Stochastic Application Models for Energy Management in Multicore Systems

by

Soner Yaldız

A Thesis Submitted to the
Graduate School of Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Electrical & Computer Engineering

Koç University

September, 2006

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Soner Yaldız

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

Prof. Reha Civanlar

Assist. Prof. Alper Demir (Co-Advisor)

Assist. Prof. Lerzan Örmeci

Assist. Prof. Serdar Tasiran (Co-Advisor)

Prof. A. Murat Tekalp

Date: _____

To my mother, cumi and my father

ABSTRACT

In this thesis, we address the energy management problem that has become an important concern for computing systems. We concentrate on dynamic voltage scaling enabled multi-core systems running soft real-time applications that have been decomposed into concurrent tasks.

We first illustrate on simple examples that how energy management ignorant of the statistical nature of task workloads can be significantly suboptimal or not properly fulfill timing constraints. Based on this observation, we propose stochastic models which capture the variability of, and correlations among the workloads of tasks. The stochastic models are constructed using the workload data collected from actual application runs. Each stochastic application model suggests an energy management policy which minimizes the average energy consumption of the system while satisfying the timing constraints according to the application's tolerance to deadline misses. These policies, which exploit the variability and the correlation information captured by the stochastic application models, are obtained by novel optimization formulations. To solve the formulated optimization problems, we present two solution techniques. Our energy management policies introduce negligible overhead at runtime, since the associated optimization problems are solved offline.

We show on a popular multimedia application, MPEG2 video decoding, that there can be significant variability in and correlations among the workloads of tasks. Using the same application, we demonstrate how stochastic application models can be constructed and then present experimental results. We compare our energy management policies with two other approaches proposed in the literature.

ÖZETÇE

Bu tezde bilgisayar sistemlerinde son zamanlarda önemli bir sorun olarak ortaya çıkan enerji yönetimi ele alınmıştır. Eş zamanlı işlere bölünmüş, yarı gerçek zamanlı uygulamalara ve dinamik voltaj ayarlı, çok çekirdekli sistemlere odaklanılmıştır. Önerilen teknikler yazılımın ve donanımın önceden belirlenmiş olduğunu kabul etmektedir.

İlk olarak, basit örnekler aracılığıyla iş yüklerinin istatistiksel özelliklerinin enerji yönetimi ile olan ilişkileri incelenmiştir. Bu örneklerden elde edilen çıkarımlar doğrultusunda iş yüklerinin istatistiksel özelliklerini barındıran çeşitli rastsal modeller geliştirilmiştir. Bu rastsal modeller hem iş yüklerinin değişkenlik bilgisini hem de iş yükleri arasındaki korelasyon bilgisini barındırabilmektedir. Bu modeller gerçek uygulama yürütümlerinden alınan iş yükü verileri kullanılarak oluşturulmuştur. Her bir model, iş yüklerinin zamanlama kısıtlarına uyulmasına olasılıksal garanti sağlayarak sistemin enerji tüketimini enküçükleyen bir enerji yönetim stratejisi önermektedir. Bu stratejiler matematiksel olarak formüle edilmiş eniyileme modellerinden elde edilmekte; değişkenlik ve korelasyon bilgilerini barındıran rastsal modellerden yararlanmaktadır. Formüle edilen eniyileme problemlerini çözmek için iki farklı yöntem sunulmuştur. Eniyileme problemleri gerçek zamanda çözülmediği için önerilen stratejiler uygulamaya hesaplama yükü getirmemektedir.

Tezde sunulan modelleme ve eniyileme teknikleri, popüler bir çokluortam uygulaması olan MPEG2 video kod çözücüsüne tatbik edilmiştir. Yapılan deneyler, bu uygulamanın iş yükleri arasında kayda değer değişkenlikler ve korelasyonlar bulunduğunu ortaya koymaktadır. Rastsal modellerden elde edilen enerji yönetim stratejileri literatürde bulunan iki yaklaşımla karşılaştırılmıştır.

ACKNOWLEDGMENTS

I would like to express my gratitude to my supervisors Assist. Prof. Alper Demir and Assist. Prof. Serdar Taşıran for their invaluable mentorship, help and patience. I hope that I inherited some of their wisdom and discipline.

I would like to thank Prof. Yusuf Leblebici and Prof. Paolo Ienne at the Swiss Federal Institute of Technology at Lausanne (EPFL) for their collaboration in this project.

I am thankful to Assist. Prof. Lerzan Örmeci for critical reading of this thesis and for her valuable comments. I am also thankful to the members of my thesis committee.

I would like to acknowledge the financial support provided by the Scientific and Technical Research Council of Turkey.

I am grateful to all my friends from Koç University, for their close friendship: Ali, Deniz, Emrah, Erkan, Nesra and Tayfun. I would like to thank especially Pınar for her love and moral support.

I would like to express my deepest gratitude to my mother, brother (cumi) and my father for their everlasting love, support and sacrifices.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

AN	Activity network
CT	Cycle-time
CSO	Current-state observable
DAN	Deterministic activity network
DEM	Deterministic energy minimization
DVS	Dynamic Voltage Scaling
GP	Geometric programming
HPC	Hardware performance counter
KB	Kilo bytes
IC	Integrated circuit
IDCT	Inverse discrete cosine transform
IQ	Inverse quantization
MB	Mega bytes
MC	Motion compensation
PSO	Previous-state observable
RT	Real-time
SAN	Stochastic activity network
SEM	Stochastic energy minimization
SP	Stochastic programming
VLD	Variable length decoding

Chapter 1

INTRODUCTION

1.1 Multicore Systems

Advances in integrated circuit (IC) technologies have enriched the variety of intelligent computing systems and enabled such systems to incorporate more functionality. In addition to computers, other devices including cellular phones, personal digital assistants and portable media players have become integral parts of modern life. Two primary factors contributing to this situation are the continuous decrease in the device dimensions and the increase in the operating speed of digital circuits.

In IC technology, the performance of a computing system has been closely related to its operating speed (i.e., the clock frequency) which has scaled up to gigahertz levels. However, frequency scaling is no longer a viable option for performance improvement, since it is limited by the physical properties of integrated circuits and increases the power consumption. In addition, the overall performance of a system no longer follows the clock frequency due to the memory wall problem which refers to the increasing gap between the memory access speed and the processor operating speed. Due to these problems, multiprocessing has become a promising trend in the performance improvement of computing systems that also keeps power consumption under control.

The most recent form of multiprocessing is implemented at the chip-level where multiple cores are integrated on a single chip. In such systems, the application workload is distributed onto multiple cores designed to operate at lower frequencies instead of a high-frequency, power-hungry single core. Multicore processors are already available in the market for various computing systems from high performance computers to embedded systems. Although multiprocessing keeps power consumption under control, there is still a strong need for intelligent energy management for both portable and wall-powered multicore systems.

1.2 Power and Energy Issues

Power consumption in a digital circuit has two main components: static and dynamic. Static power consumption is primarily caused by leakage currents and is inversely proportional to decreasing device dimensions; while dynamic consumption is due to the switching activity in the circuit and is a linear function of the clock frequency. Since IC technology tends to reduce device dimensions and increase clock frequency for faster operation, power consumption has become more of an important concern for digital systems.

For wall-powered systems, power consumption is one of the primary issues since it increases the packaging and cooling cost of integrated systems with high device density. For battery-operated portable systems, energy consumption is the primary issue due to limited battery life and the fact that battery technologies usually lag far behind IC technologies. For both kinds of systems, high operating temperature due to high power and energy consumption increases the risk of failures in circuits.

In order to alleviate these problems, power and energy minimization have been addressed at various stages and levels of the design hierarchy [1, 2]. Techniques applied during system design focus mostly on the underlying hardware. For example, device-level approaches focus on reducing the power consumption due to leakage currents by using new materials for fabrication or novel device structures, while circuit and logic-level approaches focus on reducing the power consumption due to switching activity by circuit resizing and restructuring. Techniques applied at runtime, called dynamic techniques, are implemented in software where the underlying hardware components (e.g., processor, display, storage device) should support multiple operating modes with varying levels of power consumption. At runtime, the user can either completely turn off an unused component or change its operating mode. For example, a wireless network interface or a storage device can be turned off while not in use. Similarly, a processor can be switched to a low power mode when the application workload is low.

A widely-used dynamic technique for reducing the power and energy consumption of processors is dynamic voltage scaling (DVS). DVS, supported by most of the recent processors, is based on lowering the operating speed of a core by lowering its voltage and clock frequency. For example, if it is known that an application will finish earlier than its deadline at maximum voltage, then DVS can be used to slow down its execution without violating

the deadline as shown in Figure 1.1. Executing an application at a lower voltage results in a quadratic decrease in both energy and power consumption. Execution at a lower frequency results in a linear decrease in power consumption. Although the principle behind DVS is straightforward, there are several complications with its usage for real-time applications.

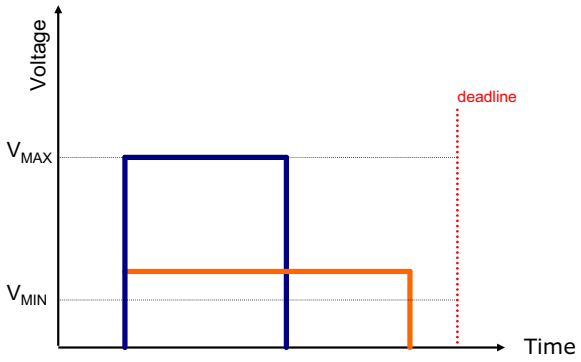


Figure 1.1: Using DVS for a task at runtime

1.3 Real-Time Applications

An application is called real-time (RT), if its execution is subject to a set of timing constraints. A large set of such applications from various domains (e.g., multimedia, entertainment and games) have a repetitive nature in which the application processes a varying amount of input data at regular time intervals. Each time interval is called a period of the application. For example, in a video decoding application, some amount of input data has to be processed in each period to obtain pictures used in the continuous play.

Depending on the criticality of timing constraints, an RT application can be classified as hard or soft. In the hard RT case, a timing constraint violation causes system failure so that tasks must be finished before the deadline in every period. For instance, the data coming from the parking sensors of a car has to be processed in a timely manner in order to prevent the car from crashing. In the soft RT case, the system has some tolerance to deadline misses so that the deadline can be violated in some of the periods according to a given probabilistic performance constraint. This tolerance provides a flexibility to be exploited for various purposes including energy management. For example, in a portable video player, some of the pictures in the sequence can be skipped, which is not critical for

the user, if significant energy savings can be achieved in return. In this thesis, we focus on reducing the energy consumption of a DVS-enabled multicore processor running a soft RT application.

1.4 Statistical Nature of Application Workload

The primary difficulty of using DVS for many periodic RT applications stems from the fact that the workload required by the input data varies over time and is unknown before the input data is processed. Consequently, one does not know exactly how much to slow down the core for a task in each period. In this thesis, we refer to this phenomenon as *workload variability*. Workload variability is primarily caused by the input data dependence of an application. If the application is hard RT, then the voltage settings should be adjusted according to the worst-case workload which limits the energy savings achievable via DVS. On the other hand, in a soft RT application, voltage settings can be lowered in some of the periods since the application has some tolerance to deadline misses.

When the application workload is decomposed into concurrent, interdependent tasks to be executed on a multicore platform, there may be correlations among the workloads of tasks. In this thesis, we refer to this phenomenon as *spatial correlation* and show that it arises in real-life applications. To our knowledge, spatial correlations have not been taken into account explicitly for energy management in the literature.

In addition to spatial correlations, the workload in a period may be correlated with the workload in other periods. For instance, the workload in one period may provide information about the workload in the next period. In this thesis, we refer to this phenomenon as *temporal correlation*. To capture temporal correlations, prediction-based approaches, in which the workload of a period is estimated using a workload history of earlier periods, have been widely used in the literature [3, 4, 5].

Workload variability, spatial and temporal correlations primarily arise from innate characteristics of an application such as the dependence of its workload on input data or its implementation in software. For example, in a digital video player, the application workload exhibits high variation depending on the compression level and the amount of motion in a scene. In a computer game, the workload in a period depends on the user input (e.g., joystick, mouse or other users in a multiplayer game) and the complexity of the game scene

(e.g., the number of objects in the scene to be processed for 3-dimensional view) [6]. In this thesis, we show that energy management ignorant of the statistical nature of task workloads can be significantly suboptimal or can not properly satisfy timing constraints. Therefore, this statistical nature needs to be taken into account for intelligent energy management.

1.5 Contributions

In this thesis, we study the energy management problem of DVS-enabled multicore systems running soft real-time applications that have been decomposed into concurrent tasks. This thesis has the following key contributions:

1. We demonstrate on a popular multimedia application, MPEG2 video decoding, that there can be significant variability in and correlations among the workloads of tasks that constitute the application. Motivated by this observation, we propose three *stochastic application models* that can effectively capture the statistical nature of task workloads, namely the workload variability, spatial and temporal correlations. We show that these models predict well the workloads of tasks in a soft real-time application.

2. We present novel mathematical formulations based on the stochastic application models which result in optimal energy management policies. These policies minimize the average energy consumption of the system and satisfy the timing constraints according to the application's tolerance to deadline misses. Our energy management policies introduce negligible overhead at runtime and are able to achieve energy savings close to the theoretical bound.

3. We show that energy minimization problems can also be formulated as optimal activity network problems that have been used in the literature. We present two solution techniques for optimal energy management policy problems one of which employs a heuristic approach previously used for activity networks.

4. We demonstrate how the memory behavior of a soft real-time application can limit the efficacy of an energy management scheme. Motivated by this observation, we adopt a

decomposed workload model in which the workload is broken down into on-chip and off-chip components. This decomposed workload model can be easily incorporated into our framework.

1.6 Organization

The organization of this thesis is as follows: In Chapter 2, we present a brief classification of the past work and a survey of closely related works. We provide a background including our assumptions and definitions in Chapter 3. In Chapter 4, we describe the stochastic application models that can capture the statistical nature of the workload. We present the optimization formulations for the energy management policies in Chapter 5. The solution techniques used for these optimization problems are discussed in Chapter 6. Chapter 7 describes how the workload data used in the construction of stochastic models is collected for a given application. In Chapter 8, we present the workload statistics of MPEG2 video decoding and discuss various stochastic models for this application. The results of experiments conducted to evaluate the effectiveness of the proposed energy management policies are reported in Chapter 9. Chapter 10 concludes the thesis including future directions.

Chapter 2

PAST WORK

2.1 Classification of the Past Work

Energy management of computing systems is a broad topic and has been addressed at various phases of system design as surveyed in [1, 2]. In this thesis, we focus on the energy minimization of multicore systems using a dynamic technique, dynamic voltage scaling (DVS), that is applied after the system has been designed. Even the scope is narrowed to dynamic techniques, the previous works in the literature can be classified in the following dimensions:

1. System architecture : Single or multicore
2. Timing constraints : Hard or soft
3. DVS implementation : Offline, online, or hybrid
4. DVS granularity : Interval, inter-task, intra-task

In this classification, the first dimension is related to the number of processing cores in the hardware system, while the second dimension reflects the criticality of the timing constraints. Hard timing constraints should be satisfied at all times, while soft timing constraints can be violated depending on the application's tolerance to deadline misses.

The third dimension indicates how voltage scaling decisions are made in an energy management scheme. An offline scheme consists of a training phase in which the workload statistics obtained from a set of representative inputs are used to determine voltage scaling decisions before runtime. The output of the training phase is stored in a look-up table and used at runtime to determine the voltage settings of tasks. The computational complexity of an offline method is negligible since all of its computations are done before runtime. On the other hand, online methods make voltage scaling decisions at runtime without any prior computation. These methods estimate the workload according to the workload experienced in the earlier periods. However, such prediction-based schemes are known to be vulnerable to sudden fluctuations in the application workload. For instance, it has been shown that

sudden workload variation reduces the accuracy of runtime workload prediction in MPEG video decoding [4, 5, 7], even if the prediction is guided by application-specific features. An online method should not introduce a significant computational overhead at runtime. A hybrid scheme is an offline method which is extended with runtime adjustments to the stored voltage scaling decisions.

The fourth dimension reflects the granularity of changing the voltage settings. In an interval-based scheme, voltage settings are altered at regular time intervals that are usually longer than the period of a repetitive application. In an inter-task scheme, the voltage setting of a task is not altered until it finishes. In an intra-task scheme, the voltage setting of a task can be readjusted during its execution. An interval-based scheme may miss short-term voltage scaling opportunities, while an intra-task one may introduce too much overhead (i.e., time and energy overhead due to frequent voltage switchings) depending on the amount of task workloads. Therefore, inter-task voltage scaling is more appropriate for parallel RT applications running on multicore systems.

Based on this classification, our energy management techniques can be summarized as offline, inter-task schemes for soft RT applications running on multicore systems. We also extend one of our energy management techniques with a runtime heuristic.

2.2 Related Work

Recently, there has been a growing interest in devising energy management schemes that take the statistical nature of the application workload into account. In this section, we provide a chronological survey of previous works which are closely related to this thesis. Some of these works address the task scheduling and the voltage scaling problems simultaneously, while some of them focus only on the voltage scaling problem assuming that tasks are already scheduled.

Paleologo et al. [8] present an energy management framework which considers the energy consumption of various components in a system each of which is modeled with a separate Markov Chain. Then, they formulate energy minimization as an optimization problem using these models. The formulated problem is equivalent to a policy optimization problem on a discrete-time Markov decision process (MDP). The discrete-time MDP in [8] is generalized to a continuous time MDP in [9], to a continuous-time semi-MDP in [10], and finally to

a time-indexed semi-MDP in [11]. In all of these studies, only the arrival rate of tasks (i.e, the number of tasks arrived in a time interval) is considered. In [12], Simunic et. al extend the framework in [11] in which the stochastic model combines both the arrival rate and the statistical characteristics of the workload. At runtime, voltage scaling decisions are made according to the changes in the arrival rate of and the changes in the execution times of tasks detected by predefined threshold values. Although, the framework in [8] and its extensions are similar to the models proposed in this thesis, these works do not consider the breakdown of the application workload into concurrent tasks for multicore systems.

Gruian [13] and Zhang et al. [14] address both the task scheduling and the voltage scaling problems with hard timing constraints. Gruian considers a set of independent tasks running on a single-core system and computes heuristically a stretching factor for each task according to its workload distribution and timing constraints before runtime [13]. At runtime, these stretching factors are used to determine the voltage settings of tasks, and the timing slacks are distributed into the tasks according to their priorities. On the other hand, Zhang et al. consider a set of tasks executed on a multicore system where there are data dependencies between tasks [14]. They formulate and solve an optimization problem in which the spatial correlations among task workloads are ignored and the worst-case workload is assumed for each task.

Hua et al. [15] study the same problem addressed in this thesis. They propose a hybrid approach for soft real-time applications running on multicore systems. The offline phase of their scheme consists of two stages both of which are based on greedy heuristics. The first stage computes a minimum workload for each task which satisfies the application's tolerance to deadline misses when tasks are executed at maximum voltage. In the second phase, the available execution time is allocated to these minimum workloads by adjusting the voltage settings of tasks. Using these allocated execution times, a latest completion time is computed for each task. At runtime, the voltage setting of each task is adjusted to finish its minimum workload until the precomputed latest finishing time.

In addition, the following recent studies focus on the statistical nature of the workload to be exploited for hard real-time applications: Leung and Hu [16] allocate the available execution time to tasks by formulating and solving an optimization problem which still guarantees the worst-case scenario. Andrei et al. [17] compute heuristically a set of voltage

settings for each task using the best, average and the worst-case workloads. At runtime, their algorithm selects one of the precomputed voltage settings depending on the start time of a task. Scordino and Bini [18] study the intra-task voltage scaling for a single task. Given a workload distribution and a deadline, their formulation results in a low voltage setting, a high voltage setting and a switching instant that minimizes the expected energy consumption. Finally, Xu et al. present a hybrid approach for the voltage scaling of a set of tasks [19]. In the offline phase, they allocate available execution time to tasks in a way to minimize the expected energy consumption. At runtime, they adjust the speed of the next task according to this allocated time and the remaining time.

This thesis is distinguished from the previous works primarily by capturing all aspects of the statistical workload, namely the workload variability, spatial correlations and temporal correlations, using stochastic models. This thesis also differs by targeting multicore systems and soft real-time applications. The problems of task scheduling and assignment are planned to be incorporated into the models and formulations provided in this thesis as future work.

Chapter 3

BACKGROUND

3.1 Software and Hardware Assumptions

A parallel application is composed of a set of computational tasks in which a task does not interrupt the execution of another task. There can be precedence relations among tasks due to data dependencies so that a task may need to wait until several other tasks have finished. Tasks without a data dependency in between can be executed concurrently on separate cores. Such a parallel application can be represented by a task graph [14, 20].

A task graph is a directed acyclic graph and can be defined by a pair (V, E) in which V is a set of nodes and E is a set of directed edges connecting these nodes. Each node $u \in V$ corresponds to a computational task, while each edge $(u, v) \in E$ represents a precedence relation between tasks u and v. An edge may also represent a communication channel associated with a cost. A node without any incoming (outgoing) edges is called a source (sink) node. A sample task graph is shown in Figure 3.1 for an application with four tasks. In this task graph, tasks 1, 2 and 3 are source nodes, while tasks 3 and 4 are sink nodes. Tasks 1, 2 and 3 can be executed concurrently, while the execution of task 4 has to wait until tasks 1 and 2 are finished.

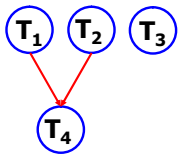


Figure 3.1: A simple task graph with four tasks

In this thesis, a parallel application is executed on a system which contains multiple cores with DVS capability placed on a single die. These cores do not have to be identical and can have different computational capabilities. The operating voltage of each core can be adjusted independently and continuously within a specified range. It is assumed that

the system architecture and the mapping of the task graph onto the cores are given. In other words, we assume that tasks are already scheduled (i.e., the order of execution is determined) and assigned to the cores in the system. An example mapping of the task graph in Figure 3.1 to a dual-core system is shown in Figure 3.2. In Figure 3.2, tasks 1 and 3 are assigned to the first core (denoted by C_1), while tasks 2 and 4 are assigned to the second core (denoted by C_2). Since tasks 1 and 3 are assigned to the same core in Figure 3.2, a new edge (denoted by the dashed arrow) is added to the task graph to represent a precedence relation due to the assignment of both tasks to the same core.

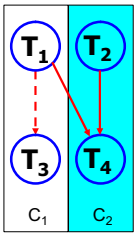


Figure 3.2: A task graph scheduled and assigned to a dual-core system

The execution time of task u is denoted by t_u. t_u is a function of the task's workload L_u and the speed setting of the underlying core

$$t_u = CT(V_{ddu})\, L_u \tag{3.1}$$

In (3.1), CT is called cycle-time (i.e., the length of a clock cycle in seconds) which is a measure of the speed of a core. CT is a function of the core's voltage setting V_{ddu}

$$CT(V_{ddu}) = k_{ct}\, \frac{V_{ddu}}{(V_{ddu} - V_{th})^{a_{ct}}} \tag{3.2}$$

in which k_{ct}, a_{ct} and V_{th} are technology related constants [14]. L_u is defined in terms of clock cycles which is independent of the core speed. (3.1) and (3.2) imply that the execution time of a task is a nonlinear function of the voltage setting. The constants in (3.2) are derived from a commercial processor with DVS capability in which the core voltage can range from 0.85 (V_{ddmin}) to 1.55 (V_{ddmax}) volts [21].

The finishing time of all tasks in a task graph is called completion time and denoted by F_{DAG}. Given L_u's and V_{ddu}'s, F_{DAG} and the finishing time of each task u (denoted by F_u) can be found by a topological traversal of the mapped task graph. The computational

complexity of this operation is $O(N + |E|)$ in which N and $|E|$ denote respectively the number of nodes and edges in the task graph.

In a repetitive RT application, all tasks in a task graph are executed periodically according to a set of timing constraints. These timing constraints appear either as deadlines on finishing times of individual tasks (D_u on F_u) or a global deadline on the completion time (D_{DAG} on F_{DAG}). In a hard RT application, all timing constraints must be satisfied in each period. On the other hand, in a soft RT application, these constraints must be satisfied by a probability of P_{CON}, called the probabilistic performance constraint.

3.2 Power and Energy Consumption

The dynamic power consumed due to the switching activity in a core executing task u can be expressed as follows

$$P_u^{dyn} = \alpha_{0\to1}\, C_u\, V_{dd_u}^2\, f(V_{dd_u}) \tag{3.3}$$

In (3.3), $\alpha_{0\to1}$ is the switching activity factor that is a measure of the switching probability in the digital circuitry. C_u denotes the effective switching capacitance that is a measure of the core utilization by a task. $f(V_{dd_u})$ is the operating frequency of the core that is equal to the inverse of the cycle-time given in (3.2).

The dynamic energy consumed by a core to execute task u at V_{dd_u} can be calculated by multiplying (3.3) by the execution time expressed in (3.2) which results in

$$E_u^{dyn} = C_u\, L_u\, V_{dd_u}^2 \tag{3.4}$$

for $\alpha_{0\to1} = 1$. The total energy consumed in a particular period for a total of N tasks can be computed as follows

$$E^{tot} = \sum_{u=1}^{N} E_u^{dyn} \tag{3.5}$$

The overhead energy consumption due to altering the voltage setting of a core and the leakage energy consumption are neglected for simplicity. However, these energy consumptions can be easily included in the models and formulations provided in this thesis. According to the voltage range used in this thesis, maximum energy savings by DVS for a task is inherently bounded by a factor of approximately 3.3 (i.e., $(V_{ddmax}/V_{ddmin})^2$).

3.3 Energy Management

In this section, several definitions related to energy management are given. An energy management action $\mathcal{V}$ contains a voltage setting for each task in a task graph and can be defined as follows:

Definition 1 *An energy management action $\mathcal{V}$ is a mapping from the set of nodes V of a task graph to a set of voltage settings in the range $[V_{ddmin}, V_{ddmax}]$.*

In a soft real-time application, either a single or multiple actions can be used at runtime. In the latter case, actions are altered at the beginning of periods. A set of actions is called an energy management policy. A policy chooses one of its actions deterministically according to some predefined condition observed in a period. Each unique condition is called a state of the application. In this sense, a state and an energy management policy can be defined as follows:

Definition 2 *Given a task graph (V, E), a state $\boldsymbol{s}$ denotes a condition which holds for task workloads in a particular period or in a set of periods.*

Definition 3 *An energy management policy β defines a mapping from a set of states $\mathcal{S}$ to a set of actions $\mathcal{V}$.*

The remaining three definitions pertain to the types of energy management problems.

Definition 4 *Given a task graph (V, E) and a set of timing constraints, deterministic energy minimization (DEM) is the problem of finding a single optimal action $\mathcal{V}$ which minimizes the total energy consumption subject to these constraints for a given set of task workloads.*

If a DEM problem is solved for every particular workload combination that can occur during periodic execution, then a theoretical bound for the maximum energy savings is found. This policy which maps every unique set of task workloads to a unique action is called β_{BND} and is used to evaluate the effectiveness of the energy minimization policies proposed in this thesis. β_{BND} can not be used in reality, since the knowledge of task workloads in a particular period is not available in advance at runtime.

Definition 5 *Given a task graph (V, E) and a set of timing constraints, hard energy minimization (HEM) is the problem of finding a single optimal action $\mathcal{V}$ which minimizes the average energy consumption subject to these constraints for randomly varying task workloads.*

An HEM policy problem is an instance of the DEM problem in which each task has its worst-case workload. Since this policy meets timing constraints even if tasks have their maximum workloads, it can be used for a hard RT application.

Definition 6 *Given a task graph (V, E), a set of timing constraints and a probabilistic performance constraint P_{CON}, soft (stochastic) energy minimization (SEM) is the problem of finding an optimal policy β which minimizes the average energy consumption and satisfies the timing constraints with a probability of P_{CON} for randomly varying task workloads.*

This thesis focuses on the stochastic energy minimization problem for multicore systems in which the goal is to find optimal policies (with single action or multiple actions) before runtime. Such a policy can be used for a soft RT application. Although the models and the formulations provided in this thesis concentrate on the energy minimization of the cores in a processor, the energy consumption of other system components (e.g., communication interface, disk drives, displays) can also be included into the presented framework. As mentioned in Section 3.1, these components and the data transfers in between can be represented by adding extra nodes and edges to the task graph. If these components also have power management features similar to DVS, an energy management action, defined above, can be extended with extra power modes for such components similar to voltage settings. Therefore, this thesis can be generalized to handle components other than the main processor with minor modifications.

Chapter 4

STOCHASTIC APPLICATION MODELS

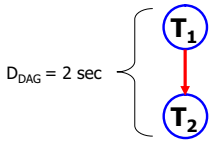
In this chapter, three stochastic application models are presented. Each model suggests a policy for the stochastic energy minimization problem. The first model, presented in Section 4.1, can capture workload variability and spatial correlations. This model suggests an energy management policy with a single action. The other two models, described in Section 4.2, can capture temporal correlations in addition to workload variability and spatial correlations. These models suggest energy management policies with multiple actions. Both sections start with a motivating example for the associated stochastic model and continue with its description. The problem of devising an optimal policy for each stochastic model is addressed in the next chapter.

4.1 Stochastic Model for a Single-Action Policy

4.1.1 Motivating Example

In this section, the motivation behind capturing workload variability and spatial correlations is illustrated on a simple application composed of two tasks running on a single core system. The task graph of this simple application and the marginal workload distributions of tasks are presented in Figure 4.1. Tasks have to be finished within a deadline of 2 seconds. The workload of each task can be either 2 or 10 instructions with equal probability. The processing core has two operating modes where it can execute 6 instructions-per-second in slow mode and 10 instructions-per-second in fast mode.

Figure 4.2a, b and c show three extreme cases in which task workloads are independent, fully positively and fully negatively correlated. For example, in Figure 4.2c, task workloads are fully negatively correlated so that the probability of the workload combinations $(2, 2)$ and $(10, 10)$ are zero. In all of the three cases, each task still has the same marginal workload distribution shown in Figure 4.1b. Let β_{ind} be an energy management policy which ignores spatial correlations and assumes that task workloads are independent (represented



a) Task graph

Task \ Workload	2	10
L_1	0.5	0.5
L_2	0.5	0.5

b) Marginal workload distribution

Figure 4.1: (a) The task graph and (b) marginal workload distributions for a simple application

L_2 \ L_1	2	10
2	0.25	0.25
10	0.25	0.25

a) Independent

L_2 \ L_1	2	10
2	0.5	0
10	0	0.5

b) Positive Correlation

L_2 \ L_1	2	10
2	0	0.5
10	0.5	0

c) Negative Correlation

Figure 4.2: Joint workload distribution in three extreme cases

by Figure 4.2a) in the following two scenarios.

In the first scenario, let the probabilistic performance constraint be 0.75. Given this performance constraint, β_{ind} operates the core in slow mode in which the core can execute 12 instructions within 2 seconds. In this mode, β_{ind} meets the deadline for all workload combinations except $(10, 10)$ to achieve a completion probability of 0.75 according to Figure 4.2a. However, if task workloads are positively correlated (as represented by Figure 4.2b), then β_{ind} achieves a completion probability of 0.50 instead of the required 0.75, since the true probability of $(10, 10)$ is higher due to positive correlation. Therefore, β_{ind} is unable to achieve the desired performance.

In the second scenario, let the application be hard real-time requiring a completion probability of 1 ($P_{CON} = 1$). Given this performance constraint, β_{ind} operates the core in fast mode in which the processor is able to execute 20 instructions within 2 seconds to meet the deadline in all workload combinations. However, if task workloads are negatively correlated (as represented by Figure 4.2c), β_{ind} results in excessive energy consumption since the workload combination of $(10, 10)$ never happens due to negative correlation. These two scenarios show that energy management ignorant of workload variability and spatial correlations may not achieve the desired probabilistic performance or may achieve suboptimal energy savings.

4.1.2 Model Description

In this model, task workloads (L_u's) are treated as discrete random variables with arbitrary distributions. The distribution of task workloads is represented by a joint probability mass function called p_{joint} defined over a multidimensional sample space $\mathbf{L}_{space}$ as follows

$$\mathrm{p}_{joint}(l_1, l_2, .., l_N) = \mathbf{prob}\{L_u = l_u, u = 1, 2, .., N\} \tag{4.1}$$

For instance, the p_{joint} for the workload distribution in Figure 4.2b can be described as follows

$$\begin{aligned} \mathrm{p}_{joint}(2,2) &= 0.5 \\ \mathrm{p}_{joint}(10,10) &= 0.5 \\ \mathrm{p}_{joint}(2,10) &= \mathrm{p}_{joint}(10,2) = 0 \end{aligned}$$

As this example shows, p_{joint} captures both workload variability (i.e., incorporates workload distribution of each task) and spatial correlations (i.e., incorporates correlation between any two tasks) for a given application. However, it can not capture temporal correlations, since it does not include timing information. In other words, task workloads are assumed to be distributed according to p_{joint} in every period independent of time. As a result of this assumption, we can obtain, from this stateless stochastic model, an optimal energy management policy β_{OFF} which uses the same action in each period at runtime. In other words, β_{OFF} contains a fixed voltage setting for every task to be used at runtime in each period. This policy and its underlying model can be defined as follows:

Definition 7 *A stochastic model for a single-action policy β_{OFF} is defined by a joint probability mass function p_{joint} which represents the distribution of task workloads in a sample space $\mathbf{L}_{space}$.*

How p_{joint} is obtained for a target application is described in Chapter 7.

4.2 Stateful Stochastic Models for Multi-Action Policies

4.2.1 Motivating Example

In this section, the motivation behind capturing temporal correlations in addition to workload variability and spatial correlations is illustrated on the same application and the system

described in Section 4.1.1. For this purpose, let task workloads be positively correlated as represented by Figure 4.2b and the desired completion probability be 1. Let β_{single} be a policy which is aware of spatial correlations and uses a single action. β_{single} will operate the core in fast mode in which the processor is able to execute 20 instructions within 2 seconds and the deadline is met for all workload combinations.

Suppose that this application has an inherent feature which results in a workload combination of $(2, 2)$ or $(10, 10)$. In addition, let this feature be observable at the beginning of each period before an action has been chosen. In such a case, the periodic execution of this application can be modeled as a discrete-time stochastic process with a unique state for each particular workload combination. The significance of this stateful model stems from the fact that states convey valuable information about task workloads in each period. From this stateful model, one can derive a policy β_{multi} which can capture temporal correlations as follows: β_{multi} operates the core in slow (fast) mode whenever the application is in the first (second) state. β_{multi} will definitely save more energy than β_{single}, since it can slow down the core in periods in which the feature indicates a workload combination of $(2, 2)$. This example shows that temporal correlations can be captured by stateful stochastic models, and a stateful model can result in more effective policies with multiple actions.

4.2.2 Model Description

The stateful model for the motivating example presented in the previous section can be generalized in two main directions. First, each state can be associated with a workload distribution rather than a single workload combination. As an example, states in a model can be associated with the dashed and the dotted workload distributions shown in Figure 4.3. In this figure, the solid line represents the aggregate workload distribution. Since the workload distributions in states are considerably different in their means and variances, an effective policy with two actions can be devised using such a stateful model as described in the next chapter.

Second, in the motivating example, the application-specific feature was observable at the beginning of each period. However, it is possible that a feature may be observable at the end of a period. In this case, the state of the feature provides information about task workloads for the past period but not for the current period. Such a feature can still be

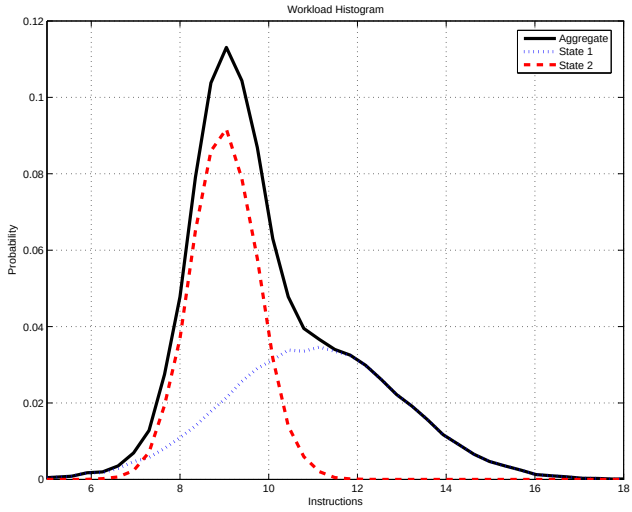


Figure 4.3: Aggregate and state workload distributions

used, if the states are correlated over time. For example, if state $\mathbf{s}_i$ is most likely to be followed by state $\mathbf{s}_j$, then $\mathbf{s}_i$ is able to provide information for the workload distribution in the succeeding period. In other words, the workload distribution in the current state $\mathbf{s}_j$, which is not observable before the execution, is indicated by the $\mathbf{s}_i$. This distinction of stateful models according to the time of observability results in the following two models:

Current-State Observable (CSO)

In this model, the state information is observable at the beginning of a period as in the motivating example. Each state i, denoted by $\mathbf{s}_i$, is observed with a probability of π_i. The joint workload distribution in $\mathbf{s}_i$ is denoted by $\mathrm{p}_{joint|i}$ and computed by conditioning p_{joint} on states. The aggregate workload distribution p_{joint} can be expressed in terms of $\mathrm{p}_{joint|i}$'s using Bayes' formula as follows

$$\mathrm{p}_{joint}(l_1, .., l_N) = \sum_{i=1}^{m} \pi_i \, \mathrm{p}_{joint|i}(l_1, .., l_N) \tag{4.2}$$

in which m denotes the total number of states.

A current-state observable model suggests a policy which uses a separate action $\mathcal{V}_i$ for $\mathbf{s}_i$. At runtime, the action associated with the current state is chosen at the beginning of each period. Consequently, each action is optimized according to the conditional joint workload

distribution in the associated state. The improvement brought by β_{CSO} over β_{OFF} depends on the dissimilarity of joint workload distributions in states. If task workloads have exactly the same distribution in each state, then a multi-action policy will converge into a single-action policy which assumes that tasks have the same workload distribution in each period. This model can be formally defined as follows:

Definition 8 *A current-state observable model for a multi-action policy (β_{CSO}) is a tuple ($\mathcal{S}$, $\mathbf{\Pi}$, $\boldsymbol{P_{JOINT|\mathcal{S}}}$) where*

- *$\mathcal{S}$ is the set of states, $\boldsymbol{s}_i \in \mathcal{S}$ denotes the i^{th} state*
- *$\mathbf{\Pi}$ is the steady state distribution vector, $\pi_i \in \mathbf{\Pi}$ denotes the probability that $\boldsymbol{s}_i$ is observed*
- *$\boldsymbol{P_{JOINT|\mathcal{S}}}$ is the set of conditional joint workload distribution functions, $p_{joint|i} \in \boldsymbol{P_{JOINT|\mathcal{S}}}$ denotes the conditional joint workload distribution in $\boldsymbol{s}_i$*

Previous-State Observable (PSO)

In this model, states are observable at the end of each period after execution. Therefore, only the state of the previous period is known before execution. It is assumed that state transitions possess the Markov property, and consequently, this model corresponds to a Markov Chain [22]. This Markov Chain is expected to be irreducible and recurrent for state limiting probabilities to exist, since it is desired that the policy will use all of its actions in the long run. Transitions between states is represented with a matrix $\mathbf{A}$ in which a_{ij} is the probability of transition from $\mathbf{s}_i$ to $\mathbf{s}_j$. Both the state and the workload distribution of the current period depends on the outgoing transition probabilities (a_{ij}'s) of the previous state.

A previous-state observable model suggests a policy which uses a separate action for each state. At runtime, the action associated with the state of the previous period is chosen at the beginning of the current period. Since the state of the current period is not known before execution, each action $\mathcal{V}_i$ is optimized not according to $\mathrm{p}_{joint|i}$, but according to the workload distribution in all states reachable from $\mathbf{s}_i$ in a single transition. This joint workload distribution of tasks experienced by $\mathcal{V}_i$ is represented with p^i_{joint} and is computed by conditioning on reachable states from $\mathbf{s}_i$ in a single transition as follows

$$\mathrm{p}^i_{joint}(l_1, .., l_N) = \sum_{j=1}^{m} a_{ij}\ \mathrm{p}_{joint|j}(l_1, .., l_N) \tag{4.3}$$

Consequently, the effectiveness of a β_{PSO} policy depends on the dissimilarity of p^{i}_{joint}'s rather than $\mathrm{p}_{joint|i}$'s. The unlikeliness of $\mathrm{p}_{joint|i}$'s is necessary, but not sufficient for the effectiveness of a β_{PSO} policy, since state transition probabilities may destroy the unlikeliness as expressed in (4.3). This model can be formally defined as follows:

Definition 9 *A previous-state observable model for a multi-action policy (β_{PSO}) is a tuple ($\mathcal{S}$, $\boldsymbol{A}$, $\boldsymbol{P_{JOINT|\mathcal{S}}}$) where*

- *$\mathcal{S}$ is the set of states, $\boldsymbol{s}_i \in \mathcal{S}$ denotes the i^{th} state*
- *$\boldsymbol{A}$ is the transition probability matrix, a_{ij} denotes the probability of transition from $\boldsymbol{s}_i$ to $\boldsymbol{s}_j$*
- *$\boldsymbol{P_{JOINT|\mathcal{S}}}$ is the set of conditional joint workload distribution functions, $p_{joint|i} \in \boldsymbol{P_{JOINT|\mathcal{S}}}$ is the conditional joint workload distribution in $\boldsymbol{s}_i$*

Chapter 5

OPTIMAL ENERGY MANAGEMENT POLICIES

In this chapter, we formulate deterministic and stochastic energy minimization as optimization problems. The optimization formulations for stochastic energy minimization employ the stochastic application models presented in the previous chapter. In Section 5.3, we provide a brief background on activity network models and discuss the similarities between energy minimization problems and activity network problems. The solution techniques for these optimization problems are presented in the next chapter.

5.1 Optimization Formulation for Deterministic Energy Minimization

The objective of a deterministic energy minimization problem is to minimize the total energy consumption for a given set of task workloads $(l_1, .., l_N)$. This objective can be expressed as follows

$$E^{tot} = \sum_{u=1}^{N} C_u \, l_u \, V_{dd_u}^2 \tag{5.1}$$

The minimization of (5.1) is constrained by both the supply voltage range and the timing constraints. Using (5.1) as the objective, the optimization formulation for a DEM problem for a particular set of task workloads $(l_1, l_2, .., l_n)$ can be stated as follows:

minimize

$\sum_{u=1}^{N} C_u \, l_u \, V_{dd_u}^2$

subject to

$t_u - CT(V_{dd_u}) \, l_u = 0$	for each node u	$(C.1)$
$t_u - F_u \leq 0$	for each source node u	$(C.2)$
$F_u + t_v - F_v \leq 0$	for each edge (u, v)	$(C.3)$
$F_u - D_u \leq 0$	for each node u	$(C.4)$
$F_u - D_{DAG} \leq 0$	for each sink node u	$(C.5)$
$V_{dd_u} \in [V_{dd_{min}}, V_{dd_{max}}]$	for each node u	$(C.6)$

in which V_{ddu}'s are the variables. In this formulation, the first set of equality constraints, denoted by $(C.1)$, establish the relation between task execution times and voltage settings. In $(C.2) - (C.5)$, the finishing time of tasks (F_u's) are used as a set of auxiliary variables in specifying the precedence relations between tasks and the timing constraints [14, 23].

5.2 Optimization Formulations for Stochastic Energy Minimization

In this section, we first provide a brief background on stochastic programming. Then, we formulate SEM problems as stochastic programs based on stochastic application models.

5.2.1 Stochastic Programming

Stochastic programming (SP) refers to an optimization problem which involves uncertainty in its problem parameters. This uncertainty may either appear in the objective or the constraints of a problem. An SP problem has the following general form:

$$\begin{aligned} &\text{minimize} \\ &\qquad f_0(x, w) \\ &\text{subject to} \\ &\qquad f_i(x, w) \leq 0 \quad i = 1, .., m \end{aligned}$$

in which x is a set of optimization variables and w is a set of randomly varying problem parameters [24]. In an SP problem, the exact distribution of w is not always known, and only a set of samples for w may be available. When a decision has to be made *here-and-now* according to the available distribution information, the corresponding optimization model is called anticipative. For an anticipative model, a reliability level $\alpha \in [0, 1]$ for the satisfaction of constraints can be defined as follows

$$\mathbf{prob}\{w | f_i(x, w) \leq 0, i, .., m\} \geq \alpha \tag{5.2}$$

The expression in (5.2) is called a joint chance constraint, since the inequality needs to hold for all $f_i(x, w)$'s for this constraint to be satisfied. Such an anticipative model in which the expected value of $f_0(x, w)$ is minimized can be stated as follows:

minimize

$$\mathbf{E}\ [f_0(x, w)]$$

subject to

$$\mathbf{prob}\{w | f_i(x, w) \leq 0, i, .., m\} \geq \alpha$$

In stochastic energy minimization, randomly varying task workloads correspond to problem parameters in an SP problem. The exact distribution of task workloads is in fact unknown for a given application, but derived by a data collection methodology that will be explained in Chapter 7. Using this imperfect distribution information, the goal is to make an offline (here-and-now) decision for voltage settings that will be used at runtime. Consequently, all of the formulations for SEM problems presented in the following sections fit in the generic form of an anticipative stochastic programming model.

5.2.2 Formulation for a Single-Action Policy

The objective of a single-action policy β_{OFF} is to minimize the average energy consumption of a soft RT application while meeting a probabilistic performance constraint. When task workloads are defined as random variables, the total energy consumed in a period expressed in (3.5) becomes a random-valued function. Consequently, the average energy consumed by an action can be computed using the expectation of E^{tot} as follows

$$\mathbf{E}\ \left[E^{tot}\right] = \sum_{u=1}^{N} \mathbf{E}\ \left[E_u^{dyn}\right] = \sum_{u=1}^{N} C_u\ \mathbf{E}\ [L_u]\ {V_{dd}}_u^2 \tag{5.3}$$

in which $\mathbf{E}\ [\cdot]$ denotes the expectation operator. The expected value of a task workload L_u can be computed using p_{joint}. The expression in (5.3) is the objective function of the optimization problem. The minimization of (5.3) is constrained by both the supply voltage range and the probabilistic performance constraint. The probabilistic performance constraint is specified as a joint chance constraint over the probability space defined by p_{joint} as follows

$$\mathbf{prob}\left\{\begin{array}{lll} & (t_u - CT({V_{dd}}_u)\ L_u = 0 & \text{for each node } u) \\ \wedge & (t_u - F_u \leq 0 & \text{for each source node } u) \\ \wedge & (F_u + t_v - F_v \leq 0 & \text{for each edge } (u, v)) \\ \wedge & (F_u - D_u \leq 0 & \text{for each node } u) \\ \wedge & (F_u - D_{DAG} \leq 0 & \text{for each sink node } u) \end{array}\right\} \geq P_{CON} \tag{5.4}$$

in which P_{CON} represents the tolerance of the target application to deadline misses. This chance constraint states that the probability that all timing constraints are satisfied should be at least P_{CON}. The completion probability achieved by an action $\mathcal{V}$ (i.e., the left hand side of (5.4)) is denoted by $\mathrm{P}_{SAT}(\mathcal{V})$. The subspace of $\mathbf{L}_{space}$ in which timing constraints are satisfied with a given action $\mathcal{V}$ is denoted by $\mathbf{L}_{space}^{\mathcal{V}}$.

Using (5.3) as the objective and (5.4) as a constraint, the optimization problem for a single-action policy can be stated as follows:

$$\begin{aligned} &\text{minimize} \\ &\qquad \textstyle\sum_{u=1}^{N} C_u \ \mathbf{E}\left[L_u\right] V_{dd_u}^2 \\ &\text{subject to} \\ &\qquad \mathrm{P}_{SAT}(\mathcal{V}) \geq P_{CON} \\ &\qquad V_{dd_u} \in [V_{ddmin}, V_{ddmax}] \end{aligned}$$

in which V_{dd_u}'s are the optimization variables.

A Runtime Heuristic on a Single-Action Policy

Although the action found in the previous section is optimal for a given $\mathbf{L}_{space}$ and P_{CON}, it is possible to save energy further by adjusting voltage settings to reclaim timing slacks occurring at runtime. This can be achieved by lowering a voltage setting V_{dd_u} in $\mathcal{V}$ when the value of L_u is lower than expected in a particular period while preserving the condition that P_{CON} is satisfied. P_{CON} can be preserved by assuring that each period which was in $\mathbf{L}_{space}^{\mathcal{V}}$ before adjustment stays in $\mathbf{L}_{space}^{\mathcal{V}}$ even if some of the voltage settings are lowered. This can be guaranteed as follows: If it is detected before the execution of task u that there is going to be slack even if all tasks after u (including u) have their worst-case workload combination in $\mathbf{L}_{space}^{\mathcal{V}}$, then it is safe to lower V_{dd_u}. Since the workload for task u is unknown, V_{dd_u} can be adjusted assuming that task u has its maximum workload in $\mathbf{L}_{space}^{\mathcal{V}}$ to preserve P_{CON}. To implement this heuristic at runtime, the following two quantities are defined:

Definition 10 *$L_{u,max}$ is the maximum workload of task u in $\mathbf{L}_{space}^{\mathcal{V}}$.*

Definition 11 *$t_{u,s}^{\mathcal{V}}$ is the start time for task u for the worst-case path in $\mathbf{L}_{space}^{\mathcal{V}}$ starting at task u and ending at a sink task, when all tasks are executed at $\mathcal{V}$.*

Both quantities can be computed off-line for each task u using $\mathbf{L}^{\mathcal{V}}_{space}$ and a topological traversal of the task graph. This computation takes linear time in the size of the task graph and the number of points in $\mathbf{L}^{\mathcal{V}}_{space}$. $L_{u,max}$ and $t^{\mathcal{V}}_{u,s}$ are used at runtime before the execution of every task as shown in Figure 5.1.

ONLINE
1 $\Delta t \leftarrow t^{\mathcal{V}}_{u,s} - t_{current}$
2 **if** $\Delta t > 0$
3 **then**
4 $V_{dd_u}{}' = CT^{-1}(\,(\,CT(V_{dd_u})\,L_{u,max} + \Delta t)/L_{u,max}\,)$

Figure 5.1: Runtime adjustments to voltage settings

In Figure 5.1, $t^{\mathcal{V}}_{u,s}$ is used to detect slack before the execution of each task. This operation ensures that a slack has occurred at runtime, even if the worst-case path from task u to a sink node in $\mathbf{L}^{\mathcal{V}}_{space}$ will be experienced in this period. Then, $L_{u,max}$ is used as a reference to lower V_{dd_u}. This operation preserves the completion probability, since $L_{u,max}$ is larger than or equal to the workload of task u in $t^{\mathcal{V}}_{u,s}$. The computational overhead of this heuristic, which requires only a few arithmetic operations and a comparison, is negligible when compared to a typical task workload. This runtime heuristic is used only for a β_{OFF} policy and denoted by β_{ONL} in the rest of the thesis.

5.2.3 Formulation for Multi-Action Policies

Current-State Observable

A β_{CSO} policy is based on a current-state observable model and uses a separate action for each state. Consequently, the objective function of a β_{CSO} policy can be computed by conditioning the expected energy consumption on states as follows

$$\mathbf{E}\left[E^{tot}\right] = \sum_{i=1}^{m} \pi_i \; \mathbf{E}\left[E^{tot}\,|\mathbf{s}_i\right] \tag{5.5}$$

for a total of m states, in which

$$\mathbf{E}\left[E^{tot}\,|\mathbf{s}_i\right] = \sum_{u=1}^{N} \mathbf{E}\left[E_u^{dyn}\,|\mathbf{s}_i\right] \tag{5.6}$$

$$\mathbf{E}\left[E_u^{dyn}\,|\mathbf{s}_i\right] = C_u\ \mathbf{E}\left[L_u\,|\mathbf{s}_i\right] V_{dd_{u,i}}^2 \tag{5.7}$$

$V_{dd_{u,i}}$ is the voltage setting for task u in action $\mathcal{V}_i$, and $\mathbf{E}\left[L_u\,|\mathbf{s}_i\right]$ can be computed using $\mathrm{p}_{joint|i}$. The completion probability of a set of actions is also calculated by conditioning P_{SAT} on states

$$\mathrm{P}_{SAT} = \sum_{i=1}^{m} \pi_i\, \mathrm{P}_{SAT_i}(\mathcal{V}_i) \tag{5.8}$$

in which $\mathrm{P}_{SAT_i}(\mathcal{V}_i)$ is the completion probability achieved by action i in $\mathbf{s}_i$. As mentioned in Section 4.2.2, $\mathrm{p}_{joint|i}$ represents the conditional joint workload distribution in $\mathbf{s}_i$.

Using (5.5) as the objective and (5.8) as a constraint, the optimization problem based on a current-state observable model can be stated as follows:

minimize

$$\textstyle\sum_{i=1}^{m} \pi_i\ \mathbf{E}\left[E^{tot}\,|\mathbf{s}_i\right]$$

subject to

$$\textstyle\sum_{i=1}^{m} \pi_i\, \mathrm{P}_{SAT_i}(\mathcal{V}_i) \geq P_{CON}$$

$$V_{dd_{u,i}} \in [V_{dd_{min}}, V_{dd_{max}}]$$

with a total of $N \times m$ optimization variables (i.e., $V_{dd_{u,i}} \in \mathcal{V}_i$ for task u in action i).

Previous-State Observable

A β_{PSO} policy is based on a previous-state observable model and uses a separate action for each state. β_{PSO} uses its action $\mathcal{V}_i$ whenever the previous period is in state $\mathbf{s}_i$. The frequency of using action j in state i is denoted by π_{ij} which satisfies

$$\sum_{i=1}^{m} \sum_{j=1}^{m} \pi_{ij} = 1 \tag{5.9}$$

Since each action i is used once state i is observed, each π_{ij} can be expressed as follows

$$\pi_{ij} = \pi_j\, a_{ji} \tag{5.10}$$

Consequently, the objective function for this problem is computed by conditioning the expected energy consumption both on states and actions as follows

$$\mathbf{E}\left[E^{tot}\right] = \sum_{i=1}^{m} \sum_{j=1}^{m} \pi_{ij}\ \mathbf{E}\left[E^{tot}\,|\mathbf{s}_i, \mathcal{V}_j\right] \tag{5.11}$$

for a total of m states in which

$$\mathbf{E}\left[E^{tot}\,|\mathbf{s}_i,\mathcal{V}_j\right] = \sum_{u=1}^{N} \mathbf{E}\left[E_u^{dyn}\,|\mathbf{s}_i,\mathcal{V}_j\right] \tag{5.12}$$

$$\mathbf{E}\left[E_u^{dyn}\,|\mathbf{s}_i,\mathcal{V}_j\right] = C_u\ \mathbf{E}\left[L_u\,|\mathbf{s}_i\right] V_{dd\,u,j}^2 \tag{5.13}$$

Similarly, the completion probability of a set of actions is calculated by conditioning P_{SAT} on both states and actions as follows

$$\mathrm{P}_{SAT} = \sum_{i=1}^{m}\ \sum_{j=1}^{m} \pi_{ij}\,\mathrm{P}_{SAT\,i}(\mathcal{V}_j) \tag{5.14}$$

in which $\mathrm{P}_{SAT\,i}(\mathcal{V}_j)$ is the completion probability achieved by action j in the conditioned probability space defined by $\mathrm{p}_{joint|i}$.

Using (5.11) as the objective and (5.14) as a constraint, the optimization problem for this model can be stated as follows:

minimize

$$\textstyle\sum_{i=1}^{m}\ \sum_{j=1}^{m} \pi_{ij}\ \mathbf{E}\left[E^{tot}\,|\mathbf{s}_i,\mathcal{V}_j\right]$$

subject to

$$\textstyle\sum_{i=1}^{m}\ \sum_{j=1}^{m} \pi_{ij}\,\mathrm{P}_{SAT\,i}(\mathcal{V}_j) \geq P_{CON}$$

$$V_{dd\,u,j} \in [V_{dd\,min}, V_{dd\,max}]$$

with a total of $N \times m$ variables (i.e., $V_{dd\,u,j} \in \mathcal{V}_j$ for task u in action j).

The optimization problem formulated above corresponds to a policy optimization problem on a Markov Decision Process in which the objective is to minimize the long run average value of a cost function and the decision variables are voltage settings [25]. In our case, the policy is both stationary and deterministic, since each state is associated with a particular action. In addition, state transition probabilities are independent of the actions.

5.3 Energy Minimization and Activity Networks

Activity networks (AN) have been used to model applications from various domains including project management, parallel processing and timing analysis of digital circuits [23]. An activity network contains a set of activities with a set of precedence relations represented by a directed acyclic graph. Each activity u has a duration d_u, which is a function of a design variable x_u. The finishing time of all activities for a given set of d_u's in an activity

network is called the makespan t_{mk}. Activity networks can be classified as deterministic or stochastic according to the nature of durations as explained next.

In a deterministic activity network (DAN), activity durations are deterministic functions of design variables. In other words, design variables directly determine activity durations. A typical objective in an optimal DAN problem is to find x_u's that minimize the makespan. This problem can be formulated as follows:

minimize

$$t_{mk}$$

subject to

$$\begin{aligned} & d_u(x_u) - F_u \leq 0 && \text{for each source node } u \\ & F_u + d_v(x_v) - F_v \leq 0 && \text{for each edge } (u, v) \\ & F_u - t_{mk} \leq 0 && \text{for each sink node } u \end{aligned}$$

in which x_u's are the optimization variables and F_u denotes the finishing time of activity u.

In a stochastic activity network (SAN), design variables determine the distribution of activity durations. In other words, an activity duration is a random-valued function so that the makespan of a SAN also becomes a random quantity. A typical objective in an optimal SAN is to minimize the average value of the makespan or to keep it below a threshold value t_{TH} according to a reliability level α as follows

$$\mathbf{prob}\{t_{mk} \leq t_{TH}\} \geq \alpha \tag{5.15}$$

Given these definitions, it can be observed that the task graph representation of a parallel application is quite similar to an activity network where the execution time t_u (voltage setting V_{dd_u}) of a task corresponds to the duration d_u (design variable x_u) of an activity. The completion time defined for a task graph is the same as the makespan of an activity network. Consequently, a DEM problem can easily be converted into an optimal DAN problem, while a single-action SEM problem can be formulated as an optimal SAN problem. Using these conversions, a solution technique devised for optimal activity network problems can be used for energy minimization problems. This issue is further discussed in the next chapter.

Chapter 6

SOLUTION TECHNIQUES FOR OPTIMAL POLICY PROBLEMS

This chapter describes the solution techniques used for the optimal energy management policy problems formulated in the previous chapter. The solution of a deterministic energy minimization problem is discussed in Section 6.1. Two approximation-based solution techniques for a stochastic energy minimization problem are discussed in Section 6.2.

6.1 Solution of a Deterministic Energy Minimization Problem

The optimization problem formulated for DEM in Section 5.1 has a convex objective function. Since cycle-time is also a convex function of the voltage setting as expressed in (3.2), a DEM problem is of type convex programming. Therefore, the global minimum of a DEM problem can be found efficiently. In this thesis, we formulate DEM problems as geometric programs.

A geometric program (GP) is a convex optimization problem which can be solved globally and efficiently [26]. A GP problem has the following generic form:

$$\begin{aligned} &\text{minimize} \\ &\qquad f_0(x) \\ &\text{subject to} \\ &\qquad f_i(x) \le 1 \qquad\qquad i = 1, .., m \\ &\qquad g_j(x) = 1 \qquad\qquad j = 1, .., k \end{aligned}$$

in which x is a vector of variables, $g_i(x)$'s are monomials, and $f_0(x)$ and $f_i(x)$'s are posynomials. In the context of geometric programming, a monomial is a real-valued function g of x with the form

$$g(x) = c \prod_{i=1}^{N} x_i^{a_i} \tag{6.1}$$

in which $c > 0$ and $a_i \in \mathbf{R}$. A posynomial f of x is a sum of monomials of the form

$$f(x) = \sum_{j=1}^{K} \{c_j \prod_{i=1}^{N} x_i^{a_i}\} \tag{6.2}$$

in which $c_j > 0$. According to these definitions, monomials are closed under multiplication and division, while posynomials are closed under addition and multiplication. Although these forms seem to be restrictive, there exist several transformation techniques to formulate a problem as a GP. Further information on geometric programming and its extensions can also be found in [26].

Given these definitions, it can be observed that the objective function of a DEM problem is already a posynomial. The nonlinear relationship between the execution time and the voltage setting (expressed in (3.1) and (3.2)) can be expressed as a posynomial by introducing an extra variable $V_{ddu}^{'}$ for each V_{ddu} to eliminate the subtraction operation in the denominator of (3.2) as follows

$$t_u = k_{ct}\, V_{ddu}\, (V_{ddu}^{'})^{-a_{ct}}\, L_u \tag{6.3}$$

and adding the following inequality constraint in the formulation.

$$V_{ddu}^{'} + V_{th} < V_{ddu} \tag{6.4}$$

Using (6.3) and (6.4), a DEM problem for a particular set of task workloads $(l_1, l_2, .., l_n)$ is formulated as the following GP:

minimize

$\sum_{u=1}^{N} C_u\, l_u\, V_{ddu}^2$

subject to

$(k_{ct}\, V_{ddu}\, (V_{ddu}^{'})^{-a_{ct}}\, l_u)/t_u^{-1} \leq 1$	for each node u
$(V_{ddu}^{'} + V_{th})/V_{ddu} \leq 1$	for each node u
$t_u/F_u \leq 1$	for each source node u
$(F_u/F_v) + (t_v/F_v) \leq 1$	for each edge (u, v)
$F_u/D_u \leq 1$	for each node u
$F_u/D_{DAG} \leq 1$	for each sink node u
$V_{ddu}/V_{ddmax} \leq 1$	for each node u
$V_{ddmin}/V_{ddu} \leq 1$	for each node u

in which the variables are V_{ddu}'s. The solution of this optimization problem results in the optimal voltage settings for a particular set of task workloads. In this thesis, each DEM problem is formulated as a GP and solved using GGPLAB [27].

6.2 Solution of a Stochastic Energy Minimization Problem

6.2.1 Solution Methods for Stochastic Programs

A common difficulty encountered in stochastic programming (SP) problems is the high cost of evaluating or differentiating the chance constraint, especially when the set of problem parameters (denoted by w in Section 5.2.1) is continuously distributed. It is also possible that the analytical gradient of this function is not available. Therefore, the solution approach for an SP problem heavily depends on the distribution of the random parameters and the structure of the objective and constraint functions. Solution methods for SP problems can be classified as approximation-based techniques or stochastic techniques [24].

The key idea behind approximation-based schemes is to convert an SP problem into a deterministic equivalent problem, the solution of which is well-studied. The simplest form of this transformation is to use the average values of the random vector w in the objective and constraint functions. Another approach is to represent w with another random vector w' which simplifies the evaluation of the objective and constraint functions. For instance, a continuously-distributed w can be approximated with a discretely-distributed w', or a discretely-distributed w can be approximated with a w' which has a coarser distribution.

Stochastic techniques are based on random search guided by the quasi-gradients of the objective and constraint functions. In this context, quasi-gradient refers to the gradient of these functions computed at a particular value of w, rather than using its distribution. Quasi-gradients are used to guide a search in the solution space and find a feasible solution which is not guaranteed to be optimal. Stochastic techniques are preferred when the distribution of w is not available or the computation of gradients are too expensive.

6.2.2 Solution of a Stochastic Energy Minimization Problem

The objective functions in the SEM formulations presented in Section 5.2 use the expected or conditional expected values of task workloads. Each formulation has a quadratic objective

function which is both continuous and differentiable over its domain. Thus, the uncertainty in these problems is contained in the joint chance constraints in which a probabilistic performance constraint P_{CON} corresponds to a reliability level.

Since task workloads are assumed to be discretely-distributed quantities, each joint chance constraint can be computed by explicit enumeration in the sample space $\mathbf{L}_{space}$. This is achieved by computing the finishing times of tasks and the whole graph for every observation, i.e., a unique $N-$tuple of workload values, in $\mathbf{L}_{space}$ for given task voltages. This operation requires a computational complexity of $O(L\,(N+|E|))$ in which L is the number of points in $\mathbf{L}_{space}$, N and $|E|$ are the number of tasks and edges in the task graph respectively.

Using this approach, the completion probability of an action in the whole space ($\mathrm{P}_{SAT}(\mathcal{V})$) can be evaluated by enumerating all points in the $\mathbf{L}_{space}$ as follows

$$\mathrm{P}_{SAT}(\mathcal{V}) = \sum_{k=1}^{|\mathbf{L}_{space}|} \mathrm{I}_k \, \mathrm{p}_{joint}(l_{(1,k)}, .., l_{(N,k)}) \tag{6.5}$$

in which $l_{(u,k)}$ represents the workload of task u at the k^{th} point, and I_k is an indicator variable denoting whether the timing constraints are satisfied for the k^{th} point or not.

Evaluation of $\mathrm{P}_{SAT\,i}(\mathcal{V}_i)$ requires enumeration of points which occur with nonzero probability in state $\mathbf{s}_i$. The set of these points is denoted by $\mathbf{L}_{space,i}$. Using this subspace, $\mathrm{P}_{SAT\,i}(\mathcal{V}_i)$ can be evaluated as follows

$$\mathrm{P}_{SAT\,i}(\mathcal{V}_i) = \sum_{k=1}^{|\mathbf{L}_{space,i}|} \mathrm{I}_k \, \mathrm{p}_{joint|i}(l_{(1,k)}, .., l_{(N,k)}) \tag{6.6}$$

$\mathrm{P}_{SAT\,i}(\mathcal{V}_j)$ can also be evaluated by explicit enumeration in a similar manner.

Using these evaluation methods for joint chance constraints, each SEM formulation becomes a constrained nonlinear optimization problem. In this thesis, a sequential quadratic programming solver (*fmincon* provided in MATLAB [28]) is used to solve this type of problems in which the gradient of a chance constraint is computed by finite differences. The only approximation in this solution technique is the usage of expected values in the objective function, while each constraint function is computed precisely using the sample space. Since all optimization problems presented in this thesis are solved offline, the computational complexity of evaluating a constraint function (i.e., $O(L\,(N+|E|))$) is not a major concern. Additionally, this complexity can be reduced by approximating $\mathbf{L}_{space}$ with a new sample

space in which task workloads are represented using a relatively smaller number of discrete values. In this thesis, this solution technique is particularly used for the single-action policy problems. For multi-action policy problems, the solution technique described in the next section is used.

6.2.3 Solution of the Deterministic Approximation of a Stochastic Energy Minimization Problem

Our experiments have shown that the solution technique presented in the previous section has efficiency problems when it is applied to optimal policy problems based on stateful models. This inefficiency is due to the fact that the chance constraint function is not numerically well-behaved in the cases of β_{CSO} and β_{PSO}. Therefore, we devised a new, robust solution technique to solve the optimal policy problems based on stateful models. In this solution technique, an SEM problem is approximated by a two-level optimization problem in which the second level is equivalent to a DEM problem. This technique employs a heuristic approximation, which is explained next, used for stochastic activity networks.

In [23], a heuristic approach has been proposed for an optimal SAN problem that appears in IC design. In this approach, a SAN problem is approximated by a DAN problem in which margin coefficients are used to account for the statistical variation of activity durations. In the approximated DAN problem, each activity duration is expressed as a linear function of its mean and standard deviation as follows

$$d_u{}'(x_u) = \mu(d_u(x_u)) + \kappa_u \, \sigma(d_u(x_u)) \tag{6.7}$$

In (6.7), $\mu(d_u(x_u))$ and $\sigma(d_u(x_u))$ denote the mean and standard deviation of $d_u(x_u)$ respectively, and κ_u is the margin coefficient. For each given set of margin coefficients, the following DAN problem is solved:

minimize

$$t_{mk}$$

subject to

$$\begin{aligned}
& d_u{}'(x_u) - F_u \leq 0 && \text{for each source node } u \\
& F_u + d_v{}'(x_v) - F_v \leq 0 && \text{for each edge } (u, v) \\
& F_u - t_{mk} \leq 0 && \text{for each sink node } u
\end{aligned}$$

in which x_u's are the optimization variables. The optimal x_u's for a given set of margin coefficients are used to evaluate the probabilistic chance constraint expressed in (5.15). For choosing the optimal margin coefficients, several schemes have been discussed in [23]. One of these schemes is based on an exhaustive search in the space of margin coefficients. In another scheme, margin coefficients are chosen by an iterative search in which an approximated DAN problem is solved and margin coefficients are updated according to some rule in each iteration.

We use the heuristic approach based on margin coefficients in [23] to solve the two-level formulation of a stochastic energy minimization problem. In our solution technique, the search in the space of margin coefficients is not performed heuristically but formulated as an optimization problem. In a two-level formulation of an SEM problem, the goal of the first level is to find the optimal margin coefficients which result in minimum expected energy consumption while satisfying the probabilistic performance constraint. The second level, which is equivalent to a DEM problem described in Section 5.1, is solved for each margin coefficient to find the optimal action which minimizes the expected energy consumption while satisfying the timing constraints.

The two-level formulation for a single-action policy problem incorporates a single margin coefficient for all tasks in the task graph. This optimization problem can be formulated as follows:

1st Level		
minimize	$E^{tot'}_{\kappa}$	
subject to	$\mathrm{P}_{SAT'_{\kappa}} \geq P_{CON}$	
2nd Level	for each given value of κ	
minimize	$\sum_{u=1}^{N} C_u \ \mathbf{E}\left[L_u\right] V_{dd_u}^2$	
subject to		
	$t_u - CT(V_{ddu})\, l_u = 0$	for each node u
	$t_u - F_u \leq 0$	for each source node u
	$F_u + t_v - F_v \leq 0$	for each edge (u, v)
	$F_u - D_u \leq 0$	for each node u
	$F_u - D_{DAG} \leq 0$	for each sink node u
	$V_{dd_u} \in [V_{dd_{min}}, V_{dd_{max}}]$	for each node u

In this formulation, the variable of the first level is the margin coefficient, while the variables of the second level are the voltage settings in the single action $\mathcal{V}$. The workload l_u of each task in the second level is approximated as follows

$$l_u = \mu(L_u) + \kappa\, \sigma(L_u) \tag{6.8}$$

Evaluation of $E^{tot'}$ and $\mathrm{P}_{SAT}{}'$ for every particular κ value requires the solution of the second-level problem. $E^{tot'}$ and $\mathrm{P}_{SAT}{}'$ can be expressed as follows

$$E^{tot'}{}_{\kappa} = \sum_{u=1}^{N} C_u\ \mathbf{E}\ [L_u]\ (V_{ddu})^2$$

$$\mathrm{P}_{SAT}{}'_{\kappa} = \mathrm{P}_{SAT}(\mathcal{V})$$

in which V_{ddu}'s and $\mathcal{V}$ are found by the solution of the second level problem for κ.

The two-level formulation for a multi-action policy problem incorporates a separate margin coefficient for each action. This optimization problem can be formulated as follows:

1st Level		
minimize	$\sum_{i=1}^{m} \pi_i\, E^{tot'}{}_{\kappa_i}$	
subject to	$\sum_{i=1}^{m} \pi_i\, \mathrm{P}_{SAT}{}'_{\kappa_i} \geq P_{CON}$	
2nd Level	for each given value of κ_i	
minimize	$\sum_{u=1}^{N} C_u\ \mathbf{E}\ [L_{ui}]\, V_{dd\,u,i}^2$	
subject to		
	$t_u - CT(V_{ddu,i})\, l_{u,i} = 0$	for each node u
	$t_u - F_u \leq 0$	for each source node u
	$F_u + t_v - F_v \leq 0$	for each edge (u, v)
	$F_u - D_u \leq 0$	for each node u
	$F_u - D_{DAG} \leq 0$	for each sink node u
	$V_{ddu} \in [V_{ddmin}, V_{ddmax}]$	for each node u

In this formulation, the variables of the first level are the margin coefficients, while the variables of the second level are the voltage settings in $\mathcal{V}_i$. The workload of $l_{u,i}$ of each task in the second level is expressed as follows

$$l_{u,i} = \mu(L_{ui}) + \kappa_i\, \sigma(L_{ui}) \tag{6.9}$$

For β_{CSO}, L_{ui} in (6.9) represents the statistical workload of task u in state $\mathbf{s}_i$. Therefore, $\mu(L_{ui})$ and $\sigma(L_{ui})$ can be calculated easily using conditional workload distribution $\mathrm{p}_{joint|i}$ defined in Section 4.2.2. ${E^{tot}}'_{\kappa_i}$ and ${\mathrm{P}_{SAT}}'_{\kappa_i}$ can be expressed as follows

$$
\begin{aligned}
{E^{tot}}'_{\kappa_i} &= \sum_{u=1}^{N} C_u \ \mathbf{E}\left[L_u \,|\mathbf{s}_i\right] V_{dd}{}^2_{u,i} \\
{\mathrm{P}_{SAT}}'_{\kappa_i} &= \mathrm{P}_{SAT\,i}(\mathcal{V}_i)
\end{aligned}
$$

For β_{PSO}, L_{ui} represents the statistical workload of task u in all states reachable within single transition from state i. This distribution is represented by p^i_{joint} defined in Section 4.2.2. $\mu(L_{ui})$ and $\sigma(L_{ui})$ can be expressed as follows

$$
\begin{aligned}
\mu(L_{ui}) &= \sum_{j=1}^{m} a_{ij} \ \mathbf{E}\left[L_u \,|\mathbf{s}_j\right] \\
\sigma(L_{ui}) &= \sum_{j=1}^{m} a_{ij}\, \sigma(L_u \,|\mathbf{s}_j)
\end{aligned}
$$

${E^{tot}}'_{\kappa_i}$ and ${\mathrm{P}_{SAT}}'_{\kappa_i}$ can be expressed as follows

$$
\begin{aligned}
{E^{tot}}'_{\kappa_i} &= \sum_{j=1}^{m} a_{ij} \sum_{u=1}^{N} C_u \ \mathbf{E}\left[L_u \,|\mathbf{s}_j\right] V_{dd}{}^2_{u,i} \\
{\mathrm{P}_{SAT}}'_{\kappa_i} &= \sum_{u=1}^{N} a_{ij}\, \mathrm{P}_{SAT\,j}(\mathcal{V}_i)
\end{aligned}
$$

The two-level formulation of an SEM has two major advantages. First, the number of variables in the first level reduces from $N \times m$ to m where N is the number of tasks and m is the number of actions (states). Although the second level problem has N variables, it can be solved efficiently and globally since it is a DEM problem which does not contain a computationally expensive performance constraint. Furthermore, memoization can been used for ${E^{tot}}'$ and ${\mathrm{P}_{SAT}}'$ to speed up the solution of a two-level problem. Secondly, ${\mathrm{P}_{SAT}}'_{\kappa}$ (${\mathrm{P}_{SAT}}'_{\kappa_i}$) is smoother than $\mathrm{P}_{SAT}(\mathcal{V})$ ($\mathrm{P}_{SAT}(\mathcal{V}_i)$), because ${\mathrm{P}_{SAT}}'_{\kappa}$ (${\mathrm{P}_{SAT}}'_{\kappa_i}$) increases monotonically with κ (κ_i). This smoothness eliminates the inaccuracy due to the discontinuity of $\mathrm{P}_{SAT}(\mathcal{V}_i)$.

In this thesis, a two-level optimization problem is solved using *fmincon*, and the gradients of the objective and the constraint functions in the first stage are computed by finite differences. Each DEM problem in the second level is solved using geometric programming

described in Section 6.1. The only drawback of this solution technique is the fact that spatial correlations among task workloads are no longer exploited explicitly, since the variation of task workloads are taken into account using a single margin for each task. To incorporate correlations into a two-level formulation, one can use a unique margin coefficient for each task. However, this will destroy the smoothness of the chance constraints. In Chapter 9, the approximation-based, two-level solution technique is compared to the one described in the previous section for a single-action policy.

Chapter 7

DATA COLLECTION FOR STOCHASTIC APPLICATION MODELS

This chapter focuses on the collection of workload data which is used to construct a stochastic model for a given application. The following section provides a brief survey of workload characterization methods. In Section 7.2, the characterization method used in this thesis is described in detail. An extended workload model which can lead to more efficient energy management policies is motivated and discussed in Section 7.3.

7.1 Overview of Workload Characterization Methods

In the context of computing systems, workload refers to the amount of work required by an application (or a task). Workload characterization refers to a detailed examination of the execution of an application with sample input data on a computing system called the host machine. The workload of soft RT applications, e.g., multimedia applications, has been characterized in several earlier studies in which major points of interest were instruction frequencies, branch and loop statistics, potential for parallelism and memory behavior [29, 30, 31]. For this purpose, benchmarking suites which contain a set of synthetic or real-life applications and a set of representative inputs have been used. However, the input data provided in these benchmarking suites is usually quite small resulting in an inadequate representation of the statistical distribution of task workloads. For instance, for MPEG2 video decoding, input data in these suites contains at most tens of frames which correspond to few seconds of video. Therefore, it is necessary to use a relatively larger set of input data to characterize the statistical distribution of task workloads. In this sense, we define workload characterization as the compilation of workload distributions at task granularity in a given parallel application. We express the workload of a task in terms of clock cycles that is independent of the speed of the host machine. In the scope of energy management, we focus on the total number of clock cycles spent for and the memory access behavior of a given application. In the remaining of this section, we provide a brief overview of methods

which can be used for statistical workload characterization.

Workload characterization usually appears in the performance evaluation and analysis of hardware and software systems. In general, the goal of performance evaluation is to determine whether a design meets a set of specifications, while analysis refers to the identification of bottlenecks in a design. In the software community, these concepts are related to profiling the execution of an application for code optimizations, and the primary issues are resource utilization, memory behavior and parallelism. In the hardware community, performance evaluation and analysis are more related to design space exploration for new architectures where the primary issues are throughput and latency.

Tools used for performance evaluation and analysis can be mainly classified as simulation-based or profiling-based. Simulation-based approaches can be categorized as trace-driven, execution-driven or instruction set emulation. Trace-driven simulators, such as Dinero IV [32], are mostly used to simulate new memory systems. In such a simulator, the application is first executed on a host machine and the desired events occurred during this execution are stored in a time-stamped list called the trace file. Then, this trace file is fed into a memory system simulator. In an execution-driven simulator such as Valgrind [33], the two steps of the trace-driven simulation are interleaved. Events observed during the execution are directly sent to the memory simulator. Instruction set emulation is the most detailed approach in which both a memory system and a target processor are simulated on a host machine. As a result, the instruction set architecture of target and host machines are allowed to be different. A widely-used instruction set emulator is SimpleScalar [34] which offers simulation at various granularities and can emulate several instruction sets.

In a profiling-based approach, the desired information is collected during the execution of an application on a host machine. This information can be collected either by code instrumentation (i.e., application source code is annotated with special instructions) or by sampling (i.e., a profiler runs as a separate process and probes the target application periodically). Profiling-based approaches can either be implemented entirely in software such as gprof [35], or be hardware-assisted utilizing hardware performance counters. Hardware performance counters (HPC), explained further in the next section, are special registers in a processor that are dedicated to observe certain types of events occurred during execution.

Simulation-based approaches are usually more accurate, detailed and flexible, since sim-

ulation can provide full observability during execution, include timing information, and enable the user to isolate the interesting blocks or events in the system. Moreover, simulation allows performance evaluation of a hardware system that has not been built yet. However, simulation may require a large storage space and a significant amount of time, since it slows down the execution of an application significantly. This slow-down makes simulation time an important bottleneck for executing an application over a large set of inputs. Therefore, we adopt a profiling-based approach and use HPC's for statistical workload characterization.

7.2 Workload Characterization Using Hardware Performance Counters

7.2.1 Hardware Performance Counters

Hardware performance counters are model-specific registers located in a processing core. These counters increase the observability in a system at runtime for architecture performance evaluation or software optimization. HPC's can be programmed via another set of registers to count the occurrence of various events including branches, cache accesses and misses, and to count the number of instructions and clock cycles. Although HPC's are available in most of the modern processors, they are not standardized. As a result, the number of available counters and countable events vary from processor to processor. In this thesis, HPC's are accessed using the low-level interface of the Performance Application Programming Interface (PAPI) [36], which introduces very low overhead in terms of execution time and is easy to use.

Due to unavailability of a multicore system described in Section 3.1, the workload data is collected on a single-core system with an Intel Pentium M (Dothan) processor, which has two programmable HPC's and 512 MB of main memory [37]. During all executions, the processing core is operated at its maximum clock frequency (1.5 GHz). This processor has a cache system which includes an L1 instruction (32 KB) , an L1 data (32KB) and a unified L2 cache (2048 KB) on the chip. This host machine has Linux as the operating system (kernel v.2.6.16), and its kernel is patched with perfctr (v.2.6.21), which is a driver required by PAPI (v.3.2.1) to access HPC's.

7.2.2 Derivation of Statistical Measures for Stochastic Models

For workload characterization, the source code of a target application is first annotated with necessary function calls to record the total number of clock cycles (using the PAPI event called PAPI_TOT_CYC) spent during the execution of each task in the application. Data collection is repeated several times for each particular input to replace context-switched periods with non-context-switched ones. Because, context switches can cause a dramatic increase in the workload of a task by flushing the pipeline and cache, and they are specific to the operation system.

To obtain the joint workload distribution of tasks (p_{joint}), the annotated application is first executed over a set of representative inputs called the *training set*. The workloads of tasks in each period are stored as separate observations in a trace file denoted by Q. Then, every unique observation in Q is identified and stored in a sample space denoted by $\mathbf{L}_{space}$ in Section 4.1.2. The probability of each unique observation $(l_1, l_2, .., l_N)$ in $\mathbf{L}_{space}$ is approximated as follows

$$\mathrm{p}_{joint}(l_1, .., l_N) = \frac{\text{Number of periods } (l_1, .., l_N) \text{ observed in } Q}{|Q|} \tag{7.1}$$

in which $|Q|$ denotes the total number of observations in Q.

For stateful models, the application feature which determines the state of an observation, is also recorded in Q in addition to workload of tasks. Then, Q is partitioned into a set of disjoint subsets for each state $\mathbf{s}_i$

$$Q = \bigcup_{i=1}^{m} Q_i \tag{7.2}$$

in which Q_i contains observations indicated as $\mathbf{s}_i$ by the feature, and m is the total number of states. The conditional workload distribution in $\mathbf{s}_i$ is approximated as follows

$$\mathrm{p}_{joint|i}(l_1, .., l_N) = \frac{\text{Number of periods } (l_1, .., l_N) \text{ observed in } Q_i}{|Q_i|} \tag{7.3}$$

The occurrence frequency of each state is computed as follows

$$\pi_i = \frac{|Q_i|}{|Q|} \tag{7.4}$$

The state transition probability matrix $\mathbf{A}$ is also computed using the trace file in which each a_{ij} is approximated as follows

$$a_{ij} = \frac{\text{Number of times state changes from } i \text{ to } j}{|Q_i|} \tag{7.5}$$

7.3 Workload Decomposition into On-Chip and Off-Chip Components

The workload expressed in terms of clock cycles in the previous section is in fact a heterogeneous quantity, and DVS is available only for a subset executed on the processing core. Therefore, ignoring this heterogeneous nature of the application workload may lead to suboptimal energy management policies as shown by the following example.

On a computing system, the workload of a task typically contains mathematical computations, memory accesses and I/O operations, all of which correspond to a set of computation and memory operations. All computation operations take place inside the main processor chip (on-chip), while some of the memory operations occur outside of the processor chip (off-chip). In other words, some of the memory accesses may hit in the on-chip cache (i.e., target data is available in the on-chip memory), while some may miss the on-chip cache and require a data transfer between the on-chip cache and the off-chip memory. In this sense, the execution of a task over time on a typical processor can be depicted by Figure 7.1a.

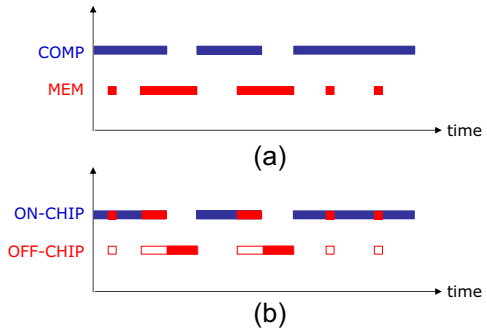


Figure 7.1: (a) Computation versus memory, (b) On-chip versus off-chip operations

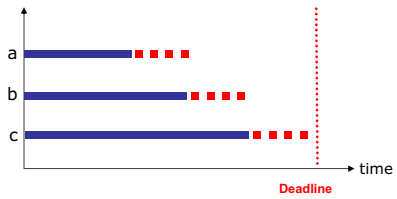


Figure 7.2: Using DVS on the decomposed workload

Modern processors employ various techniques, e.g., prefetching, buffering, speculative and out-of-order execution to reduce the amount of time the processor waits (stalls) for

off-chip accesses. However, such techniques can not fully overlap these on-chip and off-chip operations since an off-chip access lasts ten to hundred times more than an on-chip memory access or a computation does. Hence, the processor may still spend a non-negligible portion of the execution time for the off-chip memory operations. The execution flow in Figure 7.1a can be broken down into on-chip and off-chip parts as shown in Figure 7.1b. Figure 7.1b can also be represented by the line a in Figure 7.2 in which these on-chip and off-chip parts (denoted by the solid and dashed line segments respectively) are lumped into two components for simplicity.

Suppose that the line a in Figure 7.2 shows the execution at maximum voltage. If the heterogeneous nature of workload is ignored and the core speed is determined according to total execution time, this will result in a timing slack between the finishing time and the deadline as shown by line b in Figure 7.2 since the dashed component of execution time is fixed. On the other hand, if this execution time is broken down into on-chip and off-chip components, then DVS could be applied only for the solid component so that a more optimal decision can be achieved as shown by line c. This simple example shows that workload decomposition into on-chip and off-chip components may lead to more effective energy policies.

The heterogeneous nature of workload has been mostly ignored in previous studies on energy management, with two notable exceptions [38, 39]. In [38], authors propose a detailed workload model to classify applications as computation or memory-bounded and to predict maximum savings achievable with DVS in both cases. Although this model provides a valuable insight for the application workload, it is overcomplicated for statistical workload characterization of a parallel application. In [39], authors use a simpler workload model which decomposes workload into on-chip and off-chip components. They monitor these components for the workload in earlier periods at runtime via hardware performance counters and use it to predict the workload in the next period. In this thesis, we adopt the workload model in [39] and explain how this model is used in the following sections.

7.3.1 Decomposed Workload Model and Stochastic Models

The workload breakdown shown in Figure 7.1b can be approximated by assuming that every on-chip memory access overlaps with computation operations, while the processing core

stalls for every off-chip memory access. Although this assumption does not exactly reflect reality, it provides a more accurate view of the application workload for energy management compared to the homogeneous workload L_u. Based on this assumption, the decomposed workload model can be represented as a combination of the following two components:

- ${L_u}^{on}$: Clock cycles spent for the on-chip operations
- ${L_u}^{off}$: Time spent for the off-chip memory accesses

The first component ${L_u}^{on}$ is expressed in terms of processor clock cycles, because its execution time can be adjusted via DVS. The second component ${L_u}^{off}$ is expressed in seconds, because its execution time can not be scaled via DVS and depends on the underlying hardware. The execution time of a task u can be expressed in terms of ${L_u}^{on}$, ${L_u}^{off}$ and cycle-time as follows

$$t_u{}' = CT(V_{ddu})\ {L_u}^{on} + {L_u}^{off} \tag{7.6}$$

The decomposed workload model can be incorporated into the stochastic application models by replacing L_u by two new random variables ${L_u}^{on}$ and ${L_u}^{off}$. Consequently, the number of variables in each joint probability mass function is doubled. The expected dynamic energy consumed by a core to execute task u at V_{ddu} becomes

$$E_u^{dyn} = C_u\ \mathbf{E}\left[{L_u}^{on}\right] V_{ddu}^2 + C_u\ V_{ddu}\ \frac{(V_{ddu} - V_{th})^{a_{ct}}}{k_{ct}}\ \mathbf{E}\left[{L_u}^{off}\right] \tag{7.7}$$

If the core executing task u can be shut down during the time spent for ${L_u}^{off}$, then (7.7) reduces to

$$E_u^{dyn} = C_u\ \mathbf{E}\left[{L_u}^{on}\right] V_{ddu}^2 \tag{7.8}$$

Since both (7.7) and (7.8) are still convex functions of the voltage setting, the solution technique presented in Section 6.2.2 can be applied without further modifications. However, only (7.8) can be used in the second solution method presented in Section 6.2.3, since (7.7) can not be expressed as a posynomial.

7.3.2 Data Collection for the Decomposed Workload Model

The decomposed workload model is expressed in terms of PAPI events as follows:

$$
\begin{aligned}
{L_u}^{on} &= \text{PAPI\_TOT\_CYC} - C_2\ \text{PAPI\_L2\_TCM} \\
{L_u}^{off} &= CT(V_{ddmax})\ C_2\ \text{PAPI\_L2\_TCM}
\end{aligned}
$$

in which PAPI_TOT_CYC is the number of total cycles, PAPI_L2_TCM is the number of L2 cache misses, C_2 is the cost of an L2 cache miss in clock cycles, and $CT(V_{ddmax})$ is the cycle-time at maximum voltage used for conversion from clock cycles to time units. Since the host machine has an on-chip L2 cache and does not have an on-chip L3 cache, PAPI_L2_TCM is equal to the total number of off-chip cache accesses. To use this decomposed workload model, the value of C_2 has to be determined first.

To determine C_2, the resource stall counter (PAPI_RES_STL) is approximated as a measure of stalls due to L2 cache hits and misses as follows

$$\text{PAPI\_RES\_STL} = C_1\ (\text{PAPI\_L2\_TCA} - \text{PAPI\_L2\_TCM}) + C_2\ \text{PAPI\_L2\_TCM} \qquad (7.9)$$

in which PAPI_L2_TCA is the number of L2 cache accesses and C_1 is the cost of single L2 cache hit. C_1 and C_2 are found using a synthetic cache-unfriendly code segment shown in Figure 7.3.

```
1  for col 1 to N
2      do
3          for row 1 to 1000
4              do
5                  A[row][col] ← A[row − 1][col] + col
```

Figure 7.3: Code segment used to estimate the cost of a single L2 cache miss

Since a matrix is stored row-by-row in physical memory, iterating over rows for a column increases the probability of L1 and L2 cache misses. Moreover, the overlap of memory and computation operations is minimal, since each iteration depends on the result of the previous iteration. To increase accuracy further, resource stalls due to branch mispredictions are filtered out by loop unrolling which is a compiler option. Then, for a set of N values ranging from 50 to 350, this synthetic code is executed twice, since the host machine allows only two simultaneous counter measurements. PAPI also provides software multiplexing of

counters in a time-sliced manner that allows the user to count more than two events simultaneously. However, it has been shown that software multiplexing reduces the accuracy of measurements especially when the workload of a task gets lower [40]. In the first execution, L2 cache misses and resource stalls are counted. In the second execution, L2 cache misses and L2 cache accesses are counted. Since the variation in L2 cache accesses for a particular N value is found to be less than one percent, the measurements in the two passes in which the number of L2 misses are close to each other are combined. In this way, a total of 70 tuples of the form (PAPI_RES_STL, PAPI_L2_TCA, PAPI_L2_TCM) is obtained. Multiple linear regression is applied to these measurements which estimated a value of 10 cycles for C_1 and 82 cycles for C_2 with a mean absolute percentage error of 0.03. Since the estimated L2 cache hit latency matches processor specifications, it is concluded that the estimation of L2 miss latency is accurate [37]. Once C_2 is determined, the workload of an application is characterized by annotating its source code with necessary PAPI instructions to count only PAPI_L2_TCM and PAPI_TOT_CYC.

Chapter 8

CASE STUDY: MPEG2 VIDEO DECODING

This chapter studies the stochastic modeling of a popular multimedia application, MPEG2 video decoding. Section 8.1 provides a brief background on MPEG2 video decoding and its software implementation used in this thesis. Section 8.3 and Section 8.4 present the statistical characteristics of MPEG2 workload and describe four stateful stochastic application models for this application. Section 8.5 explores how the breakdown of workload into on-chip and off-chip components affect the statistical characteristics of MPEG2 workload.

8.1 Background

MPEG2 is a video compression standard in which a sequence of pictures is compressed into a sequence of data frames in a serial bitstream. In MPEG2 video decoding, the original pictures are recovered from this frame sequence in four main operations: variable length decoding (VLD), inverse quantization (IQ), inverse discrete cosine transform (IDCT) and motion compensation (MC).

Frames in an MPEG2 bitstream can be of three types: intra (I), predicted (P) and bidirectionally predicted (B) [41]. A P frame may contain references to previous I or P frames, while a B frame may contain references to previous or succeeding I or P frames. On the other hand, I frames are self-encoded, and consequently, a whole picture can be decoded from its own data. A sequence of frames between two successive I frames is called a group of pictures. Depending on the frame resolution, each frame is composed of a number of horizontal slices.

In this thesis, we used the slice-based parallel task decomposition of the MPEG2 video decoding application proposed in [41]. This decomposition is found to yield the best performance in terms of load balancing and data synchronization overhead among tasks. In this decomposition, VLD and IQ operations are lumped into a single task (denoted by VLD), while IDCT and MC operations are lumped into another task (denoted by MC) for each

slice. The task graph representing this decomposition and its assignment to four processing cores (denoted by C_1, C_2, C_3, C_4) is shown in Figure 8.1. In this figure, numbers within ovals correspond to slice numbers, and the dashed arrows represent precedence relations between tasks assigned to the same core. In this decomposition, the execution of each MC task has to wait for the completion of the VLD task of the same slice, while the MC and VLD tasks of separate slices can be executed concurrently as long as there are available cores.

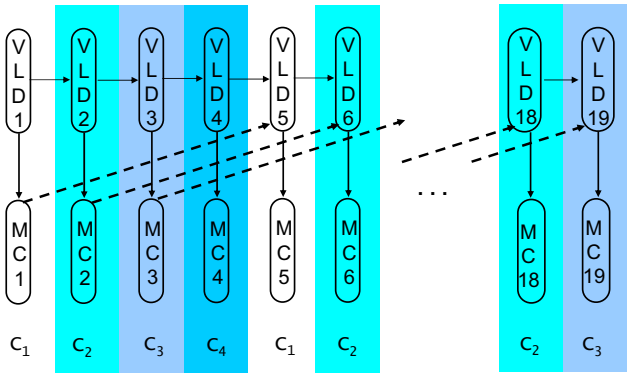


Figure 8.1: The slice-based task decomposition of MPEG2 video decoding for a frame resolution of 544×304

8.2 Input Data Set

In this thesis, two movies are (Girl With a Pearl Earring and Spiderman 2) are used as input data. Six 30-minute clips are obtained from these two movies and encoded using ffmpeg [42] with the parameters given in Table 8.1. The first three clips are extracted from the first movie, while the last three clips are extracted from the second movie. For workload characterization and stochastic modeling of MPEG2 video decoding, only two of the six clips (the 1^{st} and the 4^{th} clips) are used as the training set (defined in Section 7.2). In other words, workload distributions and associated statistical measures are obtained from the selected two clips. All energy minimization policies are also optimized using this training set. The remaining four clips are intentionally left out to determine how accurately stochastic application models can predict the workload in unknown movies.

Table 8.1: MPEG2 video encoder parameters

Frame rate	25
Frame resolution	544 × 304
Bitrate	600 kbps
Number of Slices	19
I frame distance	12
I/P frame distance	2

8.3 Stochastic Modeling of MPEG2 Workload

This section presents the MPEG2 workload statistics extracted from the training set. The training set is divided into six 10-minute pieces in some of the figures to demonstrate that the statistical properties do not only hold for the whole training set but also for its smaller subsets. In such figures, the solid line represents the whole training set, while dotted lines represent its subsets.

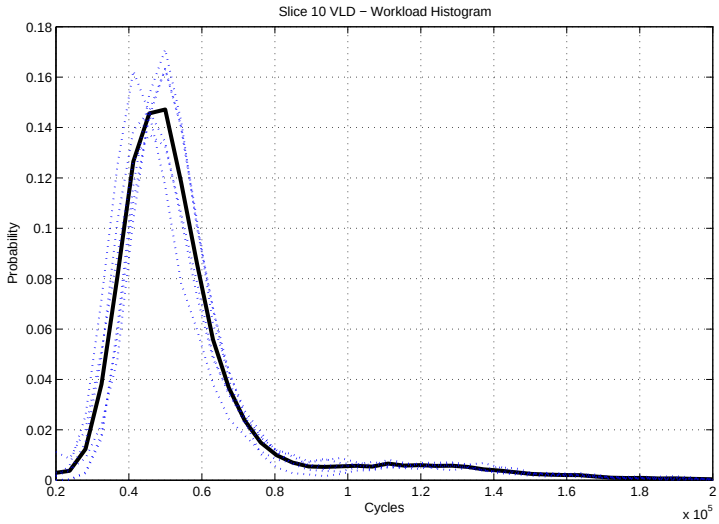


Figure 8.2: Workload histogram of the VLD task of Slice 10

Figures 8.2 and 8.3 show the workload distribution of the VLD and MC tasks of the slice in the center of each frame. The VLD and MC tasks of other slices also have similar workload distributions. In both figures, there is a significant difference among the maximum, average and the minimum workloads. For instance, the ratio of maximum to minimum and

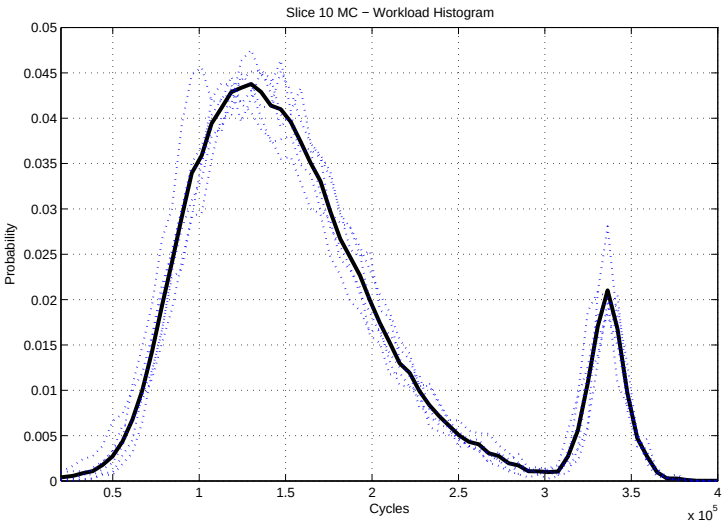


Figure 8.3: Workload histogram of the MC task of Slice 10

maximum to average workloads for the VLD task are around 18 and 5 respectively. For the MC task, these ratios are around 60 and 2.5. This large variation of task workloads indicates a significant potential in saving energy by slowing down the execution of tasks using DVS. This fact also applies to the 10-minute subsets of the training set, since the workload distributions of both tasks in these subsets are similar to the aggregate distribution.

Figure 8.4 shows the completion time distribution of the task graph in which tasks are executed at maximum voltage. In Figure 8.4, there is a significant difference among the maximum, average and the minimum completion times in which the ratio of maximum to minimum and maximum to mean are around 10 and 3 respectively. This difference supports the same conclusion derived previously for Figures 8.2 and 8.3. Moreover, the maximum completion time in Figure 8.4 is about 50% lower than the uncorrelated completion time computed by a topological traversal of the task graph in which each task has its individual worst-case workload observed in the training set. This difference means that tasks never have their individual worst-case workloads in the same period. This fact shows that ignoring spatial correlations for MPEG2 video decoding may result in suboptimal energy management policies, even for hard timing constraints.

Another important observation in Figures 8.2, 8.3 and 8.4 is the humps in the distributions. If these humps originate from an innate feature of the application, then such a feature can be used to obtain a stateful stochastic model. In the next section, several features of

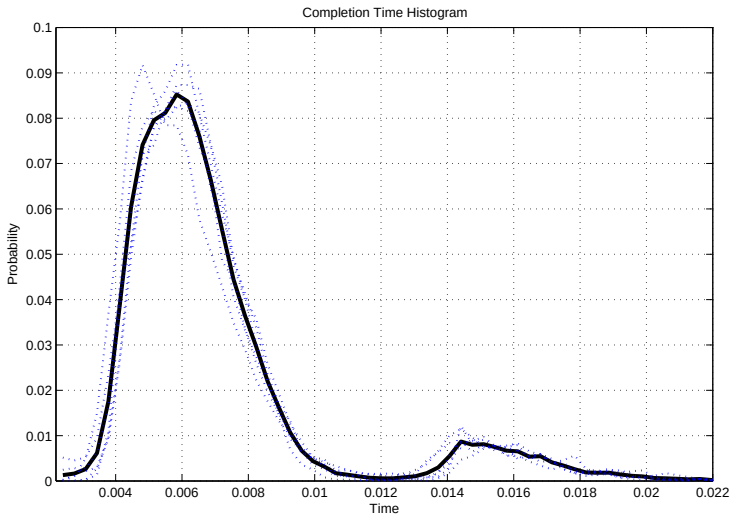


Figure 8.4: The completion time histogram

MPEG2 video decoding are explored to construct stateful stochastic models.

Finally, Figure 8.5 shows the correlation coefficients of the workloads of the VLD and MC tasks of the center slice with the workloads of other VLD and MC tasks as a function of the distance of the slices from the center slice. In this figure, the dotted lines show data obtained from a randomly selected subset of the training set, whereas the solid lines represent aggregate statistics. This figure shows that the tasks of slices close to each other are highly correlated. Moreover, the MC and VLD tasks of a particular slice are also highly correlated with each other.

8.4 Stateful Stochastic Modeling of MPEG2 Workload

In this section, two current-state observable and two previous-state observable models are proposed for MPEG2 video decoding. All models are constructed using the workload data in the training set. In these models, states are defined according to the following features which have been previously used to predict workload at runtime [3, 7, 43]:

- Type: The type of encoding used for a frame (I, P or B)
- Size: The disk space occupied by a frame in the bitstream.
- Load: The summation of task workloads in a frame.

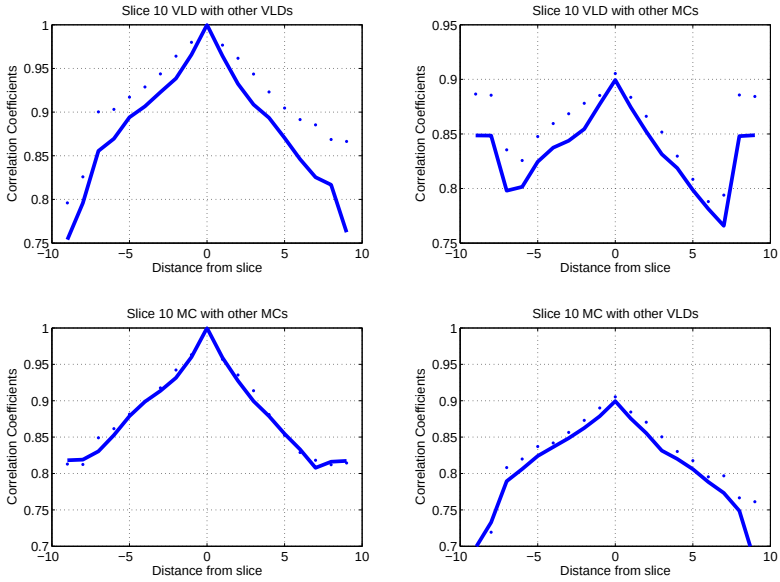


Figure 8.5: Spatial correlations between task workloads

The type information is always known before a frame is decoded. The task decomposition in [41] makes the size information available before a frame is decoded, since the bitstream is prescanned to parallelize tasks. Therefore, type and size features can be used to obtain current-state observable models. On the other hand, the load information for a frame becomes available after it is decoded. Thus, the load feature can be used in previous-state observable models.

8.4.1 Current-State Observable Models

The first current-state observable model denoted by β^T_{CSO} uses only frame type information to define states which results in the following state frequencies

$$\pi_I = 0.09 \quad \pi_P = 0.24 \quad \pi_B = 0.67$$

Figure 8.6 shows the distribution of the completion time at maximum voltage in states and in aggregate data. This figure shows that workload distributions in states (denoted by $\mathrm{p}_{joint|i}$ in the previous chapters) have considerably different means and variances. In other words, defining states according to frame type results in dissimilar workload distributions in states. Moreover, this model can capture the hump on the right as a distinct state shown by the solid line in Figure 8.6.

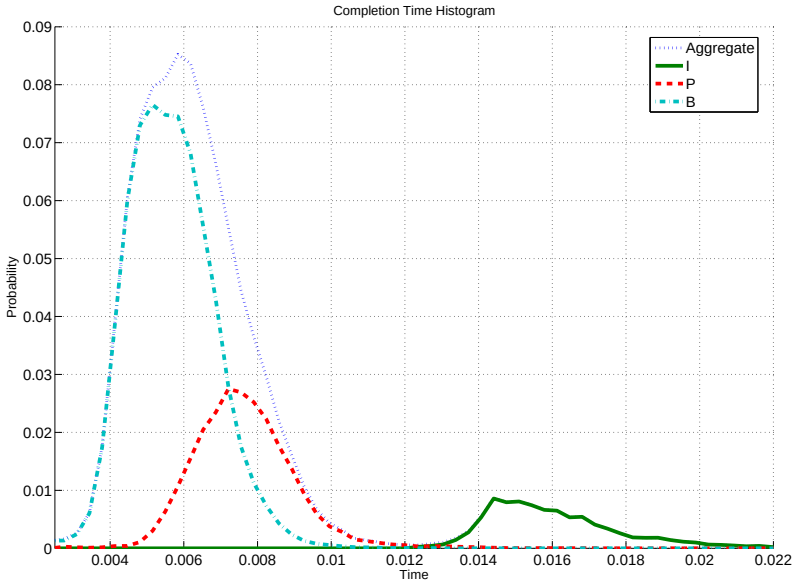


Figure 8.6: The completion time histogram of the current-state observable model with 3 states

The second current-state observable model denoted by β_{CSO}^{TS} uses both frame type and size information. For this purpose, the frame size observed for each frame type in the training set is quantized into 3 bins resulting in a total of 9 states. The three bins used for a particular type, e.g., $I-1$, $I-2$, and $I-3$, contain equal number of frames. The frequencies of states are found to be

$$\pi_{I-1} = 0.0297 \quad \pi_{I-2} = 0.0297 \quad \pi_{I-3} = 0.0297$$
$$\pi_{P-1} = 0.0814 \quad \pi_{P-2} = 0.0814 \quad \pi_{P-3} = 0.0814$$
$$\pi_{B-1} = 0.2222 \quad \pi_{B-2} = 0.2222 \quad \pi_{B-3} = 0.2222$$

Figure 8.7 shows the distribution of the completion time in states and in aggregate data. This figure shows that defining states according to both frame type and size also results in dissimilar workload distributions in states. Therefore, it is expected that the policies optimized using one of these current-state observable models will be more effective in saving energy than a single-action policy as discussed in Section 4.2.2.

8.4.2 Previous-State Observable Models

The first previous-state observable model denoted by β_{PSO}^{L} uses only the load information. For this purpose, the total frame workload observed in the training set is quantized into 3

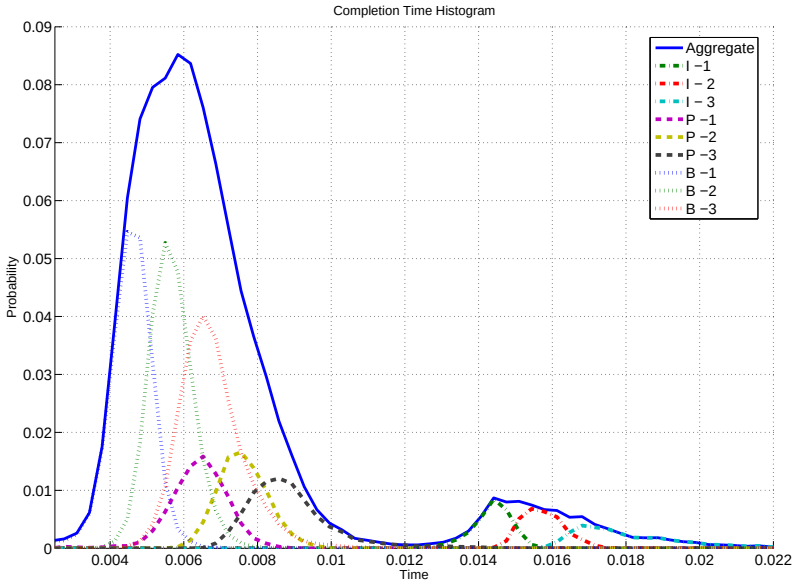


Figure 8.7: The completion time histogram of the current-state observable model with 9 states

bins with equal number of frames, which results in 3 states with equal limiting probabilities. The transition probability matrix of this model is shown in Figure 8.8.

	L1	L2	L3
L1	0.46	0.17	0.37
L2	0.23	0.38	0.39
L3	0.31	0.45	0.24

Figure 8.8: State transition matrix for the previous-state observable model with 3 states

The second previous-state observable model denoted by β_{PSO}^{LT} uses both frame type and frame load information. For this purpose, the total workload observed for each frame type in the training set is quantized into 3 bins with equal number of frames resulting in a total of 9 states. The transition probability matrix of this model is shown in Figure 8.9. Since each I and P frames are succeeded by B frames in the bitstream, the upper left 6-by-6 corner of this matrix is zero. This transition matrix leads to the same state frequencies in β_{CSO}^{TS}.

Figures 8.10 and 8.11 show the completion time distributions at maximum voltage for the previous-state observable models. In both figures, the top histograms represent the distribution in each state i obtained from $\mathrm{p}_{joint|i}$, while the bottom histograms show the

	I - L1	I - L2	I - L3	P - L1	P - L2	P - L3	B - L1	B - L2	B - L3
I - L1	0	0	0	0	0	0	0.22	0.22	0.56
I - L2	0	0	0	0	0	0	0.08	0.22	0.70
I - L3	0	0	0	0	0	0	0.16	0.33	0.51
P - L1	0	0	0	0	0	0	0.66	0.29	0.05
P - L2	0	0	0	0	0	0	0.28	0.46	0.26
P - L3	0	0	0	0	0	0	0.17	0.32	0.51
B - L1	0.04	0.04	0.07	0.21	0.10	0.07	0.43	0.04	0
B - L2	0.05	0.05	0.04	0.11	0.14	0.12	0.10	0.34	0.05
B - L3	0.05	0.04	0.02	0.05	0.13	0.18	0	0.12	0.41

Figure 8.9: State transition matrix for the previous-state observable model with 9 states

distribution experienced by each action i obtained from p^{i}_{joint}. Both $\mathrm{p}_{joint|i}$ and p^{i}_{joint} have been defined in Section 4.2. For example, $L1$ in Figure 8.10a represents the workload distribution in state $\mathbf{s}_1$ which corresponds to the first load bin, while $ACTION1(L1)$ in Figure 8.10b represents the distribution experienced by $\mathcal{V}_i$. Each action $\mathcal{V}_i$ in β^{L}_{PSO} or β^{LT}_{PSO} will be optimized using the corresponding distribution in Figure 8.10b or 8.11b.

Figures 8.10a and 8.11a show that defining states according to the load and type features results in dissimilar workload distributions in states. However, this dissimilarity is blurred due to the conditioning on the reachable states in a single transition, as expressed in 4.3. Consequently, the distributions in Figures 8.10b and 8.11b are quite similar, and the underlying models are unable to capture the humps. These observations indicate that the temporal correlation between the loads of consecutive frames is low which is due to the sudden fluctuations in task workloads at runtime. In other words, the load of the previous frame does not convey valuable information for the total workload in the current frame to be decoded. Therefore, it is expected that the policies optimized using previous-state observable models (β^{L}_{PSO} and β^{LT}_{PSO}) will be less effective in saving energy than β^{T}_{CSO} or β^{TS}_{CSO}.

8.5 On-Chip / Off-Chip Workload Decomposition

This section presents the statistics related with the breakdown of workload (L_u) into on-chip ($L_u{}^{on}$) and off-chip ($L_u{}^{off}$) components. As described in Section 7.3, L_u is measured by the counting the number of clock cycles spent during the execution of a task. $L_u{}^{off}$ is approximated by multiplying the total number of L2 cache misses with the cost of a single cache miss. All statistics are extracted from the training set.

Table 8.2 provides the average values of total cycles (PAPI_TOT_CYC), L2 cache misses (PAPI_L2_TCM) and the ratio of off-chip to on-chip workload ($L_u{}^{off}/L_u{}^{on}$) in cycles. The first three rows show the mean values for various tasks, while the last four rows show the mean values for various frame types. Table 8.2 shows that off-chip workload constitutes approximately 10% of the total workload of a task or a frame. Although an I frame has higher workload on the average, its off-chip workload is less than other frames, since I frames do not include references to other frames as explained in Section 8.1.

Table 8.2: On-Chip / Off-Chip Decomposition of the MPEG2 Workload

	Total Cycles	**L2 Misses**	$L_u{}^{off}/L_u{}^{on}$ **(in cycles)**
All tasks	92k	86	0.08
VLD	49k	83	0.16
MC	135k	88	0.06
All frames	3.5M	3280	0.08
I	8.3M	2127	0.02
P	4.0M	3314	0.07
B	2.7M	3420	0.12

Figures 8.12 and 8.13 show the decomposed workload distribution of the VLD and MC tasks of the slice in the center of each frame. In both figures, the dashed and the solid lines in the upper plots represent the on-chip and the total workload, while the solid line in the bottom plot represents the off-chip workload. Both figures demonstrate that factoring out the off-chip component from the total workload has a minimal effect on the distribution of on-chip workload. This means that the workload variability primarily originates from the

on-chip operations in MPEG2 video decoding.

Finally, Figure 8.14 shows the correlation coefficients of the workloads of the VLD and MC tasks of the center slice with the workloads of other VLD and MC tasks as a function of the distance of the slices from the center slice. In this figure, the dotted line represents the statistics obtained from the on-chip workload, while the solid line represents the statistics obtained from total workload. It is observed that spatial correlations among the on-chip workloads of tasks also do not change after decomposition. Therefore, we conclude that the workload decomposition of MPEG2 video decoding has a small effect on the statistical nature of the on-chip workload. This conclusion heavily depends on the application software and the hardware used in this thesis, but it is consistent with the literature [44]. We evaluate the effect of workload decomposition on the performance of an energy management policy in Chapter 9.

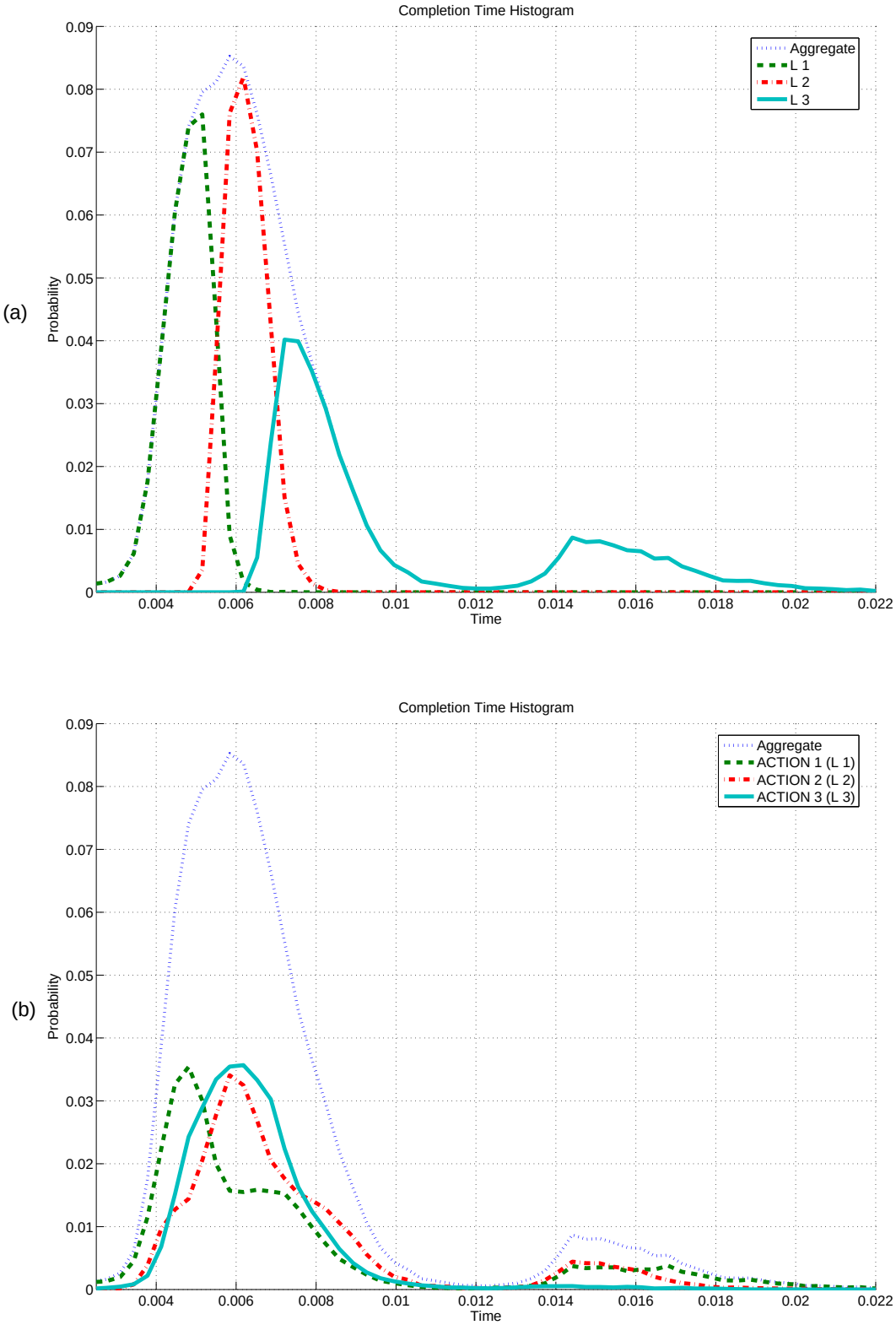


Figure 8.10: The completion time histograms a) in the states, b) experienced by the actions of the previous-state observable model with 3 states

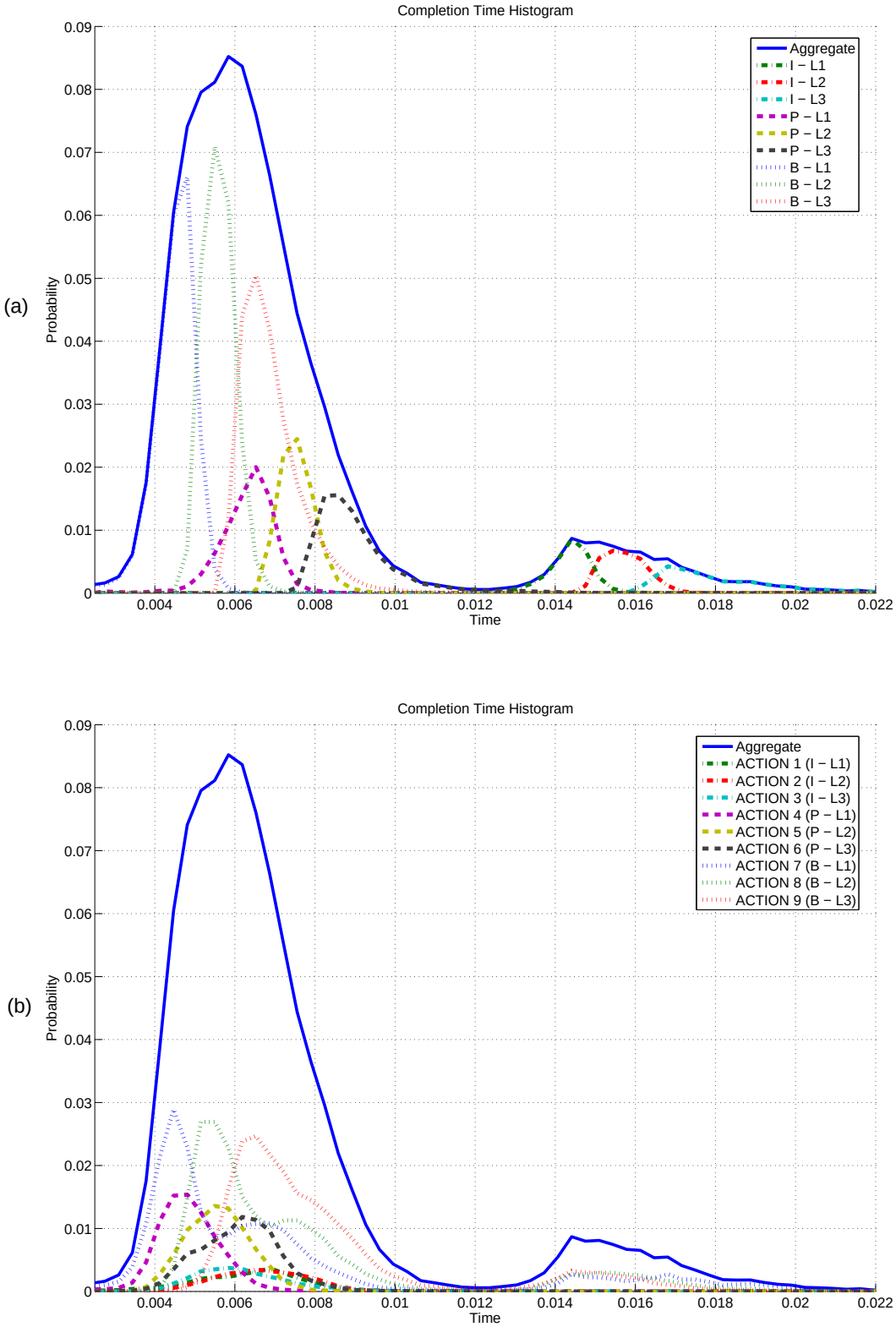


Figure 8.11: The completion time histograms a) in the states, b) experienced by the actions of the previous-state observable model with 9 states

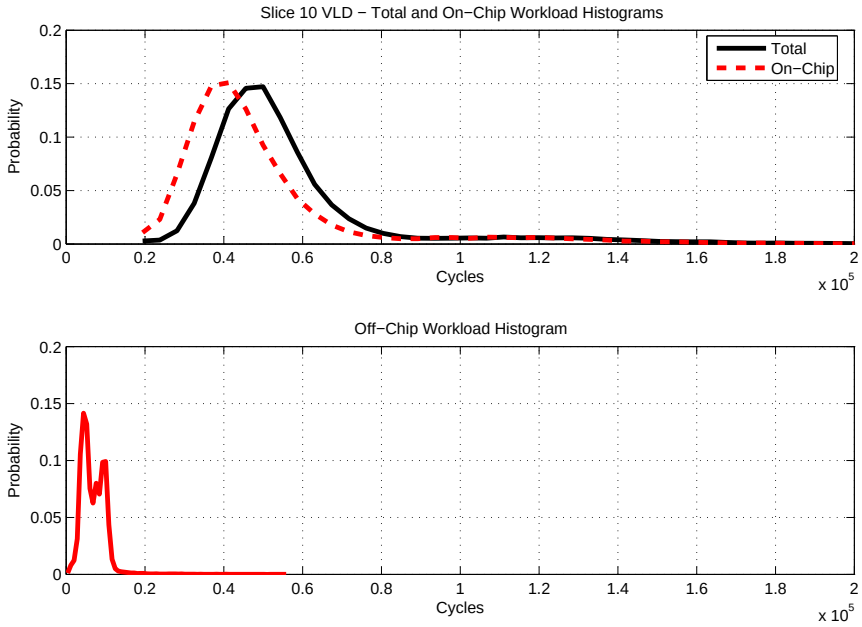


Figure 8.12: The decomposed workload histogram of the VLD task of Slice 10

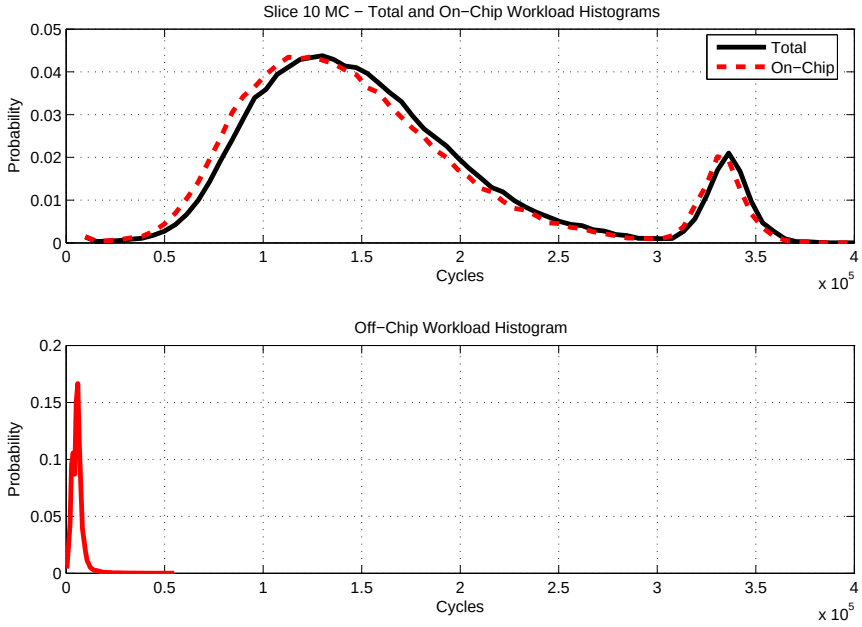


Figure 8.13: The decomposed workload histogram of the MC task of Slice 10

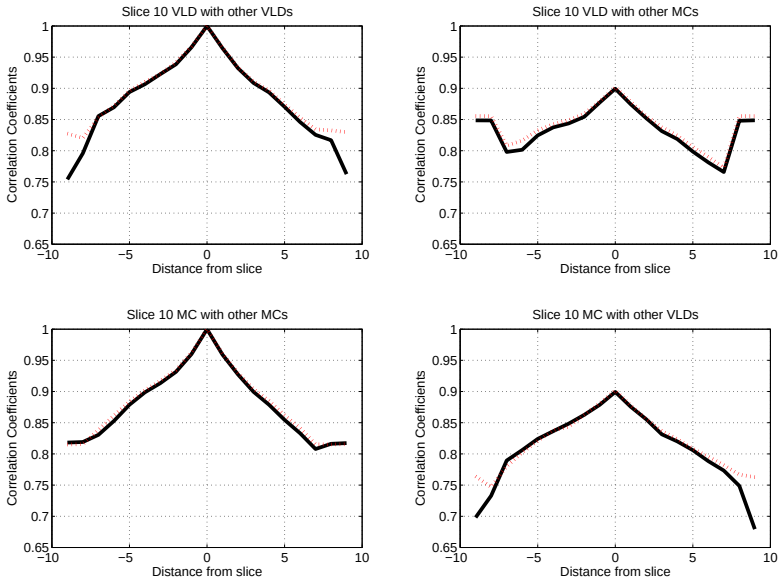


Figure 8.14: Spatial correlations between the on-chip components of task workloads

Chapter 9

EXPERIMENTS

In this chapter, the results of experiments conducted to evaluate the effectiveness of the proposed energy management policies are reported. Each experiment is performed using the MPEG2 video decoding application and the input data set described in Chapter 8. Experiments are organized in the following four sections: In Section 9.1, the single-action energy management policy and its runtime heuristic are compared to two other approaches proposed in the literature. In Section 9.2, multi-action energy management policies proposed in Section 8.4 are compared to the single-action policy. The two solution methods discussed in Section 6.2 are compared in Section 9.3. Finally, the effect of on-chip and off-chip workload decomposition on an energy management policy is studied in Section 9.4.

The following technicalities need to be explained before the results can be interpreted properly:

- All energy management policies are optimized for the following set of probabilistic performance constraints $\{0.99, 0.97, 0.95, 0.90, 0.85, 0.80\}$. In this set, $P_{CON} = 0.99$ is chosen to model a hard timing constraint, while other completion probabilities reflect different choices for the deadline miss tolerance. Since video clips are encoded at a frame rate of 25, the minimum performance constraint ($P_{CON} = 0.80$) results in a frame display rate of 20 that is still acceptable for watchability.

- All energy management schemes are optimized using stochastic models compiled from the workload data collected from the first and the fourth clips listed in each table. In other words, these two clips constitute the training set. The other four clips are intentionally left out to observe how accurately energy management policies predict the statistical workload of an unknown movie and achieve the desired completion probability.

- The energy consumption values in Tables 9.2, 9.4 and 9.5 represent the total en-

ergy consumption and are computed over the periods in which the timing constraints are satisfied by the given scheme. For each clip and the probabilistic performance constraint, the energy consumptions of the compared schemes are normalized to the theoretical lower bound achieved by β_{BND}. β_{BND}, as described in Section 3.3, uses a unique action optimized for each particular period. For each performance constraint, β_{BND} meets the timing constraints for a subset of frames in a clip which results in the minimum energy consumption. This policy is not implementable in reality since the knowledge of task workloads is not available before a frame is decoded.

- Two separate global deadlines on the completion time are used. The first deadline is called the *relaxed deadline*. This deadline reflects the case in which all tasks can be completed at V_{ddmax} even if each task has its individual worst-case workload in the training set. In other words, the relaxed deadline assumes that task workloads are distributed independently. The second deadline is called the *nominal deadline* which is the maximum completion time observed in the training set when tasks are executed at V_{ddmax}. This deadline is approximately 50% lower than the relaxed deadline and reflects a more realistic scenario, since tasks do not have their individual worst-case workload in the same period.

- The energy management schemes compared in the following first three sections are trained and simulated using the basic workload model which does not consider the computation/memory breakdown.

- In each table, the last row presents the average results over all clips.

9.1 Comparison with Previous Approaches

In this section, the following four energy management schemes are compared:

- **I** is the offline energy management scheme proposed in [14] for hard RT applications on multicore systems. This scheme uses a single action found by solving a hard energy minimization (HEM) problem described in Section 3.3 in which the individual worst-case task workloads are used. This energy management scheme assumes that the distribution of task workloads are independent.

- **II** is the hybrid energy management scheme proposed in [15] for soft RT applications on multicore systems. The offline phase of this scheme consists of two stages. The first stage computes a set of minimum workloads for tasks which satisfy the probabilistic performance constraint when tasks are executed at maximum voltage. In the second phase, the available execution time is allocated to the minimum workloads by adjusting the voltage settings of tasks. Using these allocated execution times, a latest completion time is computed for each task. At runtime, the voltage setting of each task is adjusted to finish its minimum workload until the precomputed latest finishing time.

- β_{OFF} is the single-action energy management policy described in Section 4.1.2 which captures the workload variability and the spatial correlations among task workloads. This policy finds a voltage setting for each task by formulating and solving a stochastic optimization problem. The optimized set of voltage settings are used in each period. The underlying optimization problem is solved using the method described in Section 6.2.2.

- β_{ONL} is the energy management policy which uses the voltage settings of β_{OFF} and makes runtime adjustments to reclaim timing slacks as described in Section 5.2.2.

We compare our energy management schemes with **I**, which was proposed for applications with hard timing constraints, to reveal how much energy savings can be achieved when the application has some tolerance to deadline misses. We compare our schemes with **II**, since it addresses the same problem studied in this thesis. There are two key differences between β_{OFF} and **II**: First, **II** ignores the spatial correlations among task workloads. Second, **II** is based on greedy heuristics. We do not compare the multi-action policies based on stateful models with previous approaches for a fair comparison.

All energy management schemes are trained and simulated with the relaxed deadline due to the fact that **I** and **II** fail to work with the nominal deadline. Since **I** and **II** assume the statistical independence of task workloads, they require that the deadline should be greater than or equal to the relaxed deadline.

Table 9.1 presents the completion probability achieved by **II**, β_{OFF} and β_{ONL}. **I** is not included in this table since it achieves full completion. Table 9.1 shows that **II** behaves conservatively and overshoots the completion probability by 2% to 18% for $P_{CON} < 0.99$.

Table 9.1: Comparison of the completion probability of the single-action policy with other approaches

	$P_{CON} = 0.99$			$P_{CON} = 0.97$			$P_{CON} = 0.95$		
Clip No:	**II**	β_{OFF}	β_{ONL}	**II**	β_{OFF}	β_{ONL}	**II**	β_{OFF}	β_{ONL}
1	0.99	0.99	0.99	0.98	0.97	0.97	0.98	0.95	0.95
2	0.99	0.99	0.99	0.99	0.96	0.96	0.98	0.94	0.94
3	0.99	0.99	0.99	0.99	0.97	0.97	0.98	0.95	0.95
4	0.99	0.99	0.99	0.99	0.97	0.97	0.98	0.95	0.95
5	0.99	0.99	0.99	0.99	0.98	0.98	0.99	0.96	0.96
6	0.99	0.99	0.99	0.99	0.97	0.97	0.98	0.95	0.95
AVG	0.99	0.99	0.99	0.99	0.97	0.97	0.98	0.95	0.95
	$P_{CON} = 0.90$			$P_{CON} = 0.85$			$P_{CON} = 0.80$		
1	0.97	0.91	0.91	0.96	0.86	0.86	0.95	0.80	0.80
2	0.97	0.91	0.90	0.96	0.86	0.86	0.95	0.81	0.81
3	0.97	0.90	0.90	0.96	0.85	0.85	0.94	0.80	0.80
4	0.97	0.89	0.89	0.96	0.84	0.84	0.95	0.80	0.80
5	0.98	0.89	0.89	0.96	0.84	0.84	0.93	0.79	0.82
6	0.97	0.90	0.90	0.97	0.84	0.84	0.96	0.79	0.79
AVG	0.97	0.90	0.90	0.96	0.85	0.85	0.95	0.80	0.80

Table 9.2: Comparison of the energy consumption of the single-action policy with previous approaches

	$P_{CON} = 0.99$				$P_{CON} = 0.97$				$P_{CON} = 0.95$			
Clip No:	**I**	**II**	β_{OFF}	β_{ONL}	**I**	**II**	β_{OFF}	β_{ONL}	**I**	**II**	β_{OFF}	β_{ONL}
1	175	109	138	108	177	115	129	108	178	122	127	109
2	175	110	139	109	174	116	127	107	176	122	126	108
3	176	109	139	109	177	116	129	108	179	122	128	109
4	175	109	139	109	178	116	129	109	181	122	129	111
5	177	110	140	110	181	116	131	112	182	123	129	112
6	175	109	139	109	177	115	129	109	179	122	128	110
AVG	176	109	139	109	177	116	129	109	179	122	128	110
	$P_{CON} = 0.90$				$P_{CON} = 0.85$				$P_{CON} = 0.80$			
1	187	138	114	108	189	148	110	107	188	159	107	105
2	186	138	114	107	189	147	110	108	190	159	108	107
3	186	137	113	107	187	148	108	106	190	157	107	106
4	183	140	111	105	185	151	107	105	189	164	107	105
5	182	141	111	105	184	149	106	104	185	149	104	102
6	183	139	112	106	183	152	107	104	185	164	105	103
AVG	185	139	113	106	186	149	108	106	188	159	106	105

On the other hand, β_{OFF} and β_{ONL} achieve the desired P_{CON} with a maximum percentage error of 2.5 for individual clips. This result shows that our stateless stochastic model captures accurately the workloads of clips which are not included in the training set, while ignoring correlations leads to pessimistic completion. Table 9.1 also shows that β_{ONL} is able to preserve the completion probability achieved by β_{OFF} not only for the two clips in the training set, but also for the remaining four clips.

Table 9.2 presents the total energy consumption achieved by all of the listed energy management schemes. The energy consumptions of β_{OFF} and β_{ONL} are considerably lower

than the energy consumption of **I**. It is noteworthy that β_{ONL} consumes 60% less energy than **I** for $P_{CON} = 0.99$, since β_{ONL} considers the spatial correlations. This shows that even a 1% tolerance to deadline misses results in a significant potential for energy savings and our policies are able to exploit this potential. β_{ONL} consumes 6% to 50% less energy than **II**, since the latter scheme behaves conservatively and overaccomplishes the completion probability constraint. Even β_{OFF}, which uses a fixed voltage setting for each task, consumes less energy than **II** does for $P_{CON} < 0.95$. It is also remarkable that the energy consumption of β_{ONL} is within 10% of the theoretical bound β_{BND}.

9.2 Comparison of Single-Action and Multi-Action Policies

In this section, β_{OFF} and β_{ONL} are compared to the following four multi-action energy management policies based on stateful stochastic application models:

- β^{L}_{PSO} is based on a previous-state observable model with 3 states which are defined according to frame load.

- β^{LT}_{PSO} is based on a previous-state observable model with 9 states which are defined according to both frame type and frame load.

- β^{T}_{CSO} is based on a current-state observable model with 3 states which are defined according to frame type.

- β^{TS}_{CSO} is based on a current-state observable model with 9 states which are defined according to both frame type and frame size.

The compared energy management schemes are trained and simulated with the nominal deadline.

Table 9.3 presents the completion probability achieved by each policy. It is seen that all of the policies achieve the desired P_{CON} at the nominal deadline with a maximum percentage error of 2.5 for an individual clip. Thus, all of the stochastic models predict the workload of unknown clips accurately.

Table 9.4 presents the energy consumption of the policies. Note that the energy consumption of β_{OFF} and β_{ONL} have increased when compared to Table 9.2, since the deadline

is tighter in this case. The following conclusions can be derived from the results shown in Table 9.4:

- The energy consumption of each multi-action policy is less than or close to the energy consumption of the single-action policy β_{OFF} with a few exceptions. The policies based on the previous-state (current-state) observable models save up to 10% (40%) more energy than β_{OFF}. This shows that stateful stochastic application models result in more effective policies which can exploit temporal correlations.

- The policies based on the current-state observable models (i.e., β^{T}_{CSO} and β^{TS}_{CSO}) are more effective than the policies based on the previous-state observable models (i.e., β^{L}_{PSO} and β^{LT}_{PSO}). This result is expected since the workload distributions experienced by the actions of β^{T}_{CSO} and β^{TS}_{CSO} were found to be dissimilar in their means and variances as shown in Figures 8.6 and 8.7. On the other hand, state transition probabilities have blurred the dissimilarity between the workload distributions experienced by the actions of β^{L}_{PSO} and β^{LT}_{PSO} as shown in Figures 8.10 and 8.11.

- The energy consumption of β^{LT}_{PSO} is within 3% of the energy consumption of β^{L}_{PSO} for $P_{CON} \leq 0.97$. This observation shows that using the frame type information in addition to frame load information does not result in higher energy savings.

- The energy consumption of β^{TS}_{CSO} is within 4% of the energy consumption of β^{T}_{CSO} except $P_{CON} = 0.90$. This observation shows that using the frame size information in addition to frame type information does not result in higher energy savings.

- At lower completion probabilities, β^{T}_{CSO} and β^{TS}_{CSO} consume less energy than β_{ONL} does. It is notable that the energy consumptions of β^{T}_{CSO} and β^{TS}_{CSO} are within 5% of the theoretical bound β_{BND}. Therefore, it is concluded that β^{T}_{CSO} and β^{TS}_{CSO} do not even require runtime adjustments for $P_{CON} < 0.95$. It is still possible to extend multi-action policies with simple runtime adjustments to exploit the timing slacks occurred at runtime similar to the one proposed in Section 5.2.2.

- When compared to the results in Table 9.2, β_{ONL} improves more upon β_{OFF} in terms of energy savings at the nominal deadline. This improvement originates either from

workload distributions or from the relationship between the cycle-time and the voltage setting expressed in (3.2). Because, the derivative of the cycle-time with respect to the voltage setting decreases as the voltage setting decreases.

9.3 Comparison of the Solution Methods for the Stochastic Energy Minimization Problem

In this section, the two approximation-based methods used to solve a stochastic energy minimization problem are compared for a single-action policy at the nominal deadline. The key difference between these solution methods is the fact that the former technique exploits spatial correlations explicitly.

Table 9.3: Comparison of the completion probability of the single-action and multi-action policies

	$P_{CON} = 0.99$						$P_{CON} = 0.97$						$P_{CON} = 0.95$					
Clip No:	β_{OFF}	β_{ONL}	β^{L}_{PSO}	β^{LT}_{PSO}	β^{T}_{CSO}	β^{TS}_{CSO}	β_{OFF}	β_{ONL}	β^{L}_{PSO}	β^{LT}_{PSO}	β^{T}_{CSO}	β^{TS}_{CSO}	β_{OFF}	β_{ONL}	β^{L}_{PSO}	β^{LT}_{PSO}	β^{T}_{CSO}	β^{TS}_{CSO}
1	0.99	0.99	0.99	0.99	0.99	0.99	0.97	0.97	0.97	0.96	0.97	0.97	0.95	0.95	0.95	0.95	0.95	0.95
2	0.99	0.99	0.99	0.99	0.99	0.99	0.97	0.97	0.96	0.96	0.97	0.97	0.95	0.95	0.95	0.95	0.95	0.95
3	0.99	0.99	0.99	0.99	0.99	0.99	0.97	0.97	0.97	0.96	0.97	0.97	0.95	0.95	0.95	0.95	0.95	0.95
4	0.99	0.99	0.99	0.99	0.99	0.99	0.97	0.97	0.97	0.97	0.97	0.97	0.95	0.95	0.95	0.95	0.95	0.95
5	0.99	0.99	0.99	0.99	0.99	0.99	0.98	0.98	0.97	0.97	0.97	0.97	0.96	0.96	0.94	0.94	0.95	0.96
6	0.99	0.99	0.99	0.99	0.99	0.99	0.97	0.97	0.97	0.97	0.97	0.97	0.95	0.95	0.94	0.94	0.95	0.95
AVG	0.99	0.99	0.99	0.99	0.99	0.99	0.97	0.97	0.97	0.97	0.97	0.97	0.95	0.95	0.95	0.95	0.95	0.95
	$P_{CON} = 0.90$						$P_{CON} = 0.85$						$P_{CON} = 0.80$					
1	0.91	0.91	0.91	0.91	0.91	0.90	0.86	0.86	0.86	0.86	0.85	0.85	0.80	0.80	0.80	0.81	0.80	0.79
2	0.91	0.90	0.90	0.90	0.90	0.91	0.86	0.86	0.86	0.86	0.86	0.86	0.81	0.81	0.81	0.81	0.81	0.79
3	0.90	0.90	0.90	0.90	0.90	0.90	0.85	0.85	0.85	0.85	0.85	0.86	0.80	0.80	0.80	0.80	0.80	0.79
4	0.89	0.89	0.89	0.89	0.89	0.90	0.84	0.84	0.84	0.84	0.85	0.85	0.80	0.80	0.80	0.79	0.80	0.81
5	0.89	0.89	0.89	0.89	0.89	0.89	0.84	0.84	0.84	0.85	0.84	0.84	0.79	0.82	0.80	0.79	0.78	0.78
6	0.90	0.90	0.90	0.90	0.89	0.90	0.84	0.84	0.84	0.85	0.84	0.84	0.79	0.79	0.80	0.79	0.78	0.79
AVG	0.90	0.90	0.90	0.90	0.90	0.90	0.85	0.85	0.85	0.85	0.85	0.85	0.80	0.80	0.80	0.80	0.80	0.79

Table 9.4: Comparison of the energy consumption of the single-action and multi-action policies

	$P_{CON} = 0.99$						$P_{CON} = 0.97$						$P_{CON} = 0.95$					
Clip No:	β_{OFF}	β_{ONL}	β^{L}_{PSO}	β^{LT}_{PSO}	β^{T}_{CSO}	β^{TS}_{CSO}	β_{OFF}	β_{ONL}	β^{L}_{PSO}	β^{LT}_{PSO}	β^{T}_{CSO}	β^{TS}_{CSO}	β_{OFF}	β_{ONL}	β^{L}_{PSO}	β^{LT}_{PSO}	β^{T}_{CSO}	β^{TS}_{CSO}
1	174	118	175	158	129	125	161	118	151	147	126	122	157	119	148	143	129	114
2	175	119	176	158	130	125	162	118	151	148	127	122	158	119	148	143	130	112
3	174	119	175	158	129	124	161	118	151	147	125	121	157	119	149	144	129	112
4	174	119	175	159	129	126	161	119	154	151	126	122	158	121	152	147	130	116
5	173	119	174	157	128	124	161	120	149	147	124	120	156	121	140	139	127	117
6	174	119	175	158	129	124	161	119	150	147	125	121	156	120	143	139	128	114
AVG	174	119	175	158	129	125	161	119	151	148	126	121	157	120	147	143	129	114
	$P_{CON} = 0.90$						$P_{CON} = 0.85$						$P_{CON} = 0.80$					
1	125	114	126	123	111	106	116	112	116	115	105	103	112	110	113	116	104	100
2	126	114	126	124	111	105	117	112	116	116	104	102	113	110	114	116	103	101
3	125	114	125	123	111	105	116	112	115	115	105	103	113	110	113	117	104	103
4	125	113	125	123	109	105	117	112	116	115	104	102	114	111	114	116	104	106
5	123	113	123	122	107	103	115	111	114	114	102	100	112	109	113	114	101	100
6	124	113	124	123	108	104	115	111	115	115	102	101	112	109	113	115	101	100
AVG	125	114	125	123	110	105	116	112	115	115	104	102	113	110	113	116	103	102

Table 9.5: Comparison of the energy consumption of the policies obtained by two different solution techniques

	$P_{CON} = 0.99$		$P_{CON} = 0.97$		$P_{CON} = 0.95$		$P_{CON} = 0.90$	
Clip No:	β_{OFF}	β_{OFF}^{2-LEV}	β_{OFF}	β_{OFF}^{2-LEV}	β_{OFF}	β_{OFF}^{2-LEV}	β_{OFF}	β_{OFF}^{2-LEV}
1	174	184	161	165	157	159	125	126
2	175	185	162	165	158	161	126	126
3	174	184	161	165	157	160	125	126
4	174	184	161	166	158	161	125	126
5	173	183	161	165	156	159	123	124
6	174	184	161	165	156	159	124	125
AVG	174	184	161	165	157	160	125	126

Table 9.5 presents the energy consumption of the single-action policies. In this table, the policy obtained by the first solution method described in Section 6.2.2 is denoted by β_{OFF}. The policy obtained by the heuristic-based solution method described in Section 6.2.3 is denoted by β_{OFF}^{2-LEV}. It is observed that the energy consumption of β_{OFF}^{2-LEV} is within 5% of the energy consumption of β_{OFF}. Therefore, it is concluded that the heuristic-based solution technique used in the optimization of multi-action policies works fairly well for the stochastic energy minimization problem. This result also shows that capturing spatial correlations results in higher energy savings as the performance constraint increases.

9.4 Effect of Workload Decomposition into On-Chip and Off-Chip Components

In this section, the effect of on-chip/off-chip workload decomposition on a single-action energy management policy is studied. For this purpose, the following policies are compared at the nominal deadline:

- β_{OFF} is the single-action policy which ignores the heterogeneous nature of the workload and is optimized according to sum of on-chip and off-chip components. In other words, β_{OFF} assumes that both on-chip and off-chip workload is executed on the core and their execution time can be scaled by DVS.

- β_{OFF}^{decomp} is the single-action policy optimized using the decomposed workload data. β_{OFF}^{decomp} adjusts voltage settings according to the on-chip workload only as described in Section 7.3.1.

Table 9.6 presents the completion probability achieved by β_{OFF} and β_{OFF}^{decomp}. For the results shown in this table, both β_{OFF} and β_{OFF}^{decomp} are applied to the decomposed workload in which the execution time of the off-chip workload can not be scaled by DVS. It is observed that β_{OFF} slightly overaccomplishes the desired completion probability as P_{CON} decreases. This result is expected since the voltage settings in β_{OFF} are optimized according to the total workload. However, ignoring the heterogeneous nature of the workload has a small effect on the completion probability of β_{OFF}, since the off-chip workload is insignificant compared with the on-chip workload for MPEG2 video decoding.

For all of the results presented in the previous sections, the energy consumed by a core to execute a task was calculated using (3.4) in which workload decomposition is ignored. When the workload is decomposed into on-chip and off-chip components, and the core can be shut down during the off-chip workload, (3.4) reduces to (7.8). However, this reduction ignores the energy spent for the off-chip workload which can overshadow the energy savings achieved by a policy. For instance, if the energy spent for the off-chip workload is equal to the energy spent for the on-chip workload, then a 50% reduction in the energy spent for the on-chip workload will correspond to only a 25% reduction in the total energy consumed in the system. Therefore, it is necessary to consider the energy spent not only for the on-chip workload, but also for the off-chip workload. For this purpose, we compare the energy consumption of β_{OFF} which is computed by (3.4) to the energy consumption of β_{OFF}^{2-LEV} which is computed as follows

$$E_u^{dyn} = C_u \ \mathbf{E}\left[{L_u}^{on}\right] V_{dd_u}^2 + C_u \ \mathbf{E}\left[{L_u}^{off}\right] V_{dd_{max}}^2 \tag{9.1}$$

In (9.1), the energy spent for the off-chip workload is approximated as the energy spent for the same amount of on-chip workload executed at maximum voltage. This approximation, which does not claim to be realistic, assumes that the amount of power dissipated in the core and in the memory subsystem, which includes the interface circuitry in between, are equal.

Table 9.7 compares the energy consumption of β_{OFF} (as expressed in (3.4)) and β_{OFF}^{decomp} (as expressed in (9.1)). In this table, each energy consumption value is normalized to the energy consumption at the maximum voltage. It is observed that the energy consumption of β_{OFF} is within 4% (11%) of the energy consumption of β_{OFF}^{decomp} for $P_{CON} \geq 0.95$ ($P_{CON} < 0.95$). This shows that ignoring on-chip /off-chip workload decomposition for MPEG2 video decoding does not have a significant effect on the results provided in the previous sections. This conclusion is also consistent with the fact that the off-chip workload is insignificant compared with the on-chip workload for this application. However, it appears that the heterogeneous nature of task workloads should be considered even for computation-dominated applications, such as MPEG2 video decoding, for intelligent energy management.

Table 9.6: Effect of workload decomposition on the completion probability of a single-action policy

	$P_{CON} = 0.99$		$P_{CON} = 0.97$		$P_{CON} = 0.95$		$P_{CON} = 0.90$		$P_{CON} = 0.85$		$P_{CON} = 0.80$	
Clip No:	β_{OFF}	β_{OFF}^{decomp}	β_{OFF}	β_{OFF}^{decomp}	β_{OFF}	β_{OFF}^{decomp}	β_{OFF}	β_{OFF}^{decomp}	β_{OFF}	β_{OFF}^{decomp}	β_{OFF}	β_{OFF}^{decomp}
1	0.99	0.99	0.97	0.97	0.95	0.95	0.91	0.91	0.88	0.86	0.83	0.80
2	0.99	0.99	0.97	0.97	0.95	0.95	0.91	0.90	0.88	0.86	0.84	0.81
3	0.99	0.99	0.97	0.97	0.95	0.95	0.90	0.90	0.87	0.85	0.83	0.80
4	0.99	0.99	0.97	0.97	0.96	0.95	0.90	0.89	0.86	0.84	0.83	0.80
5	0.99	0.99	0.98	0.98	0.96	0.96	0.90	0.89	0.86	0.85	0.83	0.80
6	0.99	0.99	0.97	0.97	0.95	0.95	0.90	0.90	0.86	0.84	0.82	0.79
AVG	0.99	0.99	0.97	0.97	0.95	0.95	0.90	0.90	0.87	0.85	0.83	0.80

Table 9.7: Effect of workload decomposition on the energy consumption due to both on-chip and off-chip workload

	$P_{CON} = 0.99$		$P_{CON} = 0.97$		$P_{CON} = 0.95$		$P_{CON} = 0.90$		$P_{CON} = 0.85$		$P_{CON} = 0.80$	
Clip No:	β_{OFF}	β_{OFF}^{decomp}	β_{OFF}	β_{OFF}^{decomp}	β_{OFF}	β_{OFF}^{decomp}	β_{OFF}	β_{OFF}^{decomp}	β_{OFF}	β_{OFF}^{decomp}	β_{OFF}	β_{OFF}^{decomp}
1	74	74	64	66	59	61	42	45	37	40	35	38
2	74	74	64	65	59	60	42	44	37	40	35	38
3	74	74	65	66	59	61	41	44	37	40	34	38
4	74	74	65	66	59	61	41	44	36	39	34	37
5	75	75	66	67	60	62	41	45	37	41	34	39
6	74	74	65	66	59	61	41	44	37	40	34	38
AVG	74	74	65	66	59	61	41	44	37	40	34	38

Chapter 10

CONCLUSIONS

10.1 Summary

Enabled by the advances in integrated circuit technologies, various computing systems (e.g., laptops, game consoles and cellular phones) have become integral parts of our lives. Almost all of such complex systems conceived in the near future will arguably have a multicore architecture for increased performance and functionality. These systems, either battery-operated or wall-powered, require intelligent energy management schemes, since energy consumption has become a primary concern in their design and implementation.

In this thesis, we study the energy management problem for multicore systems running real-time applications that have been decomposed into concurrent and interdependent tasks. We focus on soft real-time applications with repetitive behavior which correspond to a large class of multimedia applications. We represent such an application and its tolerance to deadline misses by a task graph and a completion probability constraint respectively. We assume that the system architecture has been determined and each core has dynamic voltage scaling capability. We also assume that the tasks constituting the application are already scheduled and assigned to the cores in the system. Given these assumptions, our goal is to find optimal energy management policies that minimize the expected energy consumption of the cores and satisfy a given completion probability constraint.

The primary difficulty in this problem is the unpredictability of task workloads that vary over time and usually depend on the input data of the application. We show on simple examples how energy management ignorant of the variability of and the correlations among task workloads can be significantly suboptimal or not properly fulfill timing constraints. Then, we propose three stochastic application models which capture this statistical nature of task workloads. These stochastic models are constructed using the workload data collected from actual application runs. From each stochastic application model, we obtain an optimal energy management policy by a novel mathematical formulation. For these formu-

lations, we present two efficient solution methods one of which employs a heuristic approach previously used in activity network problems. Our energy management policies introduce negligible overhead at runtime, since they require only table look-ups and the associated optimization problems are solved offline. We also show how the presented stochastic modeling and optimization framework can be generalized to handle the implications of the memory behavior of an application and the energy consumption of hardware components other than the main processor.

We demonstrate our framework using a popular multimedia application, MPEG2 video decoding. Experimental results show that there is significant potential in intelligent energy management that can exploit application characteristics. The proposed energy management policies predict the application workload and achieve the desired completion probability accurately even for the input data that is not included in the training of the underlying stochastic application models. These policies are more effective than two previous approaches in the literature. The energy consumption achieved by our policies are even close to the theoretical bound in most of the cases.

10.2 Future Directions

This thesis can be extended and generalized in the following directions:

- Task scheduling and assignment in fact affect the amount of energy savings achievable by an energy management scheme. Therefore, scheduling and assignment problems need to be combined with the energy minimization problem to achieve a more comprehensive and general energy management framework.

- In this thesis, we assumed that all tasks should be finished within a prespecified time interval. However, in a multimedia application, the output of the application is usually buffered especially if the streaming input data is not a live feed. In such a case, an energy management policy presented in this thesis will probably achieve a higher completion probability, since the timing slacks occurred in a period will be added to the successor periods. Therefore, the presented energy management framework can be generalized to consider the applications the output of which are buffered.

- Finally, in the stateful stochastic models, the states are defined using a known feature of the application. However, it is possible that an application may not have such observable features that convey information about the workload. In such a case, a stateful stochastic model of the application can still be obtained using Hidden Markov Models [45]. Therefore, our energy management framework can be extended with a fourth stateful model and an associated policy.

BIBLIOGRAPHY

[1] Srinivas Devadas and Sharad Malik, "A survey of optimization techniques targeting low power vlsi circuits," in *DAC '95: Proceedings of the 32nd ACM/IEEE conference on Design automation*, New York, NY, USA, 1995, pp. 242–247, ACM Press.

[2] Vasanth Venkatachalam and Michael Franz, "Power reduction techniques for microprocessor systems," *ACM Comput. Surv.*, vol. 37, no. 3, pp. 195–237, 2005.

[3] D. Son, C. Yu, and H. Kim, "Dynamic voltage scaling on MPEG decoding," in *International Conference of Parallel and Distributed System (ICPADS)*, 2001.

[4] Kihwan Choi, Karthik Dantu, Wei-Chung Cheng, and Massoud Pedram, "Frame-based dynamic voltage and frequency scaling for a mpeg decoder," in *ICCAD '02: Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, New York, NY, USA, 2002, pp. 732–737, ACM Press.

[5] C. Isci, A. Buyuktosunoglu, and M. Martonosi, "Long-term workload phases: duration predictions and applications to DVFS," *Micro, IEEE*, vol. 25, no. 5, pp. 39–51, 2005.

[6] Yan Gu, Samarjit Chakraborty, and Wei Tsang Ooi, "Games are up for dvfs," in *DAC '06: Proceedings of the 43rd annual conference on Design automation*, New York, NY, USA, 2006, pp. 598–603, ACM Press.

[7] Andy C. Bavier, A. Brady Montz, and Larry L. Peterson, "Predicting mpeg execution times," in *SIGMETRICS '98/PERFORMANCE '98: Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, New York, NY, USA, 1998, pp. 131–140, ACM Press.

[8] G. A. Paleologo, L. Benini, A. Bogliolo, and G. De Micheli, "Policy optimization for dynamic power management," in *DAC '98: Proceedings of the 35th annual conference on Design automation*, New York, NY, USA, 1998, pp. 182–187, ACM Press.

[9] Qinru Qiu and Massoud Pedram, "Dynamic power management based on continuous-time markov decision processes," in *DAC '99: Proceedings of the 36th ACM/IEEE conference on Design automation*, New York, NY, USA, 1999, pp. 555–561, ACM Press.

[10] Tajana Simunic, Giovanni de Micheli, and Luca Benini, "Event-driven power management of portable systems," in *ISSS '99: Proceedings of the 12th international symposium on System synthesis*, Washington, DC, USA, 1999, p. 18, IEEE Computer Society.

[11] Tajana Simunic, Luca Benini, Peter Glynn, and Giovanni De Micheli, "Dynamic power management for portable systems," in *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, New York, NY, USA, 2000, pp. 11–19, ACM Press.

[12] Tajana Simunic, Luca Benini, Andrea Acquaviva, Peter Glynn, and Giovanni De Micheli, "Dynamic voltage scaling and power management for portable systems," in *DAC '01: Proceedings of the 38th conference on Design automation*, New York, NY, USA, 2001, pp. 524–529, ACM Press.

[13] Flavius Gruian, "Hard real-time scheduling for low-energy using stochastic data and dvs processors," in *ISLPED '01: Proceedings of the 2001 international symposium on Low power electronics and design*, New York, NY, USA, 2001, pp. 46–51, ACM Press.

[14] Yumin Zhang, Xiaobo Sharon Hu, and Danny Z. Chen, "Task scheduling and voltage selection for energy minimization," in *DAC '02: Proceedings of the 39th conference on Design automation*, New York, NY, USA, 2002, pp. 183–188, ACM Press.

[15] Shaoxiong Hua, Gang Qu, and Shuvra S. Bhattacharyya, "Energy-efficient multiprocessor implementation of embedded software," *Lecture Notes in Computer Science*, vol. 2855, 2003.

[16] Lap-Fai Leung, Chi-Ying Tsui, and Xiaobo Sharon Hu, "Exploiting dynamic workload variation in low energy preemptive task scheduling," in *DATE '05: Proceedings of the*

conference on Design, Automation and Test in Europe, Washington, DC, USA, 2005, pp. 634–639, IEEE Computer Society.

[17] Alexandru Andrei, Marcus T. Schmitz, Petru Eles, Zebo Peng, and Bashir M. Al Hashimi, "Quasi-static voltage scaling for energy minimization with time constraints," in *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, Washington, DC, USA, 2005, pp. 514–519, IEEE Computer Society.

[18] C. Scordino and E. Bini, "Optimal speed assignment for probabilistic execution times," in *2 ndWorkshop on Power-Aware Real-Time Computing (PARC05), NJ, Sept*, 2005.

[19] Ruibin Xu, Daniel Mossé, and Rami Melhem, "Minimizing expected energy in real-time embedded systems," in *EMSOFT '05: Proceedings of the 5th ACM international conference on Embedded software*, New York, NY, USA, 2005, pp. 251–254, ACM Press.

[20] Robert P. Dick, *Multiobjective Synthesis Of Low-Power Real-Time Distributed Eembedded Systems*, Ph.D. thesis, November 2002.

[21] Intel PXA270 Processor, "Electrical, mechanical, and thermal specification," *Available from http://www.intel.com/design/pca/products/pxa27x/techdocs.htm.*

[22] Sheldon M. Ross, "Introduction to probability models," January 2003.

[23] SJ Kim, S. Boyd, S. Yun, D. Patil, and M. Horowitz, "A heuristic for optimizing stochastic activity networks with applications to statistical digital circuit sizing, 2005," *Manuscript. Available from www. stanford. edu/boyd/heur_san_opt. html.*

[24] David G. Luenberger, *Linear and nonlinear programming*, Addison-Wesley, 2 edition, 1984.

[25] Frederick S. Hillier amd Gerald J. Lieberman, "Introduction to operations research," 2001.

[26] S. Boyd, SJ Kim, L. Vandenberghe, and A. Hassibi, "A tutorial on geometric programming," *Revised for Optimization and Engineering. Available from www. stanford. edu/boyd/gp_tutorial. html.*

[27] A. Mutapcic, K. Koh, S. Kim, and S. Boyd, "ggplab version 0.9, A Matlab Toolbox for Geometric Programming," January 2006.

[28] fmincon, "Optimization toolbox, matlab, the mathworks, inc.," *Available from http://www.mathworks.com/access/helpdesk/help/toolbox/optim/.*

[29] Chunco Lee, Miodrag Potkonjak, and William H. Mangione-Smith, "Mediabench: A tool for evaluating and synthesizing video communications systems," in *Proceedings of the 30th AnnualACM/IEEE International Symposium on Microarchitecture*, December 1997, pp. 330–335.

[30] Nathan T. Slingerland and Alan Jay Smith, "Design and characterization of the berkeley multimedia workload," vol. 8, no. 4, pp. 315–327, July 2002.

[31] Jason E. Fritts, Frederick W. Steiling, and Joseph A. Tucek, "Mediabench ii video: expediting the next generation of video systems research," in *Proceedings of SPIE Embedded Processors for Multimedia and Communications II*, January 2005, pp. 79–93.

[32] Dinero IV, "Trace-driven uniprocessor cache simulator," *Available from http://www.cs.wisc.edu/ markhill/DineroIV.*

[33] ValGrind, "A suite of tools for debugging and profiling linux programs," *Available from http://valgrind.org/.*

[34] SimpleScalar, "Simplescalar llc," *Available from http://www.simplescalar.com.*

[35] gprof, "The gnu profiler," *Available from http://www.gnu.org/software/binutils/manual/gprof-2.9.1/gprof.html.*

[36] PAPI, "Performance application programming interface," *Available from http://icl.cs.utk.edu/papi/.*

[37] Intel, "Ia-32 intel architecture optimization reference manual," *Available from http://www.intel.com/design/pentium4/manuals/248966.htm.*

[38] Fen Xie, Margaret Martonosi, and Sharad Malik, "Compile-time dynamic voltage scaling settings: opportunities and limits," in *PLDI '03: Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, New York, NY, USA, 2003, pp. 49–62, ACM Press.

[39] Kihwan Choi, Ramakrishna Soma, and Massoud Pedram, "Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to on-chip computation times," in *DATE '04: Proceedings of the conference on Design, automation and test in Europe*, Washington, DC, USA, 2004, p. 10004, IEEE Computer Society.

[40] Wiplove Mathur and Jeanine Cook, "Improved estimation for software multiplexing of performance counters," in *Proceedings of the IEEE International Symposium on Modeling, Analysis, Simulation of Computer and Telecommunication Systems*, September 2005.

[41] E. Iwata and K. Olukotun, "Exploiting coarse-grain parallelism in the mpeg-2 algorithm," 1998.

[42] FFMPEG, "Multimedia system," *Available from http://ffmpeg.mplayerhq.hu/.*

[43] J. Pouwelse, K. Langendoen, and HJ Sips, "Application-directed voltage scaling," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 11, no. 5, pp. 812–826, 2003.

[44] Christopher J. Hughes, Praful Kaul, Sarita V. Adve, Rohit Jain, Chanik Park, and Jayanth Srinivasan, "Variability in the execution of multimedia applications and implications for architecture," in *ISCA '01: Proceedings of the 28th annual international*

symposium on Computer architecture, New York, NY, USA, 2001, pp. 254–265, ACM Press.

[45] L.R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, February 1989.

VITA

Soner Yaldız was born in Çine, Aydın, Turkey on March 7, 1981. He received his B.S. degree in Microelectronics from SABANCI University, Istanbul, in 2004. From September 2004 to September 2006, he worked as a teaching and research assistant at Center for Advanced Design Technologies (CADT) at KOÇ University, Turkey. His project on stochastic application models for multicore energy minimization was sponsored by The Scientific and Technical Research Council of Turkey (TÜBİTAK) since March 2004. He has presented a paper about energy management of multicore systems in ESTIMEDIA'06 (New York, U.S.).