Wiley Series on Parallel and Distributed Computing • Albert Y. Zomaya, Series Editor

ENERGY-EFFICIENT DISTRIBUTED COMPUTING SYSTEMS

Edited by
Albert Y. Zomaya • Young Choon Lee

WILEY
IEEE
IEEE computer society

ENERGY-EFFICIENT DISTRIBUTED COMPUTING SYSTEMS

Edited by

Albert Y. Zomaya
Young Choon Lee

A JOHN WILEY & SONS, INC., PUBLICATION

ENERGY-EFFICIENT DISTRIBUTED COMPUTING SYSTEMS

WILEY SERIES ON PARALLEL AND DISTRIBUTED COMPUTING

Editor: Albert Y. Zomaya

A complete list of titles in this series appears at the end of this volume.

ENERGY-EFFICIENT DISTRIBUTED COMPUTING SYSTEMS

Edited by

Albert Y. Zomaya
Young Choon Lee

A JOHN WILEY & SONS, INC., PUBLICATION

Cover Image: Baris Simsek/iStockphoto

Published by John Wiley & Sons, Inc., Hoboken, New Jersey
Published simultaneously in Canada

Library of Congress Cataloging-in-Publication Data:

Zomaya, Albert Y.
Energy-efficient distributed computing systems / Albert Y. Zomaya, Young Choon Lee.
p. cm.
ISBN 978-0-470-90875-4 (hardback)
1. Computer networks–Energy efficiency. 2. Electronic data processing–Distributed processing–Energy conservation. 3. Green technology. I. Lee, Young Choon, 1973– II. Title.
TK5105.5.Z66 2012
004′.36–dc23

2011042246

Printed in the United States of America

ISBN: 9780470908754

10 9 8 7 6 5 4 3 2 1

To our families for their help, support, and patience.

CONTENTS

PREFACE

The scope of energy-efficient computing is not limited to main computing components (e.g., processors, storage devices, and visualization facilities), but it can expand to a much larger range of resources associated with computing facilities, including auxiliary equipment, water used for cooling, and even physical and floor space that these resources occupy. Energy consumption in computing facilities raises various monetary, environmental, and system performance concerns.

Recent advances in hardware technologies have improved the energy consumption issue to a certain degree. However, it still remains a serious concern for energy-efficient computing because the amount of energy consumed by computing and auxiliary hardware resources is affected substantially by their usage patterns. In other words, resource underutilization or overloading incurs a higher volume of energy consumption when compared with efficiently utilized resources. This calls for the development of various software energy-saving techniques and new algorithms that are more energy efficient.

This book, *Energy-Efficient Distributed Computing Systems*, seeks to provide an opportunity for researchers to explore different energy consumption issues and their impact on the design of new computing systems. The book is quite timely since the field of distributed computing as a whole is undergoing many changes. Vast literature exists today on such energy consumption paradigms and frameworks and their implications for a wide range of distributed platforms.

The book is intended to be a virtual roundtable of several outstanding researchers, which one might invite to attend a conference on energy-efficient computing systems. Of course, the list of topics that is explored here is by no means exhaustive, but most of the conclusions provided here should be extended to other computing platforms that are not covered here. There was a decision to limit the number of chapters while providing more pages for contributing

authors to express their ideas, so that the book remains manageable within a single volume.

We also hope that the topics covered in this book will get the readers to think of the implications of such new ideas on the developments in their own fields. The book endeavors to strike a balance between theoretical and practical coverage of innovative problem-solving techniques for a range of distributed platforms. The book is intended to be a repository of paradigms, technologies, and applications that target the different facets of energy consumption in computing systems.

The 26 chapters were carefully selected to provide a wide scope with minimal overlap between the chapters to reduce duplications. Each contributor was asked that his/her chapter should cover review material as well as current developments. In addition, the choice of authors was made so as to select authors who are leaders in their respective disciplines.

ALBERT Y. ZOMAYA
YOUNG CHOON LEE

ACKNOWLEDGMENTS

First and foremost, we would like to thank and acknowledge the contributors to this volume for their support and patience, and the reviewers for their useful comments and suggestions that helped in improving the earlier outline of the book and presentation of the material. Also, I should extend my deepest thanks to Simone Taylor and Diana Gialo from Wiley (USA) for their collaboration, guidance, and most importantly, patience in finalizing this handbook. Finally, I would like to acknowledge the efforts of the team from Wiley's production department for their extensive efforts during the many phases of this project and the timely manner in which the book was produced.

ALBERT Y. ZOMAYA
YOUNG CHOON LEE

CONTRIBUTORS

PRITI, AGHERA, University of California, San Diego, CA, USA.

AL-NASHIF, YOUSSIF, NSF Center for Autonomic Computing, The University of Arizona, USA.

AYOUB, RAID, University of California, San Diego, CA, USA.

BERRAL, JOSEP LL., Computer Architecture Dept. and Department of Software, UPC-Barcelona Tech., Catalonia, Spain.

BILAL, KASHIF, Department of Computer Science, North Dakota State University, Fargo, ND, USA.

BOLOORI, ALI JAVADZADEH, Centre for Distributed and High Performance Computing, School of Information Technologies, University of Sydney, NSW, Australia.

BORGETTO, DAMIEN, University Paul Sabatier, Toulouse, France.

BOUVRY, PASCAL, Faculty of Sciences, Technology, and Communications, University of Luxembourg, Luxembourg.

CAMERON, KIRK W., Virginia Tech, VA, USA.

CASANOVA, HENRI, University of Hawai'i at Manoa, Hawai'i, USA.

COMITO, CARMELA, DEIS, University of Calabria, Rende (CS), Italy.

DA COSTA, GEORGES, University Paul Sabatier, Toulouse, France.

DELICATO, FLAVIA C., Computer Science Department, Federal University of Rio de Janeiro—RN, Brazil.

DHIMAN, GAURAV, University of California, San Diego, CA, USA.

FENG, WU-CHUN, Virginia Tech, Blacksburg, Virginia, USA.

JOSEPH. O. FITO, Computer Architecture Dept. and Barcelona Supercomputing Center, UPC-Barcelona Tech., Catalonia, Spain.

GAVALDA, RICARD, Department of Software, UPC-Barcelona Tech., Catalonia, Spain.

GE, RONG, The Department of Mathematics, Statistics, and Computer Science, Marquette University, WI, USA.

GHANI, NASIR, Department of Electrical and Computer Engineering, University of New Mexico, Albuquerque, NM, USA.

GOIRI, INIGO, Computer Architecture Dept. and Barcelona Supercomputing Center, UPC-Barcelona Tech., Catalonia, Spain.

GONG, JIAYU, Department of Electrical and Computer Engineering, Wayne State University, MI, USA.

DE GROOT, MARTIN, CSIRO ICT Center, Epping, NSW, Australia.

GUITART, JORDI, Computer Architecture Dept. and Barcelona Supercomputing Center, UPC-Barcelona Tech., Catalonia, Spain.

GURUMURTHI, SUDHANVA, Dept. of Computer Science, University of Virginia, Charlottesville, VA, USA.

HARIRI, SALIM, NSF Center for Autonomic Computing, The University of Arizona, USA.

HARTENSTEIN, REINER, Department of Computer Science, Kaiserslautern University of Technology, Kaiserslautern, Germany.

HSU, CHUNG-HSING, Oak Ridge National Laboratory, Oak Ridge, TN, USA.

JIANG, WEIRONG, Juniper Networks, Inc., Sunnyvale, CA, USA.

JULIA, FERRAN, Computer Architecture Dept., UPC-Barcelona Tech., Catalonia, Spain.

KANDEMIR, MAHMUT, Pennsylvania State University, PA, USA.

KHAN, SAMEE ULLAH, Department of Electrical and Computer Engineering, North Dakota State University, Fargo, ND, USA.

KHARGHARIA, BITHIKA, Cisco Systems, Inc., Durham, NC, USA.

KIM, JONG-KOOK, School of Electrical Engineering, Korea University, Korea.

LEE, YOUNG CHOON, Centre for Distributed and High Performance Computing, School of Information Technologies, University of Sydney, NSW, Australia.

LEFEVRE, LAURENT, INRIA, Ecole Normale Superieure de Lyon, University of Lyon, France.

LEINGANG, JAMES, Department of Electrical and Computer Engineering, North Dakota State University, Fargo, ND, USA.

LI, JUAN, Department of Computer Science, North Dakota State University, Fargo, ND, USA.

LI, KEQIN, State University of New York, New Paltz, NY, USA.

LINDBERG, PEDER, Department of Electrical and Computer Engineering, North Dakota State University, Fargo, ND, USA.

LUO, HAOTING, NSF Center for Autonomic Computing, The University of Arizona, AZ, USA.

LYSAKER, DANIEL, Department of Electrical and Computer Engineering, North Dakota State University, Fargo, ND, USA.

MANZANARES, ADAM, Los Alamos National Laboratory, Los Alamos, NM, USA.

MIN-ALLAH, NASRO, Department of Computer Science, COMSATS Institute of Information Technology, Pakistan.

NARAYANAN, SRI HARI KRISHNA, Argonne National Laboratory, IL, USA.

NIKZAD, NIMA, University of California, San Diego, CA, USA.

NOU, RAMON, Computer Architecture Dept. and Barcelona Supercomputing Center, UPC-Barcelona Tech., Catalonia, Spain.

ORGERIE, ANNE-CECILE, Ecole Normale Superieure de Lyon, Lyon, France.

OZTURK, OZCAN, Bilkent University, Turkey.

PARASHAR, MANISH, NSF Cloud and Autonomic Computing Center and Rutgers Discovery Informatics Institute, Rutgers University, NJ, USA.

PEDRAM, MASSOUD, University of Southern California, Los Angeles, CA, USA.

PIERSON, JEAN-MARC, University Paul Sabatier, Toulouse, France.

PIRES, PAULO F., Computer Science Department, Federal University of Rio de Janeiro - RN, Brazil.

PRASANNA, VIKTOR K., University of Southern California, Los Angeles, CA, USA.

QIN, XIAO, Auburn University, Auburn, AL, USA.

RIZVANDI, NIKZAD BABAII, Centre for Distributed and High Performance Computing, School of Information Technologies, University of Sydney, NSW, Australia.

RODERO, IVAN, NSF Cloud and Autonomic Computing Center and Rutgers Discovery Informatics Institute, Rutgers University, NJ, USA.

RONG, PENG, Brocade Communications Systems, San Jose, CA, USA.

ROSING, TAJANA SIMUNIC, University of California, San Diego, CA, USA.

RUAN, XIAOJUN, Auburn University, Auburn, AL, USA.

SIVASUBRAMANIAM, ANAND, Dept. of Computer Science and Engineering, The Pennsylvania State University, PA, USA.

TAHERI, JAVID, Centre for Distributed and High Performance Computing, School of Information Technologies, University of Sydney, NSW, Australia.

TALIA, DOMENICO, DEIS, University of Calabria, Rende (CS), Italy.

TORRES, JORDI, Computer Architecture Dept. and Barcelona Supercomputing Center, UPC-Barcelona Tech., Catalonia, Spain.

TRUNFIO, PAOLO, DEIS, University of Calabria, Rende (CS), Italy.

WANG, CHEN, CSIRO ICT Center, Epping, NSW, Australia.

XU, CHENG-ZHONG, Department of Electrical and Computer Engineering, Wayne State University, MI, USA.

YIN, SHU, Auburn University, Auburn, AL, USA.

YOUSIF, MAZIN, T-Systems International, Inc., Portland, OR, USA.

ZAPPI, PIERO, University of California, San Diego, CA, USA.

ZOMAYA, ALBERT Y., Centre for Distributed and High Performance Computing, School of Information Technologies, University of Sydney, NSW, Australia.

CHAPTER 1

POWER ALLOCATION AND TASK SCHEDULING ON MULTIPROCESSOR COMPUTERS WITH ENERGY AND TIME CONSTRAINTS

KEQIN LI

1.1 INTRODUCTION

1.1.1 Energy Consumption

Performance-driven computer development has lasted for over six decades. Computers have been developed to achieve higher performance. As of June 2010, three supercomputers have achieved petaflops speed: Cray Jaguar (224,162 processors, 1.759 petaflops), Dawning Nebulae (120,640 processors, 1.271 petaflops), and IBM Roadrunner (122,400 processors, 1.042 petaflops) [1]. According to Moore's law of computing hardware, the following quantities increase (decrease) exponentially, doubling (halving) approximately every 2 years: the number of transistors per integrated circuit (cost per transistor), processing speed, memory/storage capacity (cost per unit of information), and network capacity [2].

While performance/cost has increased dramatically, power consumption in computer systems has also increased according to Moore's law. To achieve higher computing performance per processor, microprocessor manufacturers have doubled the power density at an exponential speed over decades, which will soon reach that of a nuclear reactor [3]. Such increased energy consumption causes severe economic, ecological, and technical problems.

- *Economic Impact*. Computer systems consume tremendous amount of energy and natural resources. It has been reported that desktop computers in the United States account for over 10% of commercial electricity

Energy-Efficient Distributed Computing Systems, First Edition.
Edited by Albert Y. Zomaya and Young Choon Lee.

consumption [4]. A large-scale multiprocessor computing system consumes millions of dollars of electricity and natural resources every year, equivalent to the amount of energy used by tens of thousands US households [5]. A large data center such as Google can consume as much electricity as does a city. Furthermore, the cooling bill for heat dissipation can be as high as 70% of the above cost [6]. Supercomputers are making less efficient use of space, which often results in the design and construction of new machine rooms or even entirely new buildings.

- *Ecological Impact*. Desktop computers produce as much carbon dioxide (CO_2) as millions of cars. A recent report reveals that the global information technology industry generates as much greenhouse gas as the world's airlines, about 2% of global CO_2 emissions [7]. The heat dissipation problem gets increasingly worse because of higher computing speeds, shrinking packages, and growing energy-hungry applications such as multimedia and communications.
- *Technical Impact*. Large-scale multiprocessor computers require expensive packaging and cooling technologies, and demand for sophisticated fault-tolerant mechanisms that deal with decreased reliability due to heat dissipation caused by increased energy consumption. Despite sophisticated cooling facilities constructed to ensure proper operation, the reliability of large-scale multiprocessor computing systems is measured in hours, and the main source of outage is hardware failure caused by excessive heat. It is conceivable that a supercomputing system with 10^5 processors would spend most of its time in checkpointing and restarting [8].

It is clear that there are compelling economic, environmental, and technical reasons for emphasis on energy efficiency.

1.1.2 Power Reduction

Power conservation is critical in many computation and communication environments and has attracted extensive research activities. For high performance supercomputers, energy-aware design has significance impact on system performance. It is noticed that performance per rack equals to performance per watt times watt per rack, where watt per rack is determined by thermal cooling capabilities and can be considered as a constant of order 20 kW for an air-cooled rack. Therefore, it is the performance per watt term that determines the rack performance. It is found that in terms of performance per watt, the low frequency and low power embedded IBM PowerPC consistently outperforms high frequency and high power microprocessors by a factor of 2–10. This is one of the main reasons why IBM chose the low power design for the Blue Gene/L supercomputer that was developed around a processor with moderate frequency. In mobile computing and communication environments, efficient processor power management increases the lifetime of battery operated devices such as hand-held mobile computers and portable embedded systems. Energy efficiency is a major

design constraint in these portable devices, since battery technology has not been developed in the same pace as semiconductor industry.

Reducing processor energy consumption has been an important and pressing research issue in recent years. There has been increasing interest and importance in developing high performance and energy-efficient computing systems. There exists a large body of literature on power-aware computing and communication. The reader is referred to References [3, 9–11] for comprehensive surveys.

There are two approaches to reducing power consumption in computing systems. The first approach is the method of thermal-aware hardware design, which can be carried out at various levels, including device level power reduction, circuit and logic level techniques, and architecture level power reduction (low power processor architecture adaptations, low power memories and memory hierarchies, and low power interconnects). Low power consumption and high system reliability, availability, and usability are main concerns of modern high performance computing system development. In addition to the traditional performance measure using FLOPS, the Green500 list uses FLOPS per watt to rank the performance of computing systems, so that the awareness of other performance metrics such as energy efficiency and system reliability can be raised [12]. All the current systems that can achieve at least 400 MFLOPS/W are clusters of low power processors, aiming to achieve high performance/power and performance/space. For instance, the Dawning Nebulae, currently the world's second fastest computer, which achieves peak performance of 2.984 PFLOPS, is also the fourth most energy-efficient supercomputer in the world with an operational rate of 492.64 MFLOPS/W [12]. Intel's Tera-scale research project has developed the world's first programmable processor that delivers supercomputer-like performance from a single 80-core chip, which uses less electricity than most of today's home appliances and achieves over 16.29 GFLOPS/W [13].

The second approach to reducing energy consumption in computing systems is the method of power-aware software design at various levels, including operating system level power management, compiler level power management, application level power management, and cross-layer (from transistors to applications) adaptations. The power reduction technique discussed in this chapter belongs to the operating system level, which we elaborate in the next section.

1.1.3 Dynamic Power Management

Software techniques for power reduction are supported by a mechanism called *dynamic voltage scaling* (equivalently, dynamic frequency scaling, dynamic speed scaling, and dynamic power scaling). Many modern components allow voltage regulation to be controlled through software, for example, the BIOS or applications such as PowerStrip. It is usually possible to control the voltages supplied to the CPUs, main memories, local buses, and expansion cards [14]. Processor power consumption is proportional to frequency and the square of supply voltage. A power-aware algorithm can change supply voltage and frequency at appropriate times to optimize a combined consideration of

performance and energy consumption. There are many existing technologies and commercial processors that support dynamic voltage (frequency, speed, power) scaling. SpeedStep is a series of dynamic frequency scaling technologies built into some Intel microprocessors that allow the clock speed of a processor to be dynamically changed by software [15]. LongHaul is a technology developed by VIA Technologies, which supports dynamic frequency scaling and dynamic voltage scaling. By executing specialized operating system instructions, a processor driver can exercise fine control on the bus-to-core frequency ratio and core voltage according to how much load is put on the processor [16]. LongRun and LongRun2 are power management technologies introduced by Transmeta. LongRun2 has been licensed to Fujitsu, NEC, Sony, Toshiba, and NVIDIA [17].

Dynamic power management at the operating system level refers to supply voltage and clock frequency adjustment schemes implemented while tasks are running. These energy conservation techniques explore the opportunities for tuning the energy-delay tradeoff [18]. Power-aware task scheduling on processors with variable voltages and speeds has been extensively studied since the mid-1990s. In a pioneering paper [19], the authors first proposed an approach to energy saving by using fine grain control of CPU speed by an operating system scheduler. The main idea is to monitor CPU idle time and to reduce energy consumption by reducing clock speed and idle time to a minimum. In a subsequent work [20], the authors analyzed offline and online algorithms for scheduling tasks with arrival times and deadlines on a uniprocessor computer with minimum energy consumption. These research have been extended in References [21–27] and inspired substantial further investigation, much of which focus on real-time applications, namely, adjusting the supply voltage and clock frequency to minimize CPU energy consumption while still meeting the deadlines for task execution. In References [28–42] and many other related work, the authors addressed the problem of scheduling independent or precedence constrained tasks on uniprocessor or multiprocessor computers where the actual execution time of a task may be less than the estimated worst-case execution time. The main issue is energy reduction by slack time reclamation.

1.1.4 Task Scheduling with Energy and Time Constraints

There are two considerations in dealing with the energy-delay tradeoff. On the one hand, in high performance computing systems, power-aware design techniques and algorithms attempt to maximize performance under certain energy consumption constraints. On the other hand, low power and energy-efficient design techniques and algorithms aim to minimize energy consumption while still meeting certain performance requirements. In Reference 43, the author studied the problems of minimizing the expected execution time given a hard energy budget and minimizing the expected energy expenditure given a hard execution deadline for a single task with randomized execution requirement. In Reference 44, the author considered scheduling jobs with equal requirements on multiprocessors. In Reference 45, the authors studied the relationship among parallelization,

performance, and energy consumption, and the problem of minimizing energy-delay product. In References 46, 47, the authors attempted joint minimization of energy consumption and task execution time. In Reference 48, the authors investigated the problem of system value maximization subject to both time and energy constraints.

In this chapter, we address energy and time constrained power allocation and task scheduling on multiprocessor computers with dynamically variable voltage, frequency, speed, and power as combinatorial optimization problems. In particular, we define the problem of minimizing schedule length with energy consumption constraint and the problem of minimizing energy consumption with schedule length constraint on multiprocessor computers [49]. The first problem has applications in general multiprocessor and multicore processor computing systems, where energy consumption is an important concern, and in mobile computers, where energy conservation is a main concern. The second problem has applications in real-time multiprocessing systems and environments such as parallel signal processing, automated target recognition, and real-time MPEG encoding, where timing constraint is a major requirement. Our scheduling problems are defined such that the energy-delay product is optimized by fixing one factor and minimizing the other.

1.1.5 Chapter Outline

The rest of the chapter is organized as follows: In Section 1.2, we present the power consumption model; define our power allocation and task scheduling problems on multiprocessor computers with energy and time constraints; describe various task models, processor models, and scheduling models; discuss problem decomposition and subproblems; and mention different types of algorithms. In Section 1.3, we develop optimal solution to our problems on uniprocessor computers and multiprocessor computers with given partitions of tasks, prove the strong NP-hardness of our problems, derive lower bounds for optimal solutions, and the energy-delay tradeoff theorem. In Section 1.4, we present and analyze the performance of pre-power-determination algorithms, including equal-time algorithms, equal-energy algorithms, and equal-speed algorithms. We show both numerical data and simulation results of our performance bounds. In Section 1.5, we present and analyze the performance of post-power-determination algorithms. We demonstrate both numerical data and simulation results of our performance bounds. In Section 1.6, we summarize the chapter and point out several further research directions.

1.2 PRELIMINARIES

1.2.1 Power Consumption Model

Power dissipation and circuit delay in digital CMOS circuits can be accurately modeled by simple equations, even for complex microprocessor circuits. CMOS

circuits have dynamic, static, and short-circuit power dissipation; however, the dominant component in a well-designed circuit is dynamic power consumption p (i.e., the switching component of power), which is approximately $p = aCV^2f$, where a is an activity factor, C is the loading capacitance, V is the supply voltage, and f is the clock frequency [50]. Since $s \propto f$, where s is the processor speed, and $f \propto V^\phi$ with $0 < \phi \leq 1$ [51], which implies that $V \propto f^{1/\phi}$, we know that the power consumption is $p \propto f^\alpha$ and $p \propto s^\alpha$, where $\alpha = 1 + 2/\phi \geq 3$.

Assume that we are given n independent sequential tasks to be executed on m identical processors. Let r_i represent the execution requirement (i.e., the number of CPU cycles or the number of instructions) of task i, where $1 \leq i \leq n$. We use p_i (V_i, f_i, respectively) to represent the power (supply voltage, clock frequency, respectively) allocated to execute task i. For ease of discussion, we will assume that p_i is simply s_i^α, where $s_i = p_i^{1/\alpha}$ is the execution speed of task i. The execution time of task i is $t_i = r_i/s_i = r_i/p_i^{1/\alpha}$. The energy consumed to execute task i is $e_i = p_i t_i = r_i p_i^{1-1/\alpha} = r_i s_i^{\alpha-1}$.

We would like to mention the following number of basic and important observations: (i) $f_i \propto V_i^\phi$ and $s_i \propto V_i^\phi$: Linear change in supply voltage results in up to linear change in clock frequency and processor speed; (ii) $p_i \propto V_i^{\phi+2}$ and $p_i \propto f_i^\alpha$ and $p_i \propto s_i^\alpha$: Linear change in supply voltage results in at least quadratic change in power supply and linear change in clock frequency and processor speed results in at least cubic change in power supply; (iii) $s_i/p_i \propto V_i^{-2}$ and $s_i/p_i \propto s_i^{-(\alpha-1)}$: The processor energy performance, measured by speed per watt [12], is at least quadratically proportional to the supply voltage and speed reduction; (iv) $r_i/e_i \propto V_i^{-2}$ and $r_i/e_i \propto s_i^{-(\alpha-1)}$, where r_i is the amount of work to be performed for task i: The processor energy performance, measured by work per Joule [19], is at least quadratically proportional to the supply voltage and speed reduction; (v) $e_i \propto p_i^{1-1/\alpha} \propto V_i^{(\phi+2)(1-1/\alpha)} = V_i^2$: Linear change in supply voltage results in quadratic change in energy consumption; (vi) $e_i = r_i s_i^{\alpha-1}$: Linear change in processor speed results in at least quadratic change in energy consumption; (vii) $e_i = r_i p_i^{1-1/\alpha}$: Energy consumption reduces at a sublinear speed, as power supply reduces; (viii) $e_i t_i^{\alpha-1} = r_i^\alpha$ and $p_i t_i^\alpha = r_i^\alpha$: For a given task, there exist energy-delay and power-delay tradeoffs. (Later, we will extend such tradeoff to a set of tasks, i.e., the energy-delay tradeoff theorem.)

1.2.2 Problem Definitions

The power allocation and task scheduling problems on multiprocessor computers with energy and time constraints addressed in this chapter are defined as the following optimization problems.

Problem 1.1 (Minimizing Schedule Length with Energy Consumption Constraint)

Input: A set of n independent sequential tasks, a multiprocessor computer with m identical processors, and energy constraint E.

Output: *Power supplies* $p_1, p_2, \ldots, p_n$ to the n tasks and a schedule of the n tasks on the m processors such that the schedule length is minimized and the total energy consumption does not exceed E.

Problem 1.2 (Minimizing Energy Consumption with Schedule Length Constraint)

Input: A set of n independent sequential tasks, a multiprocessor computer with m identical processors, and time constraint T.

Output: Power supplies $p_1, p_2, \ldots, p_n$ to the n tasks and a schedule of the n tasks on the m processors such that the total energy consumption is minimized and the schedule length does not exceed T.

The framework of investigation can be established based on the product of three spaces, namely, the task models, the processors models, and the scheduling models. The above research problems have many variations and extensions, depending on the task models, processors models, and scheduling models. These power allocation and task scheduling problems can be investigated in a variety of ways to consider sophisticated application environments, realistic processor technologies, and practical scheduling algorithms.

1.2.3 Task Models

Our independent sequential tasks can be extended to precedence constrained tasks, parallel tasks, and dynamic tasks, which arise in various application environments.

- *Independent and Precedence Constrained Tasks*. A set of independent tasks can be scheduled in any order. A set of n precedence constrained tasks can be represented by a partial order $\prec$ on the tasks, that is, for two tasks i and j, if $i \prec j$, then task j cannot start its execution until task i finishes. It is clear that the n tasks and the partial order $\prec$ can be represented by a directed task graph, in which, there are n vertices for the n tasks and (i, j) is an arc if and only if $i \prec j$. Furthermore, such a task graph must be a directed acyclic graph (dag).
- *Sequential and Parallel Tasks*. A sequential task requires one processor to execute. A parallel task requires several processors to execute. Assume that task i requires π_i processors to execute and any π_i of the m processors can be allocated to task i. We call π_i the *size* of task i. It is possible that in executing task i, the π_i processors may have different execution requirements. Let r_i represent the maximum execution requirement on the π_i processors executing task i. The execution time of task i is $t_i = r_i/s_i = r_i/p_i^{1/\alpha}$. Note that all the π_i processors allocated to task i have the same speed s_i for duration t_i, although some of the π_i processors may be idle for some time. The energy consumed to execute task i is $e_i = \pi_i p_i t_i = \pi_i r_i p_i^{1-1/\alpha} = \pi_i r_i s_i^{\alpha-1}$.

- *Static and Dynamic Tasks*. A set of tasks are static if they are all available for scheduling at the same time. A schedule can be determined before the execution of any task. A set of tasks are dynamic if each task has its own arrival time. A scheduling algorithm should be able to schedule currently available tasks without knowing the arrival of future tasks.

1.2.4 Processor Models

The following processor technologies can be incorporated into our power allocation and task scheduling problems.

- *Continuous and Discrete Voltage/Frequency/Speed/Power Levels*. Most existing research assume that tasks can be supplied with any power and processors can be set at any speed, that is, voltage/frequency/speed/power can be changed continuously. However, the currently available processors have only discrete voltage/frequency/speed/power settings [40, 52, 53]. Such discrete settings certainly make our optimization problems more difficult to solve.
- *Bounded and Unbounded Voltage/Frequency/Speed/Power Levels*. Much existing research also assumes that voltage/frequency/speed/power can be changed in any range. However, the currently available processors can only change voltage/frequency/speed/power in certain bounded range. Power-aware task scheduling algorithms developed with such constraints, though more complicated, will be more practically useful.
- *Regular and Irregular Voltage/Frequency/Speed/Power Levels*. Much existing research also assume that voltage/frequency/speed/power can be changed according to certain analytical and mathematical relation. However, real processors hardly follow such regular models and exhibit irregular relation among voltage, frequency, speed, and power. Such irregularity makes analytical study of algorithms very hard.
- *Homogeneous and Heterogeneous Processors*. A multiprocessor computer is homogeneous if all the processors have the same power–speed relationship. A multiprocessor computer is heterogeneous with $\alpha_1, \alpha_2, \ldots, \alpha_m$, if each processor k has its own α_k, such that power dissipation on processor k is $\propto s_k^{\alpha_k}$, where $1 \leq k \leq m$. Heterogeneity makes the scheduling of sequential tasks more difficult and the specification of parallel tasks more sophisticated.
- *Overheads for Voltage/Frequency/Speed/Power Adjustment and Idle Processors*. In reality, it takes time and consumes energy to change voltage, frequency, speed, and power. A processor also consumes energy when it is idle [40]. Although these overheads are ignored in most existing research, it would be interesting to take these overheads into consideration to produce more realistic solutions.
- *Single and Multiple Systems*. Processors can reside on a single computing system or across multiple computing systems.

1.2.5 Scheduling Models

As in traditional scheduling theory, different types of scheduling algorithms can be considered for power-aware task scheduling problems.

- *Preemptive and Nonpreemptive Scheduling*. In a nonpreemptive schedule, the execution of a task cannot be interrupted. Once a task is scheduled on a processor, the task runs with the same power supply until it is completed. In a preemptive schedule, the execution of a task can be interrupted at any time and resumed later. When the execution of a task is resumed, the task may be assigned to a different processor, supplied with different power, and executed at different speed. Depending on the processor model, such resumption may be performed with no cost or with overheads for relocation and/or voltage/frequency/speed/power adjustment.
- *Online and Offline Scheduling*. An offline scheduling algorithm knows all the information (execution requirements, precedence constraints, sizes, arrival times, deadlines, etc.) of the tasks to be scheduled. An online algorithm schedules the tasks in certain given order. When task j is scheduled, an online algorithm only knows the information of tasks $1, 2, \ldots, j$ but does not know the information of tasks $j + 1, j + 2, \ldots$ Current tasks should be scheduled without any knowledge of future tasks.
- *Clairvoyant and Non-Clairvoyant Scheduling*. Virtually all research in scheduling theory has been concerned with clairvoyant scheduling, where it is assumed that the execution requirements of the tasks are known a priori. However, in many applications, the execution requirement of a task is not available until the task is executed and completed. A non-clairvoyant scheduling algorithm only knows the precedence constraints, sizes, arrival times, and deadlines of the tasks and has no access to information about the execution requirements of the tasks it is to schedule. The execution requirement of a task is known only when it is completed.

1.2.6 Problem Decomposition

Our power allocation and task scheduling problems contain four nontrivial subproblems, namely, system partitioning, precedence constraining, task scheduling, and power supplying. Each subproblem should be solved efficiently, so that heuristic algorithms with overall good performance can be developed.

- *System Partitioning*. Since each parallel task requests for multiple processors, a multiprocessor computer should be partitioned into clusters of processors to be assigned to the tasks.
- *Precedence Constraining*. Precedence constraints make design and analysis of heuristic algorithms more difficult.
- *Task Scheduling*. Precedence constrained parallel tasks are scheduled together with system partitioning and precedence constraining, and it is NP-hard even when scheduling independent sequential tasks without system partitioning and precedence constraint.

- *Power Supplying*. Tasks should be supplied with appropriate powers and execution speeds, such that the schedule length is minimized by consuming given amount of energy or the energy consumed is minimized without missing a given deadline.

The above decomposition of our optimization problems into several subproblems makes design and analysis of heuristic algorithms tractable. Our approach is significantly different from most existing studies. A unique feature of our work is to compare the performance of our algorithms with optimal solutions analytically and validate our results experimentally, and not to compare the performance of heuristic algorithms among themselves only experimentally. Such an approach is consistent with traditional scheduling theory.

1.2.7 Types of Algorithms

There are naturally three types of power-aware task scheduling algorithms, depending on the order of power supplying and task scheduling.

- *Pre-Power-Determination Algorithms*. In this type of algorithms, we first determine power supplies and then schedule the tasks.
- *Post-Power-Determination Algorithms*. In this type of algorithms, we first schedule the tasks and then determine power supplies.
- *Hybrid Algorithms*. In this type of algorithms, scheduling tasks and determining power supplies are interleaved among different stages of an algorithm.

1.3 PROBLEM ANALYSIS

Our study in this chapter assumes the following models, namely, task model: independent, sequential, static tasks; processor model: a single system of homogeneous processors with continuous and unbounded and regular voltage/frequency/speed/power levels and without overheads for voltage/frequency/speed/power adjustment and idle processors; scheduling model: nonpreemptive, offline, clairvoyant scheduling. The above combination of task model, processor model, and scheduling model yields the easiest version of our power allocation and task scheduling problems.

1.3.1 Schedule Length Minimization

1.3.1.1 Uniprocessor computers. It is clear that on a uniprocessor computer with energy constraint E, the problem of minimizing schedule length with energy consumption constraint is simply to find the power supplies $p_1, p_2, \ldots, p_n$, such that the schedule length

$$T(p_1, p_2, \ldots, p_n) = \frac{r_1}{p_1^{1/\alpha}} + \frac{r_2}{p_2^{1/\alpha}} + \cdots + \frac{r_n}{p_n^{1/\alpha}}$$

is minimized and the total energy consumed $e_1 + e_2 + \cdots + e_n$ does not exceed E, that is,

$$F(p_1, p_2, \ldots, p_n) = r_1 p_1^{1-1/\alpha} + r_2 p_2^{1-1/\alpha} + \cdots + r_n p_n^{1-1/\alpha} \leq E.$$

Notice that both the schedule length $T(p_1, p_2, \ldots, p_n)$ and the energy consumption $F(p_1, p_2, \ldots, p_n)$ are viewed as functions of $p_1, p_2, \ldots, p_n$.

We can minimize $T(p_1, p_2, \ldots, p_n)$ subject to the constraint $F(p_1, p_2, \ldots, p_n) = E$ by using the Lagrange multiplier system:

$$\nabla T(p_1, p_2, \ldots, p_n) = \lambda \nabla F(p_1, p_2, \ldots, p_n),$$

where λ is a Lagrange multiplier. Since

$$\frac{\partial T(p_1, p_2, \ldots, p_n)}{\partial p_i} = \lambda \cdot \frac{\partial F(p_1, p_2, \ldots, p_n)}{\partial p_i},$$

that is,

$$r_i \left(-\frac{1}{\alpha}\right) \frac{1}{p_i^{1+1/\alpha}} = \lambda r_i \left(1 - \frac{1}{\alpha}\right) \frac{1}{p_i^{1/\alpha}},$$

where $1 \leq i \leq n$, we have $p_i = 1/\lambda(1-\alpha)$, for all $1 \leq i \leq n$. Substituting the above p_i into the constraint $F(p_1, p_2, \ldots, p_n) = E$, we get $R\,(1/\lambda(1-\alpha))^{1-1/\alpha} = E$, where $R = r_1 + r_2 + \cdots + r_n$ is the total execution requirement of the n tasks. Therefore, we obtain $p_i = 1/\lambda(1-\alpha) = (E/R)^{\alpha/(\alpha-1)}$, for all $1 \leq i \leq n$.

The above discussion is summarized in the following theorem, which gives the optimal power supplies and the optimal schedule length.

Theorem 1.1 *On a uniprocessor computer, the schedule length is minimized when all tasks are supplied with the same power* $p_i = (E/R)^{\alpha/(\alpha-1)}$, *where* $1 \leq i \leq n$. *The optimal schedule length is* $T_{\text{OPT}} = R^{\alpha/(\alpha-1)}/E^{1/(\alpha-1)}$.

1.3.1.2 Multiprocessor computers. Let us consider a multiprocessor computer with m processors. Assume that a set of n tasks is partitioned into m groups, such that all the tasks in group k are executed on processor k, where $1 \leq k \leq m$. Let R_k denote group k and the total execution requirement of the tasks in group k. For a given partition of the n tasks into m groups $R_1, R_2, \ldots, R_m$, we are seeking power supplies that minimize the schedule length.

Let E_k be the energy consumed by all the tasks in group k. We observe that by fixing E_k and adjusting the power supplies for the tasks in group k to the same power $(E_k/R_k)^{\alpha/(\alpha-1)}$ according to Theorem 1.1, the total execution time of the tasks in group k can be minimized to $T_k = R_k^{\alpha/(\alpha-1)}/E_k^{1/(\alpha-1)}$. Therefore, the problem of finding power supplies $p_1, p_2, \ldots, p_n$, which minimize the schedule length is equivalent to finding $E_1, E_2, \ldots, E_m$, which minimize the

schedule length. It is clear that the schedule length is minimized when all the m processors complete their execution of the m groups of tasks at the same time T, that is, $T_1 = T_2 = \cdots = T_m = T$, which implies that $E_k = R_k^\alpha / T^{\alpha-1}$. Since $E_1 + E_2 + \cdots + E_m = E$, we have

$$\frac{R_1^\alpha + R_2^\alpha + \cdots + R_m^\alpha}{T^{\alpha-1}} = E,$$

that is,

$$T = \left(\frac{R_1^\alpha + R_2^\alpha + \cdots + R_m^\alpha}{E} \right)^{1/(\alpha-1)}$$

and

$$E_k = \left(\frac{R_k^\alpha}{R_1^\alpha + R_2^\alpha + \cdots + R_m^\alpha} \right) E.$$

Thus, we have proved the following theorem.

Theorem 1.2 *For a given partition $R_1, R_2, \ldots, R_m$ of* n *tasks into* m *groups on a multiprocessor computer, the schedule length is minimized when all the tasks in group* k *are supplied with the same power $(E_k/R_k)^{\alpha/(\alpha-1)}$, where*

$$E_k = \left(\frac{R_k^\alpha}{R_1^\alpha + R_2^\alpha + \cdots + R_m^\alpha} \right) E$$

for all $1 \le k \le m$. The optimal schedule length is

$$T_{\text{OPT}} = \left(\frac{R_1^\alpha + R_2^\alpha + \cdots + R_m^\alpha}{E} \right)^{1/(\alpha-1)}$$

for the above power supplies.

1.3.2 Energy Consumption Minimization

1.3.2.1 Uniprocessor computers. It is clear that on a uniprocessor computer with time constraint T, the problem of minimizing energy consumption with schedule length constraint is simply to find the power supplies $p_1, p_2, \ldots, p_n$, such that the total energy consumption

$$E(p_1, p_2, \ldots, p_n) = r_1 p_1^{1-1/\alpha} + r_2 p_2^{1-1/\alpha} + \cdots + r_n p_n^{1-1/\alpha}.$$

is minimized and the schedule length $t_1 + t_2 + \cdots + t_n$ does not exceed T, that is,

$$F(p_1, p_2, \ldots, p_n) = \frac{r_1}{p_1^{1/\alpha}} + \frac{r_2}{p_2^{1/\alpha}} + \cdots + \frac{r_n}{p_n^{1/\alpha}} \le T$$

The energy consumption $E(p_1, p_2, \ldots, p_n)$ and the schedule length $F(p_1, p_2, \ldots, p_n)$ are viewed as functions of $p_1, p_2, \ldots, p_n$.

We can minimize $E(p_1, p_2, \ldots, p_n)$ subject to the constraint $F(p_1, p_2, \ldots, p_n) = T$ by using the Lagrange multiplier system:

$$\nabla E(p_1, p_2, \ldots, p_n) = \lambda \nabla F(p_1, p_2, \ldots, p_n),$$

where λ is a Lagrange multiplier. Since

$$\frac{\partial E(p_1, p_2, \ldots, p_n)}{\partial p_i} = \lambda \cdot \frac{\partial F(p_1, p_2, ..., p_n)}{\partial p_i},$$

that is,

$$r_i \left(1 - \frac{1}{\alpha}\right) \frac{1}{p_i^{1/\alpha}} = \lambda r_i \left(-\frac{1}{\alpha}\right) \frac{1}{p_i^{1+1/\alpha}},$$

where $1 \le i \le n$, we have $p_i = \lambda/(1-\alpha)$, for all $1 \le i \le n$. Substituting the above p_i into the constraint $F(p_1, p_2, \ldots, p_n) = T$, we get $R\left((1-\alpha)/\lambda\right)^{1/\alpha} = T$ and $p_i = \lambda/(1-\alpha) = (R/T)^\alpha$, for all $1 \le i \le n$.

The above discussion gives rise to the following theorem, which gives the optimal power supplies and the minimum energy consumption.

Theorem 1.3 *On a uniprocessor computer, the total energy consumption is minimized when all tasks are supplied with the same power* $p_i = (R/T)^\alpha$, *where* $1 \le i \le n$. *The minimum energy consumption is* $E_{\text{OPT}} = R^\alpha / T^{\alpha-1}$.

1.3.2.2 Multiprocessor computers. By Theorem 1.3, the energy consumed by tasks in group k is minimized as $E_k = R_k^\alpha / T^{\alpha-1}$ by allocating the same power $(R_k/T)^\alpha$ to all the tasks in group k without missing the time deadline T. The minimum energy consumption is simply

$$E_1 + E_2 + \cdots + E_m = \frac{R_1^\alpha + R_2^\alpha + \cdots + R_m^\alpha}{T^{\alpha-1}}.$$

The following result gives the optimal power supplies that minimize energy consumption for a given partition of n tasks into m groups on a multiprocessor computer.

Theorem 1.4 *For a given partition* $R_1, R_2, \ldots, R_m$ *of* n *tasks into* m *groups on a multiprocessor computer, the total energy consumption is minimized when all the tasks in group* k *are supplied with the same power* $(R_k/T)^\alpha$, *where* $1 \le k \le m$. *The minimum energy consumption is*

$$E_{\text{OPT}} = \frac{R_1^\alpha + R_2^\alpha + \cdots + R_m^\alpha}{T^{\alpha-1}}$$

for the above power supplies.

1.3.3 Strong NP-Hardness

The *sum of powers* problem is defined as follows:

Problem 1.3 (Sum of Powers)

Input: A set of integers $\{r_1, r_2, \ldots, r_n\}$ and an integer $m \geq 2$.

Output: A partition of the set into m disjoint subsets, where the sum of integers in subset k is R_k, $1 \leq k \leq m$, such that $R_1^\alpha + R_2^\alpha + \cdots + R_m^\alpha$ is minimized.

Theorems 1.2 and 1.4 imply that on a multiprocessor computer, the problem of minimizing schedule length with energy consumption constraint and the problem of minimizing energy consumption with schedule length constraint are equivalent to finding a partition $R_1, R_2, \ldots, R_m$ of the n tasks into m groups such that $R_1^\alpha + R_2^\alpha + \cdots + R_m^\alpha$ is minimized. This is exactly the same problem as the sum of powers problem. Hence, we have reached the following theorem.

Theorem 1.5 *On a multiprocessor computer with $m \geq 2$ processors, the problem of minimizing schedule length with energy consumption constraint and the problem of minimizing energy consumption with schedule length constraint are equivalent to the sum of powers problem.*

We can easily prove that the sum of powers problem is NP-hard even when $m = 2$ and $\alpha = 2$. We use a reduction from the well-known *partition problem* [54], that is, to decide whether there is a partition of a set of integers $\{r_1, r_2, \ldots, r_n\}$ into two disjoint subsets, such that $R_1 = R_2$, where R_1 and R_2 are the sums of integers in the two subsets. Let $R = R_1 + R_2$ be the sum of all integers. Since $R_1^2 + R_2^2 = R_1^2 + (R - R_1)^2 = 2(R_1 - R/2)^2 + R^2/2$, we know that $R_1^2 + R_2^2$ is minimized as $R^2/2$ if and only if $R_1 = R/2$, that is, there is a partition. Actually, the following result is known in Reference 54 (p. 225).

Theorem 1.6 *The sum of powers problem is NP-hard in the strong sense for all rational $\alpha > 1$. Consequently, on a multiprocessor computer with $m \geq 2$ processors, the problem of minimizing schedule length with energy consumption constraint and the problem of minimizing energy consumption with schedule length constraint are NP-hard in the strong sense.*

1.3.4 Lower Bounds

Assume that $R_1, R_2, \ldots, R_m$ are continuous variables. By using a Lagrange multiplier system, it is easy to show that the multivariable function

$$f(R_1, R_2, \ldots, R_m) = R_1^\alpha + R_2^\alpha + \cdots + R_m^\alpha$$

subject to the constraint $R_1 + R_2 + \cdots + R_m = R$ is minimized when $R_1 = R_2 = \cdots = R_m = R/m$. If there exists such a partition, we have the optimal schedule

length $T_{\text{OPT}} = ((m/E)\,(R/m)^{\alpha})^{1/(\alpha-1)}$, by Theorem 1.2. Of course, in general, there may not exist such a partition and the above quantity can only serve as a lower bound for the optimal schedule length. The following theorem gives a lower bound for the optimal schedule length T_{OPT} for the problem of minimizing schedule length with energy consumption constraint.

Theorem 1.7 *For the problem of minimizing schedule length with energy consumption constraint on a multiprocessor computer, we have the following lower bound:*

$$T_{\text{OPT}} \geq \left(\frac{m}{E}\left(\frac{R}{m}\right)^{\alpha}\right)^{1/(\alpha-1)}$$

for the optimal schedule length.

Similarly, we know that if there exists a partition that results in $R_1 = R_2 = \cdots = R_m = R/m$, the minimum total energy consumption could be $E_{\text{OPT}} = m\,(R/m)^{\alpha}\,/T^{\alpha-1}$ by Theorem 1.4. The following theorem gives a lower bound for the minimum energy consumption E_{OPT} for the problem of minimizing energy consumption with schedule length constraint.

Theorem 1.8 *For the problem of minimizing energy consumption with schedule length constraint on a multiprocessor computer, we have the following lower bound:*

$$E_{\text{OPT}} \geq m\left(\frac{R}{m}\right)^{\alpha}\frac{1}{T^{\alpha-1}}$$

for the minimum energy consumption.

Since it is infeasible to compute optimal solutions in reasonable amount of time, the lower bounds in Theorems 1.7 and 1.8 can be used to evaluate the performance of heuristic algorithms when they are compared with optimal solutions.

1.3.5 Energy-Delay Trade-off

The lower bounds in Theorems 1.7 and 1.8 essentially state the following important theorem.

$\text{ET}^{\alpha-1}$ Lower Bound Theorem (Energy-Delay Trade-off Theorem). *For any execution of a set of tasks with total execution requirement* R *on* m *processors with schedule length* T *and energy consumption* E*, we must have the following tradeoff:*

$$ET^{\alpha-1} \geq m\left(\frac{R}{m}\right)^{\alpha}$$

by using any scheduling algorithm.

The above energy-delay tradeoff theorem implies that our power allocation and task scheduling problems are defined such that the energy-delay product is optimized by fixing one factor and minimizing the other.

Notice that the lower bounds in Theorems 1.7 and 1.8 and the energy-delay tradeoff theorem are applicable to various sequential task models (independent or precedence constrained, static or dynamic tasks), various processor models (regular homogeneous processors with continuous or discrete voltage/frequency/speed/power levels, bounded or unbounded voltage/frequency/speed/power levels, with/without overheads for voltage/frequency/speed/power adjustment, and idle processors), and all scheduling models (preemptive or nonpreemptive, online or offline, clairvoyant, or non-clairvoyant scheduling). These lower bounds have also been extended to parallel tasks [55].

1.4 PRE-POWER-DETERMINATION ALGORITHMS

1.4.1 Overview

We observe that for independent sequential tasks considered in this chapter, we only need to deal with two subproblems, namely, scheduling tasks and determining power supplies. Depending on which subproblem is solved first, we have two types of power-aware task scheduling algorithm, namely, pre-power-determination algorithms and post-power-determination algorithms.

In pre-power-determination algorithms, we first determine power supplies and then schedule the tasks. Let A_1-A_2 denote a pre-power-determination algorithm, where A_1 is an algorithm for power allocation and A_2 is an algorithm for task scheduling. Algorithm A_1-A_2 works as follows: First, algorithm A_1 is used to assign powers to the n tasks. Second, algorithm A_2 is used to produce a schedule of the n tasks (whose execution times are known) on the m processors.

In this section, we consider the following pre-power-determination algorithms:

- *Equal-Time Algorithms* (ET-A). The power supplies $p_1, p_2, \ldots, p_n$ are determined in such a way that all the n tasks have the identical execution time, that is, $t_1 = t_2 = \cdots = t_n$.
- *Equal-Energy Algorithms* (EE-A). The power supplies $p_1, p_2, \ldots, p_n$ are determined in such a way that all the n tasks consume the same amount of energy, that is, $e_1 = e_2 = \cdots = e_n$.
- *Equal-Speed Algorithms* (ES-A). All the n tasks are supplied with the same power and executed at the same speed, that is, $p_1 = p_2 = \cdots = p_n$ and $s_1 = s_2 = \cdots = s_n$.

In all the above algorithms, A is any task scheduling algorithm.

We propose to use the classic list scheduling algorithm [56] and its variations to solve the task scheduling problem.

- *List Scheduling* (LS). The algorithm works as follows to schedule a list of tasks 1, 2, $\ldots$, n. Initially, task k is scheduled on processor k, where

$1 \le k \le m$, and tasks $1, 2, \ldots, m$ are removed from the list simultaneously. On the completion of a task k, the first unscheduled task in the list, that is, task $m+1$, is removed from the list and scheduled to be executed on processor k. This process repeats until all tasks in the list are finished.

Algorithm LS has many variations depending on the strategy used in the initial ordering of the tasks. We mention two of them here.

- *Largest Requirement First* (LRF). This algorithm is the same as the LS algorithm, except that the tasks are arranged such that $r_1 \ge r_2 \ge \cdots \ge r_n$.
- *Smallest Requirement First* (SRF). This algorithm is the same as the LS algorithm, except that the tasks are arranged such that $r_1 \le r_2 \le \cdots \le r_n$.

We call algorithm LS and its variations simply as *list scheduling algorithms*.

Notice that for equal-time algorithms ET-*A*, since all tasks have the same execution time, all list scheduling algorithms generate the same schedule. Hence, we basically have one algorithm ET-LS. However, for equal-energy algorithms, EE-*A*, and equal-speed algorithms, ES-*A*, different list scheduling algorithms generate different schedules and have different performance. Therefore, we will distinguish algorithms EE-SRF, EE-LS, EE-LRF, and ES-SRF, ES-LS, ES-LRF.

1.4.2 Performance Measures

Let T_A denote the length of the schedule produced by algorithm A and E_A denote the total amount of energy consumed by algorithm A. The following performance measures are used to analyze and evaluate the performance of our power allocation and task scheduling algorithms.

Definition 1.1 The *performance ratio* of an algorithm A that solves the problem of minimizing schedule length with energy consumption constraint is defined as $\beta_A = T_A / T_{\text{OPT}}$. If $\beta_A \le B$, we call B a *performance bound* of algorithm A. The *asymptotic performance ratio* of algorithm A is defined as $\beta_A^\infty = \lim_{R/r^* \to \infty} \beta_A$ (by fixing m), where $r^* = \max\{r_1, r_2, \ldots, r_n\}$ is the maximum task execution requirement. If $\beta_A^\infty \le B$, we call B an *asymptotic performance bound* of algorithm A. Algorithm A is called *asymptotically optimal* if $\beta_A^\infty = 1$.

Definition 1.2 The *performance ratio* of an algorithm A that solves the problem of minimizing energy consumption with schedule length constraint is defined as $\gamma_A = E_A / E_{\text{OPT}}$. If $\gamma_A \le C$, we call C a *performance bound* of algorithm A. The *asymptotic performance ratio* of algorithm A is defined as $\gamma_A^\infty = \lim_{R/r^* \to \infty} \gamma_A$ (by fixing m), where $r^* = \max\{r_1, r_2, \ldots, r_n\}$ is the maximum task execution requirement. If $\gamma_A^\infty \le C$, we call C an *asymptotic performance bound* of algorithm A. Algorithm A is called *asymptotically optimal* if $\gamma_A^\infty = 1$.

When tasks have random execution requirements, T_A, T_{OPT}, β_A, β_A^∞, B, E_A, E_{OPT}, γ_A, γ_A^∞, and C are all random variables. Let $\overline{x}$ be the expectation of a random variable x.

Definition 1.3 If $\beta_A \leq B$, then $\overline{B}$ is an *expected performance bound* of algorithm A. If $\beta_A^{\infty} \leq B$ then $\overline{B}$ is an *expected asymptotic performance bound* of algorithm A.

Definition 1.4 If $\gamma_A \leq C$ then $\overline{C}$ is an *expected performance bound* of algorithm A. If $\gamma_A^{\infty} \leq C$ then $\overline{C}$ is an *expected asymptotic performance bound* of algorithm A.

1.4.3 Equal-Time Algorithms and Analysis

1.4.3.1 Schedule length minimization. To solve the problem of minimizing schedule length with energy consumption constraint E by using the equal-time algorithm ET-LS, we notice that $t_1 = t_2 = \cdots = t_n = t$, that is, $t_i = r_i / p_i^{1/\alpha} = t$, for all $1 \leq i \leq n$, where t is the identical task execution time. The above equation gives $p_i = \left(r_i/t\right)^{\alpha}$, where $1 \leq i \leq n$. Since the total energy consumption is

$$r_1 p_1^{1-1/\alpha} + r_2 p_2^{1-1/\alpha} + \cdots + r_n p_n^{1-1/\alpha} = E,$$

namely,

$$\frac{r_1^{\alpha} + r_2^{\alpha} + \cdots + r_n^{\alpha}}{t^{\alpha-1}} = E,$$

we get

$$t = \left(\frac{r_1^{\alpha} + r_2^{\alpha} + \cdots + r_n^{\alpha}}{E}\right)^{1/(\alpha-1)}.$$

Therefore, the schedule length of algorithm ET-LS is

$$T_{\text{ET-LS}} = \left\lceil \frac{n}{m} \right\rceil t = \left\lceil \frac{n}{m} \right\rceil \left(\frac{r_1^{\alpha} + r_2^{\alpha} + \cdots + r_n^{\alpha}}{E}\right)^{1/(\alpha-1)}.$$

By Theorem 1.7, the performance ratio of algorithm ET-LS is

$$\beta_{\text{ET-LS}} = \frac{T_{\text{ET-LS}}}{T_{\text{OPT}}} \leq m \left\lceil \frac{n}{m} \right\rceil \left(\frac{r_1^{\alpha} + r_2^{\alpha} + \cdots + r_n^{\alpha}}{R^{\alpha}}\right)^{1/(\alpha-1)}.$$

The above discussion is summarized in the following theorem.

Theorem 1.9 *By using the equal-time algorithm ET-LS to solve the problem of minimizing schedule length with energy consumption constraint on a multiprocessor computer, the schedule length is*

$$T_{\text{ET-LS}} = \left\lceil \frac{n}{m} \right\rceil \left(\frac{r_1^{\alpha} + r_2^{\alpha} + \cdots + r_n^{\alpha}}{E}\right)^{1/(\alpha-1)}.$$

The performance ratio is $\beta_{\text{ET-LS}} \le B_{\text{ET-LS}}$, *where the performance bound is*

$$B_{\text{ET-LS}} = m \left\lceil \frac{n}{m} \right\rceil \left(\frac{r_1^\alpha + r_2^\alpha + \cdots + r_n^\alpha}{(r_1 + r_2 + \cdots + r_n)^\alpha} \right)^{1/(\alpha-1)}.$$

1.4.3.2 Energy consumption minimization. To solve the problem of minimizing energy consumption with schedule length constraint T by using the equal-time algorithm ET-LS, we notice that enough energy $E_{\text{ET-LS}}$ should be given such that $T_{\text{ET-LS}} = T$, that is,

$$\left\lceil \frac{n}{m} \right\rceil \left(\frac{r_1^\alpha + r_2^\alpha + \cdots + r_n^\alpha}{E_{\text{ET-LS}}} \right)^{1/(\alpha-1)} = T.$$

The above equation implies that the energy consumed by algorithm ET-LS is

$$E_{\text{ET-LS}} = \left(\left\lceil \frac{n}{m} \right\rceil \frac{1}{T} \right)^{\alpha-1} \left(r_1^\alpha + r_2^\alpha + \cdots + r_n^\alpha \right).$$

By Theorem 1.8, the performance ratio of algorithm ET-LS is

$$\gamma_{\text{ET-LS}} = \frac{E_{\text{ET-LS}}}{E_{\text{OPT}}} \le \left(m \left\lceil \frac{n}{m} \right\rceil \right)^{\alpha-1} \left(\frac{r_1^\alpha + r_2^\alpha + \cdots + r_n^\alpha}{R^\alpha} \right).$$

The above discussion is summarized in the following theorem.

Theorem 1.10 *By using the equal-time algorithm ET-LS to solve the problem of minimizing energy consumption with schedule length constraint on a multiprocessor computer, the energy consumed is*

$$E_{\text{ET-LS}} = \left(\left\lceil \frac{n}{m} \right\rceil \frac{1}{T} \right)^{\alpha-1} \left(r_1^\alpha + r_2^\alpha + \cdots + r_n^\alpha \right).$$

The performance ratio is $\gamma_{\text{ET-LS}} \le C_{\text{ET-LS}}$, *where the performance bound is*

$$C_{\text{ET-LS}} = \left(m \left\lceil \frac{n}{m} \right\rceil \right)^{\alpha-1} \left(\frac{r_1^\alpha + r_2^\alpha + \cdots + r_n^\alpha}{(r_1 + r_2 + \cdots + r_n)^\alpha} \right).$$

1.4.4 Equal-Energy Algorithms and Analysis

1.4.4.1 Schedule length minimization. To solve the problem of minimizing schedule length with energy consumption constraint E by using an equal-energy algorithm EE-A, where A is a list scheduling algorithm, we notice that $e_1 = e_2 = \cdots = e_n = E/n$, that is, $e_i = r_i p_i^{1-1/\alpha} = E/n$, for all

$1 \le i \le n$, where E/n is the identical energy consumption of the n tasks. The above equation gives $p_i = \left(E/nr_i\right)^{\alpha/(\alpha-1)}$, $s_i = p_i^{1/\alpha} = \left(E/nr_i\right)^{1/(\alpha-1)}$, and $t_i = r_i/s_i = r_i^{\alpha/(\alpha-1)}\,(n/E)^{1/(\alpha-1)}$, where $1 \le i \le n$.

Let $A(t_1, t_2, \ldots, t_n)$ represent the length of the schedule produced by algorithm A for n tasks with execution times $t_1, t_2, \ldots, t_n$, where A is a list scheduling algorithm. We notice that for all $x \ge 0$, we have $A(t_1, t_2, \ldots, t_n) = xA(t_1', t_2', \ldots, t_n')$, if $t_i = xt_i'$ for all $1 \le i \le n$. That is, the schedule length is scaled by a factor of x if all the task execution times are scaled by a factor of x. Therefore, we get the schedule length of algorithm EE-A as

$$T_{\text{EE-}A} = A(t_1, t_2, \ldots, t_n) = A(r_1^{\alpha/(\alpha-1)}, r_2^{\alpha/(\alpha-1)}, \ldots, r_n^{\alpha/(\alpha-1)}) \left(\frac{n}{E}\right)^{1/(\alpha-1)}.$$

By Theorem 1.7, the performance ratio of algorithm EE-A is

$$\beta_{\text{EE-}A} = \frac{T_{\text{EE-}A}}{T_{\text{OPT}}} \le \frac{mn^{1/(\alpha-1)} A(r_1^{\alpha/(\alpha-1)}, r_2^{\alpha/(\alpha-1)}, \ldots, r_n^{\alpha/(\alpha-1)})}{R^{\alpha/(\alpha-1)}}.$$

By using any list scheduling algorithm A, we get

$$A(t_1, t_2, \ldots, t_n) \le \frac{t_1 + t_2 + \cdots + t_n}{m} + t^*,$$

where $t^* = \max\{t_1, t_2, \ldots, t_n\}$ is the longest task execution time. Hence, we obtain

$$\begin{aligned}\beta_{\text{EE-}A} &\le \frac{n^{1/(\alpha-1)} \left((r_1^{\alpha/(\alpha-1)} + r_2^{\alpha/(\alpha-1)} + \cdots + r_n^{\alpha/(\alpha-1)}) + m(r^*)^{\alpha/(\alpha-1)}\right)}{R^{\alpha/(\alpha-1)}} \\ &= n^{1/(\alpha-1)} \left(\frac{r_1^{\alpha/(\alpha-1)} + r_2^{\alpha/(\alpha-1)} + \cdots + r_n^{\alpha/(\alpha-1)}}{R^{\alpha/(\alpha-1)}} + m\left(\frac{r^*}{R}\right)^{\alpha/(\alpha-1)}\right),\end{aligned}$$

where $r^* = \max\{r_1, r_2, \ldots, r_n\}$ is the maximum task execution requirement. The asymptotic performance ratio of algorithm EE-A is

$$\beta_{\text{EE-}A}^{\infty} = \lim_{R/r^* \to \infty} \beta_{\text{EE-}A} \le \frac{n^{1/(\alpha-1)}(r_1^{\alpha/(\alpha-1)} + r_2^{\alpha/(\alpha-1)} + \cdots + r_n^{\alpha/(\alpha-1)})}{R^{\alpha/(\alpha-1)}}.$$

The above discussion is summarized in the following theorem.

Theorem 1.11 *By using an equal-energy algorithm EE-*A *to solve the problem of minimizing schedule length with energy consumption constraint on a multiprocessor computer, the schedule length is*

$$T_{\text{EE-}A} = A(r_1^{\alpha/(\alpha-1)}, r_2^{\alpha/(\alpha-1)}, \ldots, r_n^{\alpha/(\alpha-1)}) \left(\frac{n}{E}\right)^{1/(\alpha-1)}.$$

The performance ratio is

$$\beta_{\text{EE-}A} \le n^{1/(\alpha-1)} \left(\frac{r_1^{\alpha/(\alpha-1)} + r_2^{\alpha/(\alpha-1)} + \cdots + r_n^{\alpha/(\alpha-1)}}{R^{\alpha/(\alpha-1)}} + m \left(\frac{r^*}{R} \right)^{\alpha/(\alpha-1)} \right).$$

As $R/r^* \to \infty$, the asymptotic performance ratio is $\beta^{\infty}_{\text{EE-}A} \le B_{\text{EE-}A}$, where the asymptotic performance bound is

$$B_{\text{EE-}A} = \frac{n^{1/(\alpha-1)}(r_1^{\alpha/(\alpha-1)} + r_2^{\alpha/(\alpha-1)} + \cdots + r_n^{\alpha/(\alpha-1)})}{(r_1 + r_2 + \cdots + r_n)^{\alpha/(\alpha-1)}}.$$

1.4.4.2 Energy consumption minimization. To solve the problem of minimizing energy consumption with schedule length constraint T by using an equal-energy algorithm EE-A, we notice that enough energy $E_{\text{EE-}A}$ should be given such that $T_{\text{EE-}A} = T$, that is,

$$A(r_1^{\alpha/(\alpha-1)}, r_2^{\alpha/(\alpha-1)}, \ldots, r_n^{\alpha/(\alpha-1)}) \left(\frac{n}{E_{\text{EE-}A}} \right)^{1/(\alpha-1)} = T.$$

The above equation implies that the energy consumed by algorithm EE-A is

$$E_{\text{EE-}A} = \frac{n}{T^{\alpha-1}} \left(A(r_1^{\alpha/(\alpha-1)}, r_2^{\alpha/(\alpha-1)}, \ldots, r_n^{\alpha/(\alpha-1)}) \right)^{\alpha-1}.$$

By Theorem 1.8, the performance ratio of algorithm EE-A is

$$\begin{aligned} \gamma_{\text{EE-}A} &= \frac{E_{\text{EE-}A}}{E_{\text{OPT}}} \\ &\le n \left(\frac{mA(r_1^{\alpha/(\alpha-1)}, r_2^{\alpha/(\alpha-1)}, \ldots, r_n^{\alpha/(\alpha-1)})}{R^{\alpha/(\alpha-1)}} \right)^{\alpha-1} \\ &\le n \left(\frac{r_1^{\alpha/(\alpha-1)} + r_2^{\alpha/(\alpha-1)} + \cdots + r_n^{\alpha/(\alpha-1)}}{R^{\alpha/(\alpha-1)}} + m \left(\frac{r^*}{R} \right)^{\alpha/(\alpha-1)} \right)^{\alpha-1}. \end{aligned}$$

The asymptotic performance ratio of algorithm EE-A is

$$\gamma^{\infty}_{\text{EE-}A} = \lim_{R/r^* \to \infty} \gamma_{\text{EE-}A} \le \frac{n(r_1^{\alpha/(\alpha-1)} + r_2^{\alpha/(\alpha-1)} + \cdots + r_n^{\alpha/(\alpha-1)})^{\alpha-1}}{R^{\alpha}}.$$

The above discussion is summarized in the following theorem.

Theorem 1.12 *By using an equal-energy algorithm EE-*A *to solve the problem of minimizing energy consumption with schedule length constraint on a multiprocessor computer, the energy consumed is*

$$E_{\text{EE-}A} = \frac{n}{T^{\alpha-1}} \left(A(r_1^{\alpha/(\alpha-1)}, r_2^{\alpha/(\alpha-1)}, \ldots, r_n^{\alpha/(\alpha-1)}) \right)^{\alpha-1}.$$

The performance ratio is

$$\gamma_{\text{EE-}A} \leq n \left(\frac{r_1^{\alpha/(\alpha-1)} + r_2^{\alpha/(\alpha-1)} + \cdots + r_n^{\alpha/(\alpha-1)}}{R^{\alpha/(\alpha-1)}} + m \left(\frac{r^*}{R} \right)^{\alpha/(\alpha-1)} \right)^{\alpha-1}.$$

As $R/r^* \to \infty$*, the asymptotic performance ratio is* $\gamma^{\infty}_{\text{EE-}A} \leq C_{\text{EE-}A}$*, where the asymptotic performance bound is*

$$C_{\text{EE-}A} = \frac{n(r_1^{\alpha/(\alpha-1)} + r_2^{\alpha/(\alpha-1)} + \cdots + r_n^{\alpha/(\alpha-1)})^{\alpha-1}}{(r_1 + r_2 + \cdots + r_n)^{\alpha}},$$

1.4.5 Equal-Speed Algorithms and Analysis

1.4.5.1 Schedule length minimization. To solve the problem of minimizing schedule length with energy consumption constraint E by using an equal-speed algorithm ES-A, we notice that $p_1 = p_2 = \cdots = p_n = p$, that is,

$$E = r_1 p^{1-1/\alpha} + r_2 p^{1-1/\alpha} + \cdots + r_n p^{1-1/\alpha} = R p^{1-1/\alpha},$$

which gives $p = (E/R)^{\alpha/(\alpha-1)}$. Since $s_1 = s_2 = \cdots = s_n = s$, we get $s = p^{1/\alpha} = (E/R)^{1/(\alpha-1)}$ and $t_i = r_i/s = r_i\,(R/E)^{1/(\alpha-1)}$. Hence, we get the schedule length of algorithm ES-A as

$$T_{\text{ES-}A} = A(t_1, t_2, \ldots, t_n) = A(r_1, r_2, \ldots, r_n) \left(\frac{R}{E} \right)^{1/(\alpha-1)}.$$

By Theorem 1.7, the performance ratio of algorithm ES-A is

$$\beta_{\text{ES-}A} = \frac{T_{\text{ES-}A}}{T_{\text{OPT}}} \leq \frac{A(r_1, r_2, \ldots, r_n)}{R/m}.$$

By using any list scheduling algorithm A, we get

$$A(r_1, r_2, \ldots, r_n) \leq \frac{R}{m} + r^*,$$

which implies that

$$\beta_{\text{ES-}A} \leq 1 + \frac{m r^*}{R}.$$

It is clear that for a fixed m, $\beta_{\text{ES-}A}$ can be arbitrarily close to 1 as R/r^* becomes large.

The above discussion yields the following theorem.

Theorem 1.13 *By using an equal-speed algorithm ES-*A *to solve the problem of minimizing schedule length with energy consumption constraint on a multiprocessor computer, the schedule length is*

$$T_{\text{ES-}A} = A(r_1, r_2, \ldots, r_n)\left(\frac{R}{E}\right)^{1/(\alpha-1)}.$$

The performance ratio is

$$\beta_{\text{ES-}A} \leq 1 + \frac{mr}{R}.$$

As $R/r^* \to \infty$, *the asymptotic performance ratio is* $\beta^{\infty}_{\text{ES-}A} = 1$.

1.4.5.2 Energy consumption minimization. To solve the problem of minimizing energy consumption with schedule length constraint T by using an equal-speed algorithm ES-A, we notice that enough energy $E_{\text{ES-}A}$ should be given such that $T_{\text{ES-}A} = T$, that is,

$$A(r_1, r_2, \ldots, r_n)\left(\frac{R}{E_{\text{ES-}A}}\right)^{1/(\alpha-1)} = T.$$

The above equation implies that the energy consumed by algorithm ES-A is

$$E_{\text{ES-}A} = \left(\frac{A(r_1, r_2, \ldots, r_n)}{T}\right)^{\alpha-1} R.$$

By Theorem 1.8, the performance ratio of algorithm ES-A is

$$\gamma_{\text{ES-}A} = \frac{E_{\text{ES-}A}}{E_{\text{OPT}}} \leq \left(\frac{A(r_1, r_2, \ldots, r_n)}{R/m}\right)^{\alpha-1} \leq \left(1 + \frac{mr^*}{R}\right)^{\alpha-1}.$$

As R/r^* becomes large, $\gamma_{\text{ES-}A}$ can be arbitrarily close to 1.

Theorem 1.14 *By using an equal-speed algorithm ES-*A *to solve the problem of minimizing energy consumption with schedule length constraint on a multiprocessor computer, the energy consumed is*

$$E_{\text{ES-}A} = \left(\frac{A(r_1, r_2, \ldots, r_n)}{T}\right)^{\alpha-1} R.$$

The performance ratio is

$$\gamma_{\text{ES-}A} \leq \left(1 + \frac{mr^*}{R}\right)^{\alpha-1}.$$

As $R/r^* \to \infty$*, the asymptotic performance ratio is* $\gamma^{\infty}_{\text{ES-}A} = 1$.

1.4.6 Numerical Data

In Table 1.1, we demonstrate numerical data for the expectation of the performance bound $B_{\text{ET-LS}}$ given in Theorem 1.9 and the expectation of the performance bound $C_{\text{ET-LS}}$ given in Theorem 1.10, where $n = 1, 2, 3, \ldots, 15$ and $\alpha = 3.0, 4.0, 5.0$. For each combination of n and α, we generate 20,000 sets of n random execution requests. In each set, the n execution requests are independent and identically distributed (i.i.d.) random variables uniformly distributed in $[0, 1]$. For each set of n random execution requests $r_1, r_2, \ldots, r_n$, we calculate $B_{\text{ET-LS}}$. The average of the 20,000 values of $B_{\text{ET-LS}}$ is reported as the expected performance bound $\overline{B}_{\text{ET-LS}}$. A similar process is performed to get the expected performance bound $\overline{C}_{\text{ET-LS}}$. The maximum 99% confidence interval of all the data in the table is also given. We observe that as n increases, $\overline{B}_{\text{ET-LS}}$ ($\overline{C}_{\text{ET-LS}}$, respectively) quickly approaches its stable value, that is, the limit $\lim_{n\to\infty} \overline{B}_{\text{ET-LS}}$ ($\lim_{n\to\infty} \overline{C}_{\text{ET-LS}}$, respectively). Both $\overline{B}_{\text{ET-LS}}$ and $\overline{C}_{\text{ET-LS}}$ increase as α increases.

TABLE 1.1 Numerical Data for the Expected Performance Bounds $\overline{B}_{\text{ET-LS}}$ and $\overline{C}_{\text{ET-LS}}$[a]

	$\alpha = 3$		$\alpha = 4$		$\alpha = 5$	
n	$\overline{B}_{\text{ET-LS}}$	$\overline{C}_{\text{ET-LS}}$	$\overline{B}_{\text{ET-LS}}$	$\overline{C}_{\text{ET-LS}}$	$\overline{B}_{\text{ET-LS}}$	$\overline{C}_{\text{ET-LS}}$
1	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000
2	1.2640340	1.6907929	1.2880188	2.4780966	1.3061883	3.8923041
3	1.3532375	1.9228341	1.3904990	3.2061946	1.4189932	5.8662235
4	1.3867956	1.9966842	1.4310920	3.4847355	1.4713283	6.5784617
5	1.3982354	2.0269460	1.4521804	3.5441669	1.4888736	6.6706692
6	1.4057998	2.0470706	1.4584030	3.5018100	1.5022930	6.5576159
7	1.4104677	2.0506264	1.4637247	3.4949204	1.5088842	6.5677028
8	1.4134329	2.0410096	1.4678481	3.4582288	1.5122321	6.3209251
9	1.4156317	2.0348810	1.4711866	3.4471772	1.5159571	6.2137416
10	1.4151582	2.0379807	1.4698276	3.4048056	1.5154895	6.1304240
11	1.4160890	2.0323743	1.4719247	3.3859182	1.5156527	6.0539296
12	1.4139975	2.0254020	1.4739329	3.3727408	1.5190419	6.0878647
13	1.4138615	2.0243764	1.4748107	3.3570116	1.5200183	6.0082183
14	1.4145436	2.0204771	1.4754312	3.3439681	1.5226907	5.8638511
15	1.4136195	2.0204157	1.4739066	3.3324817	1.5193218	5.8842350

[a] 99% confidence interval, ±2.718%.

TABLE 1.2 Numerical Data for the Expected Asymptotic Performance Bounds $\overline{B}_{\text{EE}-A}$ and $\overline{C}_{\text{EE}-A}$[a]

	$\alpha = 3$		$\alpha = 4$		$\alpha = 5$	
n	$\overline{B}_{\text{EE}-A}$	$\overline{C}_{\text{EE}-A}$	$\overline{B}_{\text{EE}-A}$	$\overline{C}_{\text{EE}-A}$	$\overline{B}_{\text{EE}-A}$	$\overline{C}_{\text{EE}-A}$
1	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000
2	1.0883879	1.1970545	1.0545042	1.1854814	1.0386468	1.1819932
3	1.1108389	1.2494676	1.0674159	1.2306497	1.0479554	1.2207771
4	1.1195349	1.2633028	1.0725624	1.2440485	1.0515081	1.2326371
5	1.1232283	1.2708740	1.0749250	1.2505880	1.0531375	1.2394887
6	1.1256815	1.2736246	1.0763288	1.2524318	1.0541672	1.2428750
7	1.1263593	1.2759891	1.0768743	1.2565030	1.0544601	1.2447639
8	1.1286415	1.2738426	1.0771574	1.2545767	1.0551808	1.2436191
9	1.1289966	1.2757221	1.0774763	1.2568825	1.0552782	1.2447769
10	1.1292004	1.2766495	1.0776918	1.2572239	1.0555105	1.2442652
11	1.1291663	1.2781207	1.0783142	1.2570462	1.0556260	1.2471731
12	1.1293388	1.2786435	1.0784883	1.2569814	1.0559882	1.2457928
13	1.1294392	1.2786065	1.0786491	1.2577500	1.0560901	1.2458283
14	1.1291546	1.2797332	1.0786508	1.2576963	1.0561657	1.2480804
15	1.1294269	1.2792081	1.0787218	1.2584264	1.0561504	1.2476369

[a] 99% confidence interval, ±0.375%.

In Table 1.2, we demonstrate numerical data for the expectation of the performance bound $B_{\text{EE-}A}$ given in Theorem 1.11 and the expectation of the performance bound $C_{\text{EE-}A}$ given in Theorem 1.12. The data are obtained using a method similar to that of Table 1.1. It is observed that as n increases, $\overline{B}_{\text{EE-}A}$ ($\overline{C}_{\text{EE-}A}$, respectively) quickly approaches its stable value. Surprisingly, both $\overline{B}_{\text{EE-}A}$ and $\overline{C}_{\text{EE-}A}$ decrease as α increases. It is clear that the asymptotic performance of equal-energy algorithms is better than the performance of equal-time algorithms, especially for large α.

1.4.7 Simulation Results

In this section, we demonstrate some experimental data. Our experimental performance evaluation is based on two performance measures, namely, normalized schedule length and normalized energy consumption.

Definition 1.5 The *normalized schedule length* NSL_A of an algorithm A that solves the problem of minimizing schedule length with energy consumption constraint is defined as

$$\text{NSL}_A = \frac{T_A}{((m/E)(R/m)^{\alpha})^{1/(\alpha-1)}}.$$

According the the above definition, the normalized schedule length of the equal-time algorithm ET-LS is

$$\mathrm{NSL}_{\text{ET-LS}} = m \left\lceil \frac{n}{m} \right\rceil \left(\frac{r_1^\alpha + r_2^\alpha + \cdots + r_n^\alpha}{R^\alpha} \right)^{1/(\alpha-1)}.$$

For an equal-energy algorithm EE-A, the normalized schedule length is

$$\mathrm{NSL}_{\text{EE-}A} = \frac{mn^{1/(\alpha-1)} A(r_1^{\alpha/(\alpha-1)}, r_2^{\alpha/(\alpha-1)}, \ldots, r_n^{\alpha/(\alpha-1)})}{R^{\alpha/(\alpha-1)}}.$$

For an equal-speed algorithm ES-A, the normalized schedule length is

$$\mathrm{NSL}_{\text{ES-}A} = \frac{A(r_1, r_2, \ldots, r_n)}{R/m}.$$

We notice that NSL_A serves as a performance bound for the performance ratio $\beta_A = T_A / T_{\mathrm{OPT}}$ of any algorithm A that solves the problem of minimizing schedule length with energy consumption constraint on a multiprocessor computer. When the r_i's are random variables, T_A, T_{OPT}, β_A, and NSL_A all become random variables. It is clear that for the problem of minimizing schedule length with energy consumption constraint, we have $\overline{\beta}_A \le \overline{\mathrm{NSL}}_A$, that is, the expected performance ratio is no greater than the expected normalized schedule length. (Recall that we use $\overline{x}$ to represent the expectation of a random variable x.)

Definition 1.6 The *normalized energy consumption* NEC_A of an algorithm A that solves the problem of minimizing energy consumption with schedule length constraint is defined as

$$\mathrm{NEC}_A = \frac{E_A}{R^\alpha / (mT)^{\alpha-1}}.$$

According the the above definition, the normalized energy consumption of the equal-time algorithm ET-LS is

$$\mathrm{NEC}_{\text{ET-LS}} = \left(m \left\lceil \frac{n}{m} \right\rceil \right)^{\alpha-1} \left(\frac{r_1^\alpha + r_2^\alpha + \cdots + r_n^\alpha}{R^\alpha} \right).$$

For an equal-energy algorithm EE-A, the normalized energy consumption is

$$\mathrm{NEC}_{\text{EE-}A} = \frac{n(mA(r_1^{\alpha/(\alpha-1)}, r_2^{\alpha/(\alpha-1)}, \ldots, r_n^{\alpha/(\alpha-1)}))^{\alpha-1}}{R^\alpha}.$$

For an equal-speed algorithm ES-A, the normalized energy consumption is

$$\mathrm{NEC}_{\text{ES-}A} = \left(\frac{A(r_1, r_2, \ldots, r_n)}{R/m} \right)^{\alpha-1}.$$

TABLE 1.3 Simulation Results for the Expected NSL[a]

n	ET-LS	EE-SRF	EE-LS	EE-LRF	ES-SRF	ES-LS	ES-LRF
30	1.4160086	1.5788606	1.5358982	1.1830203	1.2950870	1.2777859	1.0570897
40	1.4162681	1.4614432	1.4275593	1.1598898	1.2209157	1.2095402	1.0326068
50	1.4160963	1.3939270	1.3671321	1.1476734	1.1778906	1.1681927	1.0210129
60	1.4142811	1.3501833	1.3289118	1.1419086	1.1484774	1.1398580	1.0147939
70	1.4145643	1.3183999	1.2995623	1.1387841	1.1277784	1.1188316	1.0106644
80	1.4137537	1.2940370	1.2787042	1.1364289	1.1116303	1.1047328	1.0081871
90	1.4141781	1.2760247	1.2622851	1.1350882	1.0990160	1.0933288	1.0065092

[a]99% confidence interval, ±0.355%.

TABLE 1.4 Simulation Results for the Expected NEC[a]

n	ET-LS	EE-SRF	EE-LS	EE-LRF	ES-SRF	ES-LS	ES-LRF
30	2.0166361	2.4942799	2.3687384	1.3987777	1.6795807	1.6387186	1.1184317
40	2.0141396	2.1375327	2.0427624	1.3452555	1.4955667	1.4714876	1.0671827
50	2.0101674	1.9436148	1.8768266	1.3208927	1.3900636	1.3667759	1.0421333
60	2.0079074	1.8256473	1.7667130	1.3062980	1.3195213	1.2992718	1.0294409
70	2.0065212	1.7388610	1.6960039	1.2976417	1.2720398	1.2538434	1.0214559
80	2.0112500	1.6743670	1.6388005	1.2911207	1.2366120	1.2199077	1.0165503
90	2.0061604	1.6282674	1.5961585	1.2881397	1.2087291	1.1947753	1.0129208

[a]99% confidence interval, ±0.720%.

It is noticed that NEC_A is a performance bound for the performance ratio $\gamma_A = E_A/E_{\mathrm{OPT}}$ of any algorithm A that solves the problem of minimizing energy consumption with schedule length constraint on a multiprocessor computer. It is also clear that for the problem of minimizing energy consumption with schedule length constraint, we have $\overline{\gamma}_A \leq \overline{\mathrm{NEC}}_A$, that is, the expected performance ratio is no greater than the expected normalized schedule length.

Notice that for a given power allocation and task scheduling algorithm A, the expected normalized schedule length $\overline{\mathrm{NSL}}_A$ and the expected normalized energy consumption $\overline{\mathrm{NEC}}_A$ are determined by m, n, α, and the probability distribution of the r_i's. In our simulations, the number of processors is set as $m = 10$. The number of tasks is in the range $n = 30, 40, \ldots, 90$. The parameter α is set as 3. The r_i's are i.i.d. random variables with a uniform distribution in [0, 1].

In Tables 1.3 and 1.4, we show our simulation results. For each combination of n and algorithm $A \in$ {ET-LS, EE-SRF, EE-LS, EE-LRF, ES-SRF, ES-LS, ES-LRF}, we generate 5000 sets of n tasks, produce their schedules by using algorithm A, calculate their NSL_A (or NEC_A), and report the average of NSL_A (or NEC_A), which is the experimental value of $\overline{\mathrm{NSL}}_A$ (or $\overline{\mathrm{NEC}}_A$). The 99% confidence interval of all the data is also given in the same table. We observe the following facts:

- The equal-time algorithm ET-LS exhibits quite stable performance. The expected normalized schedule length $\overline{\text{NSL}}_{\text{ET-LS}}$ (the expected normalized energy consumption $\overline{\text{NEC}}_{\text{ET-LS}}$, respectively) is almost identical to the expected performance bound $\overline{B}_{\text{ET-LS}}$ ($\overline{C}_{\text{ET-LS}}$, respectively) given in Table 1.1.
- The performance of equal-energy algorithms improves as n increases. The expected normalized schedule length $\overline{\text{NSL}}_{\text{EE-}A}$ (the expected normalized energy consumption $\overline{\text{NEC}}_{\text{EE-}A}$, respectively) decreases as n increases, that is, R/r^* increases, and eventually approaches the expected performance bound $\overline{B}_{\text{EE-}A}$ ($\overline{C}_{\text{EE-}A}$, respectively) given in Table 1.2. The speed of convergence depends on algorithm A. It is clear that algorithm LRF leads to faster speed of convergence than LS and SRF.
- The performance of equal-speed algorithms improves as n increases. The expected normalized schedule length $\overline{\text{NSL}}_{\text{ES-}A}$ and the expected normalized energy consumption $\overline{\text{NEC}}_{\text{ES-}A}$ decrease as n increases, that is, R/r^* increases, and eventually approaches 1, as claimed in Theorems 1.13 and 1.14. Again, algorithm LRF leads to faster speed of convergence than LS and SRF.
- The performance of the three list scheduling algorithms are ranked as SRF, LS, LRF, from the worst to the best. Algorithm EE-LRF performs noticeably better than EE-SRF and EE-LS. Similarly, Algorithm ES-LRF performs noticeably better than ES-SRF and ES-LS. This is not surprising since LRF schedules tasks with long execution times earlier and cause less imbalance of task distribution among the processors. On the other hand, SRF schedules tasks with short execution times earlier, and tasks with long execution times scheduled later cause more imbalance of task distribution among the processors. It is known that LRF exhibits better performance in other scheduling environments.
- The equal-time algorithm ET-LS performs better than equal-energy algorithms EE-SRF and EE-LS for small n. As n gets larger, ET-LS performs worse than EE-A and ES-A for all A. The equal-speed algorithm ES-A performs better than the equal-energy algorithm EE-A for all A. For large n, the performance of the seven pre-power-determination algorithms are ranked as ET-LS, EE-SRF, EE-LS, EE-LRF, ES-SRF, ES-LS, ES-LRF, from the worst to the best.

1.5 POST-POWER-DETERMINATION ALGORITHMS

1.5.1 Overview

As mentioned earlier, both the problem of minimizing schedule length with energy consumption constraint and the problem of minimizing energy consumption with schedule length constraint on a multiprocessor computer are equivalent to the sum of powers problem in the sense that they can be solved by finding

a partition $R_1, R_2, \ldots, R_m$ of the n tasks into m groups such that the sum of powers $R_1^\alpha + R_2^\alpha + \cdots + R_m^\alpha$ is minimized. Such a partition is essentially a schedule of the n tasks on m processors. Once a partition (i.e., a schedule) is determined, Theorems 1.2 and 1.4 can be used to decide actual power supplies, which minimize either schedule length or energy consumption. This is exactly the idea of post-power-determination algorithms, where we first schedule the tasks and then determine power supplies, that is, power supplies $p_1, p_2, \ldots, p_n$ are determined after a schedule of the n tasks on the m processors is decided, and a schedule is produced without knowing the actual task execution times but based only on task execution requirements.

Again, we can decompose our optimization problems into two subproblems, namely, scheduling tasks and determining power supplies. We use the notation A_1-A_2 to represent a post-power-determination algorithm, where A_1 is an algorithm for task scheduling and A_2 is an algorithm for power allocation. Algorithm A_1-A_2 works as follows: First, algorithm A_1 is used to produce a schedule of the n tasks (whose execution times are unknown) by using $r_1, r_2, \ldots, r_n$ as task execution times. Second, algorithm A_2 is used to assign powers to the n tasks on the m processors. We propose to use the list scheduling algorithm and its variations to solve the scheduling problem (i.e., the sum of powers problem). Since our power allocation algorithms based on Theorems 1.2 and 1.4 yields optimal solutions, we have post-power-determination algorithm LS-OPT, SRF-OPT, and LRF-OPT.

1.5.2 Analysis of List Scheduling Algorithms

1.5.2.1 Analysis of algorithm LS. Let P_{LS} be the sum of powers of the partition of a list of tasks into m groups produced by algorithm LS, and P_{OPT} be the minimum sum of powers of an optimal partition of the list of tasks. The following theorem characterizes the performance of algorithm LS in solving the sum of powers problem.

Theorem 1.15 *By using algorithm LS to solve the sum of powers problem for a list of tasks, we have* $P_{\text{LS}}/P_{\text{OPT}} \leq B_{\text{LS}}$, *where the performance bound is*

$$B_{\text{LS}} = \max_{\substack{1 \leq m' \leq m-1 \\ 0 \leq r \leq 1/m'}} \left\{ \frac{(m-m')\left(\frac{1-m'r}{m}\right)^\alpha + m'\left(\frac{1-m'r}{m} + r\right)^\alpha}{\left(r \leq \frac{1}{m}\right) \; ? \; \frac{1}{m^{\alpha-1}} : r^\alpha + (m-1)\left(\frac{1-r}{m-1}\right)^\alpha} \right\}.$$

(Note: An expression in the form $(c) \; ? \; u : v$ *means that if a boolean condition* c *is true, the value of the expression is* u*; otherwise, the value of the expression is* v.*)*

The proof of the above theorem is lengthy and sophisticated. The interested reader is referred to Reference 49 for the proof.

1.5.2.2 Analysis of algorithm LRF. Let P_{LRF} be the sum of powers of the partition of a list of tasks into m groups produced by algorithm LRF. The following theorem characterizes the performance of algorithm LRF in solving the sum of powers problem.

Theorem 1.16 *By using algorithm LRF to solve the sum of powers problem for a list of tasks, we have* $P_{\mathrm{LRF}}/P_{\mathrm{OPT}} \leq B_{\mathrm{LRF}}$, *where the performance bound is*

$$B_{\mathrm{LRF}} = m^{\alpha-1}\left(\max_{1\leq m'\leq m-1}\left\{(m-m')\left(\frac{m+1-m'}{m(m+1)}\right)^{\alpha}+m'\left(\frac{2m+1-m'}{m(m+1)}\right)^{\alpha}\right\}\right).$$

The above theorem can be proved by following the same reasoning in the proof of Theorem 1.15. Again, the interested reader is referred to Reference 49 for the proof.

1.5.3 Application to Schedule Length Minimization

Theorem 1.15 can be used to analyze the performance of algorithm LS-OPT, which solves the problem of minimizing schedule length with energy consumption constraint on a multiprocessor computer. By Theorem 1.2, the schedule length produced by algorithm LS-OPT is $T_{\mathrm{LS-OPT}} = \left(P_{\mathrm{LS}}/E\right)^{1/(\alpha-1)}$, where P_{LS} is the sum of powers of the partition produced by algorithm LS. Also, the optimal schedule length is $T_{\mathrm{OPT}} = \left(P_{\mathrm{OPT}}/E\right)^{1/(\alpha-1)}$, where P_{OPT} is the minimum sum of powers of an optimal partition. Hence, we get

$$\beta_{\mathrm{LS-OPT}} = \frac{T_{\mathrm{LS-OPT}}}{T_{\mathrm{OPT}}} = \left(\frac{P_{\mathrm{LS}}}{P_{\mathrm{OPT}}}\right)^{1/(\alpha-1)} \leq B_{\mathrm{LS}}^{1/(\alpha-1)}.$$

Notice that the condition $R/r^* \to \infty$ is equivalent to $r \to 0$ in Theorem 1.15, and it is easy to see that $\lim_{r\to 0} B_{\mathrm{LS}} = 1$. Thus, we have $\beta_{\mathrm{LS-OPT}}^{\infty} = \lim_{R/r^*\to\infty} \beta_{\mathrm{LS-OPT}} \leq \lim_{r\to 0} B_{\mathrm{LS}}^{1/(\alpha-1)} = 1$.

Theorem 1.17 *By using algorithm LS-OPT to solve the problem of minimizing schedule length with energy consumption constraint on a multiprocessor computer, the schedule length is*

$$T_{\mathrm{LS-OPT}} = \left(\frac{P_{\mathrm{LS}}}{E}\right)^{1/(\alpha-1)}.$$

The performance ratio is $\beta_{\mathrm{LS-OPT}} \leq B_{\mathrm{LS-OPT}} = B_{\mathrm{LS}}^{1/(\alpha-1)}$, *where* B_{LS} *is given by Theorem 1.15. As* $R/r^* \to \infty$, *the asymptotic performance ratio is* $\beta_{\mathrm{LS-OPT}}^{\infty} = 1$.

The following theorem can be obtained in a way similar to that of Theorem 1.17.

Theorem 1.18 *By using algorithm LRF-OPT to solve the problem of minimizing schedule length with energy consumption constraint on a multiprocessor computer, the schedule length is*

$$T_{\mathrm{LRF-OPT}} = \left(\frac{P_{\mathrm{LRF}}}{E}\right)^{1/(\alpha-1)}.$$

The performance ratio is $\beta_{\mathrm{LRF-OPT}} \le B_{\mathrm{LRF-OPT}} = B_{\mathrm{LRF}}^{1/(\alpha-1)}$, *where* B_{LRF} *is given by Theorem 1.16. As* $R/r^* \to \infty$, *the asymptotic performance ratio is* $\beta_{\mathrm{LRF-OPT}}^{\infty} = 1$.

1.5.4 Application to Energy Consumption Minimization

Theorem 1.15 can be used to analyze the performance of algorithm LS-OPT, which solves the problem of minimizing energy consumption with schedule length constraint on a multiprocessor computer. By Theorem 1.4, the energy consumption of the schedule produced by algorithm LS-OPT is $E_{\mathrm{LS-OPT}} = P_{\mathrm{LS}}/T^{\alpha-1}$, where P_{LS} is the sum of powers of the partition produced by algorithm LS. Also, the minimum energy consumption of an optimal schedule is $E_{\mathrm{OPT}} = P_{\mathrm{OPT}}/T^{\alpha-1}$, where P_{OPT} is the minimum sum of powers of an optimal partition. Hence, we get $\gamma_{\mathrm{LS-OPT}} = E_{\mathrm{LS-OPT}}/E_{\mathrm{OPT}} = P_{\mathrm{LS}}/P_{\mathrm{OPT}} \le B_{\mathrm{LS}}$. The asymptotic performance ratio $\gamma_{\mathrm{LS-OPT}}^{\infty}$ can be obtained in a way similar to that of Theorem 1.17.

Theorem 1.19 *By using algorithm LS-OPT to solve the problem of minimizing energy consumption with schedule length constraint on a multiprocessor computer, the energy consumed is*

$$E_{\mathrm{LS-OPT}} = \frac{P_{\mathrm{LS}}}{T^{\alpha-1}}.$$

The performance ratio is $\gamma_{\mathrm{LS-OPT}} \le C_{\mathrm{LS-OPT}} = B_{\mathrm{LS}}$, *where* B_{LS} *is given by Theorem 1.15. As* $R/r^* \to \infty$, *the asymptotic performance ratio is* $\gamma_{\mathrm{LS-OPT}}^{\infty} = 1$.

The following theorem can be obtained in a way similar to that of Theorem 1.19.

Theorem 1.20 *By using algorithm LRF-OPT to solve the problem of minimizing energy consumption with schedule length constraint on a multiprocessor computer, the energy consumed is*

$$E_{\mathrm{LRF-OPT}} = \frac{P_{\mathrm{LRF}}}{T^{\alpha-1}}.$$

The performance ratio is $\gamma_{\mathrm{LRF-OPT}} \le C_{\mathrm{LRF-OPT}} = B_{\mathrm{LRF}}$, *where* B_{LRF} *is given by Theorem 1.16. As* $R/r^* \to \infty$, *the asymptotic performance ratio is* $\gamma_{\mathrm{LRF-OPT}}^{\infty} = 1$.

TABLE 1.5 Numerical Data for the Performance Bounds $B_{\text{LS-OPT}}$ and $C_{\text{LS-OPT}}$

	$\alpha = 3$		$\alpha = 4$		$\alpha = 5$	
m	$B_{\text{LS-OPT}}$	$C_{\text{LS-OPT}}$	$B_{\text{LS-OPT}}$	$C_{\text{LS-OPT}}$	$B_{\text{LS-OPT}}$	$C_{\text{LS-OPT}}$
2	1.3660254	1.8660254	1.3999105	2.7434735	1.4212571	4.0802858
3	1.4168919	2.0075827	1.4721932	3.1907619	1.5098182	5.1963533
4	1.4517046	2.1074462	1.4886206	3.2987700	1.5361359	5.5682478
5	1.5235253	2.3211293	1.5274255	3.5635275	1.5430156	5.6686715
6	1.5653646	2.4503664	1.5695451	3.8665303	1.5814389	6.2547465
7	1.6075236	2.5841321	1.5955042	4.0615694	1.6094683	6.7101114
8	1.6621450	2.7627259	1.6149005	4.2115046	1.6277417	7.0200781
9	1.7031903	2.9008574	1.6495521	4.4884680	1.6399180	7.2325010
10	1.7406107	3.0297256	1.6757104	4.7054035	1.6627810	7.6443430

TABLE 1.6 Numerical Data for the Performance Bounds $B_{\text{LRF-OPT}}$ and $C_{\text{LRF-OPT}}$

	$\alpha = 3$		$\alpha = 4$		$\alpha = 5$	
m	$B_{\text{LRF-OPT}}$	$C_{\text{LRF-OPT}}$	$B_{\text{LRF-OPT}}$	$C_{\text{LRF-OPT}}$	$B_{\text{LRF-OPT}}$	$C_{\text{LRF-OPT}}$
2	1.1547005	1.3333333	1.1885514	1.6790123	1.2141069	2.1728395
3	1.1858541	1.4062500	1.2382227	1.8984375	1.2806074	2.6894531
4	1.2165525	1.4800000	1.2568900	1.9856000	1.3012612	2.8672000
5	1.2360331	1.5277778	1.2893646	2.1435185	1.3286703	3.1165123
6	1.2453997	1.5510204	1.3018050	2.2061641	1.3496519	3.3180818
7	1.2593401	1.5859375	1.3116964	2.2568359	1.3585966	3.4069214
8	1.2636090	1.5967078	1.3236611	2.3191587	1.3675714	3.4978408
9	1.2727922	1.6200000	1.3284838	2.3446000	1.3781471	3.6073000
10	1.2771470	1.6311044	1.3351801	2.3802336	1.3833651	3.6622436

1.5.5 Numerical Data

In Table 1.5, we demonstrate numerical data for the performance bounds in Theorems 1.17 and 19. For each combination of $\alpha = 3, 4, 5$ and $m = 2, 3, \ldots, 10$, we show $B_{\text{LS-OPT}}$ and $C_{\text{LS-OPT}}$.

In Table 1.6, we demonstrate numerical data for the performance bounds in Theorems 1.18 and 20. For each combination of $\alpha = 3, 4, 5$ and $m = 2, 3, \ldots, 10$, we show $B_{\text{LRF-OPT}}$ and $C_{\text{LRF-OPT}}$.

It is clear that algorithm LRF leads to improved performance compared with algorithm LS. Tighter performance bounds can be obtained by more involved analysis.

1.5.6 Simulation Results

In this section, we demonstrate some experimental data.

TABLE 1.7 Simulation Results for the Expected NSL[a]

n	SRF-OPT	LS-OPT	LRF-OPT
30	1.0535521	1.0374620	1.0024673
40	1.0303964	1.0214030	1.0008078
50	1.0195906	1.0134978	1.0003326
60	1.0136363	1.0092786	1.0001669
70	1.0100516	1.0068138	1.0000894
80	1.0076977	1.0052356	1.0000527
90	1.0060781	1.0041218	1.0000335

[a] 99% confidence interval, ±0.058%.

For a post-power-determination algorithm A-OPT, where A is a list scheduling algorithm, the normalized schedule length is

$$\mathrm{NSL}_{A-\mathrm{OPT}} = \left(\frac{R_1^\alpha + R_2^\alpha + \cdots + R_m^\alpha}{m(R/m)^\alpha} \right)^{1/(\alpha-1)},$$

where $R_1, R_2, \ldots, R_n$ is a partition into m groups produced by algorithm A for n tasks. The normalized energy consumption is

$$\mathrm{NEC}_{A-\mathrm{OPT}} = \frac{R_1^\alpha + R_2^\alpha + \cdots + R_m^\alpha}{m(R/m)^\alpha}.$$

In Tables 1.7 and 1.8, we show our simulation results. For each combination of n and algorithm $A \in \{$ SRF-OPT, LS-OPT, LRF-OPT $\}$, we generate 5000 sets of n tasks, produce their schedules by using algorithm A, calculate their NSL_A (or NEC_A), and report the average of NSL_A (or NEC_A), which is the experimental value of $\overline{\mathrm{NSL}}_A$ (or $\overline{\mathrm{NEC}}_A$). The 99% confidence interval of all the data in the same table is also given. We observe the following facts:

- The performance of the three post-power-determination algorithms are ranked as SRF-OPT, LS-OPT, LRF-OPT, from the worst to the best.
- The post-power-determination algorithms perform better (as measured by $\overline{\mathrm{NSL}}_A$ and $\overline{\mathrm{NEC}}_A$) than the pre-power-determination algorithms, although there is no direct comparison among the performance bounds given in Theorems 1.9, 1.11, 1.13, 1.17, and 1.18, and the performance bounds given in Theorems 1.10, 1.12, 1.14, 1.19, and 1.20.

1.6 SUMMARY AND FURTHER RESEARCH

We have investigated nonpreemptive offline non-clairvoyant scheduling of independent sequential static tasks on a single computing system of homogeneous processors with continuous and unbounded and regular voltage/frequency/speed/power levels and without overheads for voltage/frequency/speed/power

TABLE 1.8 Simulation Results for the Expected NEC[a]

n	SRF-OPT	LS-OPT	LRF-OPT
30	1.1102206	1.0765611	1.0051583
40	1.0619973	1.0427680	1.0016418
50	1.0395262	1.0268312	1.0006819
60	1.0274261	1.0187010	1.0003373
70	1.0201289	1.0136876	1.0001829
80	1.0154632	1.0104982	1.0001088
90	1.0122283	1.0082873	1.0000684

[a] 99% confidence interval, ±0.117%.

adjustment and idle processors. We have developed and analyzed pre-power-determination and post-power-determination algorithms, which solve the problems of minimizing schedule length with energy consumption constraint and minimizing energy consumption with schedule length constraint. The performance of all our algorithms is compared with optimal solutions. It is found that the best algorithm among all our algorithms in this chapter is LRF-OPT, whose performance ratio is very close to optimal.

Possible further research can be directed toward precedence constrained tasks, parallel tasks, discrete and/or bounded voltage/frequency/speed/power levels, heterogeneous processors, and online scheduling. These extensions to our study in this chapter are likely to yield analytically tractable algorithms.

ACKNOWLEDGMENT

The materials presented in Sections 1.3 and 1.5 are based in part on the author's work in Reference 49.

REFERENCES

1. Available at http://www.top500.org/.
2. Available at http://en.wikipedia.org/wiki/Moore's_law.
3. Venkatachalam V, Franz M. Power reduction techniques for microprocessor systems. ACM Comput Surv 2005;37(3):195–237.
4. Srivastava MB, Chandrakasan AP, Rroderson RW. Predictive system shutdown and other architectural techniquesfor energy efficient programmable computation. IEEE Trans Very Large Scale Integr (VLSI) Syst 1996;4(1):42–55.
5. Gara MA, Blumrich D, Chen GL.-T, Chiu P, Coteus ME, et al. Overview of the Blue Gene/L system architecture. IBM J Res Dev 2005;49(2/3):195–212.
6. Feng W-C. The importance of being low power in high performance computing. CTWatch Quarterly; 2005;1(3), Los Alamos National Laboratory.
7. Available at http://www.foxnews.com/story/0,2933,479127,00.html.

8. Graham SL, Snir M, Patterson CA, editors. Getting up to speed: the future of supercomputing. *Committee on the Future of Supercomputing*. National Research Council, National Academies Press; Washington, D.C. 2005.
9. Albers S. Energy-efficient algorithms. Commun ACM 2010;53(5):86–96.
10. Benini L, Bogliolo A, De Micheli G. A survey of design techniques for system-level dynamic power management. IEEE Trans Very Large Scale Integrat (VLSI) Syst 2000;8(3):299–316.
11. Unsal OS, Koren I. System-level power-aware design techniques in real-time systems. Proc IEEE 2003;91(7):1055–1069.
12. Available at http://www.green500.org/.
13. Available at http://techresearch.intel.com/articles/Tera-Scale/1449.htm.
14. Available at http://en.wikipedia.org/wiki/Dynamic/voltage/scaling.
15. Available at http://en.wikipedia.org/wiki/SpeedStep.
16. Available at http://en.wikipedia.org/wiki/LongHaul.
17. Available at http://en.wikipedia.org/wiki/LongRun.
18. Stan MR, Skadron K. Guest editors' introduction: power-aware computing. IEEE Comput 2003;36(12):35–38.
19. Weiser M, Welch B, Demers A, Shenker S. Scheduling for reduced CPU energy. In: Proceedings of the 1st USENIXSymposium on Operating Systems Design and Implementation; 1994. pp. 13–23, Monterey, California.
20. Yao F, Demers A, Shenker S. A scheduling model for reduced CPU energy. In: Proceedings of the 36thIEEE Symposium on Foundations of Computer Science; 1995. pp. 374–382, Milwaukee, Wisconsin.
21. Bansal N, Kimbrel T, Pruhs K. Dynamic speed scaling to manage energy and temperature. In: Proceedings of the 45th IEEE Symposium on Foundation of Computer Science; 2004. pp. 520–529, Rome, Italy.
22. Chan H-L, Chan W-T, Lam T-W, Lee L-K, Mak K-S, Wong PWH. Energy efficient online deadline scheduling. In: Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms; 2007. pp. 795–804, New Orleans, Louisiana.
23. Kwon W-C, Kim T. Optimal voltage allocation techniques for dynamically variable voltage processors. ACM Trans Embedded Comput Syst 2005;4(1):211–230.
24. Li M, Liu BJ, Yao FF. Min-energy voltage allocation for tree-structured tasks. J Comb Optim 2006;11:305–319.
25. Li M, Yao AC, Yao FF. Discrete and continuous min-energy schedules for variable voltage processors. Proc Natl Acad Sci U S A 2006;103(11):3983–3987.
26. Li M, Yao FF. An efficient algorithm for computing optimal discrete voltage schedules. SIAM J Comput 2006;35(3):658–671.
27. Yun H-S, Kim J. On energy-optimal voltage scheduling for fixed-priority hard real-time systems. ACM Trans Embedded Comput Syst 2003;2(3):393–430.
28. Aydin H, Melhem R, Mossé D, Mejía-Alvarez P. Power-aware scheduling for periodic real-time tasks. IEEE Trans Comput 2004;53(5):584–600.
29. Hong I, Kirovski D, Qu G, Potkonjak M, Srivastava MB. Power optimization of variable-voltage core-based systems. IEEE Trans Comput Aided Des Integr Circ Syst 1999;18(12):1702–1714.

30. Im C, Ha S, Kim H. Dynamic voltage scheduling with buffers in low-power multimedia applications. ACM Trans Embedded Comput Syst 2004;3(4):686–705.
31. Krishna CM, Lee Y-H. Voltage-clock-scaling adaptive scheduling techniquesfor low power in hard real-time systems. IEEE Trans Comput 2003;52(12):1586–1593.
32. Lee Y-H, Krishna CM. Voltage-clock scaling for low energy consumption infixed-priority real-time systems. Real-Time Syst 2003;24(3):303–317.
33. Lorch JR, Smith AJ. PACE: a new approach to dynamic voltage scaling. IEEE Trans Comput 2004;53(7):856–869.
34. Mahapatra RN, Zhao W. An energy-efficient slack distribution techniquefor multimode distributed real-time embedded systems. IEEE Trans Parallel Distrib Syst 2005;16(7):650–662.
35. Quan G, Hu XS. Energy efficient DVS schedule for fixed-priority real-time systems. ACM Trans Embedded Comput Syst 2007;6(4):Article No. 29.
36. Shin D, Kim J. Power-aware scheduling of conditional task graphs in real-time multiprocessor systems. In: Proceedings of the International Symposium on Low Power Electronics and Design; Seoul, Korea; 2003. pp. 408–413.
37. Shin D, Kim J, Lee S. Intra-task voltage scheduling for low-energy hard real-time applications. IEEE Des Test Comput 2001;18(2):20–30.
38. Yang P, Wong C, Marchal P, Catthoor F, Desmet D, Verkest D, Lauwereins R. Energy-aware runtime scheduling for embedded-multiprocessor SOCs. IEEE Des Test Comput 2001;18(5):46–58.
39. Zhong X, Xu C-Z. Energy-aware modeling and scheduling for dynamic voltage scaling with statistical real-time guarantee. IEEE Trans Comput 2007;56(3):358–372.
40. Zhu D, Melhem R, Childers BR. Scheduling with dynamic voltage/speed adjustmentusing slack reclamation in multiprocessor real-time systems. IEEE Trans Parallel Distrib Syst 2003;14(7):686–700.
41. Zhu D, Mossé D, Melhem R. Power-aware scheduling for AND/OR graphs in real-time systems. IEEE Trans Parallel Distrib Syst 2004;15(9):849–864.
42. Zhuo J, Chakrabarti C. Energy-efficient dynamic task scheduling algorithms for DVS systems. ACM Trans Embedded Comput Syst 2008;7(2):Article No. 17.
43. Barnett JA. Dynamic task-level voltage scheduling optimizations. IEEE Trans Comput 2005;54(5):508–520.
44. Bunde DP. Power-aware scheduling for makespan and flow. In: Proceedings of the 18th ACM Symposium on Parallelism in Algorithms and Architectures; 2006. pp. 190–196, Cambridge, Massachusetts
45. Cho S, Melhem RG. On the interplay of parallelization, program performance, and energy consumption. IEEE Trans Parallel Distrib Syst 2010;21(3):342–353.
46. Khan SU, Ahmad I. A cooperative game theoretical technique for joint optimization of energy consumption and response time in computational grids. IEEE Trans Parallel Distrib Syst 2009;20(3):346–360.
47. Lee YC, Zomaya AY. Energy conscious scheduling for distributed computing systemsunder different operating conditions. IEEE Trans Parallel Distrib Syst. 2011;22(8):1374–1381
48. Rusu C, Melhem R, Mossé D. Maximizing the system value while satisfying time and energy constraints. In: Proceedings of the 23rd IEEE Real-Time Systems Symposium; 2002. pp. 256–265, Austin, Texas

49. Li K. Performance analysis of power-aware task scheduling algorithmson multiprocessor computers with dynamic voltage and speed. IEEE Trans Parallel Distrib Syst 2008;19(11):1484–1497.

50. Chandrakasan AP, Sheng S, Brodersen RW. Low-power CMOS digital design. IEEE J Solid-State Circ 1992;27(4):473–484.

51. Zhai B, Blaauw D, Sylvester D, Flautner K. Theoretical and practical limits of dynamic voltage scaling. In: Proceedings of the 41st Design Automation Conference; California, USA; 2004. pp. 868–873.

52. Intel, *Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor—White Paper*, March 2004.

53. Qu G. What is the limit of energy saving by dynamic voltage scaling. In: Proceedings of the International Conference on Computer-Aided Design; 2001. pp. 560–563, San Jose, California

54. Garey MR, Johnson DS. *Computers and Intractability—A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman; 1979.

55. Li K. Energy efficient scheduling of parallel tasks on multiprocessor computers. J Supercomput. 2012;60(2):223–247

56. Graham RL. Bounds on multiprocessing timing anomalies. SIAM J Appl Math 1969;2:416–429.

CHAPTER 2

POWER-AWARE HIGH PERFORMANCE COMPUTING

RONG GE and KIRK W. CAMERON

2.1 INTRODUCTION

High performance computing (HPC) is indispensable for scientific discovery and technological revolution. Today's HPC computers are able to perform peta (10^{15}) floating-point operations per second by using hundreds of thousands of processing units. Such unprecedented computational capability enables scientists to solve complex problems previously deemed intractable. With the aid of HPC systems, scientists are able to make breakthroughs in a wide spectrum of fields such as nanoscience, fusion, climate modeling, and astrophysics [1, 2].

Computational capability, albeit growing at a relatively healthy rate, still remains a bottleneck to the advances of many national and global priority grand challenge problems. Understanding and mitigating the effects of global warming require finer resolution simulation with regional details in more complex models. Facilitating regional adaption to climate variability and change needs 1000-fold increase to exaflop-scale (10^{18}) [3] in computing power. Nuclear energy science and engineering simulations also require similar computing power to create robust, predictive simulations that have quantifiable uncertainties for reactors over 40–60 year lifetime [4]. Ten exaflop-scale of computing power is necessary for computational simulations of highly efficient combustion design for transportation. To meet the demand of these applications, petascale computers will continue to increase in performance and exascale computers are expected to debut around 2018.

Power and energy are key challenges in future HPC systems, and they must be efficiently used for HPC to be affordable, scalable, and available. Large-scale computer systems have to use massive numbers of processing units for designed peak performance. Consequently, the power and energy consumption is huge.

Energy-Efficient Distributed Computing Systems, First Edition.
Edited by Albert Y. Zomaya and Young Choon Lee.

For example, the Jaguar system at Oak Ridge Nation Laboratory that is ranked as the #1 supercomputer in the top 500 list in November 2009 contains 26,520 compute nodes and 224,162 cores, and delivers 1.76 petaflops at the expense of 7 MW of electrical power [5]. As a rule of thumb, 1 MW of power incurs $1 million energy bill and requires additional 1 MW of power in cooling. Powering and cooling Jaguar and another petascale system to be built by 2012 will cost Oakridge National Lab and University of Tennessee $33 million annually. An exascale computer could consist of ~100 million of processor cores [6]. Even using today's *most* energy-efficient design such as PowerXCell architecture, the exascale computer will consume 1.3 GW of power. Such power consumption is unacceptable. *The practical power is limited around 20 MW*. At present, about 90% of the data center failures and unavailability are due to power outages. The larger power requirement for the future systems is certain to exacerbate data center failures and unavailability.

Energy-efficient HPC requires not only efficient hardware and system design but also aggressive runtime power management to adapt the power use to application demand. Without adaptive power allocation, the exascale systems with the most power-efficient design technology will be around 67 MW [6]. Runtime adaptive power allocation must be applied in order to reduce the system power to acceptable levels *with 20 MW*. With the adaptive power allocation, an application segment that demands very high floating-point operations and operates entirely out of registers can throttle/shut down memory or communication, and a different application segment that demands high DRAM bandwidth and relatively little floating-point operations can throttle down floating-point units.

Adaptive power allocation is challenging because it is employed at component level and on application segments. It requires fine-grain, detailed power and energy profiles of every system component, particularly the processing units, the memory and storage subsystems, and networking. Meanwhile, adaptive power allocation must not sacrifice performance for HPC applications, which in turn require in-depth understanding of the detailed interrelation between application performance and power consumption. Segment level performance quantification for the variety of HPC applications will be challenging, especially provided that their execution patterns vary with the underlying architecture and system scale.

We are in dire need of techniques for obtaining fine-grain, detailed power profiles of system components to identify best adaptive power allocations for energy efficiency. Most existing power profiling research is focused on a specific single computer component at microarchitecture level such as processor [7, 8], disk [9], memory [8], and networking interface [10] using simulation, direct measurement, and analytical estimation. Neither of these techniques can be adapted to all the components on the system nor can they be integrated easily to reveal the power profile of the entire system. Some have studied the power efficiency of parallel and distributed systems at the system or building level [11–13]. However, coarse grained power profiling is not particularly useful for determining exactly where and how power is consumed by an application and the individual components in a distributed system.

We are also in dire need of analytical models for quantifying and predicting the performance impact of adaptive power allocations. Most existing scalability models focus on the performance speedup from parallel processing. They are unable to capture the combined performance scalability at multiple dimensions, including system size, power modes of components, and affinity of processing units.

In this chapter, we introduce the models and techniques that are designed to meet these needs and enable adaptive power allocation for HPC systems and applications. Specifically, the models and techniques include the following:

- *PowerPack hardware/software toolkit* for fine-grain, detailed power profiling of multiple system components, synchronized with application segments in HPC systems.
- *Practical power and performance models* for quantifying the interrelation between application performance and power consumption on HPC systems capable of power adaptation.
- *Model-directed adaptive power allocation* for performance-constrained energy-efficient computing.

The rest of the chapter is organized as follows. Some background about current technology and power consumption are introduced in the next section. Section 2.4 presents the PowerPack tool for fine-grain power and energy profiling for HPC applications and systems. Section 2.5 presents the Power-Aware Speedup model for quantifying the performance impact of dynamic processor power allocation. Section 2.6 describe a model-directed design of adaptive power allocation. Section 2.7 concludes our work.

2.2 BACKGROUND

2.2.1 Current Hardware Technology and Power Consumption

2.2.1.1 Processor power. Complementary metal oxide semiconductor (CMOS) is a technology for constructing integrated circuits for computer components, including microprocessors, microcontrollers, and static RAM. The power consumption of silicon CMOS logic circuits [14] is approximated by

$$P = ACV^2 f + P_{\text{short}} + P_{\text{leak}}. \tag{2.1}$$

The power consumption of CMOS logic consists of three components: dynamic power $P_{\text{d}} = ACV^2 f$, which is caused by the charging and discharging of the capacitive load on each gate's output; short circuit power P_{short}, which is caused by the short-circuit current momentarily flowing within the cell; and leak power P_{leak}, which is caused by leakage current regardless of the gate's state. Here f is the operating frequency, A is the activity of the gates in the system (reduced activity corresponds to smaller A value), C is the total capacitance seen

by the gate outputs, and V is the supply voltage. Dynamic power P_d dominates total power P, accounting for 70% or more on today's CMOS devices, P_{short} typically accounts for 10–30% and P_{leak} accounts for about 1% [15] of total P.[1]

The dominant dynamic power suggests three key strategies for power reduction on today's microprocessors and caches.

1. Using two processing units with a frequency f consumes less power compared with using a single processing unit with doubled frequency $2f$ for the same computational capacity. Multicore architecture follows this strategy. Multicore technology uses multiple slower, low power cores instead of faster single cores. At present, quad-core and hexa-core processors are commonplace on production high performance computing systems, and processors with tens to hundreds of cores have merged [19].
2. Reducing voltage and frequency can exponentially reduce power. Both low power design and power-aware dynamic voltage and frequency scaling (DVFS) technology follow this strategy. Low power processors such as PowerPC used in BlueGene systems run at low frequency. In contrast, high performance processors such as Intel Xeon and AMD Opteron run at high frequency with higher power consumption. A DVFS processor can be switched among several performance states, each determined by a pair of voltage and frequency. Normally, the frequency is proportional to the voltage, and scaling a processor to low frequency results in cubic power reduction on DVFS processors. DVFS technology is available on most commodity server processors. The performance states are normally controlled by operating systems or users.
3. Low activity consumes less power compared to high activity.

2.2.1.2 Memory subsystem power. DRAM is the mainstream memory technology used in HPC systems. Commodity DRAM devices have only recently begun to address power concerns as low power DRAM devices have become standard for applications in mobile phones and portable electronics [6].

To understand the DRAM device power consumption, it is necessary to understand the basic functionality of DRAM devices. DRAM devices have several operating states, including idle, refresh, precharge, active, read, and write. During the idle state the DRAM clock and input buffers are turned off. Refresh is periodically performed on the cells to keep information. In order to activate a bank with a row for reading or writing, the bank must first be charged. The charge is carried out by the precharge operation. The charge allows the chip to sense a particular row and amplify the signal from that row. The active operation presents bank and row addresses and causes a read of the wordline into a sense amplifier. Activating the row is also known as *opening* the row. Once the row has been activated or "opened", Read command is possible to that row by steering

[1] In the future, leakage power (P_{leak}) will grow in significance with continuing decreases in feature size and increases in processor frequency and reducing leakage power will also be important [16–18].

the data from the sense amplifiers, through registers and pipelines to the output. A write operation is similar to a read operation. It begins with precharge and then activate. The data to be written is driven into the cells by the sense amplifiers.

The power consumption of the memory subsystem varies with the states. The power consumption can be classified into three levels: background power, activation power, and read/write power. Background power is the base power consumed by the memory cells for storing data, and it is aggregated over idle, refresh, and precharging states. Activation power is the extra power consumed by addressing the row and copying the data to the sensor amplifier. Read/write power are the power consumption for steering the actual data from the sensor amplifier. The activation power and read/write power are normally larger than the background power.[2] Figure 2.1 shows the power consumption of a DDR3 [20].

The power breakdown on the commodity DRAM device suggests the following:

- Reducing the number of memory rows that are opened reduces power consumption of memory subsystem.
- Reducing the number of reads/writes *and* thus reducing power consumption.

Good memory locality reduces both activation and read/write power consumption.

2.2.2 Performance

Execution time T (or delay D for the same meaning) is the ultimate measure of performance for an application on a given system [21]. The execution time

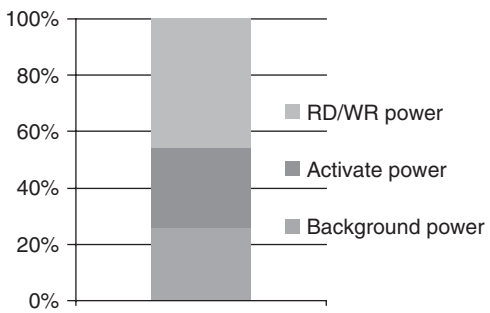

Figure 2.1 The memory subsystem power consumption of a commodity DRAM device. The power consumed by opening a row is a little more than background power. The read/write power dominates the power consumption of the memory subsystem.

[2]The numbers are extracted from Reference 20.

of an application is affected by CPU speed, memory hierarchy, and application execution pattern.

The sequential execution time $T(1)$ for a program on a single processor consists of two parts: the time that the processor is busy executing instructions T_{comp} and the time that the processor waits for data from the local memory system T_{mem} [22], that is,

$$T(1) = T_{\text{comp}}(1) + T_{\text{mem}}(1). \tag{2.2}$$

Memory access is expensive: the latency for a single memory access is equivalent to the time for the CPU to execute hundreds of instructions. Memory time T_{mem} can account for up to 50% of execution time for an application that hits the closest cache for 99% of the data accesses.

The parallel execution time on n processors $T(n)$ consists of additional items that are denoted as parallel overhead. Synchronization time, $T_{\text{sync}}(n)$, is due to load imbalance and serialization. Communication time, $T_{\text{comm}}(n)$, occurs when the processor is stalled waiting for a data transfer from or to a remote processing node. The time that the processor is busy executing extra work, $T_{\text{extra}}(n)$, is due to parallel data decomposition and task assignment. The parallel execution time is formalized as

$$T(n) = T_{\text{comp}}(n) + T_{\text{mem}}(n) + T_{\text{sync}}(n) + T_{\text{comm}}(n) + T_{\text{extra}}(n). \tag{2.3}$$

Parallel overhead, the sum of $T_{\text{sync}}(n)$, $T_{\text{comm}}(n)$, and $T_{\text{extra}}(n)$), is quite expensive. For example, the network communication time for a single piece of data can be as large as the computation time for thousands of instructions. Moreover, parallel overhead tends to increase with the number of processing nodes.

The ratio of sequential execution time to parallel execution time on n processors is the parallel speedup, that is,

$$\text{speedup}(n) = \frac{T(1)}{T(n)}. \tag{2.4}$$

Ideally, the speedup grows linearly with the number of processors for a fixed-size problem and is equal or close to n when n processors are used. However, the achieved speedup for real applications is typically sublinear due to parallel overhead.

2.2.3 Energy Efficiency

The energy (E) consumed by an application is the aggregated power over its execution time. Energy can be calculated as the product of the average power during its execution and the time interval between the starting time t_1 and finishing time $t_2 = t_1 + D$, where D is the delay:

$$E = \int_{t_1}^{t_2} P \mathrm{d}t = P_{\text{avg}} \times (t_2 - t_1) = P_{\text{avg}} \times T. \tag{2.5}$$

Equation 2.5 suggests that energy reduction requires less execution time, smaller average power, or both.

Energy efficiency is measured by energy-delay product (e.g., $E \cdot D$ or $E \cdot D^2$) [23]. Lower numbers represent "better" efficiency.

2.3 RELATED WORK

We focus our discussion in three close related subareas: power profiling on large-scale systems, performance scalability on power-aware parallel systems, and adaptive power allocation for energy-efficient computing. Owing to the space limit, we limit our discussions in embedded systems [24] and mobile systems [25], even though these systems can share same techniques for energy efficiency.

2.3.1 Power Profiling

There are three primary approaches used to profile power of systems and components such as simulators, direct measurements, and performance-counter-based models.

2.3.1.1 Simulator-based power estimation. The power simulators are normally built on or used in conjunction with performance simulators. The performance simulators provide resource usage counts, while the power simulators estimate energy consumption using power models for the resources. Most of the power simulators are at architecture level and for single computer components such as processor, memory, disk, or interconnect. Few of them are at system level and estimate power consumption of software and applications.

Component Power Simulators. Several processor power simulators are available. These simulators are usually built on instruction-level performance simulators and add extra modules to performance simulator to estimate power consumption. Wattch [7] and SimplePower [8, 26] model the total processor power as the dominant dynamic power CMOS chips without considering leakage power and short-circuit power. These two simulators are based on SimpleScalar [27] performance simulator. PowerTimer [28] is a simplified version of Wattch for PowerPC processors. TEM2P2EST [29] and the Cai-Lim model [30] are also built on SimpleScalar [27] but models both dynamic and leakage power. Power simulators for other major computer components are also available. For example, DRAMSim [31] simulates the power consumption of DRAM system. Orion [10] simulates interconnection network power at the architectural level based on the performance simulator LSE [32]. Dempsey [33] simulates the power consumption of hard disk drives.

System Power Simulators. The system power simulators simulate the power consumption of applications or software. Softwatt [34] quantifies the power behavior of both the application and operating system based on SimOS [35]. Powerscope [36] samples system activity by periodically recording the program

counter (PC) and process identifier (PID) of the currently executing process, collects and stores current, and then maps the energy to specific processes and procedures.

2.3.1.2 Direct measurements. Power can be directly measured both intrusively [37, 38] and nonintrusively. The intrusive measurements require inserting precision resistors into the power supply lines to components under study and use power meters to measure the voltage drop on the resistor. The current through the component is calculated by the voltage drop over the resistor divided by its resistance. The nonintrusive approach [39, 40] uses ammeters to measure the current flow of the power supply lines directly.

Tiwari et al. [40] use ammeters to measure the current drawn by a processor while running programs on an embedded system and develope a power model to estimate power cost. Isci and Martonosi [39] use ammeters to measure the power for P4 processors to derive their event-count-based power model. Bellosa et al. [37] derive CPU power by measuring current on a precision resistor inserted between the power line and supply for a Pentium II CPU; they use this power to validate their event-count-based power model and save energy. Joseph et al. [38] use precision resistor to measure power for a Pentium Pro processor. These approaches can be extended to measure single processor system power. Flinn et al. [41] use a multimeter to sample the current being drawn by a laptop from its external power source.

2.3.1.3 Event-based estimation. Most high-end CPUs have a set of hardware counters to count performance events such as cache hit/miss, and memory load. If power is mainly dissipated by these performance events, power can be estimated based on performance counters. Isci and Martonosi [39] develope a runtime power monitoring model that correlates performance event counts with CPU subunit power dissipation on real machines. CASTLE [38] does similar work on performance simulators (SimpleScalar) instead of real machines. Joule Watcher [37] also correlates power with performance events, the difference is that it measures the energy consumption for a single event, such as a floating-point operation and L2 cache access, and uses this energy consumption for energy-aware scheduling.

2.3.2 Performance Scalability on Power-Aware Systems

Parallel speedup models analytically describe how application performance scales with the number of concurrent computing processors. Such models often also indicate performance bound and limiting factors. Several models have been proposed [42–49], each focusing on a unique limiting factor. The first parallel speedup model is Amdahl's law [42]. Amdahl's Law states that speedup is limited by the fraction of the workload that cannot be computed in parallel. Amdahl's law is also known as *strong scaling*, as problem sizes under study will not change with the number of computing units. Later, Gustafson [45] argues

that one tends to scale problem size, instead of keeping it constant, for accuracy when more computing units available, and thus proposes a fixed-time speedup model. It quantifies the ability to scale up workload with the number of computational nodes while maintaining the same execution time. As memory wall emerges as a limiting factor of performance, Sun and Ni [47] argue that the problem size can be scaled further in large memory systems to gain more speedup and increase simulation accuracy. They accordingly present a memory-bounded speedup model, where speedup is constrained by main memory size. While fixed-time speedup and memory-bounded speedup models study how to scale workload to gain parallel speedup and simulation accuracy, whereas isoefficiency metric proposed by Grama et al. [44] studies how to scale workload to obtain same parallel computing efficiency. These models are powerful in analyzing parallel performance and scalability in parallel computing but limited to conventional systems. They are problematic when used to analyze the performance effects of power mode on the emerging power-scalable systems. The analytical model proposed by Cho and Melhem [50] studies the interaction between parallelism and energy consumption. This model assumes that frequency scaling unanimously affects executions of any workload and thus cannot guide the design of adaptive power allocation.

2.3.3 Adaptive Power Allocation for Energy-Efficient Computing

Adaptive power allocation in HPC is premised by energy savings without performance impact for applications. Adaptive power allocation can be deployed at node level by consolidating the application to a few servers and shutting down unused nodes [51, 52]. Adaptive power allocation can also be deployed at the component level. At present, major computer components such as processor, disk, memory [53], and monitor accommodate Advanced Configuration and Power Interface (ACPI) components. These components have one or more active modes and several sleep modes. Usually the deeper sleep modes consume less power but require longer time to transit to active modes. These components can be put to sleep states to reduce power consumption when there are no user and no system accesses. Only processor has more than one active mode in production systems, while other components such as multiple-speed disks [54, 55] are studied in simulations. If the processor is designed with high performance, it is normally capable of DVFS and can be switched among several active performance states, each determined by a pair of frequency and voltage. If the processor is designed with low power, it is normally capable of clock gating. Clock gating does not change processor frequency. However, it prevents clock from propagating to portions of the circuitry, so that the flip-flops in the portion do not change state and their dynamic power consumption is zero.

Adaptive power allocation using DVFS is studied most on high performance systems for scientific computation. Experiments on real systems show that energy is more efficiently utilized by scaling processor to low power modes during its slack times, that is, communication [56–59], memory access [60, 61], load