solution is checked in the hope of finding a better solution. A memory structure is maintained to store recent visited solutions within the search space and prevent the algorithm from visiting these solution again.

On the other hand, particle swarm optimization and genetic algorithms use a population-based approach to maintain and improve multiple candidate solutions, using the characteristics of the population to guide the search. In particle swarm optimization, a population of candidate solutions is spread over the search space. These candidate solutions are referred to as particles. Each particle moves around

in the search space based on a simple function of its position and velocity. Each particle's movement is guided by both its local best known position as well as the best known positions discovered by other particles. As a result, the particles are expected to swarm toward the best solutions. In a genetic algorithm, a population of candidate solutions evolves towards better solutions during the process of evolution. Candidate solutions are usually represented as a string or an array. A fitness function is defined to evaluate the quality of the candidate solution. The genetic algorithm starts with a randomly generated population of candidate solutions. These candidate solutions are then evaluated using the defined fitness function. Next, a new population of candidate solutions are generated from the current population using the principles of genetic crossover and mutation [5, 25, 68]. In crossover, a pair of of parent strings is selected from the current population with the probability of selection being an increasing function of fitness. With some crossover probability, the pair is crossed over at randomly chosen point to form two new strings. Next, the two new candidate solutions are mutated at random points with some mutation probability. The newly generated population of candidate solutions then replaces the current population. This process of fitness evaluation, crossover and mutation is then repeated iteratively, until the process does not find any better candidate solutions after a number of iterations.

3.2 Multiprocessor Systems with a Single Energy Source

Most multiprocessor systems have a single energy source from which each processing element draws its power. In order to maximize the lifetime of such a multiprocessor system, the total energy consumption of the system must be minimized. The most common way to solve this problem is to divide it into two sub-problems. In the first sub-problem, the tasks are mapped to the processing elements and the mapping is usually improved iteratively based on the feasibility and energy consumption of the generated schedule. This is known as the task mapping (TM) sub-problem. In the second sub-problem, it is assumed that the mapping of tasks to processing elements is known and the tasks are scheduled/ordered and assigned to various voltage levels so as to minimize the total energy consumption. We shall define this as the task scheduling and voltage scaling (TSVS) sub-problem. Figure 3.1 shows the typical flow in solving this energy-aware scheduling problem. The TSVS sub-problem is highlighted by the shaded rectangle.

There are some papers [34, 50, 54] that assume that the task ordering is known and focus only on voltage scaling. In [54], Schmitz et al. propose a heuristic that is based on energy gradient and takes into account the power variations among the tasks. While this approach is suitable for heterogeneous multiprocessor systems its

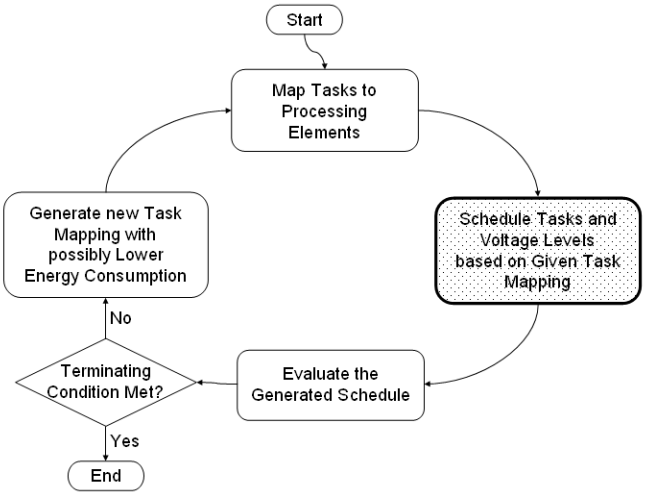


Figure 3.1: Typical flow for solving energy-aware scheduling problem for dependent tasks

performance is dependent on the granularity of the time quantum used in the approach. As the size of the time quantum decreases, more energy is reduced but the computation time also increases. There are a few studies that use the integer linear programming approach. Zhang et al. [50] formulate the voltage scaling problem as an integer linear programming (ILP) problem for a fixed task ordering and without considering communication time and energy. Andrei et al. [34] use a mixed integer linear programming (MILP) method to solve the combined problem of voltage scaling and adaptive body biasing assuming a known task ordering. However, for both approaches, the long runtime of the optimal formulation makes it impractical to be used within a task mapping and scheduling algorithm. Yanhong et al. [21] propose a scheduling algorithm with low computational complexity using a critical path track and update scheme to update the scaling factor of each critical path

and distribute the slack over the tasks. The low computational complexity of the algorithm makes it suitable to be used within a task mapping and scheduling algorithm.

There are also many papers [23, 36, 39, 60, 61] that focus solely on the TSVS subproblem. Gruian et al. [60] use a list scheduling heuristic with a priority function based on the average energy consumption. Whenever an infeasible schedule is found, the priorities of the tasks are dynamically increased and the tasks are rescheduled. However, the average energy and priority function used in the algorithm is calculated based on the assumption that the energy consumption and computation time of a task is the same on all the processors. Therefore, it is not suitable for scheduling tasks on heterogeneous multiprocessor systems. Luo et al. [61] try to minimize the energy consumption by evenly distributing the slack among the tasks. While this approach is suitable for homogeneous multiprocessor systems, it is not optimized for heterogeneous multiprocessor systems due to the variation of the power consumption across different processing elements. In [39], Gorjiara et al. propose a fast heuristic by randomly slowing down some of the high-power tasks. Tasks with higher power consumption have higher probabilities of being slowed down. More recently, the authors propose another stochastic-based scheduling algorithm [23, 36] that is faster and more energy-efficient. In this approach, they randomly slow down or speed up the tasks based on their energy gradient and

execution delays. Tasks with higher energy gradients and lower execution delays are assigned higher probabilities of being slowed down. Due to the random slowing down or speeding up of the tasks, this algorithm is able to avoid being trapped in local minima and therefore it is able to find better solutions more easily. The nature of the algorithm allows it to be used for heterogeneous multiprocessor systems. In addition, the low computation time of the algorithm makes it suitable for use within a task mapping algorithm.

There are not many literature that considers task mapping, task ordering and voltage scaling at the same time. Leung et al. [40] formulate the whole problem of task mapping, task ordering and voltage scaling as a mixed integer non-linear programming (MINLP) problem with continuous voltage levels. However, since their runtime is very long, they propose a divide-and-conquer approach to speed up the optimization process at the expense of losing the optimality of their solution. In [52], Schmitz et al. propose a strategy that also considers task mapping, task ordering and voltage scaling. In their strategy, they use a list scheduling heuristic where the priorities of the tasks are generated using a genetic algorithm (GA) and voltage scaling of the tasks is done using [54]. This is then nested inside another GA that is used to determine the optimal mapping of the tasks to the processing elements. The genetic algorithms used in this approach allow the user to search through a larger exploration space and avoid local minima, resulting in good solutions being

found. Although this approach is able to obtain good solutions compared to other approaches such as [40], the optimization time is still relatively high due to the nested nature of the GA algorithms.

During runtime, tasks may not require their WCETs to complete, resulting in slacks being generated. Dynamic energy-aware scheduling algorithms are then used during runtime to reclaim the slacks and reduce the total energy consumption further. Yang et al. [57] propose a two-phase strategy for runtime scheduling on multiprocessor system. In this strategy, the tasks are grouped into clusters called thread frame. The runtime scheduling options are set during the design-time phase. During the runtime phase, the scheduler just chooses the suitable scheduling option. Although the runtime complexity of this approach is low, by grouping the tasks into thread frames, the amount of energy reduction may be limited. Zhu et al. [44] propose the concept of slack sharing among the processors for homogeneous systems. They later extend the concept to applications that are modelled using AND/OR graphs [49]. Mishra et al. [46] propose a greedy approach in which the whole slack that is generated by a task will be reclaimed by its immediate successor tasks to reduce their energy consumption. While this approach is simple, it ensures that the deadlines of the tasks will be met while the constant order complexity of the algorithm means that the runtime scheduling overhead is minimal. Kang et

al. [9] propose to apply static slack allocation schemes during runtime to a subset of tasks in order to derive a more energy-efficient schedule while requiring less runtime overhead when compared to applying the static schemes to all the tasks during runtime. While this approach is able to dynamically derive a more energy-efficient schedule, it does not guarantee that the deadlines of the task will be met. Therefore it is not suitable for runtime scheduling of hard real-time applications.

From the literature, it is observed that most of the researchers focus their research on solving a subset of the problem that we are aiming to solve. Most apply DVS to uniprocessor or homogeneous multiprocessor systems to generate energy efficient schedules. Furthermore, their research are also usually focused on the TSVS sub-problem or the voltage scaling problem, assuming that the task mapping is known. The few literature that addresses task mapping, task ordering and voltage scaling for heterogeneous multiprocessor systems requires a high optimization time in order to achieve a reasonably good solution. This thesis tries to minimize the energy consumption in a heterogeneous multiprocessor system by considering task mapping, task ordering and voltage scaling in an integrated way. In doing so, we are able to generate energy-efficient schedules using much less optimization time.

3.3 Multiprocessor Systems with Distributed Energy Sources

In tightly coupled battery-operated multiprocessor systems where processors share the same energy source, minimizing the total energy consumption of the system also maximize its lifetime. However, the same cannot be said for some systems in which each processor has its own energy source. An example is a wireless sensor network (WSN). In this type of system, minimizing the total energy consumption may not necessarily maximize the lifetime of the system. If many of the tasks are allocated to a single processor, the energy source of that processor is going to drain much faster than the other processors, resulting in a shorter system lifetime as a whole. In order to maximize the lifetime of the system, the tasks have to be allocated in a balanced way according to the available energy capacities of each processor.

To address this problem, Yu et al. [33] proposed a 3-phase heuristic approach for task mapping, task ordering and voltage scaling in a WSN. In the first phase, the tasks are grouped into clusters by eliminating communications with high execution times. Next, the clusters are assigned to the sensor nodes in a way such that the norm-energies of the sensor nodes are balanced. Here, the norm-energy is defined as the total energy consumption of the tasks scheduled on a node normalized by the

remaining energy of that node. In the last phase, the voltage levels of the tasks are adjusted to reduce the energy consumption further. However, the 3-phase heuristic approach is only applicable to a WSN with homogeneous sensor nodes.

The thesis proposes a new heuristic scheduling algorithm that is that can be applied to heterogeneous sensor nodes. The algorithm tries to minimize the energy consumption of the tasks on each sensor node while keeping the energy consumption as balanced as possible among the sensor nodes. In doing so, we are able to achieve a much longer lifetime of the WSN while the deadline miss rate of the generated schedules are also much lower.

Chapter 4

Static Energy-aware Scheduling Strategies for Systems with Single Energy Source

Since hard real-time applications have strict deadline requirements, in order to guarantee that the deadline constraints are not violated, static energy-aware scheduling algorithms are usually used to generate static energy optimized schedules in advance. These algorithms use the WCETs of the task so that the deadline constraints will still be met in the worst-case scenario. In this chapter, we present a few static energy-aware scheduling strategies for multiprocessor systems. First, an energy gradient-based multiprocessor scheduling algorithm will be described. The algorithm, referred to as *Energy Gradient-based Multiprocessor Scheduling (EGMS)*, is designed to schedule task precedence graphs under deadline constraints. The second algorithm, *EGMS with Intra-task Voltage Scaling (EGMSIV)*, extends EGMS

and utilizes a Linear Programming (LP) method for intra-task voltage scaling. We then present an adaptation of the proposed EGMS/EGMSIV algorithms for TM or TSVS only. Lastly, the performance of EGMS and EGMSIV are evaluated through rigorous simulation experiments.

4.1 Design of Energy Gradient-based Multiprocessor Scheduling (EGMS)

In energy-aware scheduling of task precedence graphs on heterogeneous embedded multiprocessor platforms, there are three main factors that affect the quality of the solution obtained: the mapping of tasks onto processors, the ordering of the tasks and the selection of the voltage levels of the processors. Most literature [23, 36, 39, 60, 61] considers the ordering of tasks and voltage scaling in an integrated approach for the TSVS sub-problem. However, the mapping of tasks to processors is usually considered separately. During task mapping optimization, the TSVS algorithm that is used has to be invoked repeatedly to obtain the best feasible and energy-efficient schedule for every task mapping that is generated in the process, regardless of the quality of the task mapping. In addition, only the final schedule that is generated by the TSVS for each task mapping is considered during this optimization process. However, the TSVS process itself may also be

useful in guiding the task mapping optimization process towards a more energy-efficient schedule at a faster rate. This is one of the main factors that is considered in the design of the EGMS algorithm.

The EGMS algorithm takes into consideration task mapping, task ordering and voltage scaling in an integrated manner. A schedule is first generated based on an initial task mapping. In each optimization step, a task is re-mapped to a new processor and/or voltage level such that the total energy consumption of the schedule is reduced as much as possible while the slack is decreased as little as possible. In this way, the task mapping as well as the task scheduling and voltage scaling are optimized at the same time based on the current partially optimized schedule. In doing so, it is hoped that an optimized energy-efficient schedule can be derived in a shorter time.

Figure 4.1 defines some notations that would be used in the description of the EGMS algorithm. We define the makespan of a schedule as the period of time required to completely process all the tasks.

The pseudo-code of the EGMS algorithm is presented in Algorithm 1. An initial schedule is first generated by assigning tasks to the processors that can complete their execution in the shortest amount of time at the highest voltage level (lines

Notations used in EGMS

$M_p(i)$	:	Processor to which T_i is currently mapped, $1 \leq i \leq N_t$
$M_v(i)$	:	Voltage level to which T_i is currently mapped, $1 \leq i \leq N_t$
e	:	Energy consumption of the current schedule
m_s	:	Makespan of the current schedule
$M_{p_{best}}(i)$	:	Processor to which T_i is mapped in the best schedule generated so far, $1 \leq i \leq N_t$
$M_{v_{best}}(i)$	:	Voltage level to which T_i is mapped in the best schedule generated so far, $1 \leq i \leq N_t$
e_{best}	:	Minimum energy consumption of the best schedule generated so far
$m_{s_{best}}$	:	Makespan of the best schedule generated so far
$numIter$	:	Number of successive iterations without significant improvement
$T_{selected}$	:	Task selected to be re-mapped
$P_{selected}$	:	Processor to which $T_{selected}$ is mapped
$V_{selected}$	:	Voltage level to which $T_{selected}$ is mapped
$e_{selected}$	:	Total energy consumption when $T_{selected}$ is re-mapped
$m_{s_{selected}}$	:	Makespan of schedule when $T_{selected}$ is re-mapped
$p_{r_{selected}}$	:	Priority calculated when $T_{selected}$ is re-mapped to $P_{selected}$ at voltage $V_{selected}$

Figure 4.1: Notations used in EGMS algorithm

2-5). The tasks are then scheduled using the Critical Path-based Task Ordering (CPTO) algorithm (line 6). In each iteration of the while loop (lines 8-31), a task is selected to be re-mapped to a new processor and/or a new voltage level such that the total energy consumption is reduced and no deadlines are violated by using the SelectRemapTask() algorithm (line 9). If such a task can be found, the current schedule is updated (lines 10-14). This process continues until no tasks can be re-mapped without violating the deadlines. When this happens, the energy consumption of the schedule cannot be reduced further. If the current schedule is feasible and has a lower energy consumption than the best schedule obtained so far, the best schedule is updated with the current schedule (lines 18-26). However,

this best schedule may not be the global optimum schedule. In order to obtain a better solution, 50% of the tasks in the current schedule are randomly reassigned to other processors at the highest voltage levels to generate a new initial schedule (lines 28-29). The whole process of task re-mapping is then repeated starting from the new initial mapping until there is no significant improvement in the energy consumption ($> 1\%$) of n successive schedules. Here, n is a user-defined parameter that determines the terminating condition of the algorithm. It shall be noted that by reassigning the tasks and applying the algorithm repeatedly, the total energy consumption is lowered further at the expense of an increase in optimization time.

We shall now give the complexity of the EGMS algorithm. Let C_{CPTO} and C_{SEL} be the complexities of CPTO() and SelectRemapTask() respectively. (The complexities of these two algorithms shall be shown later.) Lines 2 to 5 execute in $O(N_t \cdot N_p)$ time. In the while loop, the steps from lines 9 to 14 are repeated until the current schedule cannot be optimized further. Let η be the average number of times these steps are repeated. When the current schedule cannot be optimized further, the steps from lines 15 to 30 are executed to generate a new initial schedule before repeating the steps from lines 9 to 14 again. The steps from lines 15 to 30 are repeated $O(n)$ times. The complexity of the while loop is therefore given by $O(n(\eta \cdot C_{SEL} + N_t + C_{CPTO}))$. Hence, the total complexity of EGMS is $O(N_t \cdot N_p + C_{CPTO} + n(\eta \cdot C_{SEL} + N_t + C_{CPTO}))$. It shall be shown later that this

Algorithm 1 : EGMS()

1: $e_{best} \leftarrow \infty$
2: **for all** T_i **do** /* Assign tasks to fastest processors */
3: $M_p(i) \leftarrow$ Fastest-executing processor for T_i
4: $M_v(i) \leftarrow$ Maximum voltage level
5: **end for**
6: CPTO(M_p, M_v, e, m_s) /* Schedule tasks based on initial processor and voltage mapping */
7: $numIter \leftarrow 0$
8: **while** $numIter < n$ **do**
9: $T_{selected}$ = SELECTREMAPTASK($P_{selected}$, $V_{selected}$, $e_{selected}$, $m_{s_{selected}}$) /* Find a task to be re-mapped */
10: **if** $T_{selected} \neq -1$ **then** /* Task to be re-mapped is found, update current mapping */
11: $M_p(T_{selected}) \leftarrow P_{selected}$
12: $M_v(T_{selected}) \leftarrow V_{selected}$
13: $e \leftarrow e_{selected}$
14: $m_s \leftarrow m_{s_{selected}}$
15: **else** /* No tasks can be re-mapped without violating deadline */
16: $numIter++$
17: **if** feasible schedule found **then**
18: **if** $e < e_{best}$ **then** /* Better schedule is found, update the best schedule found so far */
19: **if** $\frac{e}{e_{best}} < 0.99$ **then** /* Improvement > 1% */
20: $numIter \leftarrow 0$
21: **end if**
22: $e_{best} \leftarrow e$
23: $m_{s_{best}} \leftarrow m_s$
24: $M_{p_{best}} \leftarrow M_p$
25: $M_{v_{best}} \leftarrow M_v$
26: **end if**
27: **end if**
28: Randomly assign 50% of tasks to other processors at maximum voltage level
29: CPTO(M_p, M_v, e, m_s)
30: **end if**
31: **end while**

complexity is dominated by C_{SEL} and so it can be given as $O(n \cdot \eta \cdot C_{SEL})$.

Algorithm 2 shows the CPTO() algorithm that is use to generate a schedule. Based on the given processor and voltage level mapping, all the communication edges between two tasks that are mapped on different processors are first replaced using tasks where the execution time is equal to the communication time (line 3). For

each task, the length of the critical path from that task is calculated and its start and end times are initialized (lines 4-8). The tasks are then scheduled based on their critical paths (lines 9-26). Tasks with longer critical paths have higher priorities and are scheduled first. The makespan and energy consumption of the schedule are also calculated in the process. The complexity of CPTO() is given as follows: Let N_a be the total number of computational and communication tasks and N_{succ} be the average number of successors of a task. Let us assume that the tasks are already in topological order. For line 4-8, the lengths of all the critical paths can be calculated in $O(N_a \cdot N_{succ})$ time. A fibonacci heap is used to implement the priority queue Q. Therefore insertion into Q (line 9) is $O(1)$ and removal from Q (line 11) is $O(logN_a)$ amortized time. Line 24 requires $O(N_{succ})$ time for execution. The while loop is executed N_a times. Thus, the total complexity of CPTO() is $O(N_a \cdot N_{succ} + N_a \cdot (logN_a + N_{succ})) = O(N_a \cdot (logN_a + N_{succ}))$.

Algorithm 3 shows the SelectRemapTask() algorithm that is used to select the best task to be re-mapped, as well as the processor and voltage level it should be re-mapped to. In this algorithm, we consider the cases when T_i is re-mapped to processor P_j at voltage level $V(j,k)$ for all i,j and k. For each $< T_i, P_j, V(j,k) >$ triplet, CPTO() is invoked to obtain the energy consumption e' and makespan m'_s of the new schedule generated by the re-mapping (lines 7-11). The priority p_r is then calculated if this new schedule is feasible and has a lower energy consumption

Algorithm 2 : CPTO(M_p, M_v, e, m_s)

```
1: e ← 0
2: m_s ← 0
3: Replace the communication between any 2 tasks that are scheduled on different processors
   with a task, where the execution time is the communication time
4: for all T_j do        /* Both computational and communication tasks */
5:    Calculate l_cp(j)        /* Length of critical path starting from T_j */
6:    t_start(j) ← 0        /* Initialize start time of T_j to 0 */
7:    t_end(j) ← 0        /* Initialize end time of T_j to 0 */
8: end for
9: Insert tasks with no incoming edges into priority queue Q, where tasks are sorted in decreasing
   values of l_cp
10: while Q is not empty do
11:    Remove T_j from front of Q.        /* Get task with largest critical path length */
12:    if T_j is communication task then
13:       e ← e+Communication energy
14:       t_start(j) ← Earliest time communication bus is free
15:       t_end(j) ← t_start(j)+ Communication time
16:    else        /* T_j is computational task */
17:       e ← e + e^wc(j, M_p(j), M_v(j))
18:       t_start(j) ← Earliest time PE_{M_p(j)} is free
19:       t_end(j) ← t_start(j) + t^wc(j, M_p(j), M_v(j))
20:    end if
21:    if t_end(j) > m_s then
22:       m_s ← t_end(j)
23:    end if
24:    Remove T_j and all outgoing edges from task graph
25:    Insert tasks with no incoming edges into priority queue Q
26: end while
```

than the current schedule:

$$p_r = \begin{cases} \frac{e-e'}{m'_s-m_s} & \text{if } m'_s > m_s \\ w \times (e-e') & \text{if } m'_s \leq m_s \end{cases} \tag{4.1}$$

Here, two cases are considered. In the first case, the new schedule has a lower energy consumption but a longer makespan. Here, the concept of energy gradient

is used to calculate the priority so that schedules that give the largest reduction in energy consumption with the least increase in makespan will be assigned higher priorities. Most of the schedules will be in this case. In the second case, the new schedule has both a lower energy consumption and a shorter makespan. In this case, higher priorities are assigned to schedules that result in larger reduction of energy consumption. An arbitrary large constant w is used in the calculation of the priority so as to assign higher priorities to these schedules compared to the schedules in the first case. This is because schedules that reduce both the energy consumption and the makespan are much more preferred than those in the first case.

It shall be noted that there is no need to invoke CPTO() and calculate the priorities for all $< T_i, P_j, V(j,k) >$ triplets. For a task that is re-mapped to a particular processor, the highest voltage level l can be obtained such that the total energy consumption is lower than the current consumption for all voltages $\leq l$ (line 5). This step takes $O(logN_v)$ time. Therefore there is no need to consider the schedules for voltages greater than l in the inner-most k-loop (lines 6-25). In addition, whenever an infeasible schedule is generated for a particular voltage level, there is no need to consider the lower voltage levels as well (lines 22-23). As the schedule becomes more optimized, the number of feasible voltage levels that can be mapped to also decreases. Although the overall worst-case complexity of SelectRemapTask()

Algorithm 3 :
SELECTREMAPTASK($P_{selected}, V_{selected}, e_{selected}, m_{s_{selected}}$)

```
1: T_selected ← −1
2: p_{r_selected} ← −∞
3: for i ← 1 to N_t do
4:   for j ← 1 to N_p do
5:     Select the highest voltage l such that the total energy consumption is reduced
6:     for k ← l to 1 do        /* No need to consider voltage levels higher than l */
7:       curProc ← M_p(i), M_p(i) ← j        /* Re-map T_i to PE_j at V(j,k) */
8:       curVoltage ← M_v(i), M_v(i) ← k
9:       CPTO(M_p, M_v, e', m'_s)        /* Obtain schedule based on new mapping */
10:      M_p(i) ← curProc        /* Revert back to original mapping */
11:      M_v(i) ← curVoltage
12:      if m'_s ≤ d then        /* Schedule is feasible */
13:        Calculate priority p_r using Eq.(4.1)
14:        if p_r > p_{r_selected} then        /* Highest priority found so far */
15:          p_{r_selected} ← p_r
16:          T_selected ← i
17:          P_selected ← j
18:          V_selected ← k
19:          e_selected ← e'
20:          m_{s_selected} ← m'_s
21:        end if
22:      else        /* Deadline violated, no need to consider lower voltage levels */
23:        break
24:      end if
25:    end for
26:  end for
27: end for
28: return T_selected
```

is $O(N_t \cdot N_p \cdot (logN_v + N_v \cdot C_{CPTO})) = O(N_t \cdot N_p \cdot N_v \cdot C_{CPTO})$, the average-case complexity is usually much smaller.

4.2 Design of EGMS with Intra-task Voltage Scaling (EGMSIV)

The EGMS algorithm is extended to scenarios where intra-task voltage scaling shall be used. In intra-task voltage scaling, a fraction of a task can be executed at a particular voltage level while the rest of the task is executed at another voltage level on the same processor. It is assumed that there is negligible overhead involved when a processor changes its voltage level.

In order to introduce intra-task voltage scaling in the algorithm, the following issue needs to be addressed. Given the processor mapping and ordering of each task, how should the voltage levels be assigned to the tasks so that the total energy consumption is minimized and all tasks meet their deadline requirements?

This can be formulated into a Linear Programming (LP) problem in the following way. Let $M_p(i)$ be the given processor mapping of each task T_i. For any two tasks that are connected by an edge in the task precedence graph and are scheduled on different processors, a communication task C_j with communication time $t_c(j)$ is required to transfer data between the two processors. Let N_c be the total number of such communication tasks to be scheduled on the bus. Let $y(i, M_p(i), k)$ denote the fraction of T_i that executes on the given processor $M_p(i)$ at the voltage level

$V(M_p(i), k)$. Let $t_s^{T_i}$ and $t_e^{T_i}$ denote the start and end times of the execution of T_i. In addition, the start and end times of the execution of C_j are denoted using $t_s^{C_j}$ and $t_e^{C_j}$ respectively. The voltage scaling problem is then formulated as shown in Figure 4.2.

LP Formulation for Voltage Scaling Problem

Minimize
$E = \sum_{i=1}^{N_t} \sum_{k=1}^{N_v} (y(i, M_p(i), k) \cdot e^{wc}(i, M_p(i), k)) + P_b \cdot \sum_{j=1}^{N_c} (t_e^{C_j} - t_s^{C_j})$

Subjected to

1) Fraction of T_i assigned to $V(M_p(i), k) \leq 1$):
$0 \leq y(i, M_p(i), k) \leq 1$,
for $i = 1, 2, \cdots, N_t$; $k = 1, 2, \cdots, N_v$

2) Fractions of T_i must add up to exactly one:
$\sum_{k=1}^{N_v} y(i, M_p(i), k) = 1$,
for $i = 1, 2, \cdots, N_t$

3) End time of T_i = Start time of T_i + Execution time of T_i:
$t_e^{T_i} = t_s^{T_i} + \sum_{k=1}^{N_v} (y(i, M_p(i), k) \cdot t^{wc}(i, M_p(i), k)) \leq d$,
for $i = 1, 2, \cdots, N_t$

4) End time of C_j = Start time of C_j + Communication time of C_j:
$t_e^{C_j} = t_s^{C_j} + t_c(j) \leq d$,
for $j = 1, 2, \cdots, N_c$

5) Tasks scheduled on the same processor must not overlap:
$t_s^{T_i} \geq t_e^{T_a}$,
for $i = 1, 2, \cdots, N_t$, where T_a and T_i are scheduled on the same processor and T_i is the next task to be executed after T_a.

6) Communications scheduled on the bus must not overlap:
$t_s^{C_j} \geq t_e^{C_b}$,
for $j = 1, 2, \cdots, N_c$, where C_b and C_j are scheduled on the bus and C_j is the next edge to be scheduled after C_b.

7) Precedence constraints are observed:
$t_s^{T_i} \geq t_e^{C_p}$, $t_s^{C_p} \geq t_e^{T_q}$,
for $i = 1, 2, \cdots, N_t$, where T_q and T_i are scheduled on different processors, C_p is the communication task between them, and T_i is the immediate successor of T_q.

Figure 4.2: Voltage Scaling using LP formulation

This LP formulation of the voltage scaling problem can be solved optimally in polynomial time using currently available LP solvers if the formulation is feasible. The solution to this problem will give the optimized energy consumption, the fraction of tasks to be scheduled on each voltage level, the start and end times of the tasks on the processors that they are mapped to, as well as the start and end times of the communication tasks on the bus. However, it should be noted that the optimality of this solution applies only to the given processor mapping and ordering of the tasks.

With this LP formulation of the voltage scaling problem, the EGMS algorithm can be extended to introduce intra-task voltage scaling in the following way. When the current schedule is feasible and cannot be optimized further (line 17 of EGMS()), the processor mapping and task ordering of the current schedule is based on the assumption that no intra-task voltage scaling is used. In order to obtain the energy consumption for the case with intra-task voltage scaling, the LP formulation is applied based on the same processor mapping and ordering of the tasks as the current schedule. This additional step of applying the LP formulation is inserted between lines 17 and 18 of the EGMS algorithm. Since the current schedule is feasible, the LP formulation is also feasible. As such, the final schedule will reflect the minimized energy consumption obtainable using the algorithm for scheduling the tasks with intra-task voltage scaling. In addition, the EGMS algorithm is

employed repeatedly until there is no significant improvement in the solution after n successive iterations. Hence, one needs to apply this formulation $O(n)$ times to avoid the solution being trapped in local minima.

4.3 Adaptation of EGMS/EGMSIV for TM or TSVS only

While EGMS and EGMSIV consider task mapping, task ordering and voltage scheduling in an integrated way, they can be modified to address the TM as well as the TSVS sub-problems separately. A brief description of the changes required to modify them is shown below.

1) *EGMS for Task Mapping Only (EGMS-TM)*

   To use EGMS for task mapping only, the CPTO() algorithm (lines 6 and 29 of EGMS(), line 9 of SelectRemapTask()) shall be replaced with any TSVS algorithms that the user desires. In addition, since the voltage scaling is done by the TSVS algorithms, there is no need to consider the voltage level mapping. Therefore, lines 4, 12 and 25 from EGMS(), as well as lines 5, 8, 11 and 18 from SelectRemapTask() shall be removed. In addition, in SelectRemapTask(), there is no need to consider the inner-most k-loop. Instead, the steps inside the k-loop should be moved to the outer j-loop.

2) *EGMS/EGMSIV for Task Scheduling and Voltage Scaling Only (EGMS-TSVS/EGMSIV-TSVS)*

To use EGMS/EGMS-IV for task scheduling and voltage scaling only, it is assumed that the task mapping is already given. Since no task mapping is required, lines 3, 11 and 24 from EGMS() as well as lines 5, 7, 10 and 17 from SelectRemapTask() can be removed. The j-loop in SelectRemapTask() is also removed. In addition, since fast TSVS algorithms are often required, the steps in the k-loop (lines 8-21) shall be applied for the next lower voltage level only.

4.4 Performance of EGMS and EGMSIV

4.4.1 Energy Optimization without Task Mapping

Let us consider the TSVS sub-problem where the mapping of tasks to processors is assumed to be given. The EGMS-TSVS and EGMSIV-TSVS algorithms will be compared with ASG-VTS [23, 36]. As described in Section 3.2, ASG-VTS is an ultra-fast algorithm that produces energy-efficient schedules without intra-task voltage scaling. An on-line version of the DVS tool that uses ASG-VTS is available at [3]. The algorithms are implemented using C++ in a Cygwin environment on a Pentium-IV/3.2GHz/2GB RAM PC running Windows XP. 1000 task graphs comprising a maximum of 100 tasks are generated using TGFF [64]. The lp_solve

[2] library are used for solving the LP formulation of the voltage scaling problem in the EGMSIV-TSVS algorithm. The time required to execute the tasks on the processors at the highest voltage level were defined in an expected time to compute (ETC) matrix which was generated using the method described in [62]. The ETC matrix is generated for high task as well as high machine heterogeneity ($V_{task} = 0.5, V_{mach} = 0.5$) condition. It is assumed that each processor has four voltage levels at 0.9V, 1.7V, 2.5V and 3.3V. The mean task execution time (μ_{task}) was set as 10 and the mean power consumption of each processor at maximum voltage level (μ_{power}) was set as 100. The maximum power ratings for the processors were randomly generated using a gamma distribution with α and β parameters calculated as follows:

$$\alpha = \frac{1}{V_{mach}^2} \tag{4.2}$$

$$\beta = \frac{\mu_{power}}{\alpha} \tag{4.3}$$

Figure 4.3 shows the deadline miss rate achieved by the ASG-VTS, EGMS-TSVS and EGMSIV-TSVS algorithms. Out of the 1000 task graphs, ASG-VTS is unable to find a feasible schedule for 5.3% of the task graphs while EGMS-TSVS and EGMSIV-TSVS are unable to find a feasible for 5.0% and 4.8% of the task graphs

respectively. We compare the performance of the three algorithms using the remaining 945 task graphs where feasible solutions can be found for all 3 algorithms. We normalized the optimization time by the values obtained using ASG-VTS for comparison purposes. Figure 4.4 shows the average energy savings achieved by the 3 algorithms while Figure 4.5 show the geometric mean of the normalized optimization time required by the 3 algorithms to obtain a feasible schedule. It is observed that the geometric mean of the normalized optimization time of EGMS-TSVS and EGMSIV-TSVS is about 3 to 4 times that of ASG-VTS. This is because EGMS-TSVS and EGMSIV-TSVS search through a larger exploration space than ASG-VTS, resulting in a longer optimization time. However, because EGMS-TSVS and EGMSIV-TSVS search through a larger exploration space, they are able to perform better than ASG-VTS in terms of energy minimization. On the average, there is an improvement of about 21% in terms of the amount of energy saved when EGMS-TSVS is compared to ASG-VTS. The EGMSIV-TSVS performs even better, obtaining an improvement of about 36% compared to ASG-VTS as a result of using intra-task voltage scaling.

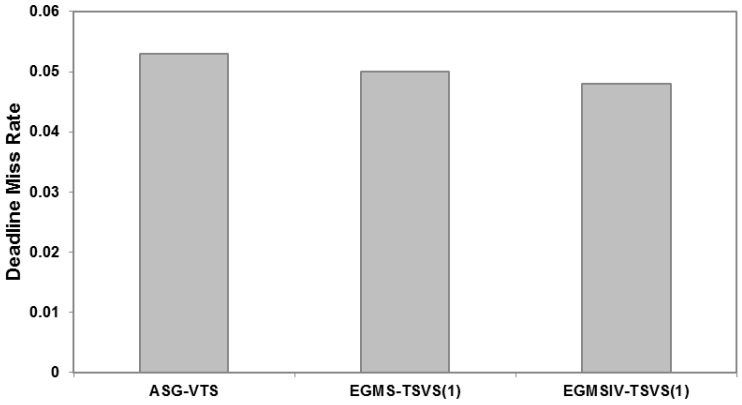


Figure 4.3: Deadline miss rate when using ASG-VTS, EGMS-TSVS and EGMSIV-TSVS for task scheduling and voltage scaling based on a given mapping of tasks to processors

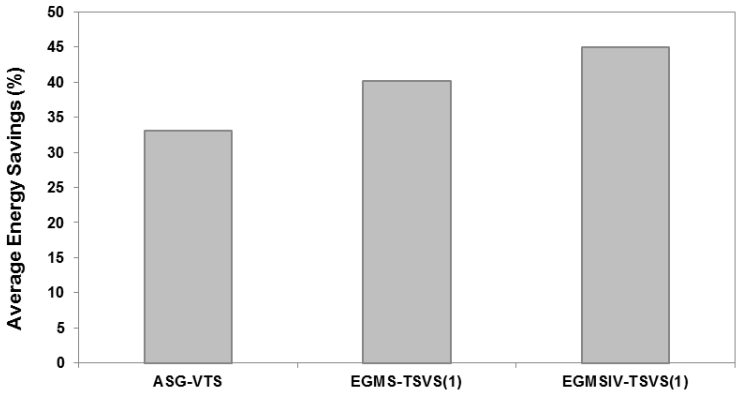


Figure 4.4: Average energy savings by ASG-VTS, EGMS-TSVS and EGMSIV-TSVS for task scheduling and voltage scaling based on a given mapping of tasks to processors

4.4.2 Energy Optimization with Task Mapping

For energy optimization with task mapping, task ordering and voltage scaling, the EGMS and EGMSIV algorithms are compared to the nested GA approach [52]

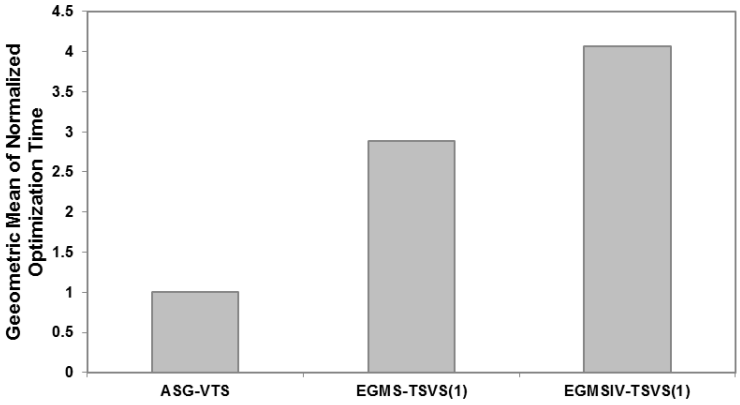


Figure 4.5: Geometric mean of the normalized optimization time required by ASG-VTS, EGMS-TSVS and EGMSIV-TSVS for task scheduling and voltage scaling based on a given mapping of tasks to processors

since it is able to generate energy-efficient schedule while considering task mapping, task ordering and voltage scaling at the same time. In the nested GA approach, a GA-based task scheduling algorithm EE-GLSA is nested within another GA-based task mapping algorithm EE-GMA in order to obtain the best processor mapping, task ordering and voltage level mapping. Due to the high runtime of the nested GA approach, we also try to replace the inner GA-based task scheduling algorithm EE-GLSA with the ASG-VTS algorithm [23, 36]. In addition, we also insert the ASG-VTS algorithm into the EGMS-TM algorithm for evaluation. These approaches are denoted as EE-GMA(ASG-VTS) and EGMS-TM(ASG-VTS) respectively. The communication delays are uniformly distributed between 1 and 5 while the power consumption of the communication bus is set at 10. The EGMS-TSVS and EGMSIV-TSVS algorithms are also inserted into EE-GMA and the results are compared. These approaches are denoted as EE-GMA(EGMS-TSVS)

Table 4.1: Scheduling strategies compared in the simulation study for energy optimization with task mapping

Strategy	Task Mapping Algorithm	Task Scheduling & Voltage Scaling Algorithm	Intra-task Voltage Scaling
EE-GMA(ASG-VTS)	EE-GMA	ASG-VTS	No
EE-GMA(EGMS-TSVS)	EE-GMA	EGMS-TSVS	No
EGMS-TM(ASG-VTS)	EGMS-TM	ASG-VTS	No
EGMS	EGMS-TM	EGMS-TSVS	No
Nested GA	EE-GMA	EE-GLSA	Yes
EE-GMA(EGMSIV-TSVS)	EE-GMA	EGMSIV-TSVS	Yes
EGMSIV	EGMS-TM	EGMSIV-TSVS	Yes

and EE-GMA(EGMSIV-TSVS) respectively. A summary of the features of the various approaches used in the simulation is shown in Table 4.1.

In the nested GA approach [52], the area penalty is omitted in the calculation of the fitness function in EE-GMA, since processor area is not a constraint in the analysis. Hence, the fitness function used in the experiment is given by:

$$F_s = E \cdot \left(1 + \frac{\sum_{i=1}^{N_s} (max(0, t_e^i - d))^2}{d^2} \right) \tag{4.4}$$

where t_e^i is the end time of the sink task T_i and N_s is the total number of sink tasks. Here, the sink tasks refer to those tasks in the task precedence graph that have no successors. The inner EE-GLSA algorithm terminates when there is no improved solution (improvement $> 1\%$) being produced for 10 successive generations. The outer EE-GMA algorithm terminates when there is no improvement in the solution for 25 successive generations. In addition, just as in [52], the calculation of the energy consumption obtained using the nested GA approach assumes the use of

intra-task voltage scaling. The EE-GMA algorithm that is used in combination of other TSVS algorithms also terminates when there is no improvement in the solution for 25 successive generations. For the EGMS-TM algorithm, n is set to 25 as well.

We randomly generate 1000 task graphs consisting of between 10 to 100 tasks and obtain the energy consumption and optimization time required by the various approaches. For the EGMS and EGMSIV algorithms, we consider the cases when n is 1, 100 and 500 respectively. For the purpose of comparison, the energy consumption and the optimization time obtained using the various methods are normalized by those obtained using the nested GA approach. Out of the 1000 task graphs, there are 869 tasks graphs in which feasible schedules can be found for all the approaches. Figure 4.6 shows the deadline miss rates that are achieved by the various approaches. We observe that the nested GA approach has the highest rate of deadline misses at 12.2% while EGMS and EGMSIV have the lowest rate of deadline misses at between 3.7% to 7.2%. We also observe that when n increase from 1 to 100 for EGMS and EGMSIV, the miss rate decreases. This is due to the larger exploration space that is being searched when n increases.

Figure 4.7 and 4.8 show the average normalized energy consumption and the geometric mean of the normalized optimization time required by the various algorithms

for the remaining 869 task graphs. The 95% confidence intervals are also shown in the figures. The first 5 algorithms in the figures use intra-task voltage scaling while the last 6 algorithms do not employ intra-task voltage scaling.

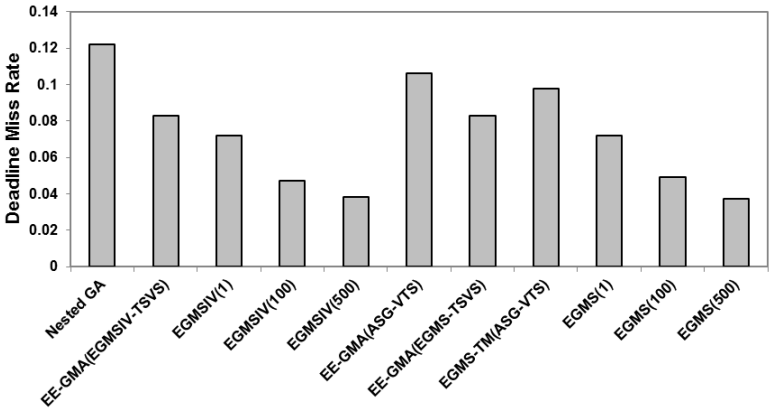


Figure 4.6: Deadline miss rate by the various algorithms for mapping optimization

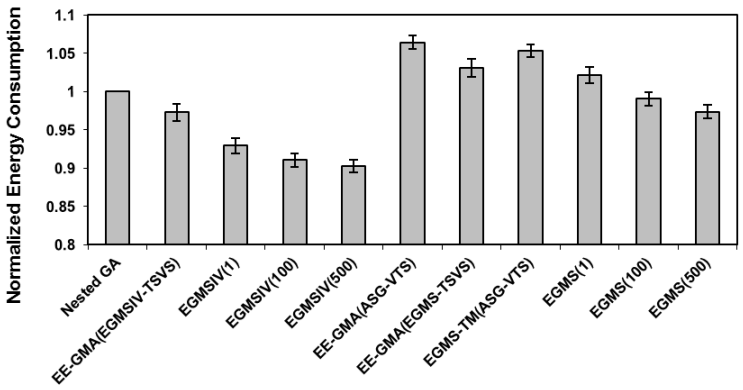


Figure 4.7: Average normalized energy consumption by the various algorithms for mapping optimization with 95% confidence intervals

For the first 5 algorithms that use intra-task voltage scaling, it can be observed that when the EE-GLSA algorithm in the nested GA approach is replaced with

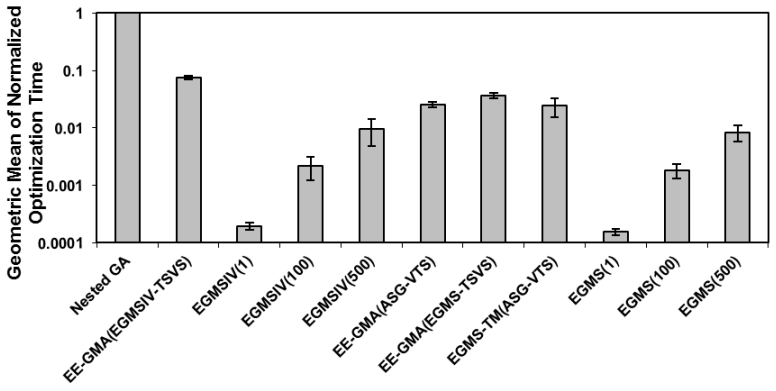


Figure 4.8: Geometric mean of the normalized optimization time required by the various algorithms for mapping optimization with 95% confidence interval

the EGMSIV-TSVS algorithm, the energy consumption can be reduced further. The EGMSIV algorithm performs even better. Compared to the nested GA approach, the EGMSIV algorithm is able to reduce the energy consumption by 7% to 10% when n increases from 1 to 500. Among the remaining 6 algorithms that do not consider intra-task voltage scaling, EE-GMA(ASG-VTS) consumes the most energy, followed by EGMS-TM(ASG-VTS), EE-GMA(EGMS-TSVS), EGMS(1), EGMS(100) and EGMS(500). It is observed that when either the EE-GMA or ASG-VTS algorithms in EE-GMS(ASG-VTS) are replaced by the EGMS-TM or EGMS-TSVS algorithms respectively, the energy consumption can be reduced further. However, the EGMS algorithm is able to obtain an even better performance. Compared to EE-GMA(ASG-VTS), EGMS reduces the energy consumption by 4% to 9% when n increases from 1 to 500.

Next, let us look at the optimization time required by the various algorithms to derive a feasible schedule. Nested GA requires between 5 and 20695 seconds to derive a feasible schedule while EE-GMA(ASG-VTS) requires between 1 and 86 seconds. EGMS-TM(ASG-VTS) requires between 1 and 55 seconds while EE-GMA(EGMS-TSVS) and EE-GMA(EGMSIV-TSVS) require up to 398 and 916 seconds for optimization respectively. The best optimization times are achieved by the EGMS and EGMSIV algorithms. Both EGMS and EGMSIV take about 0.006 to 3.1 seconds, 0.2 to 4.7 seconds and 1 to 27 seconds for optimization when $n = 1$, $n = 100$ and $n = 500$ respectively. Compared to nested GA and EE-GMA(ASG-VTS), there is a reduction of 99% and 67% in the geometric mean of the normalized optimization time respectively even when $n = 500$. From these results, it can be observed that when the number of iterations is increased, both EGMS and EGMSIV are able to obtain feasible schedules with lower energy consumption at the expense of a longer optimization time. However, this optimization time is still shorter than those required by both nested GA and EE-GMA(ASG-VTS).

In addition to the hypothetical task graphs generated using TGFF [64], the designed algorithms are also applied to some task graphs corresponding to real-life examples. The experiment is repeated using the set of task graphs used by Bambha et al. [59]. The set of task graphs consists of 2 differently implemented fast Fourier transforms (fft1, fft3), a Karplus-strong music synthesis algorithm

Table 4.2: Normalized energy consumption achieved for mapping optimization using real-life applications used in [59]

Task Graph	Normalized Energy Consumption						
	Without Intra-task Voltage Scaling				With Intra-task Voltage Scaling		
	EE-GMA (ASG-VTS)	EE-GMA (EGMS-TSVS)	EGMS-TM (ASG-VTS)	EGMS(500)	Nested GA	EE-GMA (EGMSIV-TSVS)	EGMSIV(500)
fft1	1.073	0.992	1.052	0.885	1.000	0.900	0.792
fft3	1.207	1.051	1.192	0.989	1.000	1.017	0.957
karp10	1.081	0.946	1.031	0.894	1.000	0.900	0.873
qmf4	1.034	1.043	1.015	1.014	1.000	0.980	0.968
meas	1.029	1.029	1.029	1.029	1.000	0.999	0.999

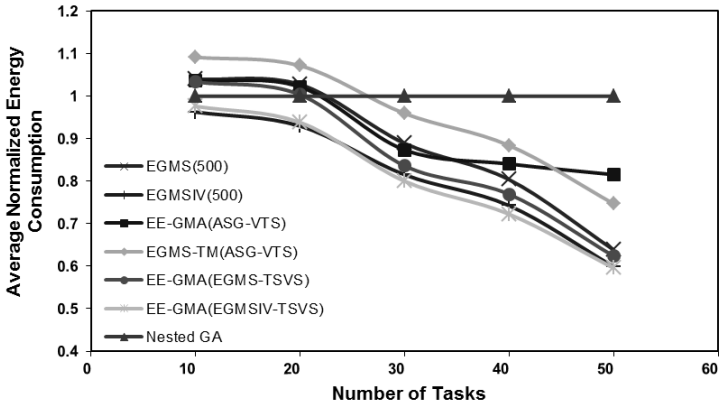
Table 4.3: Normalized optimization time required for mapping optimization using real-life applications used in [59]

Task Graph	Normalized Optimization Time						
	Without Intra-task Voltage Scaling				With Intra-task Voltage Scaling		
	EE-GMA (ASG-VTS)	EE-GMA (EGMS-TSVS)	EGMS-TM (ASG-VTS)	EGMS(500)	Nested GA	EE-GMA (EGMSIV-TSVS)	EGMSIV(500)
fft1	0.00403	0.01502	0.01546	0.00953	1.000	0.02954	0.01099
fft3	0.00025	0.00123	0.00081	0.00077	1.000	0.00111	0.00081
karp10	0.00036	0.00213	0.00144	0.00323	1.000	0.00296	0.00355
qmf4	0.1485	0.01086	0.02495	0.02242	1.000	0.05413	0.02675
meas	0.00396	0.00892	0.00188	0.00978	1.000	001390	0.01527

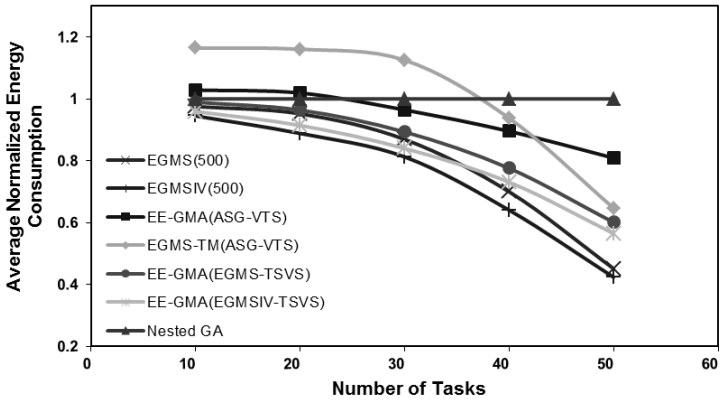
(karp10), a quadrature mirror filter bank (qmf4), and a measurement application (meas). These applications are run on multiprocessor platforms consisting of identical processors. The normalized energy consumption and normalized optimization time of the various algorithms are shown in Table 4.2 and 4.3. Again, the EGMS and EGMSIV algorithms achieve the best results in terms of energy minimization. In terms of optimization time, EE-GMA(ASG-VTS) achieves the best results for all the task graphs except for qmf4. However, the schedules generated by EE-GMA(ASG-VTS) consume about 11% more energy. From the results, it can be observed that although the algorithms are designed for heterogeneous multiprocessors, they can be used for homogeneous multiprocessors as well.

Lastly, the performance of the algorithms are evaluated when the number of tasks and processors increases. We randomly generated 1000 task graphs containing between 10 to 50 tasks each and obtained their average normalized energy consumption. The results are shown in Figure 4.9 for the cases when 3 and 6 processors are used respectively. From the figures, it is observed that when the number of tasks increases, the designed algorithms perform better. When 3 processors are used, both EGMSIV(500) and EE-GMA(EGMSIV-TSVS) have the best performance, followed by EE-GMA(EGMS-TSVS), EGMS(500), EE-GMA(ASG-VTS) and nested GA. EGMS-TM(ASG-VTS) does not perform well for smaller task graphs. However, it performs better than EE-GMA(ASG-VTS) when the number of task is large. When the number of processors is increased from 3 to 6, the EGMS and EGMSIV algorithms outperform the other algorithms, especially when the number of tasks is large. From these results, it is observed that when the number of tasks and processors increases, the search space becomes exponentially larger and therefore the EE-GMA genetic algorithm used in nested GA and EE-GMA(ASG-VTS) is unable to converge fast enough before the terminating condition is met. This is also the reason why EE-GMA(ASG-VTS) outperforms EGMS-TM(ASG-VTS) for smaller task graphs but not larger ones. However, when EE-GMA is used with the EGMS-TSVS and EGMSIV-TSVS algorithms, it is able to converge to better solutions faster. On the other hand, the EGMS and EGMSIV algorithms obtain the best performance as they are able to converge to very

good solutions by considering task mapping, task ordering and voltage scaling in an integrated way.



(a) 3 processors



(b) 6 processors

Figure 4.9: Average normalized energy consumption as the number of tasks increases

Chapter 5

Dynamic Energy-aware Scheduling Strategies for Systems with Single Energy Source

Static energy-aware scheduling algorithms use the WCETs of the tasks to generate energy efficient schedules without violating the deadline constraints. However, tasks may not require their WCETs to complete during runtime, resulting in the generation of slacks. Dynamic energy-aware scheduling algorithms are then employed during runtime to reclaim these slacks so as to reduce the energy consumption further. In this chapter, we describe two strategies to improve the runtime performance of the applications in terms of energy minimization. First, we proposed a method to improve the performance of static energy-aware scheduling

algorithms during runtime using Potential Slack for Dynamic Scheduling Considerations (PSDSC). Next, we present our Average-based Aggressive Dynamic Scheduling (AADS) algorithm that can be employed during runtime in order to minimize the overall average energy consumption.

5.1 Design of Potential Slack for Dynamic Scheduling Considerations (PSDSC)

5.1.1 Description of PSDSC

Most static energy-aware scheduling algorithms use the WCETs of the tasks in order to generate the static schedules. These schedules are energy-efficient if all the tasks require their WCETs to execute. When tasks do not require their WCETs to complete their execution most of the time, the average energy consumption may not be optimized. To generate a static schedule that is energy-efficient in the average case, we can simply replace the WCETs of the tasks with their ACETs in the static scheduling algorithms. However, the deadline requirements may not be met should the tasks' execution times be larger than their ACETs during runtime.

In order to rectify this, we propose the PSDSC method which can be incorporated

into various static energy-aware scheduling algorithms to improve their performance. During the generation of the static schedule, our proposed PSDSC takes into consideration the usage of a greedy slack reclamation scheme during runtime and tries to minimize the overall average energy consumption by using the concept of potential slack to estimate the dynamic execution speeds and energy consumption of the tasks. The greedy slack reclamation scheme [46] is a simple dynamic scheduling scheme where all the slack generated by a task during runtime is consumed by its immediate successor tasks to lower their execution speeds further. Let us first introduce the concept of potential slack that will be used in this proposed method. The potential slack ps_i of a task T_i is defined as follows:

$$ps_i = \begin{cases} 0 \\ \text{if } T_i \text{ has no precedents} \\ \\ \min_{T_p \in pred(T_i)}[(1-\Gamma_p) \times (ps_p + t^{wc}(p,j,k))] \\ \text{otherwise} \end{cases} \tag{5.1}$$

where T_p is a precedent task of T_i, Γ_p is the ratio of ACEC to WCEC of T_p and $t^{wc}(p,j,k)$ is the WCET of T_p when it is assigned to PE_j at voltage level $V(j,k)$. It should be noted that the ACEC and WCEC of a task are constant, but its ACET and WCET will vary depending on the speed at which the task is executed, as indicated in (2.7) and (2.8). The potential slack ps_i gives an estimate of the

potential amount of slack that T_i may receive from its precedent tasks based on their current processor and voltage assignments.

Next, we define the potential speed reduction factor of T_i when it is mapped to PE_j at the voltage level $V(j,k)$ as follows:

$$psrf(i,j,k) = \frac{t^{wc}(i,j,k)}{\min(t^{wc}(i,j,1), ps_i + t^{wc}(i,j,k))} \tag{5.2}$$

The potential speed reduction factor $psrf(i,j,k)$ gives an estimate of the factor by which the speed of PE_j may be reduced if T_i is assigned to it at voltage level $V(j,k)$. This factor is calculated based on the potential amount of slack that T_i may receive from its precedent tasks and is limited by the lowest voltage level $V(j,1)$ at which T_i can be executed on PE_j. Based on the potential speed reduction factor, the potential energy reduction factor of T_i when it is mapped to PE_j at the voltage level $V(j,k)$ can be calculated:

$$perf(i,j,k) = (\frac{V_{psrf(i,j,k)}}{V(j,k)})^2 \times \Gamma_i \tag{5.3}$$

where $V_{psrf(i,j,k)}$ is the voltage of PE_j at a speed that is a fraction $psrf(i,j,k)$ of the speed at voltage level $V(j,k)$. The average-case energy consumption needed

to execute T_i on PE_j at voltage level $V(j,k)$ is therefore given as follows:

$$e^{ac}(i,j,k) = perf(i,j,k) \times e^{wc}(i,j,k) \tag{5.4}$$

In order to apply the PSDSC method to the various static energy-aware scheduling algorithms, we need to incorporate the calculation of the potential slack and the average-case energy consumption into the algorithms. In each scheduling step, we calculate the average-case energy consumption $e^{ac}(i,j,k)$ of each task based on the current processor and voltage mapping of their precedent tasks and use these values to calculate the total average-case energy consumption. By doing this, we try to minimize the overall average energy consumption in the objective function. However, when calculating the start and end times of the tasks, we use their original worst-case execution times. This ensures that the generated static schedule is still feasible and meets the deadline requirements even if all the tasks require their WCETs to execute.

5.1.2 Illustrative Example for PSDSC

The exact steps of applying PSDSC differ depending on the static energy-aware scheduling algorithm into which it is to be incorporated. As an example to illustrate the workings of PSDSC, we shall incorporate the use of PSDSC into the EGMS algorithm. In our modified EGMS-PSDSC algorithm, the potential slacks of the tasks can be calculated during the task ordering step using the CPTO algorithm. Algorithm 4 shows the pseudo-code of the modified CPTO algorithm. In line 5, the critical paths $l_{cp}(j)$ are calculated based on the ACETs of the tasks. Whenever a computational task is scheduled based on its critical path (lines 17-21), we calculate the potential slack and the average-case energy consumption of the task based on the current processor and voltage mapping of its precedent tasks using (5.1)-(5.4). This average-case energy consumption is then used in the calculation of the total energy consumption e of the current schedule. However, we do not use the ACET of the task in the calculation of the makespan m_s of the current schedule. Instead, we use the task's WCET in the calculation of m_s. This helps to determine if the current schedule will still meet its deadline requirements in the worst-case scenario.

It should be noted that the potential slack and average-case energy consumption of a task varies according to the processor and voltage mapping of its precedent tasks. Therefore in the EGMS-PSDSC algorithm, before each iteration to re-map

Algorithm 4 : CPTO(M_p, M_v, e, m_s) *Modified for PSDSC*

$e \leftarrow 0$
$m_s \leftarrow 0$
Replace the communication between any 2 tasks that are scheduled on different processors with a task, where the execution time is the communication time
for all T_j **do** /* Both computational and communication tasks */
  Calculate $l_{cp}(j)$ /* Length of critical path starting from T_j using $t^{ac}(j, M_p(j), M_v(j))$ */
  $t_{start}(j) \leftarrow 0$ /* Initialize start time of T_j to 0 */
  $t_{end}(j) \leftarrow 0$ /* Initialize end time of T_j to 0 */
end for
Insert tasks with no incoming edges into priority queue Q, where tasks are sorted in decreasing values of l_{cp}
while Q is not empty **do**
  Remove T_j from front of Q. /* Get task with largest critical path length */
  if T_j is communication task **then**
    $e \leftarrow e$+Communication energy
    $t_{start}(j) \leftarrow$ Earliest time communication bus is free
    $t_{end}(j) \leftarrow t_{start}(j)$+ Communication time
  else /* T_j is computational task */
    Calculate ps_j, $psrf(j, M_p(j), M_v(j))$ and $perf(j, M_p(j), M_v(j))$
    Calculate $e^{ac}(j, M_p(j), M_v(j))$ based on $perf(j, M_p(j), M_v(j))$
    $e \leftarrow e + e^{ac}(j, M_p(j), M_v(j))$
    $t_{start}(j) \leftarrow$ Earliest time $PE_{M_p(j)}$ is free
    $t_{end}(j) \leftarrow t_{start}(j) + t^{wc}(j, M_p(j), M_v(j))$
  end if
  if $t_{end}(j) > m_s$ **then**
    $m_s \leftarrow t_{end}(j)$
  end if
  Remove T_j and all outgoing edges from task graph
  Insert tasks with no incoming edges into priority queue Q
end while

a task to a new processor and/or a new voltage level, the average-case energy consumption of each task $e^{ac}(i, j, k)$ has to be updated for all processing elements and at all voltage levels based on the current schedule obtained so far. This updated values of $e^{ac}(i, j, k)$ are then used in the calculation of the energy gradient used by the EGMS-PSDSC algorithm to select the task to be re-mapped.

The same modifications can also be applied to the EGMSIV algorithm using PS-DSC (EGMSIV-PSDSC) for intra-task voltage scaling.

To illustrate the performance improvement that can be achieved by the modified EGMS-PSDSC/EGMSIV-PSDSC algorithms, let us consider a periodic hard real-time application represented by a task graph as shown by Figure 5.1. The edges between two tasks represents the communication times required between them if they are mapped onto different processing elements. The period of the application is set at 50. Communication power is set at 1. For illustration purpose, we assume that the task graph is to be mapped to a system with two homogeneous processing elements. Each processing element supports four execution speeds at $0.25f_{max}$, $0.5f_{max}$, $0.75f_{max}$ and f_{max}, where f_{max} is the highest execution speed supported by the processing element. The worst-case execution times and energy consumptions of the tasks are shown in Tables 5.1 and 5.2 respectively. Let Γ_i, be 0.5 for all the tasks.

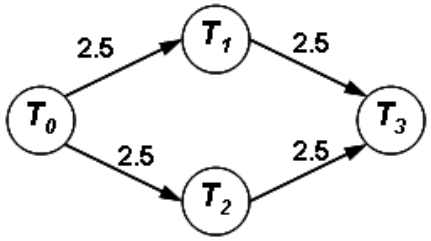


Figure 5.1: Directed acyclic graph representing the periodic hard real-time application used in the example

Table 5.1: Worst-case execution times of the tasks on different processing elements and at different voltage levels

Worst-Case Execution Times								
Task	PE_0				PE_1			
	V_0	V_1	V_2	V_3	V_0	V_1	V_2	V_3
T_0	40	20	13.3	10	40	20	13.3	10
T_1	40	20	13.3	10	40	20	13.3	10
T_2	40	20	13.3	10	40	20	13.3	10
T_3	40	20	13.3	10	40	20	13.3	10

Table 5.2: Worst-case energy consumptions of the tasks on different processing elements and at different voltage levels

Worst-Case Energy Consumptions								
Task	PE_0				PE_1			
	V_0	V_1	V_2	V_3	V_0	V_1	V_2	V_3
T_0	0.63	2.5	5.63	10	0.63	2.5	5.63	10
T_1	0.63	2.5	5.63	10	0.63	2.5	5.63	10
T_2	0.63	2.5	5.63	10	0.63	2.5	5.63	10
T_3	0.63	2.5	5.63	10	0.63	2.5	5.63	10

We generate two different static schedules using EGMSIV and EGMSIV-PSDSC. Figure 5.2 shows the static schedules that are generated using the two algorithms. We observe that if all the tasks require their WCETs to execute, EGMSIV generates a more energy-efficient schedule with 22% lower energy consumption as compared to the schedule generated by EGMSIV-PSDSC. However, if the tasks execute at their ACETs, slacks are generated during runtime. For both algorithms, we use the greedy slack reclamation scheme [46] during runtime to dynamically lower the execution speeds of the tasks in order to achieve more energy savings. In this scheme, when a task does not require its WCET to complete its execution, all its unused time will be passed to its immediate successor tasks so that they can further lower their execution speeds. Figure 5.3 shows the runtime schedules when the tasks execute at their ACETs. We observe that in this case, the schedule generated

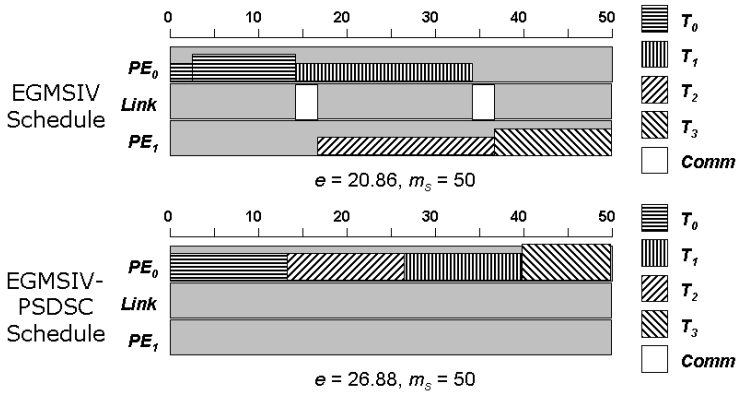


Figure 5.2: Runtime schedules when all tasks require their WCETs to complete their execution and use the dynamic greedy slack reclamation scheme to lower the energy consumption during runtime

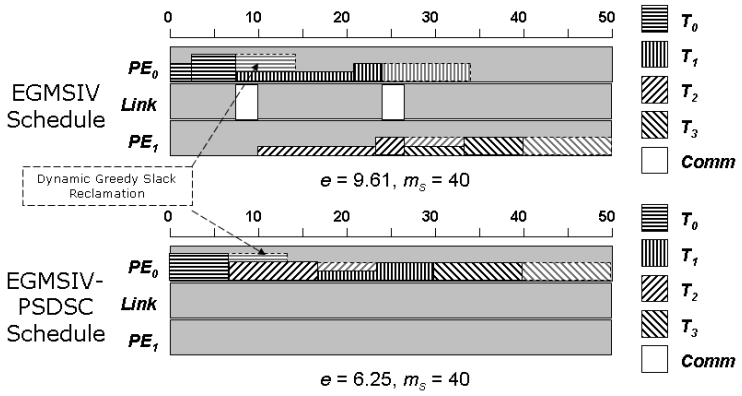


Figure 5.3: Runtime schedules when all tasks require their ACETs to complete their execution and use the dynamic greedy slack reclamation scheme to lower the energy consumption during runtime

by EGMSIV-PSDSC is more energy-efficient with 35% lower energy consumption compared to that generated by EGMSIV.

5.1.3 Performance of PSDSC

Next, we evaluate the performance of our proposed PSDSC method. PSDSC can be incorporated into various static energy-aware scheduling algorithms. First, we

shall evaluate the performance of the EGMS, EGMSIV and the nested GA (NGA) algorithms with and without incorporating our proposed PSDSC. The PSDSC method is incorporated into the nested GA algorithm by taking into consideration the use of potential slacks of the tasks during the calculation of the objective function. We set the user-defined parameter $n = 500$ as the terminating condition for the EGMS and EGMSIV algorithms when generating the static schedules. For the nested GA algorithm, we configure the inner EE-GLSA algorithm to terminate when there is no improved solution being produced for 10 successive generations while the outer EE-GMA algorithm terminates when there is no improvement after 25 successive generations.

We randomly generated task graphs comprising between 10 to 50 tasks using TGFF [64] running on 2 to 4 processors. In this experiment, we vary Γ, the ratio of ACEC to WCEC of the tasks in the task graph. For each task T_i, we normalized the actual number of execution cycles to its WCEC and randomly generate the normalized actual number of execution cycles using a Gaussian distribution with a mean of Γ and a variance of 0.1 for 1000 execution instances of the application. During runtime, all the algorithms use the dynamic greedy slack reclamation scheme [46] to further reduce the execution speeds of the tasks.

Figure 5.4 shows the average energy consumption over 1000 execution instances of

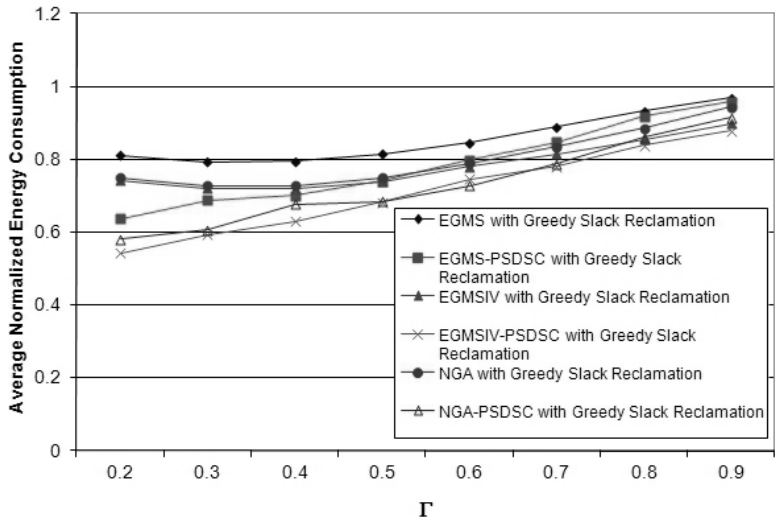


Figure 5.4: Average normalized energy consumption over 1000 execution instances using EGMS, EGMSIV, EGMS-PSDSC, EGMSIV-PSDSC, NGA and NGA-PSDSC with varying Γ

the application using the EGMS, EGMSIV and NGA algorithms with and without incorporating our proposed PSDSC. The values of energy consumption are normalized to that obtained when the static EGMS schedule is used without any dynamic slack reclamation during runtime. We observe that at $\Gamma = 0.2$, EGMS-PSDSC, EGMSIV-PSDSC and NGA-PSDSC consume 21.4%, 26.8% and 22.5% less energy when compared to EGMS, EGMSIV and NGA respectively. EGMSIV-PSDSC and NGA-PSDSC are able to achieve better improvement than EGMS-PSDSC as they consider intra-task voltage scaling to reduce the energy consumption further. We also observe that at $\Gamma \leq 0.4$, the normalized energy consumption due to EGMS, EGMSIV and NGA remains relatively constant, indicating that most of the tasks are already scheduled at the minimum possible voltage levels. As a result, the

dynamic greedy slack reclamation scheme is unable to further reduce the energy consumption of the task graphs. On the other hand, EGMS-PSDSC, EGMSIV-PSDSC and NGA-PSDSC are able to further reduce the energy consumption of the task graphs at low values of Γ by using different processor mappings and task orderings that take into consideration the use of dynamic greedy slack reclamation during runtime. As the value of Γ increases, the difference in normalized energy consumption between EGMS and EGMS-PSDSC gradually becomes smaller. The same observation can also be seen between the normalized energy consumption of EGMSIV and EGMSIV-PSDSC, as well as that of NGA and NGA-PSDSC. This is due to the fact that as Γ approaches 1, there is less dynamic slack that can be reclaimed. As a result, the performances of EGMS-PSDSC, EGMSIV-PSDSC and NGA-PSDSC gradually approach those of EGMS, EGMSIV and NGA respectively.

5.2 Design of Average-based Aggressive Dynamic Scheduling (AADS)

5.2.1 Description of AADS

In the dynamic greedy slack reclamation scheme, all the slack that is generated by a task is used by its immediate successor tasks in the task graph to lower their execution speeds and energy consumptions. As we progress down the task graph,

the amount of slack gets accumulated and when all the tasks have finished their execution, there may still be a significant amount of slack that remains unused. For example, in the two schedules shown in Figure 5.3, there are still 10 units of unused slack after the last task T_3 completes its execution.

In order to reclaim the slack more effectively, we propose the AADS dynamic scheduling algorithm. In this algorithm, we aggressively run the tasks near the root nodes of the task graph at lower speeds so that less slack will be generated and passed on to the successor tasks. In order to illustrate our proposed algorithm more clearly, let us first assume that the voltage and frequency levels of the processing elements are continuous. We obtain the expected start and end times ($st_i^{ac,avg}$ and $et_i^{ac,avg}$ respectively) at which the task T_i is to be executed if it requires its ACEC instead of WCEC to complete. Then,

$$f_i^{ac,avg} = \frac{c_i^{ac}}{(et_i^{ac,avg} - st_i^{ac,avg})} \tag{5.5}$$

where $f_i^{ac,avg}$ is the expected frequency at which T_i executes assuming that it requires its ACEC to complete. The values of $st_i^{ac,avg}$ and $et_i^{ac,avg}$ can be obtained using the predetermined task mapping and ordering of the static schedule and replacing the worst-case execution times and energy consumptions of the tasks with their corresponding average-case values. Note that $f_i^{ac,avg}$ represents the frequency

at which T_i should execute if it requires exactly its ACEC to run. However during the actual runtime, T_i may require fewer or more execution cycles to complete. In the worst-case, it will require its WCEC to complete. If T_i requires more than its ACEC to complete, we can either execute the extra cycles at the same frequency $f_i^{ac,avg}$ or at the maximum frequency of the processing element f_i^{max}. Executing the extra cycles at $f_i^{ac,avg}$ allows T_i to have a lower energy consumption, but has a greater impact on the successor tasks down the task graph as they need to execute at a higher speed (and thus consume more energy) to meet the deadlines. Alternatively, we can execute the extra cycles at f_i^{max}. This may increase the energy consumption of T_i, but will have less impact on the successor tasks, allowing them to execute at a lower speed. Let us execute the extra cycles at f_i^{max}. Therefore, the potential maximum end time of T_i may be calculated as:

$$et_i^{ac,max} = et_i^{ac,avg} + \frac{(c_i^{wc} - c_i^{ac})}{f_i^{max}} \tag{5.6}$$

This end time must not be larger than the end time generated by the ALAP (As Late As Possible) schedule in order to guarantee that the deadline constraints are met:

$$et_i^{ac,max} \leq et_i^{ALAP} \tag{5.7}$$

Therefore, we can execute the task T_i at two different frequencies. For the first c_i^{ac} cycles, T_i executes at the frequency f_i^{dyn}:

$$f_i^{dyn} = \min\left(\frac{c_i^{ac}}{\min(et_i^{ac,avg}, et_i^{dyn}) - st_i^{ac,avg}}, f_i^{max}\right) \tag{5.8}$$

where

$$et_i^{dyn} = et_i^{ALAP} - \frac{(c_i^{wc} - c_i^{ac})}{f_i^{max}} \tag{5.9}$$

For the next $(c_i^{wc} - c_i^{ac})$ cycles, T_i executes at f_i^{max}.

The ACEC of a task is the long term average number of execution cycles required to complete the task. During runtime, the task may require less than its ACEC to complete for a period of time, and then switch to its WCEC for execution for the subsequent period of time. To optimize the energy consumption further and adapt to the changing operating environment, we replaced the long term ACEC of T_i with a moving average $\eta_{i,t}$ instead during runtime:

$$\eta_{i,t} = \max(\frac{1}{n_w}\sum_{j=t-n_w}^{t-1} c_{i,j}^{actual}, c_{i,t-1}^{actual}) \tag{5.10}$$

where n_w is the size of the moving window, t represents the t-th execution instance of the application and $c_{i,j}^{actual}$ is the actual number of execution cycles required to run T_i to completion during the j-th execution instance of the application. $\eta_{i,t}$ uses the average number of execution cycles for the previous n_w execution instances of the application to adapt to the changing executing environment. At the same time, if there is a sudden sharp increase in the number of execution cycles in the previous execution instance, the algorithm will quickly adapt to this sharp increase to reduce the occurrence of the tasks running at the maximum speeds of the processing elements. Therefore during the t-th execution instance of the application, the frequency at which to execute the first $\eta_{i,t}$ cycles of T_i can now be calculated as:

$$f_{i,t}^{dyn} = \min\left(\frac{\eta_{i,t}}{\min(et_i^{ac,avg}, et_i^{dyn}) - st_{i,t}^{dyn}}, f_i^{max}\right) \tag{5.11}$$

where $st_{i,t}^{dyn}$ is the actual start time of T_i during the t-th execution instance of the application. T_i will then execute the next $(c_i^{wc} - \eta_{i,t})$ cycles at f_i^{max}.

In practical scenarios where the available execution speeds of the processing elements are discrete, we can execute the first portion of the task on PE_j at two consecutive speeds $f(j,k)$ and $f(j,k+1)$ such that $f(j,k) \leq f_{i,t}^{dyn} \leq f(j,k+1)$, as described in [63]. In this case, we execute the task T_i on PE_j at the following

frequencies:

$$\begin{cases} f(j,k) & \text{for } \lfloor \frac{f(j,k)\cdot(f(j,k+1)-f_{i,t}^{dyn})}{f_{i,t}^{dyn}\cdot(f(j,k+1)-f(j,k))} \cdot \eta_{i,t} \rfloor \text{ cycles} \\ f(j,k+1) & \text{for the next } \lceil \frac{f(j,k+1)\cdot(f_{i,t}^{dyn}-f(j,k))}{f_{i,t}^{dyn}\cdot(f(j,k+1)-f(j,k))} \cdot \eta_{i,t} \rceil \text{ cycles} \\ f(j,N_v) & \text{for the remaining } c_i^{wc} - \eta_{i,t} \text{ cycles} \end{cases} \quad (5.12)$$

5.2.2 Illustrative Example for AADS

To illustrate the workings of the AADS algorithm, let us consider the example in Section 5.1.2. Suppose the tasks in the task graph have been requiring their ACECs to complete for the previous n_w execution instances. In this case, $\eta_{i,t} = c_i^{ac}$ for each task T_i. Instead of using the dynamic greedy slack reclamation, we apply our AADS algorithm during runtime to the static schedules generated by EGMSIV and EGMSIV-PSDSC. The resulting runtime schedules are shown in Figure 5.5. From the runtime schedules, we observe that the AADS algorithm tries to aggressively lower the execution speeds of the tasks in order to reduce the amount of slack that will be generated and passed to the successor tasks. Compared to the dynamic greedy slack reclamation scheme, AADS consumes 16% less energy when used with the static schedule generated by EGMSIV and 27% less energy when used

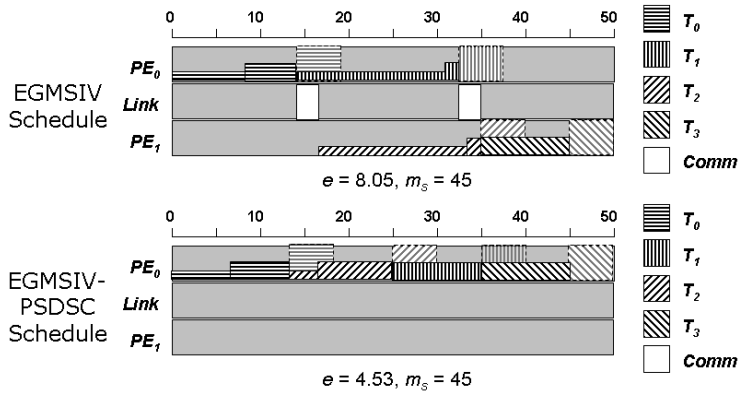


Figure 5.5: Runtime schedules when all tasks require their ACETs to complete their execution and use the AADS algorithm to lower the energy consumption during runtime

with the static schedule generated by EGMSIV-PSDSC.

The schedule shown in Figure 5.5 is based on the ideal case when all the tasks require their ACETs to complete. Now, let us assume that starting from the next instance of execution, the tasks now require their WCETs to complete. During the next instance of execution, $\eta_{i,t}$ is still c_i^{ac} and the resulting runtime schedules are shown in Figure 5.6. We can observe that $\eta_{i,t}$ underestimates the number of execution cycles of the tasks, resulting in significant portions of the tasks being executed at f_i^{max}. This increases the energy consumption substantially and results in more energy consumed when compared to the schedules in Figure 5.2.

However, our AADS algorithm tries to adapt quickly to the changing conditions. In subsequent instances of execution, $\eta_{i,t}$ will be updated to c_i^{wc} and the resulting

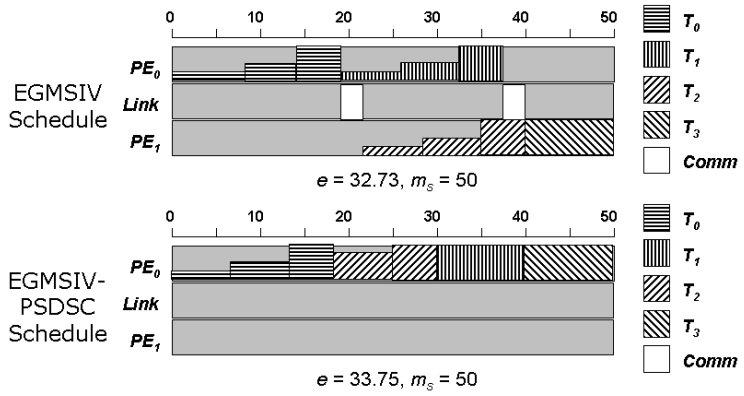


Figure 5.6: Runtime schedules immediately after all tasks switch to their WCETs to complete their execution and use the AADS algorithm to lower the energy consumption during runtime

schedules will now become the same as those shown in Figure 5.2.

5.2.3 Performance of AADS

Next, we shall evaluate the performance of our AADS algorithm. We compare our algorithm with the dynamic greedy slack reclamation scheme when applied to the static schedules generated by EGMSIV, EGMSIV-PSDSC, NGA and NGA-PSDSC. Just as in the previous set of simulation experiments, we use randomly generated task graphs consisting of 10 to 50 tasks and run the experiment over 1000 execution instances at varying values of Γ. Figures 5.7 and 5.8 show the average normalized energy consumption over 1000 execution instances of the application.

We observe that at $\Gamma = 0.3$, AADS is able to achieve an average of 6.1% and 4.5% more energy saving compared to the dynamic greedy slack reclamation scheme

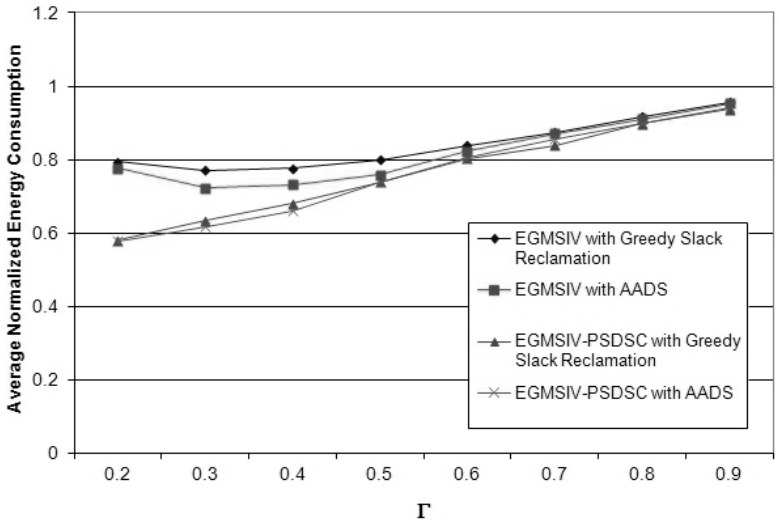


Figure 5.7: Average normalized energy consumption over 1000 execution instances when dynamic greedy slack reclamation and AADS are applied to EGMSIV and EGMSIV-PSDSC with varying Γ

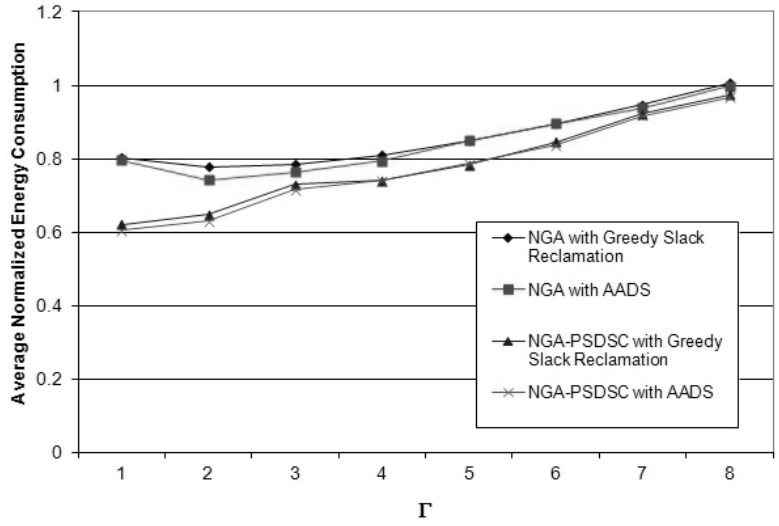


Figure 5.8: Average normalized energy consumption over 1000 execution instances when dynamic greedy slack reclamation and AADS are applied to NGA and NGA-PSDSC with varying Γ

when they are applied to the static schedules generated by EGMSIV and NGA respectively. However when $\Gamma = 0.2$, the improvement is only about 2.4% and

1.0% respectively. This is due to the fact that at $\Gamma = 0.2$, most of the tasks will be scheduled at the minimum possible voltage levels during runtime and therefore there is little difference in the average energy consumed when either AADS or dynamic greedy slack reclamation is used. For $\Gamma \geq 0.3$, the difference in average energy consumption between AADS and dynamic greedy slack reclamation becomes smaller when Γ increases as there is less dynamic slack that can be reclaimed. On the other hand, we observe that the performance of AADS is similar to that of dynamic greedy slack reclamation when applied to the static schedules generated by EGMSIV-PSDSC and NGA-PSDSC, with improvements of less than 2%. This is because EGMSIV-PSDSC and NGA-PSDSC take into consideration the workings of the dynamic greedy slack reclamation scheme when generating the static schedules and therefore the mapping and ordering of the tasks are optimized for dynamic greedy slack reclamation. As a result, the performance of the dynamic greedy slack reclamation scheme is almost as good as AADS.

Lastly, we shall examine the effect of the rate of changing execution cycles of the tasks on the performance of our AADS algorithm. We generate the actual execution cycles of the tasks such that 75% of the time the tasks execute at 20% of their WCECs while the remaining 25% of the time the tasks execute at their WCECs. In this case, $\Gamma = 0.4$. We define T as the number of execution instances before the actual execution cycles of the tasks increase from 20% to 100% of their WCECs

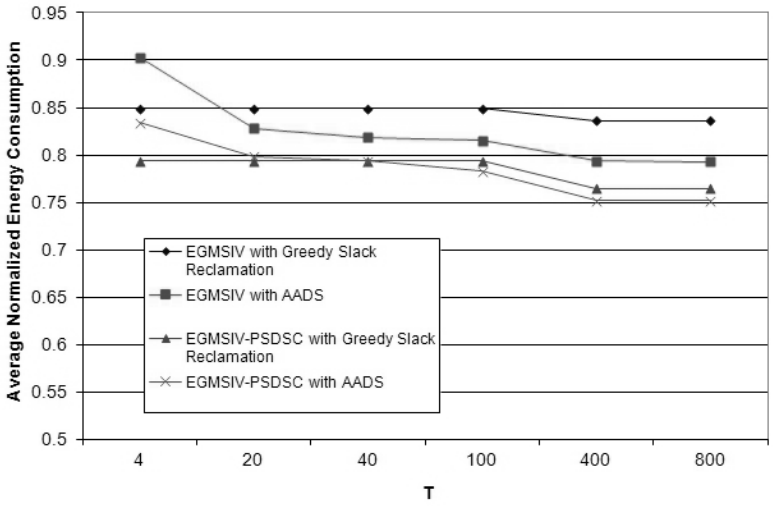


Figure 5.9: Average normalized energy consumption over 1000 execution instances with varying T

and vice versa. We vary the values of T and obtain the average normalized energy consumption over 1000 execution instances when AADS and dynamic greedy slack reclamation are applied to EGMSIV and EGMSIV-PSDSC. The results are shown in Figure 5.9.

We observe that for high values of T, the AADS outperforms the dynamic greedy slack reclamation scheme by 5.2% and 1.7% when applied to EGMSIV and EGMSIV-PSDSC respectively. However, at low values of T, the dynamic greedy slack reclamation performs better. The main reason is that at low values of T, the rate of change of the execution cycles of the tasks is high. As we observe in our illustrative example in Section 5.2.2, whenever there is a sudden sharp increase to the execution cycles of the tasks, the AADS algorithm significantly underestimates

the average execution cycles of the tasks in the next execution instance of the task graph. This results in a very unoptimized schedule in this execution instance before the AADS algorithm adjusts to the change in subsequent execution instances. When the rate of change of the execution cycles increases, the overall ACECs of the tasks remain unchanged, but the occurrence of the unoptimized runtime schedules increases. This results in an increase in the average energy consumption of the tasks. From the results of this experiment, we conclude that although AADS consumes less energy than dynamic greedy slack reclamation in general, it cannot be used for applications in which the tasks undergo sharp increase in the execution cycles at a fast rate.

Chapter 6

Energy-aware Scheduling Strategies for Systems with Distributed Energy Sources

Unlike multiprocessor systems with a single energy source, minimizing the total energy consumption does not necessary improve the lifetime of systems with distributed energy sources. One such system is the WSN. In this chapter, we present the Energy-Balanced Task Scheduling (EBTS) algorithm to address this problem. We also describe the EBTS with Dual Schedule (EBTS-DS) algorithm which extends the EBTS algorithm by generating a second static schedule that compliments the original schedule to increase the lifetime of the system further. Rigorous simulation experiments are then conducted to evaluate the performance of our proposed algorithms.

6.1 Design of Energy-Balanced Task Scheduling (EBTS)

6.1.1 Description of EBTS

Today, WSNs are used in a wide variety of applications such as health monitoring, target tracking and surveillance. These applications often require each sensor node to sense and collect information from the surrounding environment, process the information collected, and communicate the results to other sensor nodes in the network. Based on the collective information gathered from several sensor nodes, a decision can then be made to determine the action to be taken.

In many WSNs, the sensors are individually operated by battery. Efficient energy management is required to prolong the lifetime of such sensor networks. While there are many energy-aware scheduling algorithms in the literature for both homogeneous [31, 32, 44–46, 49] and heterogeneous [39, 47, 52, 54, 60, 61] multiprocessor systems, these algorithms are designed mainly for tightly coupled systems and are not suitable for wireless sensor networks. These algorithms try to minimize the overall total energy consumption of the system in order to maximize its lifetime. This works for tightly coupled system since the processors in the system share the same energy source. However, for wireless sensor network, minimizing the total

energy consumption does not necessarily maximize the lifetime of the network. For example, if many tasks are assigned to a single node, the battery on this node will be drained at a rate much faster than other nodes. As a result, the lifetime of the network is completely determined by the lifetime of this single node even when other nodes have an abundance of remaining energy. Therefore, in order to maximize the lifetime of a WSN, we should try to distribute the tasks among the sensor nodes in such a way that the energy consumption on each sensor node is as balanced as possible.

We consider a WSN with the properties as described in Section 2.3. The pseudocode for the algorithm can be seen in Algorithm 5. In Step 1, the average computational time of each task over all the sensor nodes and the average communication time of each edge over all pairs of sensor nodes are first calculated. Next, the upper rank $rank_u$, as defined in [26], is calculated recursively using the following equation:

$$rank_u(T_i) = \overline{t_i} + \max_{T_j \in succ(T_i)} (\overline{\tau_{\theta(T_i)\theta(T_j)}} + rank_u(T_j)) \tag{6.1}$$

where $\overline{t_i}$ is the average computational time of T_i over all the sensor nodes, $\overline{\tau_{\theta(T_i)\theta(T_j)}}$ is the average communication time of the edge from T_i to T_j over all sensor node pairs and $succ(T_i)$ denotes the immediate successors of T_i.

In Step 2, the tasks are assigned in descending order of upper rank. For each task, the priority is calculated assuming that the task is assigned to each sensor node. The task is then assigned to the sensor node that gives the lowest priority value, provided that its finish time when assigned to this sensor node does not exceed a threshold value $\sigma(T_i)$.

$$\sigma(T_i) = \frac{P}{m_s} \cdot f_i' \tag{6.2}$$

where m_s is the makespan of the schedule generated by a general list scheduling algorithm and f_i' is the finish time of T_i using the list scheduling algorithm. If the threshold is exceeded, the task will be assigned to the sensor node that gives the earliest finish time. A threshold is imposed to the finish time to reduce the probability of many tasks being assigned to the same sensor node as this may result in deadlines to be missed.

In Step 3, the sensor node with the largest norm-energy is first identified. Among the tasks that are assigned to this sensor node, the one that has the largest energy reduction when assigned to the next voltage level without violating the deadline constraints is selected. If no such tasks exists, the sensor node is removed permanently from the priority queue and does not need to be considered further. Otherwise, the new norm-energy of the sensor node is updated accordingly and the node is reinserted back into the priority queue. The process continues until no

Algorithm 5 EBTS

Step 1: Calculate Upper Rank
Assign computation time of tasks with the mean values over all sensor nodes. Assign communication time of edges with mean values overall all pairs of sensor nodes.
Compute the upper rank of all the tasks by traversing the DAG upwards, starting from the sink tasks.

Step 2: Assign Tasks to Sensor Nodes
Sort the tasks in non-increasing order of upper rank.
while there are unscheduled tasks **do**
  Select the first task T_i from the sorted list.
  for each sensor node PE_j in the system **do**
    Assume that T_i is allocated to PE_j.
    Compute the norm-energy η_k for all sensor nodes where $k = 1, 2, \cdots N_p$.
    Compute the finish time f_j of T_i.
    Calculate the priority function $pr_j = \alpha \cdot \max(\eta_k) + (1 - \alpha) \cdot \sum \eta_k$, where $\alpha \in [0, 1]$ is a user-defined parmeter. If T_i is a source task and another source task is already assigned to PE_j, Set $pr_j = \infty$.
  end for
  Assign T_i to the sensor node with the least value of pr_j, provided that the finish time f_j of this assignment is less than or equal to the threshold $\sigma(T_i)$.
  If no such sensor node exists, assign T_i to the sensor node that gives the smallest value of f_j.
end while

Step 3: Assign Voltage Levels
Insert the sensor nodes into a priority queue Q sorted in non-increasing order of norm-energy.

while Q is not empty **do**
  Remove the first sensor node PE_j from Q.
  Sort the tasks assigned to this sensor node in non-increasing order of the difference in energy consumption when their voltage is lowered by one level.
  for each task T_i in the sorted list **do**
    Lower voltage of T_i to the next lower voltage level.
    if deadline is not exceeded **then**
      Update the schedule.
      Insert this sensor node back into Q with the updated value of norm-energy. Break out of for loop.
    end if
  end for
end while

tasks can be executed at a lower voltage level without exceeding the deadline.

6.1.2 Illustrative Example for EBTS

We shall use the same illustrative example used in [33] to illustrate the workings of our algorithm and compare it to the 3-phase heuristic. Figure 6.1 shows the precedence relationships among the various tasks in the application task graph. These tasks are to be assigned to a WSN consisting of three homogeneous nodes. The time and energy costs of executing the tasks at different voltage levels are shown in Table 6.1. Each sensor node has an energy capacity of 1000.

In Step 1, we calculate the upper rank of each task. The result is shown in the last column of Table 6.1. Next, we arrange the tasks in descending order of upper rank. For each task, we calculate the priority function of each sensor node assuming that the task is assigned to that node. We use the value $\alpha = 0.5$ in this example. We then allocate the task to the sensor node with the lowest priority value. Table 6.2 shows the various steps involved in assigning the tasks to the sensor nodes.

Lastly, we shall apply voltage scaling to the tasks to further reduce the energy consumption. We select the sensor node with the largest value of norm-energy. Among the tasks assigned to this node, we choose one which will result in the largest reduction in energy consumption when it is assigned to the next lower voltage level without exceeding the deadline. This process of selecting the sensor node and task is then repeated until no tasks can be assigned to a lower voltage

Task	Time Cost		Energy Cost		Upper Rank
	V_h	V_l	V_h	V_l	
T_1	10	33	20	6	130
T_2	60	199	120	36	90
T_3	10	33	20	6	80
T_4	10	33	20	6	95
T_5	20	66	40	12	75
T_6	10	33	20	6	45
T_7	10	33	20	6	10

Table 6.1: Time and energy cost of each task at different voltage levels

level without exceeding the deadline. The steps are shown in Table 6.3. The final schedule is shown in Figure 6.2(a). From the example, we obtain a maximum norm-energy of 0.081 among the sensor nodes, which corresponds to a lifetime of 12 cycles. Compared to the 3-phase heuristic which gives a maximum norm-energy of 0.1 (corresponding to a lifetime of 10 cycles), our algorithm is able to provide an improvement of 20% in the lifetime of the sensor network.

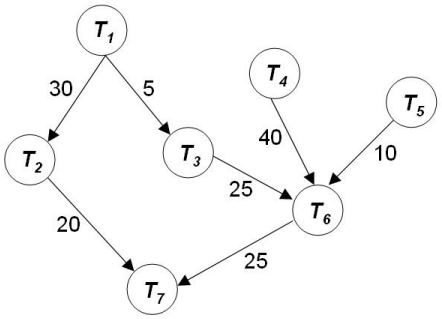


Figure 6.1: Example of a task graph.

Step	Selected Task	Sensor Node in Consideration	Priority	Selected Sensor Node	Norm-energy after Assignment
1	T_1	PE_1 PE_2 PE_3	40/1000 40/1000 40/1000	PE_1	20/1000 0/1000 0/1000
2	T_4	PE_1 PE_2 PE_3	∞ 60/1000 60/1000	PE_2	20/1000 20/1000 0/1000
3	T_2	PE_1 PE_2 PE_3	300/1000 390/1000 370/1000	PE_1	140/1000 20/1000 0/1000
4	T_3	PE_1 PE_2 PE_3	340/1000 335/1000 335/1000	PE_2	145/1000 45/1000 0/1000
5	T_5	PE_1 PE_2 PE_3	∞ ∞ 375/1000	PE_3	145/1000 45/1000 40/1000
6	T_6	PE_1 PE_2 PE_3	640/1000 415/1000 525/1000	PE_2	145/1000 75/1000 50/1000
7	T_7	PE_1 PE_2 PE_3	530/1000 495/1000 545/1000	PE_2	165/1000 115/1000 50/1000

Table 6.2: Steps to illustrate how tasks are assigned to sensor nodes using the EBTS algorithm

Step	Selected Sensor Node	Selected Task	Makespan after Voltage Scaling	Norm-energy after Voltage Scaling
0	-	-	100	PE_1: 165/1000 PE_2: 115/1000 PE_3: 50/1000
1	PE_1	T_2	239	PE_1: 81/1000 PE_2: 115/1000 PE_3: 50/1000
2	PE_2	T_3	239	PE_1: 81/1000 PE_2: 101/1000 PE_3: 50/1000
3	PE_2	T_4	239	PE_1: 81/1000 PE_2: 87/1000 PE_3: 50/1000
4	PE_2	T_6	239	PE_1: 81/1000 PE_2: 73/1000 PE_3: 50/1000
5	PE_3	T_5	239	PE_1: 81/1000 PE_2: 73/1000 PE_3: 22/1000

Table 6.3: Steps to illustrate how voltage levels are assigned using the EBTS algorithm

6.2 Design of EBTS using Dual Schedule (EBTS-DS)

6.2.1 Description of EBTS-DS

Although the EBTS algorithm tries to improve the lifetime of the sensor network by balancing the energy consumption among the sensor nodes, it is often very

difficult to obtain a perfectly energy-balanced schedule in practice. As a result, when the batteries of some sensor nodes are depleted, there is still a significant amount of remaining energy in other sensor nodes. If the tasks can be reassigned from a sensor node that is low in remaining energy to one that is high in remaining energy in an effective way, the lifetime of the network can be increased further. In view of this, we propose to use two static schedules for executing the tasks instead of a single schedule. The aim is to use a second schedule that compliments the first one such that the lifetime of the network can be increased by the combined use of both schedules compared to using each schedule individually. The EBTS-DS algorithm is shown in Algorithm 6.

In EBTS-DS, a schedule is first generated using the EBTS algorithm. This will be the first schedule S^{1st}. Next, a series of candidate schedules are generated. Each candidate schedule is generated in the following way. We assume that S^{1st} has been run for $\frac{j}{10}$ of its lifetime, where $j = 1, 2, \cdots, 9$. The remaining energy of each sensor node in the network is then calculated. Based on the remaining energy, the non-source tasks are rescheduled using the EBTS algorithm to obtain a new schedule. Since the source tasks are usually used to collect measurements from the environment, they are assigned to specific sensor nodes and cannot be reassigned. A linear program formulation is then used to calculate the maximum possible lifetime that can be obtained using both the first schedule and the current

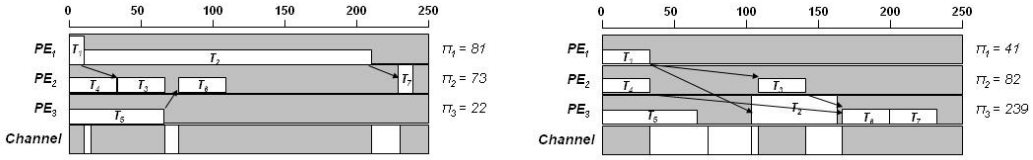
Algorithm 6 EBTS-DS

Run the EBTS algorithm to obtain an initial schedule S and the energy consumption per cycle π_i of all sensor nodes, $\forall i = 1, 2, \cdots, N_p$.
Calculate the expected lifetime L using Equation 2.16
Set the first schedule $S^{1st} \leftarrow S$, the energy consumption per cycle $\pi_i^{1st} \leftarrow \pi_i$ and the remaining energy capacity $R_i^{1st} \leftarrow R_i$, $\forall i = 1, 2, \cdots, N_p$.
Set $\delta \leftarrow \lfloor \frac{L}{10} \rfloor$.
for $j \leftarrow 1$ to 9 **do**
    Set $R_i \leftarrow (R_i^{1st} - j \times \delta \times \pi_i^{1st})$, $\forall i = 1, 2, \cdots, N_p$.
    Run the EBTS algorithm to schedule the non-source tasks using these values of R_i as the remaining energies of the sensor nodes. Update S and π_i using the new schedule.
    if S is feasible **then**
        Solve the following linear program:
        • Maximize $C_1 + C_2$
        • Subjected to $C_1 \times \pi_i^{1st} + C_2 \times \pi_i \leq R_i^{1st}$, $\forall i = 1, 2, \cdots, N_p$.
        if $(\lfloor C_1 \rfloor + \lfloor C_2 \rfloor) > L$ **then**
            Update $L \leftarrow (\lfloor C_1 \rfloor + \lfloor C_2 \rfloor)$.
            Update $\zeta^{1st} \leftarrow \lfloor C_1 \rfloor$ and $\zeta^{2nd} \leftarrow \lfloor C_2 \rfloor$, where ζ^{1st} and ζ^{2nd} are the number of cycles to run S^{1st} and S^{2nd} respectively.
            Update $S^{2nd} \leftarrow S$, $\pi_i^{2nd} \leftarrow \pi_i$ and $R_i^{2nd} \leftarrow R_i$, $\forall i = 1, 2, \cdots, N_p$.
        end if
    end if
end for
if S^{2nd} is found **then**
    Use S^{1st} for ζ^{1st} cycles followed by S^{2nd} for ζ^{2nd} cycles.
else
    Use S^{1st} for the entire lifetime of the sensor network.
end if

candidate schedule. Lastly, the candidate schedule that maximizes the network lifetime if it is used together with the first schedule is chosen.

6.2.2 Illustrative Example for EBTS-DS

Using the same example as in the previous section, we shall show how a second schedule can be used to improve the lifetime of the sensor network further. In the first step of EBTS-DS, we obtain the first schedule as illustrated in the previous section using our EBTS algorithm. As we have shown in Figure 6.2(a), this schedule



(a) Schedule 1 (b) Schedule 2

Figure 6.2: Schedules generated using EBTS-DS.

Step	Schedule	Task Assignment	Energy Consumption	No. of Cycles Consumed	Remaining Energy
0	-	-	-	-	PE_1: 1000 PE_2: 1000 PE_3: 1000
1	1	PE_1: T_1, T_2 PE_2: T_4, T_3, T_6, T_7 PE_3: T_5	PE_1: 81 PE_2: 73 PE_3: 22	11	PE_1: 109 PE_2: 197 PE_3: 758
2	2	PE_1: T_1 PE_2: T_4, T_3 PE_3: T_5, T_2, T_6, T_7	PE_1: 41 PE_2: 82 PE_3: 239	2	PE_1: 27 PE_2: 33 PE_3: 280

Table 6.4: Steps to illustrate the use of 2 consecutive schedules to improve the lifetime further

gives a lifetime of 12 cycles. Next, we generate a series of schedules and choose the schedule that maximizes the lifetime of the network when used together with the first schedule. This will be the second schedule. This schedule, as shown in Figure 6.2(b), gives a lifetime of only 4 cycles if used by itself. However, if we combine the two schedules in the way as shown in Table 6.4, we are able to obtain a lifetime of 13 cycles, which is an 8% improvement compared to using only the first schedule.

6.3 Performance of EBTS and EBTS-DS

6.3.1 WSN with Heterogeneous Sensor Nodes

First, we apply the algorithms to a WSN with heterogeneous nodes and compare their performance to the baseline case when EBTS is used without applying DVS. The maximum in-degree and out-degree of each task is set at 5. The number of source tasks in each task graph is equal to the number of sensor nodes used in the experiment. The average computation time and energy consumption of each task over all the sensor nodes at the maximum speed were randomly generated using a gamma distribution with a mean of 2msec and 4mJ respectively. The computation time and energy consumption of the task on each individual sensor node were then randomly generated using another gamma distribution with the mean equal to the average values generated earlier. It is assumed that all the tasks executed at their WCET for every periodic cycle of the application. It is also assumed that the minimum computational speed is $\frac{1}{N_v}$ of the maximum computational speed and all other levels of computational speed are uniformly distributed between the minimum and maximum speed.

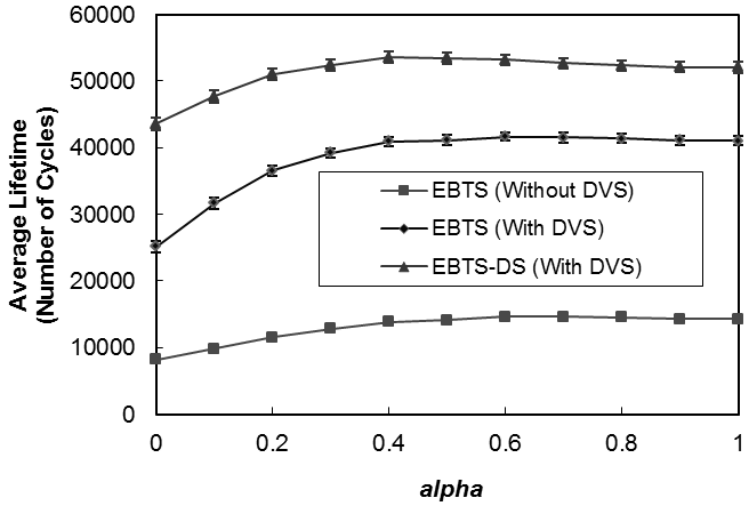
The time and energy cost of transmitting one bit of data are set to be 10 μsec and 1 μJ respectively. The number of bits of data to be transmitted between 2 tasks with precedence constraints was uniformly distributed between $200CCR(1 \pm 0.2)$,

where CCR represents the communication to computation ratio. The period of the application P was generated using the same method as described in [33] in the following way. For each task, its *distance* is defined as the number of edges in the longest path from any of the source tasks to that task. The tasks are then divided into layers according to their distance. Assuming that all the tasks in the same layer are executed in parallel, the estimated computation time required for each layer is $\overline{t_l}\lceil\frac{n_l}{m}\rceil$, where n_l is the number of tasks in that layer and $\overline{t_l}$ denotes the average computational time of the tasks in the layer. In addition, the expected number of communication activities initiated by any task is estimated to be $(d_{out} - 1)$ where d_{out} is the out-degree of the task. The total communication time required for each layer is therefore estimated as $\overline{t_l} \cdot CCR\lceil\frac{q}{K}\rceil$ where q is the sum of the expected number of communication activities initiated by the tasks in that layer. P is then calculated as the sum of computation and communication time of all the layers divided by u, where u is the utilization of the sensor network. Lastly, the remaining energy at each sensor node is uniformly generated between $(1 \pm 0.2) \times 10^6$mJ.

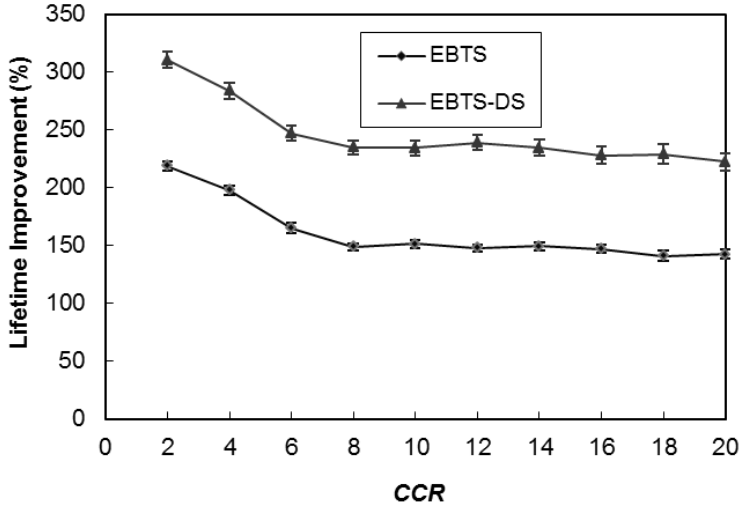
The simulation experiments are conducted for a wireless sensor network consisting of 10 sensor nodes, 8 voltage levels, 4 channels, 100 tasks (with 10 source tasks), CCR between 2 and 20, and u between 0 and 1. All the data in the experiment are obtained by averaging the values obtained using 1000 different task graphs.

We first set CCR to be 4, u to be 0.5 and varied the value of α from 0 to 1. The results are shown in Figure 6.3(a). It is observed that when $\alpha = 0$, both the EBTS and EBTS-DS algorithms tried to minimize the total energy consumption of the sensor network without considering the maximum norm-energy of each sensor node. Therefore the average lifetime of the sensor network obtained using the designed algorithms are shorter. On the other hand, when α is around 0.4 to 0.6, the average lifetimes reach the maximum values before decreasing slightly when α increases further. This is because when α is around 0.4 to 0.6, the designed algorithms take into consideration the maximum norm-energy while trying to minimize the total energy consumption at the same time. As a result, the sensor nodes have more remaining energy and the algorithms are therefore able to generate better schedules. At $\alpha = 0.5$, there is about 195% improvement in the lifetime when EBTS is used with DVS, compared to the baseline case of using EBTS without DVS. When the EBTS-DS algorithm is used to generate a second schedule, the lifetime improvement increases to 287%. The value $\alpha = 0.5$ shall be used in the subsequent experiments.

Next, CCR is varied between 2 to 20. The lifetime improvement of the designed algorithms for different values of CCR is shown in Figure 6.3(b). Here, the *lifetime improvement* is defined as $(\frac{L_{alg}}{L_{base}} - 1)$, where L_{alg} is the lifetime of the WSN when a particular algorithm is used and L_{base} is the lifetime of the WSN in the



(a) Average lifetime with varying α and $CCR = 4$



(b) Lifetime improvement with varying CCR and $\alpha = 0.5$

Figure 6.3: Performance of EBTS and EBTS-DS for WSN consisting of heterogeneous nodes (10 sensor nodes, 8 voltage levels, 4 channels, 100 tasks, $u = 0.5$). The values for the lifetime improvement are calculated as the improvement over the baseline case when EBTS is used without DVS. The vertical bars show the confidence intervals at 95% confidence level.

baseline case when the EBTS algorithm is used without DVS. When $CCR = 2$, the EBTS algorithm is able to obtain a lifetime improvement of about 220% while the EBTS-DS achieves a lifetime improvement of 301%. However, as CCR increases, this improvement decreases. At $CCR = 20$, the lifetime improvements that could be achieved using EBTS and EBTS-DS decrease to about 136% and 213% respectively. This is because as CCR increases, the communication energy becomes more significant when compared to the computational energy. Therefore, the lifetime improvement obtained by reducing the computational energy using DVS becomes more limited as a result.

Lastly, the utilization u of the sensor network is varied and the rate at which deadlines are missed is observed. Figure 6.4 shows the simulation results. It is observed that the designed algorithm provides a very low miss rate. Even when $u = 1$, only 26% of the application task graphs missed their deadlines.

6.3.2 WSN with Homogeneous Sensor Nodes

The designed algorithms are also compared with the 3-phase heuristic [33] for WSNs with homogeneous nodes. The task graphs and parameters are generated in the same way as in the previous experiment, except that the computational time and energy consumption for each task does not vary across different sensor nodes. The performance of the algorithms are compared to the baseline case when the

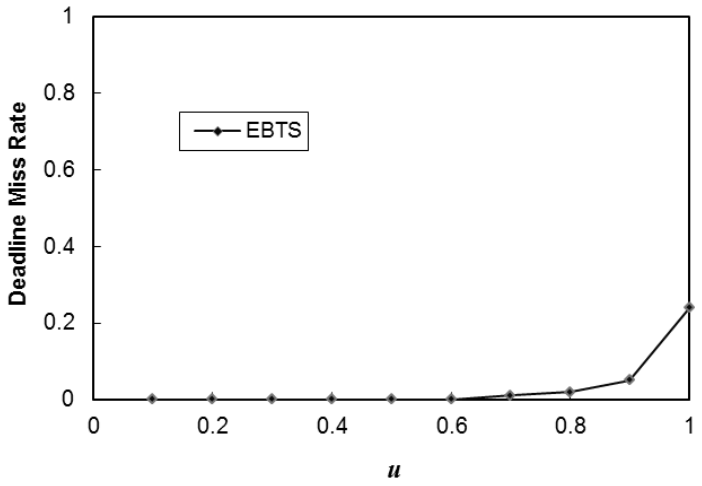
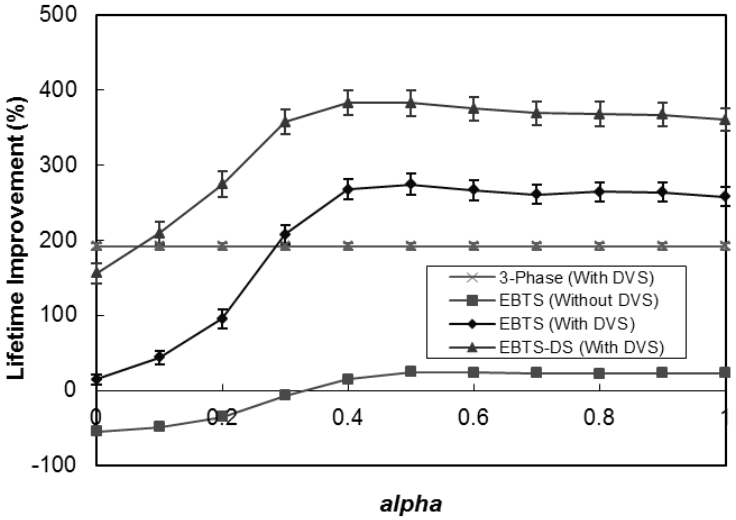


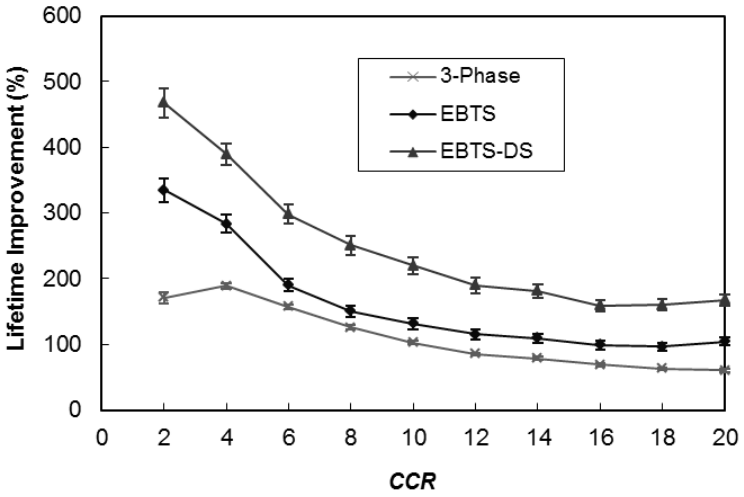
Figure 6.4: Miss rate of EBTS with varying values of u for WSN consisting of heterogeneous nodes (10 sensor nodes, 8 voltage levels, 4 channels, 100 tasks, $CCR = 0$).

3-phase heuristic is used without applying DVS.

Figure 6.5(a) shows the results when α is varied from 0 to 1. Similar results are observed as in the previous experiment. When $\alpha = 0$, the lifetime improvements of EBTS and EBTS-DS are only 15% and 154% respectively compared to 190% improvement obtained using the 3-phase heuristic. On the other hand, when $\alpha = 1$, lifetime improvement of EBTS and EBTS-DS are 257% and 362% respectively. The best performance is obtained when $\alpha = 0.5$. At this value of α, the lifetime improvement of EBTS and EBTS-DS are 269% and 381% respectively. Even when the EBTS algorithm is used without DVS, there is an improvement of 25% at $\alpha = 0.5$ compared to the baseline case of using the 3-phase heuristic without DVS.



(a) Lifetime improvement with varying α and $CCR = 4$



(b) Lifetime improvement with varying CCR and $\alpha = 0.5$

Figure 6.5: Lifetime improvement of 3-phase heuristic, EBTS and EBTS-DS for WSN consisting of homogeneous nodes (10 sensor nodes, 8 voltage levels, 4 channels, 100 tasks, $u = 0.5$). These values are calculated as the improvement over the baseline case when the 3-phase heuristic is used without DVS. The vertical bars show the confidence intervals at 95% confidence level.

Next, the performance of the algorithms with respect to varying values of CCR is studied. The results are shown in Figure 6.5(b). The lifetime improvement is calculated by comparing the lifetime generated by the algorithms to the baseline case where the 3-phase heuristic is used without DVS. When $CCR = 2$, the EBTS and EBTS-DS algorithms obtain lifetime improvements of 331% and 461% respectively while the 3-phase heuristic obtains an improvement of only 168%. However, as CCR increases, the improvement of EBTS and EBTS-DS over the 3-phase heuristic decreases. At $CCR = 20$, EBTS and EBTS-DS achieve lifetime improvements of about 104% and 167% respectively compared to 60% achieved by the 3-phase heuristic.

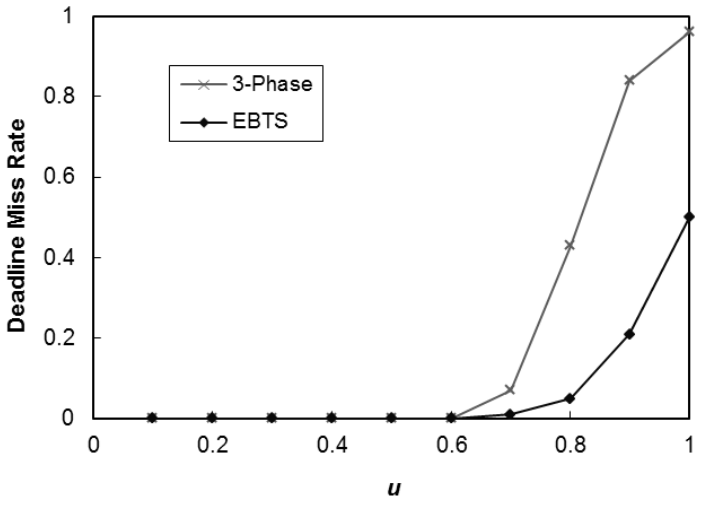


Figure 6.6: Miss rate of 3-phase heuristic and EBTS with varying values of u for WSN consisting of homogeneous nodes (10 sensor nodes, 8 voltage levels, 4 channels, 100 tasks, $CCR = 0$).

Lastly, the deadline miss rates of the EBTS algorithm and the 3-phase heuristic are studied for homogeneous WSNs. The results are shown in Figure 6.6. It is observed that the EBTS algorithm provides a lower deadline miss rate compared to the 3-phase heuristic, especially at higher values of u. At $u = 1$, the miss rate of the EBTS algorithm is 51% while the miss rate for the 3-phase heuristic is as high as 96%. From these experiments, it can be concluded that although the EBTS and EBTS-DS algorithms are designed for WSNs with heterogeneous sensor nodes, they can be used for homogeneous sensor nodes as well.

Chapter 7

Conclusion

As the demand for performance and functionality of portable embedded systems increases over the year, many applications are commonly implemented on platforms comprising of multiple homogeneous or heterogeneous processing cores. As many of these systems operate using a battery source, energy management is essential to extend the operational lifetime of the system. For hard real-time applications, there are also deadline requirements that must be strictly adhered to, otherwise it may result in a catastrophic failure of the system.

This thesis addresses the problem of scheduling a hard real-time application consisting of tasks with precedence constraints onto a multiprocessor system where the processing cores have DVS capabilities and can be either homogeneous or heterogeneous. The objective is to derive an energy-efficient schedule that maximizes the lifetime of the system while ensuring that all the deadline requirements of the

application are met. The thesis looks at this problem in various aspects and proposes a few energy-aware scheduling algorithms to address the problem.

Most portable systems operates on a single battery source. In order to maximize the lifetime of such systems, the total energy consumption should be minimized. In order to guarantee that the deadlines requirements are met, static energy-aware algorithms are usually used to generate static schedules in advance using the WCETs of the tasks. We proposed the EGMS algorithm to generate an energy-efficient static schedule that meets the deadline requirements. Unlike most scheduling algorithms in the literature that consider TM or TSVS separately, EGMS consider them in an integrated way to derive a low-energy schedule using the concept of energy gradient. EGMS does not consider intra-task voltage scaling and is useful for scheduling tasks in situations where intra-task voltage scaling cannot be used due to its high overhead. In EGMSIV, we extended our EGMS algorithm to introduce intra-task voltage scaling by using an LP formulation for the voltage scaling problem. We also describe modifications that can be made to EGMS/EGMSIV so that they can be used for task mapping (EGMS-TM) or task scheduling and voltage scaling (EGMS-TSVS/EGMSIV-TSVS) separately.

As the static energy-aware scheduling algorithms use the WCETs of the tasks to generate energy-efficient schedules, these schedules are optimized provided that the

tasks execute at their WCETs most of the time during runtime. This may not be applicable to applications where the tasks only require their WCETs to complete occasionally. In this case, the average energy consumption of the tasks may not be optimized. We proposed the PSDSC method which tries to estimate the potential amount of slack that a task may receive from its precedent tasks in the scheduling step. Based on the amount of potential slack, we can estimate the speed at which the task is expected to execute during runtime and how much energy it consumes. By using these estimated values, our PSDSC method tries to generate a schedule that optimizes the average energy consumption while still guaranteeing that the deadline requirements are not violated even in the worst-case scenario. In addition to PSDSC, we also proposed an aggressive dynamic scheduling algorithm to be used during runtime. In AADS, we aggressively lower the execution speeds of the tasks during runtime based on the average of their actual number of execution cycles in the previous execution instances of the application. In doing so, we reduce the amount of slack that will be generated and passed on to successor tasks.

While most multiprocessor systems operate using a single battery source, there are systems where each processing element has its own energy source. The WSN is one such example. To address this, we have proposed the EBTS algorithm for assigning tasks with precedence constraints to a single-hop WSN consisting of heterogeneous nodes. Our objective is to maximize the lifetime of the WSN by assigning the

tasks in a balanced way such that the lifetime of the sensor node which consumes the most energy is maximized. We also proposed the EBTS-DS algorithm, which generates a second schedule that is used to extend the lifetime of the WSN further when it is used together with the original schedule.

Finally we evaluated the performance of all our proposed algorithms with existing strategies in the literature through extensive simulation experiments. We observed improvements in maximizing the lifetime of the multiprocessor systems while ensuring that the deadlines of the application are strictly met.

From the research that is conducted in this thesis, we find that task mapping plays an important part in improving the lifetime of the multiprocessor system. In addition, we find that the conventional way of iteratively solving the TM and TSVS sub-problems separately usually requires much more iterations to guide the optimization to a reasonably good solution. By considering task mapping, task ordering and voltage scaling together in each scheduling step, the optimization process is usually faster as many redundant steps can be excluded.

While we formuate the problem with the assumption that the deadline of the tasks in the task graph is equal to its period, it is relatively straight forward to extend our proposed algorithms to cases where the deadlines of the tasks in the task graph

are different and may be earlier than the period of the task graph. This can be done by making some minor changes in the algorithms. In EGMS and EGMSIV, during the calculation of the lengths of the critical paths in the CPTO sub-routine, if a task has a deadline that is earlier than the period, we have to add the difference between the deadline and the period to the lengths of critical paths that contain this task. Similarly for EBTS and EBTS-DS, if a task has a deadline that is earlier than the period, the difference between the deadline and the period can be added to the calculation of the upper rank of that task. In this way, this task and its precedent tasks will have a higher priority of being scheduled first in order to meet the deadline constraint.

Bibliography

[1] URL http://www.ti.com/omap/.

[2] URL http://sourceforge.net/projects/lpsolve/.

[3] URL http://cecs02.cecs.uci.edu/DVS/.

[4] Dawei Li and Jie Wu, "Scheduling on Homogeneous DVFS Multiprocessor Platforms," *Energy-aware Scheduling on Multiprocessor Platforms,* Springer-Briefs in Computer Science, pp. 13-40, 2013.

[5] B.R. Rajakumar and A. George, "A new adaptive mutation technique for genetic algorithm," *Proceedings of IEEE International Conference on Computational Intelligence & Computing Research,* pp. 1-7, December 2012.

[6] Shyamal Patel, Hyung Park, Paolo Bonato, Leighton Chan, and Mary Rodgers, "A review of wearable sensors and systems with application in rehabilitation," *Journal of Neuroengineering and Rehabilitation,* 9:21, April 2012.

[7] H. Gjoreski, M. Lustrek, and M. Gams, "Accelerometer Placement for Posture Recognition and Fall Detection," *Proceedings of 7th International Conference on Intelligent Environments,* pp. 47-54, July 2011.

[8] Ziliang Zong, Adam Manzanares, Xiaojun Ruan, and Xiao Qin, "EAD and PEBD: Two Energy-Aware Duplication Scheduling Algorithms for Parallel Tasks on Homogeneous Clusters," *IEEE Transactions on Computers,* vol. 60, no. 3, pp. 360-374, March 2011.

[9] Jaeyeon Kang and Sanjay Ranka, "Dynamic Slack Allocation Algorithms for Energy Minimization on Parallel Machines," *Journal of Parallel and Distributed Computing,* vol. 70, no. 5, pp. 417430, 2010.

[10] Xin-She Yang, *Nature-Inspired Metaheuristic Algorithms (Second Edition,* Luniver Press, UK, 2010.

[11] Hande Alemdar and Cem Ersoy, "Wireless sensor networks for healthcare: A survey," *Computer Networks,* vol. 54, issue 15, pp. 2688-2710, October 2010.

[12] Jian-Jia Chen and Lothar Thiele, "Energy-Efficient Scheduling on Homogeneous Multiprocessor Platforms," *Proceedings of the 25th ACM Symposium on Applied Computing,* pp. 542-549, March 2010.

[13] El-Ghazali Talbi, *Metaheuristics: From Design to Implementation,* Wiley, 2009.

[14] Byron K. Lee and Jeffery E. Olgin, "Role of Wearable and Automatic External Defibillators in Improving Survival in Patients at Risk for Sudden Cardiac Death," *Current Treatment Options in Cardiovascular Medicine,* vol. 11, issue 5, pp. 360-365, October 2009.

[15] Leonora Bianchi, Marco Dorigo, Luca Maria Gambardella, and Walter J. Gutjahr, "A survey on metaheuristics for stochastic combinatorial optimization," *Natural Computing,* vol. 8, issue 2, pp. 239-287, June 2009.

[16] Lee Kee Goh, Bharadwaj Veeravalli, and Sivakumar Viswanathan, "Design of Fast and Efficient Energy-Aware Gradient-Based Scheduling Algorithms for Heterogeneous Embedded Multiprocessor Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 20, no. 1, pp. 1-12, January 2009.

[17] M.N. Nyan, Francis E.H. Tay, and E. Murugasu, "A wearable system for pre-impact fall detection," *Journal of Biomechanics,* vol. 41, issue 16, pp. 3475-3481, December 2008.

[18] Lee Kee Goh and Bharadwaj Veeravalli, "An Energy-Balanced Task Scheduling Heuristic for Heterogeneous Wireless Sensor Networks," *Proceedings of International Conference on High Performance Computing,* Lecture Notes in Computer Science 5374, pp. 257-268, 2008.

[19] Myo Naing Nyan, Francis Eng Hock Tay, Lee Kee Goh, Bharadwaj Veeravalli, Dagang Guo, Lin Xu, and K.L. Yap, "Low-Energy Scheduling Algorithms for Wearable Fall Pre-Impact Detection System," *Proceedings of International Symposium on Bio- and Medical Informatics and Cybernetics,* pp. 172-178, 2008.

[20] A. K. Bourke and G. M. Lyons, "A threshold-based fall-detection algorithm using a bi-axial gyroscope sensor," *Medical Engineering & Physics,* vol. 30, issue 1, pp. 84-90, January 2008.

[21] Yanhong Liu, Bharadwaj Veeravalli, and Sivakumar Viswanathan, "Novel Critical-Path based Low-Energy Scheduling Algorithms for Heterogeneous Multiprocessor Real-Time Embedded Systems," *Proceedings of International Conference on Parallel and Distributed Systems,* vol. 1, pp. 1-8, December 2007.

[22] Lee Kee Goh, Bharadwaj Veeravalli, and Sivakumar Viswanathan, "An Energy-Aware Gradient-Based Scheduling Heuristic for Heterogeneous Multiprocessor Embedded Systems," *Proceedings of International Conference on*

High Performance Computing, Lecture Notes in Computer Science 4873, pp. 331-341, 2007.

[23] Bita Gorjiara, Nader Bagherzadeh, and Pai Chou, "Ultra-Fast and Efficient Algorithm for Energy Optimization by Gradient-based Stochastic Voltage and Task Scheduling," *ACM Transactions on Design Automation of Electronic Systems,* vol. 12, no. 4, Article 39, September 2007.

[24] Changjiu Xian, Yung-Hsiang Lu, and Zhiyuan Li, "Energy-Aware Scheduling for Real-Time Multiprocessor Systems with Uncertain Task Execution Time," *Proceedings of the 44th Annual Design Automation Conference,* pp. 664-669, June 2007.

[25] Jun Zhang, H.S.-H. Chung, and Wai-Lun Lo, "Clustering-Based Adaptive Crossover and Mutation Probabilities for Genetic Algorithms," *IEEE Transactions on Evolutionary Computation,* vol. 11, no. 3, pp. 326-335, June 2007.

[26] Sanjeev Baskiyar and Kiran Kumar Palli, "Low Power Scheduling of DAGs to Minimize Finish Times," *Proceedings of International Conference on High Performance Computing,* Lecture Notes in Computer Science 4297, pp. 353-362, 2006.

[27] Ioannis Milis, "Approximating a Class of Classification Problems," *Efficient Approximation and Online Algorithms,* Lecture Notes in Computer Science,

vol. 3484, pp. 213-249, 2006.

[28] Jian-Jia Chen and Tei-Wei Kuo, "Energy-Efficient Scheduling of Periodic Real-Time Tasks over Homogeneous Multiprocessors," *Proceedings of the 2nd International Workshop on Power-Aware Real-Time Computing,* pp. 30-35, September 2005.

[29] R. Rao and S. Vrudlhula, "Energy optimal speed control of devices with discrete speed sets," *Proceedings of the 42nd Design Automation Conference,* pp. 901-904, June 2005.

[30] Yan Zhang, Zhijian Lu, J. Lach, K. Skadron, and M.R. Stan, "Optimal procrastinating voltage scheduling for hard real-time systems," *Proceedings of the 42nd Design Automation Conference,* pp. 905-908, June 2005.

[31] Jian-Jun Han and Qing-Hua Li, "Dynamic Power-Aware Scheduling Algorithms for Real-Time Task Sets with Fault-Tolerance in Parallel and Distributed Computing Environment," *Proceedings of 19th International Parallel and Distributed Processing Symposium,* April 2005.

[32] Tarek A. AlEnawy and Hakan Aydin, "Energy-Aware Task Allocation for Rate Monotonic Scheduling," *Proceedings of 11th IEEE Real-Time and Embedded Technology and Applications Symposium,* pp. 213-223, March 2005.

[33] Yang Yu and Viktor K. Prasanna, "Energy-Balanced Task Allocation for Collaborative Processing in Wireless Sensor Networks," *Mobile Networks and Applications,* vol. 10, pp. 115-131, 2005.

[34] Alexandru Andrei, Marcus T. Schmitz, Petru Eles, Zebo Peng, and Bashir M. Al-Hashimi, "Overhead-Conscious Voltage Selection for Dynamic and Leakage Energy Reduction of Time-Constrained Systems," *IEE Proceedings - Computers and Digital Techniques,* vol. 152, issue 1, pp. 28-38, January 2005.

[35] James H. Anderson and Sanjoy K. Baruah, "Energy-Efficient Synthesis of Periodic Task Systems upon Identical Multiprocessor Platforms," *Proceedings of the 24th International Conference on Distributed Computing Systems,* pp. 428-435, 2004.

[36] Bita Gorjiara, Nader Bagherzadeh, and Pai Chou, "An Efficient Voltage Scaling Algorithm for Complex SoCs with Few Number of Voltage Modes," *Proceedings of 2004 International Symposium on Low Power Electronics and Design,* pp. 381-386, 2004.

[37] L.S.Y .Wong, S. Hossain, A. Ta, J. Edvinsson, D.H. Rivas, and H. Naas, "A very low-power CMOS mixed-signal IC for implantable pacemaker applications," *IEEE Journal of Solid-State Circuits,* vol. 39, issue 12, pp. 2446-2456, December 2004.

[38] Vishnu Swaminathan and Krishnendu Chakrabarty, "Network Flow Techniques for Dynamic Voltage Scaling in Hard Real-Time Systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 23, no. 10, pp. 1385-1398, October 2004.

[39] Bita Gorjiara, Pai Chou, Nader Bagherzadeh, Mehrdad Reshadi, and David Jensen, "Fast and Efficient Voltage Scheduling by Evolutionary Slack Distribution," *Proceedings of Asia and South Pacific Design Automation Conference,* pp. 659-662, January 2004.

[40] Lap-Fai Leung, Chi-Ying Tsui, and Wing-Hung Ki, "Minimizing Energy Consumption of Multiple-Processors-Core Systems with Simultaneous Task Allocation, Scheduling and Voltage Assignment," *Proceedings of Asia and South Pacific Design Automation Conference,* pp. 647-652, January 2004.

[41] Wanghong Yuan and Klara Nahrstedt, "Energy-Efficient Soft Real-Time CPU Scheduling for Mobile Multimedia Systems," *Proceedings of the 19th ACM Symposium on Operating Systems Principles,* pp. 149-163 , December 2003.

[42] Juan de Vicente, Juan Lanchares, and Román Hermida, "Placement by thermodynamic simulated annealing," *Physics Letters A,* vol. 317, issue 5-6, pp. 415-423, October 2003.

[43] Christian Blum and Andrea Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Computing Surveys,* vol. 35, issue 3, pp. 268-308, September 2003.

[44] Dakai Zhu, Rami G. Melhem, and Bruce R. Childers, "Scheduling with Dynamic Voltage/Speed Adjustment Using Slack Reclamation in Multiprocessor Real-Time Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 14, no. 7, pp. 686-700, July 2003.

[45] Hakan Aydin and Qi Yang, "Energy-Aware Partitioning for Multiprocessor Real-Time Systems," *Proceedings of 17th International Parallel and Distributed Processing Symposium,* April 2003.

[46] Ramesh Mishra, Namrata Rastogi, Dakai Zhu, Daniel Mossé, and Rami G. Melhem, "Energy Aware Scheduling for Distributed Real-Time Systems," *Proceedings of 17th International Parallel and Distributed Processing Symposium,* April 2003.

[47] Yang Yu and Viktor K. Prasanna, "Power-Aware Resource Allocation for Independent Tasks in Heterogeneous Real-Time Systems," *Proceedings of 9th International Conference on Parallel and Distributed Systems,* pp. 341-348, December 2002.

[48] Steven M. Martin, Krisztian Flautner, Trevor Mudge, and David Blaauw, "Combined Dynamic Voltage Scaling and Adaptive Body Biasing for Lower Power Microprocessors under Dynamic Workloads," *Proceedings of International Conference on Computer Aided Design,* pp. 721-725, 2002.

[49] Dakai Zhu, Nevine AbouGhazaleh, Daniel Mossé, and Rami G. Melhem, "Power Aware Scheduling for AND/OR Graphs in Multi-Processor Real-Time Systems," *Proceedings of 31st International Conference on Parallel Processing,* pp. 593-601, August 2002.

[50] Yumin Zhang, Xiaobo Hu, and Danny Z. Chen, "Task Scheduling and Voltage Selection for Energy Minimization," *Proceedings of 39th Design Automation Conference,* pp. 183-188, June 2002.

[51] Roseanne Schott, "Wearable Defibrillator," *Journal of Cardiovascular Nursing,* vol. 16, issue 3, pp. 44-52, April 2002.

[52] Marcus T. Schmitz, Bashir M. Al-Hashimi, and Petru Eles, "Energy-Efficient Mapping and Scheduling for DVS Enabled Distributed Embedded Systems," *Proceedings of 2002 Design, Automation and Test in Europe Conference and Exposition,* pp. 514-521, March 2002.

[53] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms (Second Edition),* MIT Press and McGraw-Hill, 2001.

[54] Marcus T. Schmitz and Bashir M. Al-Hashimi, "Considering Power Variations of DVS Processing Elements for Energy Minimisation in Distributed Systems," *Proceedings of International Symposium on Systems Synthesis,* pp. 250-255, October 2001.

[55] Hakan Aydin, Rami G. Melhem, Daniel Mossé, Pedro Mejía-Alvarez, "Determining Optimal Processor Speeds for Periodic Real-Time Tasks with Different Power Characteristics," *Proceedings of the 13th EuroMicro Conference on Real-Time Systems,* pp. 225-232, June 2001.

[56] Padmanabhan Pillai and Kang G. Shin, "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems," *Proceedings of the 18th ACM Symposium on Operating Systems Principles,* pp. 89-102, 2001.

[57] P. Yang, C. Wong, P. Marchal, F. Catthoor, D. Desmet, D. Verkest, and R. Lauwereins, "Energy-aware Runtime Scheduling for Embedded-Multiprocessor SOCs," *IEEE Design & Test of Computers,* vol. 18, no. 5, pp. 4658, 2001.

[58] Curt Schurgers, Olivier Aberthorne, and Mani B. Srivastava, "Modulation Scaling for Energy-aware Communication Systems," *Proceedings of International Symposium on Low Power Electronics and Design,* pp. 96-99, 2001.

[59] Neal K. Bhamba, Shuvra S. Bhattacharyya, Jürgen Teich, and Eckart Zitzler, "Hybrid Global/Local Search Strategies for Dynamic Voltage Scaling in Embedded Multiprocassors," *Proceedings of 9th International Symposium on Hardware/Software Co-Design,* pp. 243-248, April 2001.

[60] Flavius Gruian and Krzysztof Kuchcinski, "LEneS: Task Scheduling for Low-Energy Systems Using Variable Supply Voltage Processors," *Proceedings of Asia and South Pacific Design Automation Conference,* pp. 449-455, January 2001.

[61] Jiong Luo and Niraj K. Jha, "Power-conscious Joint Scheduling of Periodic Task Graphs and Aperiodic Tasks in Distributed Real-time Embedded Systems," *Proceedings of International Conference on Computer-Aided Design,* pp. 357-364, November 2000.

[62] Shoukat Ali, Howard Jay Siegel, Muthucumaru Maheswaran, Debra A. Hensgen, and Sahra Ali, "Task Execution Time Modeling for Heterogeneous Computing Systems," *Proceedings of 9th Heterogeneous Computing Workshop,* pp. 185-199, May 2000.

[63] Tohru Ishihara and Hiroto Yasuura, "Voltage Scheduling Problem for Dynamically Variable Voltage Processors," *Proceedings of 1998 International Symposium on Low Power Electronics and Design,* pp. 197-202, August 1998.

[64] Robert P. Dick, David L. Rhodes, and Wayne Wolf, "TGFF: Task Graphs for Free," *Proceedings of 6th International Workshop on Hardware/Software Codesign,* pp. 97-101, March 1998.

[65] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," *Proceedings of IEEE International Conference on Neural Networks,* vol. 4, pp. 1942-1948, November 1995.

[66] Sanjeev Arora, David Karger, and Marek Karpinski, "Polynomial time approximation schemes for dense instances of NP-hard problems," *Proceedings of the 27th Annual ACM Symposium on Theory of Computing,* pp. 284-293, 1995.

[67] A.E. Eiben, P-E. Raué, and Zs. Ruttkay, "Genetic algorithms with multi-parent recombination," *Proceedings of the International Conference on Evolutionary Computation, The Third Conference on Parallel Problem Solving from Nature,* pp. 78-87, October 1994.

[68] M. Srinivas, and L.M. Patnaik, "Adaptive probabilities of crossover and mutation in genetic algorithms," *IEEE Transactions on Systems, Man and Cybernetics,* vol. 24, no. 4, pp. 656-667, April 1994.

[69] Anantha P. Chandrakasan, Samuel Sheng, and Robert W. Brodersen, "Low-Power CMOS Digital Design," *IEEE Journal of Solid-State Circuits,* vol. 27,

no. 4, pp. 473-484, April 1992.

[70] John H. Holland, *Adaptation in Natural and Artificial Systems,* MIT Press, Cambridge, MA, USA, 1992.

[71] Fred Glover, "Tabu Search - Part II," *ORSA Journal on Computing,* vol. 2, no. 1, pp. 4-32, 1990.

[72] Fred Glover, "Tabu Search - Part I," *ORSA Journal on Computing,* vol. 1, no. 3, pp. 190-206, 1989.

[73] David E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning,* Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1989.

[74] Fred Glover, "Future Paths for Integer Programming and Links to Artificial Intelligence," *Computers and Operations Research,* vol. 13, issue 5, pp. 533-549, May 1986.

[75] Judea Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving,* Addison-Wesley, 1984.

[76] S. Kirkpatrick, C.D. Gelatt Jr., M.P. Vecchi, "Optimization by Simulated Annealing," *Science,* vol. 220, no. 4598, pp. 671-680, May 1983.

[77] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the theory of NP-Completeness,* W. H. Freeman and Company, San Francisco, CA, 1979.