15

Algorithms and Analysis of Energy-Efficient Scheduling of Parallel Tasks

Keqin Li
State University of New York

15.1 Introduction

15.1.1 Motivation

For six decades, the concept of the performance of a computer has been equivalent to the computing speed measured by floating-point operations per second (FLOPS). The peak speed of high-performance supercomputers has increased at an exponential speed. At the same time, the peak power requirements also increase at the same rate [12]. 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 [40]. The emphasis on speed has led to the emergence of supercomputers that consume tremendous amounts of electrical power and produce so much heat that excessive cooling facilities must be constructed to ensure proper operation. Furthermore, the adoption of speed as the ultimate performance metric has caused other metrics such as reliability, availability, and usability to be largely ignored. Consequently, there has been an extraordinary increase in the total cost of ownership of a supercomputer.

Such increased energy consumption causes severe economic, ecological, and technical problems. A large-scale multiprocessor computing system consumes millions of dollars of electricity and natural

Ranka/Handbook of Energy-Aware and Green Computing K12293_C015 Page proof Page 331 2011-7-29

resources every year, equivalent to the amount of energy used by tens of thousands of U.S. households [13]. A large data center such as Google can consume as much electricity as a city. Furthermore, the cooling bill for heat dissipation can be as high as 70% of the cost mentioned previously [11]. A recent report reveals that the global information technology industry generates as much greenhouse gas as the world's airlines, about 2% of global carbon dioxide (CO_2) emissions. Despite sophisticated cooling facilities constructed to ensure proper operation, the reliability and availability (mean time between failures and interrupts) of large-scale multiprocessor computing systems is measured in hours, and the main source of outage is hardware (CPU, memory, storage, and third-party hardware) failure caused by excessive heat. It is conceivable that a supercomputing system with 10^5 processors would spend most of its time checkpointing and restarting [16]. Furthermore, the hourly cost of downtime can be as high as millions of US dollars. The cost of ownership can easily exceed the initial acquisition cost.

In recent years, there has been rapidly growing interest and importance in developing high-performance and energy-efficient computing systems (see [1,5,39,40] for comprehensive surveys). 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 most energy-efficient supercomputers in the world, so that the awareness of other performance metrics such as performance per watt, energy efficiency, system reliability, and total cost of ownership can be raised [12]. The philosophy is that high-performance supercomputers should only simulate and predict climate and weather, but should not change or create them.

It can be found from the Green500 list that all the current supercomputing systems which 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. 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.

One effective and widely adopted approach to reducing energy consumption in computing systems is the method of using a mechanism called *dynamic voltage scaling* (equivalently, dynamic frequency scaling, dynamic speed scaling, 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. 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. 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. LongRun and LongRun2 are power management technologies introduced by Transmeta. LongRun2 has been licensed to Fujitsu, NEC, Sony, Toshiba, and NVIDIA.

15.1.2 Related Research

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 trade-off [38]. Power-aware task scheduling on processors

with variable voltages and speeds has been extensively studied since mid 1990s. In a pioneering paper [41], the authors first proposed the 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 [43], the authors analyzed off-line and online algorithms for scheduling tasks with arrival times and deadlines on a uniprocessor computer with minimum energy consumption. These researches have been extended in [3,7,21,28–30,44] 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 [2,17,18,20,23,31,32,34,36,37,42,46–49] and many other related works, 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.

There are two considerations in dealing with the energy delay trade-off. On 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 goals. In [4], 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 [6], the author considered scheduling jobs with equal requirements on multi-processors. In [9], the authors studied the relationship among parallelization, performance, and energy consumption, and the problem of minimizing energy-delay product. In [19,22], the authors attempted joint minimization of energy consumption and task execution time. In [35], the authors investigated the problem of system value maximization subject to both time and energy constraints.

In [25,27], we addressed energy- and time-constrained power allocation and task scheduling on multiprocessor computers with dynamically variable voltage and frequency and speed and power as combinatorial optimization problems. In particular, we defined the problem of minimizing schedule length with energy consumption constraint and the problem of minimizing energy consumption with schedule length constraint on multiprocessor computers [25]. The first problem has applications in general multiprocessor and multi-core 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. The investigation in [25,27] is for sequential tasks. A sequential task is executed on one processor. The motivation of this chapter is to study energy-efficient scheduling of parallel tasks.

15.1.3 Our Contributions

In this chapter, we investigate energy-efficient scheduling of parallel tasks on multiprocessor computers with dynamically variable voltage and speed. As in traditional scheduling theory, our problems are defined 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 for parallel tasks on multiprocessor computers. It is noticed that power-aware scheduling of parallel tasks has not been well studied before. Most existing studies are on scheduling sequential tasks which require one processor to execute, while a parallel task requires multiple processors to execute. It is clear that our scheduling problems introduce new challenges to energy-aware computing.

Our investigation in this chapter, together with our recent study in [26], make some initial attempt to energy-efficient scheduling of parallel tasks on multiprocessor computers with dynamic voltage and speed.

It is found that our scheduling problems contain three nontrivial sub-problems, namely, system partitioning, task scheduling, and power supplying. These subproblems are described as follows:

- *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.
- *Task scheduling*: Parallel tasks are scheduled together with system partitioning, and it is NP-hard even scheduling sequential tasks without system partitioning.
- *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.

Each subproblem should be solved efficiently, so that heuristic algorithms with overall good performance can be developed.

We consider two types of algorithms to solve our energy-aware parallel task scheduling problems, depending on the order of solving the subproblems.

- *Pre-Power-Determination algorithms*: In this type of algorithms, power allocation and execution speed determination are performed before tasks are scheduled. When tasks are scheduled, their execution times are available.
- *Post-Power-Determination algorithms*: In this type of algorithms, power allocation and execution speed determination are performed after tasks are scheduled. When a system is partitioned and tasks are scheduled, their execution times are not available, and tasks are scheduled based on their execution requirements.

The decomposition of our optimization problems into three subproblems makes design and analysis of heuristic algorithms tractable. We will develop a number of pre-power-determination algorithms and post-power-determination algorithms for energy-aware scheduling of parallel tasks. Furthermore, we will show that our heuristic algorithms are able to produce solutions very close to optimum.

15.2 Background Information

15.2.1 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 [8]. Since $s \propto f$, where s is the processor speed, and $f \propto V^\gamma$ with $0 < \gamma \leq 1$ [45], which implies that $V \propto f^{1/\gamma}$, we know that power consumption is $p \propto f^\alpha$ and $p \propto s^\alpha$, where $\alpha = 1 + 2/\gamma \geq 3$. It is clear from $f \propto V^\gamma$ and $s \propto V^\gamma$ that linear change in supply voltage results in up to linear change in clock frequency and processor speed. It is also clear from $p \propto V^{\gamma+2}$ and $p \propto f^\alpha$ and $p \propto s^\alpha$ that linear change in supply voltage results in at least quadratic change in power supply, and that linear change in clock frequency and processor speed results in at least cubic change in power supply.

Assume that we are given n independent parallel tasks to be executed on m identical processors. Task i requires π_i processors to execute, where $1 \leq i \leq n$, 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 (i.e., the numbers of CPU cycles or the numbers of instructions executed on the processors). Let r_i represent the maximum execution requirement on the π_i processors executing

task i. We use p_i to represent the power supplied 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 $t_i = r_i/s_i = r_i/p_i^{1/\alpha}$. 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 $e_i = \pi_i p_i t_i = \pi_i r_i p_i^{1-1/\alpha} = \pi_i r_i s_i^{\alpha-1} = w_i s_i^{\alpha-1}$, where $w_i = \pi_i r_i$ is the amount of work to be performed for task i.

We would like to mention a number of important observations.

- $f_i \propto V_i^{\phi}$ and $s_i \propto V_i^{\phi}$: Linear change in supply voltage results in up to linear change in clock frequency and processor speed.
- $p_i \propto V_i^{\phi+2}$ 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.
- $s_i/p_i \propto V_i^{-2}$ and $s_i/p_i \propto s_i^{-(\alpha-1)}$: The processor energy performance, measured by speed per Watt, is at least quadratically proportional to the supply voltage and speed reduction.
- $w_i/e_i \propto V_i^{-2}$ and $w_i/e_i \propto s_i^{-(\alpha-1)}$: The processor energy performance, measured by work per Joule, is at least quadratically proportional to the supply voltage and speed reduction.
- $e_i \propto p_i^{1-1/\alpha} \propto V_i^{(\phi+2)(1-1/\alpha)} = V_i^2$: Linear change in supply voltage results in quadratic change in energy consumption.
- $e_i = w_i s_i^{\alpha-1}$: Linear change in processor speed results in at least quadratic change in energy consumption.
- $e_i = w_i p_i^{1-1/\alpha}$: Energy consumption reduces at a sublinear speed as power supply reduces.
- $e_i t_i^{\alpha-1} = \pi_i r_i^{\alpha}$ and $p_i t_i^{\alpha} = r_i^{\alpha}$: For a given task, there exist energy delay and power delay trade-offs. (Later, we will extend such trade-off to a set of parallel tasks, i.e., the energy delay trade-off theorem.)

15.2.2 Definitions

Problem 15.1 *(Minimizing schedule length with energy consumption Constraint)*
Input: A set of n parallel tasks with task sizes $\pi_1, \pi_2, \ldots, \pi_n$ and task execution requirements $r_1, r_2, \ldots, r_n$, 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 non-preemptive schedule of the n parallel tasks on the m processors such that the schedule length is minimized and the total energy consumed does not exceed E.

Problem 15.2 *(Minimizing energy consumption with schedule length constraint)*
Input: A set of n parallel tasks with task sizes $\pi_1, \pi_2, \ldots, \pi_n$ and task execution requirements $r_1, r_2, \ldots, r_n$, 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 non-preemptive schedule of the n parallel tasks on the m processors such that the total energy consumed is minimized and the schedule length does not exceed T.

In the previous description, the energy-aware parallel task scheduling problems on multiprocessor computers with energy and time constraints addressed in this chapter are defined as optimization problems.

When all the π_i's are identical, the scheduling problems discussed earlier are equivalent to scheduling sequential tasks discussed in [25,27]. Since both scheduling problems are NP-hard in the strong sense for all rational $\alpha > 1$ in scheduling sequential tasks [14,25], our problems for scheduling parallel tasks are also NP-hard in the strong sense for all rational $\alpha > 1$. Hence, we will develop fast polynomial time heuristic algorithms to solve these problems.

Let T_A denote the length of the schedule produced by algorithm A, and T_{OPT} denote the length of an optimal schedule. Similarly, let E_A denote the total amount of energy consumed by algorithm A, and E_{OPT} denote the minimum amount of energy consumed by an optimal schedule. The following performance measures are used to analyze and evaluate the performance of our heuristic power allocation and parallel task scheduling algorithms.

Definition 15.1 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 $\beta_A \leq B$, we call B a *performance bound* of algorithm A.

Definition 15.2 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 $\gamma_A \leq C$, we call C a *performance bound* of algorithm A.

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

Definition 15.3 If $\beta_A \leq B$, then $\bar{\beta}_A \leq \overline{B}$, where $\overline{B}$ is an *average-case performance bound* of algorithm A.

Definition 15.4 If $\gamma_A \leq C$, then $\bar{\gamma}_A \leq \overline{C}$, where $\overline{C}$ is an *average-case performance bound* of algorithm A.

15.2.3 Lower Bounds

The performance of our heuristic algorithms will be compared with optimal solutions analytically. Unfortunately, it is infeasible to compute optimal solutions in reasonable amount of time. To make such comparison possible, we have derived lower bounds for the optimal solutions in Theorems 15.1 and 15.2 [26]. The significance of these lower bounds is that they can be used to evaluate the performance of heuristic algorithms when they are compared with optimal solutions.

Let $W = w_1 + w_2 + \cdots + w_n = \pi_1 r_1 + \pi_2 r_2 + \cdots + \pi_n r_n$ denote the total amount of work to be performed for the n parallel tasks. 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 15.1 *For the problem of minimizing schedule length with energy consumption constraint in scheduling parallel tasks, we have the following lower bound*

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

for the optimal schedule length.

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 15.2 *For the problem of minimizing energy consumption with schedule length constraint in scheduling parallel tasks, we have the following lower bound*

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

for the minimum energy consumption.

The lower bounds in Theorems 15.1 and 15.2 essentially state the following important theorem.

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

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

by using any scheduling algorithm.

Therefore, our scheduling problems are defined such that the energy delay product is optimized by fixing one factor and minimizing the other.

It is noticed that when $\alpha = 3$, we have $ET^{\alpha-1} = ET^2$, which was proposed as a measure of the energy efficiency of a computation [33].

Notice that the lower bounds in Theorems 15.1 and 15.2 and the energy delay trade-off theorem are applicable to various parallel 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 non-preemptive, online or off-line, clairvoyant or non-clairvoyant scheduling). (See [27] for description of these models.)

15.3 Pre-Power-Determination Algorithms

15.3.1 Overview

In pre-power-determination algorithms, we first determine power supplies to the n parallel tasks and then partition the system and schedule the tasks. We use $A_1 - A_2$ to represent a pre-power-determination algorithm, where A_1 is a strategy for power supplying and A_2 is an algorithm for system partitioning and task scheduling. Algorithm $A_1 - A_2$ works in the following way. First, algorithm A_1 is used to allocate powers to the n tasks and to solve the subproblem of power supplying. Second, algorithm A_2 is used to partition the system with m processors and to schedule the n tasks whose execution times are known based on the power allocation, and to solve the subproblems of system partitioning and task scheduling simultaneously.

In this chapter, we consider three strategies (i.e., algorithm A_1) for power allocation.

- *Equal-time (ET)*: The power supplies $p_1, p_2, \ldots, p_n$ are determined such that all the n parallel tasks have the same execution time, i.e., $t_1 = t_2 = \cdots = t_n$.
- *Equal-energy (EE)*: The power supplies $p_1, p_2, \ldots, p_n$ are determined such that all the n parallel tasks consume the same amount of energy, i.e., $e_1 = e_2 = \cdots = e_n$.
- *Equal-speed (ES)*: The power supplies $p_1, p_2, \ldots, p_n$ are determined such that all the n parallel tasks have the same execution speed, i.e., $s_1 = s_2 = \cdots = s_n$.

In all cases, the execution times of all the n parallel tasks are available before the tasks are scheduled.

We consider the following four methods (i.e., algorithm A_2) for scheduling parallel tasks.

- SIMPLE: The n parallel tasks are put into a list, which is divided into groups, where each group contains a sublist of consecutive tasks. In the beginning and whenever a task is completed, the next group of tasks are scheduled for execution. Each group includes as many tasks as possible for simultaneous execution.

- SIMPLE*: This method works in the same way as SIMPLE, except that tasks are arranged in a nonincreasing order of sizes, i.e., $\pi_1 \geq \pi_2 \geq \cdots \geq \pi_n$, before scheduling.
- GREEDY: In the beginning and whenever a task is completed, each remaining unscheduled task is examined for possible execution, i.e., whether there are enough available processors for the task.
- GREEDY*: This method works in the same way as GREEDY, except that tasks are arranged in a nonincreasing order of sizes, i.e., $\pi_1 \geq \pi_2 \geq \cdots \geq \pi_n$, before scheduling.

Algorithms SIMPLE and GREEDY were considered in [24] for non-clairvoyant scheduling of parallel tasks, since these algorithms do not require the information of task execution times.

To summarize, we have developed 12 pre-power-determination algorithms for power allocation and parallel task scheduling, namely, ET-*A*, EE-*A*, and ES-*A*, where $A \in \{\text{SIMPLE}, \text{SIMPLE}^*, \text{GREEDY}, \text{GREEDY}^*\}$.

For n parallel tasks with sizes $\pi_1, \pi_2, \ldots, \pi_n$ and execution times $t_1, t_2, \ldots, t_n$, we use $A(\pi_1, t_1, \pi_2, t_2, \ldots, \pi_n, t_n)$ to denote the length of the schedule produced by algorithm A.

15.3.2 Analysis of Equal-Time Algorithms

15.3.2.1 Minimizing Schedule Length

To solve the problem of minimizing schedule length with energy consumption constraint E by using an equal-time algorithm ET-*A*, we have

$$t_1 = t_2 = \cdots = t_n = t,$$

where t is the identical task execution time. Since

$$t_i = \frac{r_i}{s_i} = t,$$

we get

$$s_i = \frac{r_i}{t},$$

and

$$p_i = s_i^\alpha = \left(\frac{r_i}{t}\right)^\alpha,$$

and

$$e_i = \pi_i r_i p_i^{1-1/\alpha} = \frac{\pi_i r_i^\alpha}{t^{\alpha-1}},$$

for all $1 \leq i \leq n$. Since

$$e_1 + e_2 + \cdots + e_n = E,$$

that is,

$$\frac{\pi_1 r_1^\alpha + \pi_2 r_2^\alpha + \cdots + \pi_n r_n^\alpha}{t^{\alpha-1}} = E,$$

we obtain

$$t = \left(\frac{\pi_1 r_1^\alpha + \pi_2 r_2^\alpha + \cdots + \pi_n r_n^\alpha}{E}\right)^{1/(\alpha-1)}.$$

Consequently, the length of the schedule generated by algorithm ET-A is

$$T_{\text{ET-}A} = A(\pi_1, t, \pi_2, t, \ldots, \pi_n, t).$$

By Theorem 1, the performance ratio of algorithm ET-A is

$$\beta_{\text{ET-}A} = \frac{T_{\text{ET-}A}}{T_{\text{OPT}}} \leq \frac{A(\pi_1, t, \pi_2, t, \ldots, \pi_n, t)}{((m/E)(W/m)^{\alpha})^{1/(\alpha-1)}}.$$

We observe that when all tasks have the same execution time t, the problem of scheduling parallel tasks with sizes $\pi_1, \pi_2, \ldots, \pi_n$ such that the total execution time is minimized is equivalent to the classic bin packing problem, i.e., packing n items of sizes $\pi_1, \pi_2, \ldots, \pi_n$ into bins of size m, such that the number of bins used is minimized. Let $b_1, b_2, b_3, \ldots$ be the sequence of bins used. The following four methods are well known bin packing algorithms [10].

- *Next fit (NF)*: Assume that b_i is the current bin being packed. b_i is packed as many items as possible, until there is π_j which cannot be packed into b_i. Then, a new bin b_{i+1} is opened to pack π_j.
- *Next fit decreasing (NFD)*: This algorithm is the same as *NF*, except that the sizes are sorted in a nonincreasing order before packing, i.e., $\pi_1 \geq \pi_2 \geq \cdots \geq \pi_n$.
- *First fit (FF)*: Let $b_1, b_2, \ldots, b_i$ be the bins ever used. To pack an item π_j, each bin of $b_1, b_2, \ldots, b_i$ is examined in this order to see whether π_j can be packed. π_j is packed into the first bin which can accommodate π_j. If no such bin is found, a new bin b_{i+1} is opened to pack π_j.
- *First fit decreasing (FFD)*: This algorithm is the same as *FF*, except that the sizes are sorted in a nonincreasing order before packing, i.e., $\pi_1 \geq \pi_2 \geq \cdots \geq \pi_n$.

Each parallel task scheduling algorithm A is equivalent to a bin packing algorithm A'. It is clear that for $A =$ SIMPLE, SIMPLE*, GREEDY, GREEDY*, we have $A' =$ *NF, NFD, FF, FFD*, respectively.

Let $A'(\pi_1, \pi_2, \ldots, \pi_n)$ denote the number of bins used by algorithm A'. Then, we have

$$\begin{aligned}
\text{SIMPLE}(\pi_1, t, \pi_2, t, \ldots, \pi_n, t) &= t\mathit{NF}(\pi_1, \pi_2, \ldots, \pi_n),\\
\text{SIMPLE}^*(\pi_1, t, \pi_2, t, \ldots, \pi_n, t) &= t\mathit{NFD}(\pi_1, \pi_2, \ldots, \pi_n),\\
\text{GREEDY}(\pi_1, t, \pi_2, t, \ldots, \pi_n, t) &= t\mathit{FF}(\pi_1, \pi_2, \ldots, \pi_n),\\
\text{GREEDY}^*(\pi_1, t, \pi_2, t, \ldots, \pi_n, t) &= t\mathit{FFD}(\pi_1, \pi_2, \ldots, \pi_n),
\end{aligned}$$

and

$$T_{\text{ET-}A} = tA'(\pi_1, \pi_2, \ldots, \pi_n).$$

By Theorem 1, the performance ratio of algorithm ET-A is

$$\begin{aligned}
\beta_{\text{ET-}A} &= \frac{T_{\text{ET-}A}}{T_{\text{OPT}}}\\
&\leq \frac{tA'(\pi_1, \pi_2, \ldots, \pi_n)}{((m/E)(W/m)^{\alpha})^{1/(\alpha-1)}}\\
&= \left(\frac{\pi_1 r_1^{\alpha} + \pi_2 r_2^{\alpha} + \cdots + \pi_n r_n^{\alpha}}{E}\right)^{1/(\alpha-1)} \frac{A'(\pi_1, \pi_2, \ldots, \pi_n)}{((m/E)(W/m)^{\alpha})^{1/(\alpha-1)}}\\
&= \left(\frac{m(\pi_1 r_1^{\alpha} + \pi_2 r_2^{\alpha} + \cdots + \pi_n r_n^{\alpha})^{1/(\alpha-1)}}{W^{\alpha/(\alpha-1)}}\right) A'(\pi_1, \pi_2, \ldots, \pi_n).
\end{aligned}$$

This discussion can be summarized in the following theorem.

Theorem 15.3 *By using an equal-time algorithm ET-A to solve the problem of minimizing schedule length with energy consumption constraint, the schedule length is*

$$T_{ET\text{-}A} = A(\pi_1, t, \pi_2, t, \ldots, \pi_n, t),$$

where

$$t = \left(\frac{\pi_1 r_1^\alpha + \pi_2 r_2^\alpha + \cdots + \pi_n r_n^\alpha}{E}\right)^{1/(\alpha-1)},$$

or,

$$T_{ET\text{-}A} = \left(\frac{\pi_1 r_1^\alpha + \pi_2 r_2^\alpha + \cdots + \pi_n r_n^\alpha}{E}\right)^{1/(\alpha-1)} A'(\pi_1, \pi_2, \ldots, \pi_n).$$

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

$$B_{ET\text{-}A} = \frac{A(\pi_1, t, \pi_2, t, \ldots, \pi_n, t)}{((m/E)(W/m)^\alpha)^{1/(\alpha-1)}},$$

or,

$$B_{ET\text{-}A} = \left(\frac{m\left(\pi_1 r_1^\alpha + \pi_2 r_2^\alpha + \cdots + \pi_n r_n^\alpha\right)^{1/(\alpha-1)}}{(\pi_1 r_1 + \pi_2 r_2 + \cdots + \pi_n r_n)^{\alpha/(\alpha-1)}}\right) A'(\pi_1, \pi_2, \ldots, \pi_n).$$

For the purpose of average-case performance analysis, throughout this chapter, we make the following assumptions of task sizes and task execution requirements.

- $\pi_1, \pi_2, \ldots, \pi_n$ are independent and identically distributed (i.i.d.) discrete random variables with a common probability distribution in the range $[1 \ldots m]$.
- $r_1, r_2, \ldots, r_n$ are i.i.d. continuous random variables.
- The probability distribution of the π_i's and the probability distribution of the r_i's are independent of each other.

These assumptions make the probabilistic analysis of SIMPLE feasible.

Let us consider the following packing model. Assume that there is a bag of capacity m and there are n objects of sizes $\pi_1, \pi_2, \ldots, \pi_n$. These objects are to be packed into the bag as many as possible, i.e., we need to find i such that

$$\pi_1 + \pi_2 + \cdots + \pi_i \leq m,$$

but

$$\pi_1 + \pi_2 + \cdots + \pi_i + \pi_{i+1} > m.$$

If $\pi_1, \pi_2, \ldots, \pi_n$ are i.i.d. random variables, the total size of the objects packed into the bag, i.e., $\pi_1 + \pi_2 + \cdots + \pi_i$, is also a random variable. Let $P(n, m)$ denote the mean of this random variable, and define $P(m) = \lim_{n\to\infty} P(n, m)$. In fact, if $\pi_1, \pi_2, \ldots, \pi_n$ are discrete integer random variables, we have $P_m = P(n, m)$ for all $n \geq m$, since the bag can accommodate at most m objects. It was shown in [24] that P_m can be calculated easily by using a recurrence relation for any probability distribution of the π_i's.

Furthermore, it was known that by the previously mentioned assumptions of task sizes and task execution requirements, for algorithm SIMPLE, we have

$$\mathbb{E}(\text{SIMPLE}(\pi_1, t_1, \pi_2, t_2, \ldots, \pi_n, t_n)) \approx \frac{1}{P_m}\mathbb{E}(\pi_1 t_1 + \pi_2 t_2 + \cdots + \pi_n t_n),$$

when n is large [24]. (Notation: $\mathbb{E}(x)$ and $\bar{x}$ represent the expectation of a random variable x.) Based on the previous result, we obtain

$$\begin{aligned}\mathbb{E}(\text{SIMPLE}(\pi_1, t, \pi_2, t, \ldots, \pi_n, t)) &\approx \frac{1}{P_m}\mathbb{E}(\pi_1 t + \pi_2 t + \cdots + \pi_n t)\\ &\approx \frac{1}{P_m}\mathbb{E}(t)\mathbb{E}(\pi_1 + \pi_2 + \cdots + \pi_n),\end{aligned}$$

which implies the following approximation of the average-case performance bound $\overline{B}_{\text{ET-SIMPLE}}$

$$\begin{aligned}\overline{B}_{\text{ET-SIMPLE}} &\approx \frac{\mathbb{E}(\text{SIMPLE}(\pi_1, t, \pi_2, t, \ldots, \pi_n, t))}{\mathbb{E}(((m/E)(W/m)^{\alpha})^{1/(\alpha-1)})}\\ &\approx \frac{1}{P_m}\cdot\frac{\mathbb{E}(t)\mathbb{E}(\pi_1 + \pi_2 + \cdots + \pi_n)}{\mathbb{E}(((m/E)(W/m)^{\alpha})^{1/(\alpha-1)})}\\ &\approx \frac{m}{P_m}\cdot\frac{\mathbb{E}\left(\left(\pi_1 r_1^{\alpha} + \pi_2 r_2^{\alpha} + \cdots + \pi_n r_n^{\alpha}\right)^{1/(\alpha-1)}\right)\mathbb{E}(\pi_1 + \pi_2 + \cdots + \pi_n)}{\mathbb{E}((\pi_1 r_1 + \pi_2 r_2 + \cdots + \pi_n r_n)^{\alpha/(\alpha-1)})}.\end{aligned}$$

We will provide numerical and simulation data to demonstrate the accuracy of the previous approximation.

15.3.2.2 Minimizing Energy Consumption

To solve the problem of minimizing energy consumption with schedule length constraint T by using an equal-time algorithm ET-A, we need to provide enough energy $E_{\text{ET-}A}$, so that the deadline T is met, i.e., $T_{\text{ET-}A} = T$,

$$\left(\frac{\pi_1 r_1^{\alpha} + \pi_2 r_2^{\alpha} + \cdots + \pi_n r_n^{\alpha}}{E_{\text{ET-}A}}\right)^{1/(\alpha-1)} A'(\pi_1, \pi_2, \ldots, \pi_n) = T.$$

This equation implies that the amount of energy $E_{\text{ET-}A}$ consumed by algorithm ET-A is

$$E_{\text{ET-}A} = \left(\frac{A'(\pi_1, \pi_2, \ldots, \pi_n)}{T}\right)^{\alpha-1}\left(\pi_1 r_1^{\alpha} + \pi_2 r_2^{\alpha} + \cdots + \pi_n r_n^{\alpha}\right).$$

By theorem 2, the performance ratio of algorithm ET-A is

$$\begin{aligned}\gamma_{\text{ET-}A} &= \frac{E_{\text{ET-}A}}{E_{\text{OPT}}}\\ &\le \frac{(A'(\pi_1, \pi_2, \ldots, \pi_n)/T)^{\alpha-1}\left(\pi_1 r_1^{\alpha} + \pi_2 r_2^{\alpha} + \cdots + \pi_n r_n^{\alpha}\right)}{(m/T^{\alpha-1})(W/m)^{\alpha}}\\ &= \frac{(mA'(\pi_1, \pi_2, \ldots, \pi_n))^{\alpha-1}\left(\pi_1 r_1^{\alpha} + \pi_2 r_2^{\alpha} + \cdots + \pi_n r_n^{\alpha}\right)}{W^{\alpha}}.\end{aligned}$$

This discussion basically proves the following theorem.

Theorem 15.4 *By using an equal-time algorithm* ET-*A to solve the problem of minimizing energy consumption with schedule length constraint, the energy consumed is*

$$E_{\text{ET-}A} = \left(\frac{A'(\pi_1, \pi_2, \ldots, \pi_n)}{T}\right)^{\alpha-1} \left(\pi_1 r_1^\alpha + \pi_2 r_2^\alpha + \cdots + \pi_n r_n^\alpha\right).$$

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

$$C_{\text{ET-}A} = \left(\frac{m^{\alpha-1}\left(\pi_1 r_1^\alpha + \pi_2 r_2^\alpha + \cdots + \pi_n r_n^\alpha\right)}{(\pi_1 r_1 + \pi_2 r_2 + \cdots + \pi_n r_n)^\alpha}\right)\left(A'(\pi_1, \pi_2, \ldots, \pi_n)\right)^{\alpha-1}.$$

Since $C_{\text{ET-}A} = B_{\text{ET-}A}^{\alpha-1}$, we obtain the following approximation of the average-case performance-bound $\overline{C}_{\text{ET-SIMPLE}}$, i.e.,

$$\begin{aligned}\overline{C}_{\text{ET-SIMPLE}} &= \mathbb{E}(B_{\text{ET-SIMPLE}}^{\alpha-1}) \approx \overline{B}_{\text{ET-SIMPLE}}^{\alpha-1} \\ &\approx \left(\frac{m}{P_m} \cdot \frac{\mathbb{E}\left(\left(\pi_1 r_1^\alpha + \pi_2 r_2^\alpha + \cdots + \pi_n r_n^\alpha\right)^{1/(\alpha-1)}\right)\mathbb{E}(\pi_1 + \pi_2 + \cdots + \pi_n)}{\mathbb{E}((\pi_1 r_1 + \pi_2 r_2 + \cdots + \pi_n r_n)^{\alpha/(\alpha-1)})}\right)^{\alpha-1}.\end{aligned}$$

15.3.3 Analysis of Equal-Energy Algorithms

15.3.3.1 Minimizing Schedule Length

To solve the problem of minimizing schedule length with energy consumption constraint E by using an equal-energy algorithm EE-A, we have

$$e_1 = e_2 = \cdots = e_n = \frac{E}{n}.$$

Hence, we get

$$e_i = \pi_i r_i p_i^{1-1/\alpha} = w_i p_i^{1-1/\alpha} = \frac{E}{n},$$

which gives

$$p_i = \left(\frac{E}{nw_i}\right)^{\alpha/(\alpha-1)},$$

and

$$s_i = p_i^{1/\alpha} = \left(\frac{E}{nw_i}\right)^{1/(\alpha-1)},$$

and

$$t_i = \frac{r_i}{s_i} = r_i w_i^{1/(\alpha-1)} \left(\frac{n}{E}\right)^{1/(\alpha-1)},$$

for all $1 \leq i \leq n$.

It is noticed that for all $\phi \geq 0$, we have

$$A(\pi_1, \phi t_1, \pi_2, \phi t_2, \ldots, \pi_n, \phi t_n) = \phi A(\pi_1, t_1, \pi_2, t_2, \ldots, \pi_n, t_n).$$

In other words, the schedule length is changed by a factor of ϕ if all the task execution times are changed by a factor of ϕ. This observation implies that the length of the schedule produced by algorithm EE-A is

$$\begin{aligned} T_{\text{EE-}A} &= A(\pi_1, t_1, \pi_2, t_2, \ldots, \pi_n, t_n) \\ &= A\left(\pi_1, r_1 w_1^{1/(\alpha-1)}, \pi_2, r_2 w_2^{1/(\alpha-1)}, \ldots, \pi_n, r_n w_n^{1/(\alpha-1)}\right)\left(\frac{n}{E}\right)^{1/(\alpha-1)}. \end{aligned}$$

By theorem 1, the performance ratio of algorithm EE-A is

$$\begin{aligned} \beta_{\text{EE-}A} &= \frac{T_{\text{EE-}A}}{T_{\text{OPT}}} \\ &\le \frac{A\left(\pi_1, r_1 w_1^{1/(\alpha-1)}, \pi_2, r_2 w_2^{1/(\alpha-1)}, \ldots, \pi_n, r_n w_n^{1/(\alpha-1)}\right)(n/E)^{1/(\alpha-1)}}{((m/E)(W/m)^{\alpha})^{1/(\alpha-1)}} \\ &= \frac{mn^{1/(\alpha-1)} A\left(\pi_1, r_1 w_1^{1/(\alpha-1)}, \pi_2, r_2 w_2^{1/(\alpha-1)}, \ldots, \pi_n, r_n w_n^{1/(\alpha-1)}\right)}{W^{\alpha/(\alpha-1)}} \\ &= \frac{mn^{1/(\alpha-1)} A\left(\pi_1, r_1 w_1^{1/(\alpha-1)}, \pi_2, r_2 w_2^{1/(\alpha-1)}, \ldots, \pi_n, r_n w_n^{1/(\alpha-1)}\right)}{(\pi_1 r_1 + \pi_2 r_2 + \cdots + \pi_n r_n)^{\alpha/(\alpha-1)}}. \end{aligned}$$

This discussion leads to the following theorem.

Theorem 15.5 *By using an equal-energy algorithm* EE-A *to solve the problem of minimizing schedule length with energy consumption constraint, the schedule length is*

$$T_{\text{EE-}A} = A\left(\pi_1, r_1 w_1^{1/(\alpha-1)}, \pi_2, r_2 w_2^{1/(\alpha-1)}, \ldots, \pi_n, r_n w_n^{1/(\alpha-1)}\right)\left(\frac{n}{E}\right)^{1/(\alpha-1)}.$$

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

$$B_{\text{EE-}A} = \frac{mn^{1/(\alpha-1)} A\left(\pi_1, r_1 w_1^{1/(\alpha-1)}, \pi_2, r_2 w_2^{1/(\alpha-1)}, \ldots, \pi_n, r_n w_n^{1/(\alpha-1)}\right)}{(\pi_1 r_1 + \pi_2 r_2 + \cdots + \pi_n r_n)^{\alpha/(\alpha-1)}}.$$

For algorithm SIMPLE, we have

$$\begin{aligned} &\mathbb{E}(\text{SIMPLE}\left(\pi_1, r_1 w_1^{1/(\alpha-1)}, \pi_2, r_2 w_2^{1/(\alpha-1)}, \ldots, \pi_n, r_n w_n^{1/(\alpha-1)})\right) \\ &\quad \approx \frac{1}{P_m}\mathbb{E}\left(w_1^{\alpha/(\alpha-1)} + w_2^{\alpha/(\alpha-1)} + \cdots + w_n^{\alpha/(\alpha-1)}\right). \end{aligned}$$

Hence, we get

$$\begin{aligned} \overline{B}_{\text{EE-SIMPLE}} &\approx \frac{mn^{1/(\alpha-1)}\mathbb{E}(\text{SIMPLE}\left(\pi_1, r_1 w_1^{1/(\alpha-1)}, \pi_2, r_2 w_2^{1/(\alpha-1)}, \ldots, \pi_n, r_n w_n^{1/(\alpha-1)}\right))}{\mathbb{E}((\pi_1 r_1 + \pi_2 r_2 + \cdots + \pi_n r_n)^{\alpha/(\alpha-1)})} \\ &\approx \frac{m}{P_m} \cdot \frac{n^{1/(\alpha-1)}\mathbb{E}\left(w_1^{\alpha/(\alpha-1)} + w_2^{\alpha/(\alpha-1)} + \cdots + w_n^{\alpha/(\alpha-1)}\right)}{\mathbb{E}((w_1 + w_2 + \cdots + w_n)^{\alpha/(\alpha-1)})}, \end{aligned}$$

for large n.

15.3.3.2 Minimizing Energy Consumption

To solve the problem of minimizing energy consumption with schedule length constraint T by using an equal-energy algorithm EE-A, we need to provide enough energy $E_{\text{EE-}A}$, so that the deadline T is met, i.e., $T_{\text{EE-}A} = T$,

$$A\left(\pi_1, r_1 w_1^{1/(\alpha-1)}, \pi_2, r_2 w_2^{1/(\alpha-1)}, \ldots, \pi_n, r_n w_n^{1/(\alpha-1)}\right)\left(\frac{n}{E_{\text{EE-}A}}\right)^{1/(\alpha-1)} = T.$$

The last equation implies that the amount of energy $E_{\text{EE-}A}$ consumed by algorithm EE-A is

$$E_{\text{EE-}A} = \frac{n}{T^{\alpha-1}}\left(A\left(\pi_1, r_1 w_1^{1/(\alpha-1)}, \pi_2, r_2 w_2^{1/(\alpha-1)}, \ldots, \pi_n, r_n w_n^{1/(\alpha-1)}\right)\right)^{\alpha-1}.$$

By theorem 2, the performance ratio of algorithm EE-A is

$$\begin{aligned}
\gamma_{\text{EE-}A} &= \frac{E_{\text{EE-}A}}{E_{\text{OPT}}} \\
&\leq \frac{(n/T^{\alpha-1})\left(A\left(\pi_1, r_1 w_1^{1/(\alpha-1)}, \pi_2, r_2 w_2^{1/(\alpha-1)}, \ldots, \pi_n, r_n w_n^{1/(\alpha-1)}\right)\right)^{\alpha-1}}{(m/T^{\alpha-1})(W/m)^{\alpha}} \\
&= \frac{nm^{\alpha-1}\left(A\left(\pi_1, r_1 w_1^{1/(\alpha-1)}, \pi_2, r_2 w_2^{1/(\alpha-1)}, \ldots, \pi_n, r_n w_n^{1/(\alpha-1)}\right)\right)^{\alpha-1}}{W^{\alpha}} \\
&= \frac{nm^{\alpha-1}\left(A\left(\pi_1, r_1 w_1^{1/(\alpha-1)}, \pi_2, r_2 w_2^{1/(\alpha-1)}, \ldots, \pi_n, r_n w_n^{1/(\alpha-1)}\right)\right)^{\alpha-1}}{(\pi_1 r_1 + \pi_2 r_2 + \cdots + \pi_n r_n)^{\alpha}}.
\end{aligned}$$

This discussion essentially proves the following theorem.

Theorem 15.6 *By using an equal-energy algorithm* EE-A *to solve the problem of minimizing energy consumption with schedule length constraint, the energy consumed is*

$$E_{\text{EE-}A} = \frac{n}{T^{\alpha-1}}\left(A\left(\pi_1, r_1 w_1^{1/(\alpha-1)}, \pi_2, r_2 w_2^{1/(\alpha-1)}, \ldots, \pi_n, r_n w_n^{1/(\alpha-1)}\right)\right)^{\alpha-1}.$$

The performance ratio is $\gamma_{\text{EE-}A} \leq C_{\text{EE-}A}$, *where the performance bound is*

$$C_{\text{EE-}A} = \frac{nm^{\alpha-1}\left(A\left(\pi_1, r_1 w_1^{1/(\alpha-1)}, \pi_2, r_2 w_2^{1/(\alpha-1)}, \ldots, \pi_n, r_n w_n^{1/(\alpha-1)}\right)\right)^{\alpha-1}}{(\pi_1 r_1 + \pi_2 r_2 + \cdots + \pi_n r_n)^{\alpha}}.$$

Since $C_{\text{EE-}A} = B_{\text{EE-}A}^{\alpha-1}$, we obtain the following approximation of the average-case performance-bound $\overline{C}_{\text{EE-SIMPLE}}$, i.e.,

$$\begin{aligned}
\overline{C}_{\text{EE-SIMPLE}} = \mathbb{E}(B_{\text{EE-SIMPLE}}^{\alpha-1}) &\approx \overline{B}_{\text{EE-SIMPLE}}^{\alpha-1} \\
&\approx \left(\frac{m}{P_m} \cdot \frac{n^{1/(\alpha-1)}\mathbb{E}\left(w_1^{\alpha/(\alpha-1)} + w_2^{\alpha/(\alpha-1)} + \cdots + w_n^{\alpha/(\alpha-1)}\right)}{\mathbb{E}((w_1 + w_2 + \cdots + w_n)^{\alpha/(\alpha-1)})}\right)^{\alpha-1}.
\end{aligned}$$

15.3.4 Analysis of Equal-Speed Algorithms

15.3.4.1 Minimizing Schedule Length

To solve the problem of minimizing schedule length with energy consumption constraint E by using an equal-speed algorithm ES-A, we have

$$s_1 = s_2 = \cdots = s_n = s,$$

where s is the identical task execution speed. Hence, we get

$$e_i = \pi_i r_i p_i^{1-1/\alpha} = w_i p_i^{1-1/\alpha} = w_i s_i^{\alpha-1} = w_i s^{\alpha-1},$$

for all $1 \leq i \leq n$. Since

$$\sum_{i=1}^{n} e_i = s^{\alpha-1} \sum_{i=1}^{n} w_i = W s^{\alpha-1} = E,$$

we obtain

$$s = \left(\frac{E}{W}\right)^{1/(\alpha-1)},$$

and

$$t_i = \frac{r_i}{s} = r_i \left(\frac{W}{E}\right)^{1/(\alpha-1)},$$

for all $1 \leq i \leq n$. Thus, we get the length of the schedule generated by algorithm ES-A as

$$T_{\text{ES-}A} = A(\pi_1, r_1, \pi_2, r_2, \ldots, \pi_n, r_n) \left(\frac{W}{E}\right)^{1/(\alpha-1)}.$$

By theorem 1, the performance ratio of algorithm ES-A is

$$\begin{aligned}
\beta_{\text{ES-}A} &= \frac{T_{\text{ES-}A}}{T_{\text{OPT}}} \\
&\leq \frac{A(\pi_1, r_1, \pi_2, r_2, \ldots, \pi_n, r_n)(W/E)^{1/(\alpha-1)}}{((m/E)(W/m)^{\alpha})^{1/(\alpha-1)}} \\
&= \frac{mA(\pi_1, r_1, \pi_2, r_2, \ldots, \pi_n, r_n)}{\pi_1 r_1 + \pi_2 r_2 + \cdots + \pi_n r_n}.
\end{aligned}$$

This discussion is summarized in the following theorem.

Theorem 15.7 *By using an equal-speed algorithm* ES-A *to solve the problem of minimizing schedule length with energy consumption constraint, the schedule length is*

$$T_{\text{ES-}A} = A(\pi_1, r_1, \pi_2, r_2, \ldots, \pi_n, r_n) \left(\frac{W}{E}\right)^{1/(\alpha-1)}.$$

The performance ratio is $\beta_{\text{ES-}A} \leq B_{\text{ES-}A}$, *where the performance bound is*

$$B_{\text{ES-}A} = \frac{mA(\pi_1, r_1, \pi_2, r_2, \ldots, \pi_n, r_n)}{\pi_1 r_1 + \pi_2 r_2 + \cdots + \pi_n r_n}.$$

For algorithm SIMPLE, we have

$$\overline{B}_{\text{ES-SIMPLE}} \approx \frac{m\mathbb{E}(\text{SIMPLE}(\pi_1, r_1, \pi_2, r_2, \ldots, \pi_n, r_n))}{\mathbb{E}(\pi_1 r_1 + \pi_2 r_2 + \cdots + \pi_n r_n)} \approx \frac{m}{P_m},$$

for large n.

15.3.4.2 Minimizing Energy Consumption

To solve the problem of minimizing energy consumption with schedule length constraint T by using an equal-speed algorithm ES-A, we need to provide enough energy $E_{\text{ES-}A}$, so that the deadline T is met, i.e., $T_{\text{ES-}A} = T$,

$$A(\pi_1, r_1, \pi_2, r_2, \ldots, \pi_n, r_n)\left(\frac{W}{E_{\text{ES-}A}}\right)^{1/(\alpha-1)} = T.$$

This equation implies that the amount of energy $E_{\text{ES-}A}$ consumed by algorithm ES-A is

$$E_{\text{ES-}A} = \left(\frac{A(\pi_1, r_1, \pi_2, r_2, \ldots, \pi_n, r_n)}{T}\right)^{\alpha-1} W.$$

By theorem 2, the performance ratio of algorithm ES-A is

$$\begin{aligned}
\gamma_{\text{ES-}A} &= \frac{E_{\text{ES-}A}}{E_{\text{OPT}}} \\
&\leq \frac{(A(\pi_1, r_1, \pi_2, r_2, \ldots, \pi_n, r_n)/T)^{\alpha-1} W}{(m/T^{\alpha-1})(W/m)^{\alpha}} \\
&= \frac{(A(\pi_1, r_1, \pi_2, r_2, \ldots, \pi_n, r_n))^{\alpha-1}}{(W/m)^{\alpha-1}} \\
&= \left(\frac{mA(\pi_1, r_1, \pi_2, r_2, \ldots, \pi_n, r_n)}{\pi_1 r_1 + \pi_2 r_2 + \cdots + \pi_n r_n}\right)^{\alpha-1}.
\end{aligned}$$

This discussion gives rise to the following theorem.

Theorem 15.8 *By using an equal-speed algorithm* ES-A *to solve the problem of minimizing energy consumption with schedule length constraint, the energy consumed is*

$$E_{\text{ES-}A} = \left(\frac{A(\pi_1, r_1, \pi_2, r_2, \ldots, \pi_n, r_n)}{T}\right)^{\alpha-1} W.$$

The performance ratio is $\gamma_{\text{ES-}A} \leq C_{\text{ES-}A}$, *where the performance bound is*

$$C_{\text{ES-}A} = \left(\frac{mA(\pi_1, r_1, \pi_2, r_2, \ldots, \pi_n, r_n)}{\pi_1 r_1 + \pi_2 r_2 + \cdots + \pi_n r_n}\right)^{\alpha-1}.$$

Since $C_{\text{ES-}A} = B_{\text{ES-}A}^{\alpha-1}$, we obtain the following approximation of the average-case performance-bound $\overline{C}_{\text{ES-SIMPLE}}$, i.e.,

$$\overline{C}_{\text{ES-SIMPLE}} = \mathbb{E}(B_{\text{ES-SIMPLE}}^{\alpha-1}) \approx \overline{B}_{\text{ES-SIMPLE}}^{\alpha-1} \approx \left(\frac{m}{P_m}\right)^{\alpha-1}.$$

15.3.5 Performance Data

In this section, we demonstrate numerical and simulation data for the average-case performance bounds derived for pre-power-determination algorithms in theorems 3 through 15.8.

Assume that there are $n = 1{,}000$ parallel tasks to be scheduled on a parallel computing system with $m = 128$ processors. The parameter α is set as 3. Although these parameters are for demonstration, they do not affect the observations and conclusion to be made.

The task execution requirements $r_1, r_2, \ldots, r_n$ are treated as i.i.d. continuous random variables uniformly distributed in $[0, 1)$. The task sizes $\pi_1, \pi_2, \ldots, \pi_n$ are i.i.d. discrete random variables in $[1 \ldots m]$.

We consider three types of probability distributions of task sizes with about the same expected task size $\bar{\pi}$. Let a_b be the probability that $\pi_i = b$, where $b \geq 1$.

- Uniform distributions in the range $[1 \ldots u]$, i.e., $a_b = 1/u$ for all $1 \leq b \leq u$, where u is chosen such that $(u + 1)/2 = \bar{\pi}$, i.e., $u = 2\bar{\pi} - 1$.
- Binomial distributions in the range $[1 \ldots m]$, i.e.,

$$a_b = \frac{\binom{m}{b} p^b (1-p)^{m-b}}{1 - (1-p)^m},$$

  for all $1 \leq b \leq m$, where p is chosen such that $mp = \bar{\pi}$, i.e., $p = \bar{\pi}/m$. However, the actual expectation of task sizes is

$$\frac{\bar{\pi}}{1 - (1-p)^m} = \frac{\bar{\pi}}{1 - (1 - \bar{\pi}/m)^m},$$

  which is slightly greater than $\bar{\pi}$, especially when $\bar{\pi}$ is small.
- Geometric distributions in the range $[1 \ldots m]$, i.e.,

$$a_b = \frac{q(1-q)^{b-1}}{1 - (1-q)^m},$$

  for all $1 \leq b \leq m$, where q is chosen such that $1/q = \bar{\pi}$, i.e., $q = 1/\bar{\pi}$. However, the actual expectation of task sizes is

$$\frac{1/q - (1/q + m)(1-q)^m}{1 - (1-q)^m} = \frac{\bar{\pi} - (\bar{\pi} + m)(1 - 1/\bar{\pi})^m}{1 - (1 - 1/\bar{\pi})^m},$$

  which is less than $\bar{\pi}$, especially when $\bar{\pi}$ is large.

In Table 15.1, we show the average-case performance-bound $\overline{B}_{\text{ET-SIMPLE}}$. For each combination of the expected task size $\bar{\pi} = 10, 20, 30, 40, 50, 60$ and the three probability distributions of task sizes, we

TABLE 15.1 Simulation Data for $\overline{B}_{\text{ET-SIMPLE}}$

Average Task Size	Uniform Distribution	Binomial Distribution	Geometric Distribution
10	1.4925696	1.4799331	1.5272638
20	1.5727972	1.5363483	1.6507309
30	1.6701122	1.5875832	1.7432123
40	1.7975673	1.6175625	1.7904198
50	1.9119195	1.8022895	1.8189020
60	1.9069588	1.6583340	1.8336815

show $\overline{B}_{\text{ET-SIMPLE}}$ obtained by random sampling as follows. We generate 500 sets of n parallel tasks, find $B_{\text{ET-SIMPLE}}$ by using theorem 15.3, and report the average of the 500 values of $B_{\text{ET-SIMPLE}}$, which is the experimental value of $\overline{B}_{\text{ET-SIMPLE}}$. In Table 15.2, we show our approximation of the average-case performance-bound $\overline{B}_{\text{ET-SIMPLE}}$. These data are obtained by using the same method of Table 15.1. In Table 15.3, we show the average-case performance-bound $\overline{B}_{\text{ET-GREEDY}}$.

In Table 15.4, we show the average-case performance-bound $\overline{C}_{\text{ET-SIMPLE}}$. For each combination of the expected task size $\bar{\pi} = 10, 20, 30, 40, 50, 60$ and the three probability distributions of task sizes, we show $\overline{C}_{\text{ET-SIMPLE}}$ obtained by random sampling as follows. We generate 500 sets of n parallel tasks, find $C_{\text{ET-SIMPLE}}$ by using theorem 15.4, and report the average of the 500 values of $C_{\text{ET-SIMPLE}}$, which is the experimental value of $\overline{C}_{\text{ET-SIMPLE}}$. In Table 15.5, we show our approximation of the average-case performance-bound $\overline{C}_{\text{ET-SIMPLE}}$. These data are obtained by using the same method of Table 15.4. In Table 15.6, we show the average-case performance-bound $\overline{C}_{\text{ET-GREEDY}}$.

TABLE 15.2 Simulation Data for the Approximation of $\overline{B}_{\text{ET-SIMPLE}}$

Average Task Size	Uniform Distribution	Binomial Distribution	Geometric Distribution
10	1.4838727	1.4712699	1.5175525
20	1.5673887	1.5321343	1.6497503
30	1.6657088	1.5823904	1.7527788
40	1.7812809	1.6123531	1.8235302
50	1.9030034	1.8004366	1.8632664
60	1.9571228	1.6383452	1.8903620

TABLE 15.3 Simulation Data for $\overline{B}_{\text{ET-GREEDY}}$

Average Task Size	Uniform Distribution	Binomial Distribution	Geometric Distribution
10	1.4310459	1.4299945	1.4431398
20	1.4385093	1.4304722	1.4728624
30	1.4752343	1.4447528	1.5030419
40	1.4796543	1.4884609	1.5269353
50	1.6343792	1.7111951	1.5403827
60	1.6434003	1.5080826	1.5556964

TABLE 15.4 Simulation Data for $\overline{C}_{\text{ET-SIMPLE}}$

Average Task Size	Uniform Distribution	Binomial Distribution	Geometric Distribution
10	2.2253142	2.1877799	2.3346273
20	2.4741347	2.3623308	2.7328413
30	2.7900444	2.5184688	3.0317472
40	3.2155913	2.6227083	3.1991694
50	3.6596691	3.2529483	3.3139123
60	3.6380009	2.7527856	3.3663785

TABLE 15.5 Simulation Data for the Approximation of $\overline{C}_{\text{ET-SIMPLE}}$

Average Task Size	Uniform Distribution	Binomial Distribution	Geometric Distribution
10	2.2013749	2.1650992	2.3096532
20	2.4662319	2.3443439	2.7233384
30	2.7792308	2.5079857	3.0749527
40	3.1699565	2.5930330	3.3145158
50	3.6138148	3.2443710	3.4685996
60	3.8208861	2.6877475	3.5659465

In Tables 15.7 through 15.12, we duplicate the same work for algorithms EE-SIMPLE and EE-GREEDY.

In Tables 15.13 through 15.18, we repeat the same work for algorithms ES-SIMPLE and ES-GREEDY. Notice that the data in Tables 15.14 and 15.17 are obtained by numerical calculation.

We would like to mention that the 99% confidence interval of all the data in the same table is no more than $\pm 0.8\%$.

TABLE 15.6 Simulation Data for $\overline{C}_{\text{ET-GREEDY}}$

Average Task Size	Uniform Distribution	Binomial Distribution	Geometric Distribution
10	2.0460829	2.0405601	2.0883940
20	2.0758287	2.0537063	2.1842573
30	2.1838240	2.0850937	2.2613991
40	2.1899949	2.2190162	2.3271776
50	2.6855169	2.9297280	2.3798546
60	2.7006062	2.2793767	2.4128476

TABLE 15.7 Simulation Data for $\overline{B}_{\text{EE-SIMPLE}}$

Average Task Size	Uniform Distribution	Binomial Distribution	Geometric Distribution
10	1.3463462	1.2348196	1.6112312
20	1.4298048	1.2560974	1.7861327
30	1.5369151	1.2928216	1.8688076
40	1.7015489	1.3220516	1.8833816
50	1.8260972	1.4423315	1.8774630
60	1.7745509	1.3775116	1.8612660

TABLE 15.8 Simulation Data for the Approximation of $\overline{B}_{\text{EE-SIMPLE}}$

Average Task Size	Uniform Distribution	Binomial Distribution	Geometric Distribution
10	1.3264845	1.2174092	1.5773889
20	1.4117628	1.2450593	1.7214958
30	1.5017826	1.2792598	1.7990156
40	1.6086828	1.2981559	1.8269234
50	1.7165236	1.4474411	1.8372114
60	1.7664283	1.3150274	1.8392359

TABLE 15.9 Simulation Data for $\overline{B}_{\text{EE-GREEDY}}$

Average Task Size	Uniform Distribution	Binomial Distribution	Geometric Distribution
10	1.2918285	1.1932594	1.5244995
20	1.3051615	1.1673758	1.5757269
30	1.3561090	1.1688848	1.5902360
40	1.3677546	1.1954258	1.5857617
50	1.5496023	1.3769236	1.5722843
60	1.5302645	1.2082060	1.5644404

TABLE 15.10 Simulation Data for $\overline{C}_{\text{EE-SIMPLE}}$

Average Task Size	Uniform Distribution	Binomial Distribution	Geometric Distribution
10	1.8114682	1.5255576	2.5931999
20	2.0482069	1.5777715	3.1880987
30	2.3612833	1.6696594	3.5059091
40	2.9001191	1.7473951	3.5492390
50	3.3359420	2.0831180	3.5155132
60	3.1424992	1.9012417	3.4714851

TABLE 15.11 Simulation Data for the Approximation of $\overline{C}_{\text{EE-SIMPLE}}$

Average Task Size	Uniform Distribution	Binomial Distribution	Geometric Distribution
10	1.7574598	1.4814359	2.4787496
20	1.9903160	1.5500319	2.9644179
30	2.2544732	1.6362618	3.2282039
40	2.5863755	1.6841439	3.3426538
50	2.9486812	2.0959486	3.3784349
60	3.1174110	1.7281041	3.3864316

TABLE 15.12 Simulation Data for $\overline{C}_{\text{EE-GREEDY}}$

Average Task Size	Uniform Distribution	Binomial Distribution	Geometric Distribution
10	1.6706632	1.4231051	2.3331851
20	1.7052260	1.3624873	2.5014480
30	1.8417139	1.3675744	2.5252715
40	1.8759147	1.4285966	2.4987712
50	2.3967202	1.8971794	2.4801984
60	2.3425787	1.4592520	2.4528297

TABLE 15.13 Simulation Data for $\overline{B}_{\text{ES-SIMPLE}}$

Average Task Size	Uniform Distribution	Binomial Distribution	Geometric Distribution
10	1.0612181	1.0519759	1.0904810
20	1.1183020	1.0897683	1.1905421
30	1.1916076	1.1270139	1.2693422
40	1.2985623	1.1529977	1.3120221
50	1.3915928	1.2759972	1.3375258
60	1.3963486	1.1952081	1.3525453

TABLE 15.14 Numerical Data for the Approximation of $\overline{B}_{\text{ES-SIMPLE}}$

Average Task Size	Uniform Distribution	Binomial Distribution	Geometric Distribution
10	1.0491804	1.0403200	1.0756198
20	1.1099183	1.0836783	1.1677707
30	1.1784512	1.1202275	1.2417018
40	1.2610587	1.1400834	1.2889239
50	1.3448804	1.2739033	1.3179703
60	1.3831981	1.1585732	1.3363460

TABLE 15.15 Simulation Data for $\overline{B}_{\text{ES-GREEDY}}$

Average Task Size	Uniform Distribution	Binomial Distribution	Geometric Distribution
10	1.0164086	1.0145500	1.0246841
20	1.0200092	1.0138670	1.0470670
30	1.0472299	1.0226195	1.0700968
40	1.0506965	1.0535358	1.0857458
50	1.1677252	1.2109840	1.0971333
60	1.1684183	1.0682941	1.1036731

We have the following observations from our simulations.

- The performance of the three power allocation algorithms are ranked as ET, EE, ES, from the worst to the best.
- The task scheduling algorithm GREEDY performs noticeably better than algorithm SIMPLE.
- All our approximate performance bounds are very accurate.

TABLE 15.16 Simulation Data for $\overline{C}_{\text{ES-SIMPLE}}$

Average Task Size	Uniform Distribution	Binomial Distribution	Geometric Distribution
10	1.1262599	1.1070876	1.1897538
20	1.2499551	1.1877677	1.4151224
30	1.4202129	1.2702833	1.6086410
40	1.6843206	1.3293841	1.7232975
50	1.9345899	1.6277295	1.7879323
60	1.9475815	1.4293379	1.8290383

TABLE 15.17 Numerical Data for the Approximation of $\overline{C}_{\text{ES-SIMPLE}}$

Average Task Size	Uniform Distribution	Binomial Distribution	Geometric Distribution
10	1.1007795	1.0822657	1.1569579
20	1.2319186	1.1743586	1.3636884
30	1.3887472	1.2549096	1.5418234
40	1.5902689	1.2997902	1.6613248
50	1.8087032	1.6228296	1.7370457
60	1.9132369	1.3422920	1.7858207

TABLE 15.18 Simulation Data for $\overline{C}_{\text{ES-GREEDY}}$

Average Task Size	Uniform Distribution	Binomial Distribution	Geometric Distribution
10	1.0340945	1.0288788	1.0494240
20	1.0405856	1.0282521	1.0972321
30	1.0974053	1.0459606	1.1451099
40	1.1046457	1.1095928	1.1824923
50	1.3555731	1.4671864	1.2022531
60	1.3672342	1.1413527	1.2230523

Due to space limitation, we do not provide the performance data for algorithms SIMPLE* and GREEDY*. Our main finding is that the performance of SIMPLE* is slightly better than that of SIMPLE, while the performance of GREEDY* is about the same as that of GREEDY.

15.4 Post-Power-Determination Algorithms

15.4.1 Overview

In post-power-determination algorithms, we first partition the system and schedule the tasks, and then determine power supplies to the n parallel tasks. Unfortunately, if we apply algorithm SIMPLE or GREEDY to generate a schedule of the tasks, it is not clear at all how power allocation can be performed, since power supplies also determine (and change) task execution speeds and times, and such change affects the schedule already produced. What we need to make sure is that power supplies do not change the schedule. Therefore, new system partitioning and task scheduling methods are required.

Our strategies for solving the three subproblems are described as follows.

- *System partitioning*: We use the harmonic system partitioning and processor allocation scheme, which divides a multiprocessor computer into clusters of equal sizes and schedules tasks of similar sizes together to increase processor utilization.
- *Task scheduling*: Our approach is to divide a list of tasks into sublists such that each sublist contains tasks of similar sizes which are scheduled on clusters of equal sizes. Scheduling such parallel tasks on clusters is no more difficult than scheduling sequential tasks and can be performed by list scheduling algorithms.

- *Power supplying*: We adopt a three-level energy/time/power allocation scheme for a given schedule, namely, optimal energy/time allocation among sublists of tasks (theorems 15.11 and 15.14), optimal energy allocation among groups of tasks in the same sublist (theorems 15.10 and 15.13), and optimal power supplies to tasks in the same group (theorems 15.9 and 15.12).

The post-power-determination algorithms and their analysis presented in this section are developed in [26].

15.4.2 System Partitioning

Our post-power-determination algorithms are called H_c-A, where "H_c" stands for the harmonic system partitioning scheme with parameter c to be presented later, and A is a list scheduling algorithm to be presented in the next section.

To schedule a list of n parallel tasks, algorithm H_c-A divides the list into c sublists according to task sizes (i.e., numbers of processors requested by tasks), where $c \geq 1$ is a positive integer constant. For $1 \leq j \leq c-1$, we define sublist j to be the sublist of tasks with

$$\frac{m}{j+1} < \pi_i \leq \frac{m}{j},$$

i.e., sublist j contains all tasks whose sizes are in the interval $I_j = (m/(j+1), m/j]$. We define sublist c to be the sublist of tasks with $0 < \pi_i \leq m/c$, i.e., sublist c contains all tasks whose sizes are in the interval $I_c = (0, m/c]$. The partition of $(0, m]$ into intervals $I_1, I_2, \ldots, I_j, \ldots, I_c$ is called *the harmonic system partitioning scheme* whose idea is to schedule tasks of similar sizes together. The similarity is defined by the intervals $I_1, I_2, \ldots, I_j, \ldots, I_c$. For tasks in sublist j, processor utilization is higher than $j/(j+1)$, where $1 \leq j \leq c-1$. As j increases, the similarity among tasks in sublist j increases and processor utilization also increases. Hence, the harmonic system partitioning scheme is very good at handling small tasks.

Algorithm H_c-A produces schedules of the sublists sequentially and separately. To schedule tasks in sublist j, where $1 \leq j \leq c$, the m processors are partitioned into j *clusters* and each cluster contains m/j processors. Each cluster of processors is treated as one unit to be allocated to one task in sublist j. This is basically the harmonic system partitioning and processor allocation scheme. Therefore, scheduling parallel tasks in sublist j on the j clusters where each task i has processor requirement π_i and execution requirement r_i is equivalent to scheduling a list of sequential tasks on j processors where each task i has execution requirement r_i. It is clear that scheduling of the list of sequential tasks on j processors can be accomplished by using algorithm A, where A is a list scheduling algorithm.

15.4.3 Task Scheduling

When a multiprocessor computer with m processors is partitioned into $j \geq 1$ clusters, scheduling tasks in sublist j is essentially dividing sublist j into j groups of tasks, such that each group of tasks are executed on one cluster. Such a partition of sublist j into j groups is essentially a schedule of the tasks in sublist j on m processors with j clusters. Once a partition (i.e., a schedule) is determined, we can use the methods in the next section to find power supplies.

We propose to use the list scheduling algorithm and its variations to solve the task scheduling problem. Tasks in sublist j are scheduled on j clusters by using the classic list scheduling algorithm [15] and by ignoring the issue of power supplies. In other words, the task execution times are simply the task execution requirements $r_1, r_2, \ldots, r_n$, and tasks are assigned to the j clusters (i.e., groups) by using the list scheduling algorithm, which works as follows to schedule a list of tasks $1, 2, 3 \ldots$.

- *List scheduling* (*LS*): Initially, task k is scheduled on cluster (or group) k, where $1 \leq k \leq j$, and tasks $1, 2, \ldots, j$ are removed from the list. Upon the completion of a task k, the first unscheduled

task in the list, i.e., task $j+1$, is removed from the list and scheduled to be executed on cluster 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 several of them here.

- *Largest requirement first* (*LRF*): This algorithm is the same as the LS algorithm, except that the tasks are arranged such that $r_1 \geq r_2 \geq \cdots \geq r_n$.
- *Smallest requirement first* (*SRF*): This algorithm is the same as the LS algorithm, except that the tasks are arranged such that $r_1 \leq r_2 \leq \cdots \leq r_n$.
- *Largest size first* (*LSF*): This algorithm is the same as the LS algorithm, except that the tasks are arranged such that $\pi_1 \geq \pi_2 \geq \cdots \geq \pi_n$.
- *Smallest size first* (*SSF*): This algorithm is the same as the LS algorithm, except that the tasks are arranged such that $\pi_1 \leq \pi_2 \leq \cdots \leq \pi_n$.
- *Largest task first* (*LTF*): This algorithm is the same as the LS algorithm, except that the tasks are arranged such that $\pi_1^{1/\alpha} r_1 \geq \pi_2^{1/\alpha} r_2 \geq \cdots \geq \pi_n^{1/\alpha} r_n$.
- *Smallest task first* (*STF*): This algorithm is the same as the LS algorithm, except that the tasks are arranged such that $\pi_1^{1/\alpha} r_1 \leq \pi_2^{1/\alpha} r_2 \leq \cdots \leq \pi_n^{1/\alpha} r_n$.

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

15.4.4 Power Supplying

Once the n parallel tasks are divided into c sublists and tasks in sublist j are further partitioned into j groups, power supplies which minimize the schedule length within energy consumption constraint or the energy consumption within schedule length constraint can be determined. We adopt a three-level energy/time/power allocation scheme for a given schedule, namely,

- Optimal power supplies to tasks in the same group (theorems 15.9 and 15.12)
- Optimal energy allocation among groups of tasks in the same sublist (theorems 15.10 and 15.13)
- Optimal energy/time allocation among sublists of tasks (theorems 15.11 and 15.14)

15.4.4.1 Minimizing Schedule Length

We first consider optimal power supplies to tasks in the same group. Notice that tasks in the same group are executed sequentially. In fact, we consider a more general case, i.e., n parallel tasks with sizes $\pi_1, \pi_2, \ldots, \pi_n$ and execution requirements $r_1, r_2, \ldots, r_n$ to be executed sequentially one by one. Let

$$M = \pi_1^{1/\alpha} r_1 + \pi_2^{1/\alpha} r_2 + \cdots + \pi_n^{1/\alpha} r_n.$$

The following result gives the optimal power supplies when the n parallel tasks are scheduled sequentially.

(*Note*: Due to space limitation, the proofs of all theorems in this section are omitted, and the interested reader is referred to [26].)

Theorem 15.9 *When n parallel tasks are scheduled sequentially, the schedule length is minimized when task i is supplied with power* $p_i = (E/M)^{\alpha/(\alpha-1)}/\pi_i$, *where* $1 \leq i \leq n$. *The optimal schedule length is* $T = M^{\alpha/(\alpha-1)}/E^{1/(\alpha-1)}$.

Now, we consider optimal energy allocation among groups of tasks in the same sublist. Again, we discuss group-level energy allocation in a more general case, i.e., scheduling n parallel tasks on m processors, where $\pi_i \leq m/j$ for all $1 \leq i \leq n$ with $j \geq 1$. In this case, the m processors can be partitioned into j clusters, such that each cluster contains m/j processors. Each cluster of processors are treated as one unit to be allocated to one task. Assume that the set of n tasks is partitioned into j groups, such that all the tasks in group k are executed on cluster k, where $1 \leq k \leq j$. Let M_k denote the total $\pi_i^{1/\alpha} r_i$ of the

tasks in group k. For a given partition of the n tasks into j groups, we are seeking power supplies that minimize the schedule length. Let E_k be the energy consumed by all the tasks in group k. The following result characterizes the optimal power supplies.

Theorem 15.10 *For a given partition $M_1, M_2, \ldots, M_j$ of n parallel tasks into j groups on a multiprocessor computer partitioned into j clusters, the schedule length is minimized when task i in group k is supplied with power $p_i = (E_k/M_k)^{\alpha/(\alpha-1)}/\pi_i$, where*

$$E_k = \left(\frac{M_k^\alpha}{M_1^\alpha + M_2^\alpha + \cdots + M_j^\alpha}\right) E,$$

for all $1 \leq k \leq j$. The optimal schedule length is

$$T = \left(\frac{M_1^\alpha + M_2^\alpha + \cdots + M_j^\alpha}{E}\right)^{1/(\alpha-1)},$$

for the previously mentioned power supplies.

To use algorithm H_c-A to solve the problem of minimizing schedule length with energy consumption constraint E, we need to allocate the available energy E to the c sublists. We use $E_1, E_2, \ldots, E_c$ to represent an energy allocation to the c sublists, where sublist j consumes energy E_j, and $E_1 + E_2 + \cdots + E_c = E$. By using any of the list scheduling algorithms to schedule tasks in sublist j, we get a partition of the tasks in sublist j into j groups. Let R_j be the total execution requirement of tasks in sublist j, and $R_{j,k}$ be the total execution requirement of tasks in group k, and $M_{j,k}$ be the total $\pi_i^{1/\alpha} r_i$ of tasks in group k, where $1 \leq k \leq j$. Theorem 15.11 provides optimal energy allocation to the c sublists for minimizing schedule length with energy consumption constraint in scheduling parallel tasks by using scheduling algorithms H_c-A, where A is a list scheduling algorithm. The theorem also gives the performance bound when algorithms H_c-A are used to solve the problem of minimizing schedule length with energy consumption constraint.

Theorem 15.11 *For a given partition $M_{j,1}, M_{j,2}, \ldots, M_{j,j}$ of the tasks in sublist j into j groups produced by a list scheduling algorithm A, where $1 \leq j \leq c$, and an energy allocation $E_1, E_2, \ldots, E_c$ to the c sublists, the length of the schedule produced by algorithm H_c-A is*

$$T_{H_c\text{-}A} = \sum_{j=1}^{c} \left(\frac{M_{j,1}^\alpha + M_{j,2}^\alpha + \cdots + M_{j,j}^\alpha}{E_j}\right)^{1/(\alpha-1)}.$$

The energy allocation $E_1, E_2, \ldots, E_c$ which minimizes $T_{H_c\text{-}A}$ is

$$E_j = \left(\frac{N_j^{1/\alpha}}{N_1^{1/\alpha} + N_2^{1/\alpha} + \cdots + N_c^{1/\alpha}}\right) E,$$

where $N_j = M_{j,1}^\alpha + M_{j,2}^\alpha + \cdots + M_{j,j}^\alpha$, for all $1 \leq j \leq c$, and the minimized schedule length is

$$T_{H_c\text{-}A} = \frac{\left(N_1^{1/\alpha} + N_2^{1/\alpha} + \cdots + N_c^{1/\alpha}\right)^{\alpha/(\alpha-1)}}{E^{1/(\alpha-1)}},$$

by using the previous energy allocation. The performance ratio is $\beta_{H_c\text{-}A} \leq B_{H_c\text{-}A}$, *where*

$$B_{H_c\text{-}A} = \left(\frac{\left(N_1^{1/\alpha} + N_2^{1/\alpha} + \cdots + N_c^{1/\alpha}\right)^{\alpha}}{m(W/m)^{\alpha}} \right)^{1/(\alpha-1)}.$$

Furthermore, we have

$$B_{H_c\text{-}A} \approx \left(\left(\sum_{j=1}^{c} \frac{R_j}{j} \left(\frac{2j+1}{2j+2} \right)^{1/\alpha} \right) \Big/ \left(\frac{W}{m} \right) \right)^{\alpha/(\alpha-1)}.$$

Theorems 15.10 and 15.11 give the power supply to the task i in group k of sublist j as

$$\frac{1}{\pi_i} \left(\frac{E_{j,k}}{M_{j,k}} \right)^{\alpha/(\alpha-1)} = \frac{1}{\pi_i} \left(\left(\frac{M_{j,k}^{\alpha}}{M_{j,1}^{\alpha} + M_{j,2}^{\alpha} + \cdots + M_{j,j}^{\alpha}} \right) \left(\frac{N_j^{1/\alpha}}{N_1^{1/\alpha} + N_2^{1/\alpha} + \cdots + N_c^{1/\alpha}} \right) \frac{E}{M_{j,k}} \right)^{\alpha/(\alpha-1)},$$

for all $1 \leq j \leq c$ and $1 \leq k \leq j$.

15.4.4.2 Minimizing Energy Consumption

The following result gives the optimal power supplies when n parallel tasks are scheduled sequentially.

Theorem 15.12 *When n parallel tasks are scheduled sequentially, the total energy consumption is minimized when task i is supplied with power* $p_i = (M/T)^{\alpha}/\pi_i$, *where* $1 \leq i \leq n$. *The minimum energy consumption is* $E = M^{\alpha}/T^{\alpha-1}$.

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

Theorem 15.13 *For a given partition* $M_1, M_2, \ldots, M_j$ *of n parallel tasks into j groups on a multiprocessor computer partitioned into j clusters, the total energy consumption is minimized when task i in group k is executed with power* $p_i = (M_k/T)^{\alpha}/\pi_i$, *where* $1 \leq k \leq j$. *The minimum energy consumption is*

$$E = \frac{M_1^{\alpha} + M_2^{\alpha} + \cdots + M_j^{\alpha}}{T^{\alpha-1}}$$

for the previously mentioned power supplies.

To use algorithm H_c-A to solve the problem of minimizing energy consumption with schedule length constraint T, we need to allocate the time T to the c sublists. We use $T_1, T_2, \ldots, T_c$ to represent a time allocation to the c sublists, where tasks in sublist j are executed within deadline T_j, and $T_1 + T_2 + \cdots + T_c = T$. Theorem 15.14 provides optimal time allocation to the c sublists for minimizing energy consumption with schedule length constraint in scheduling parallel tasks by using scheduling algorithms H_c-A, where A is a list scheduling algorithm. The theorem also gives the performance bound when algorithms H_c-A is used to solve the problem of minimizing energy consumption with schedule length constraint.

Theorem 15.14 *For a given partition $M_{j,1}, M_{j,2}, \ldots, M_{j,j}$ of the tasks in sublist j into j groups produced by a list scheduling algorithm A, where $1 \le j \le c$, and a time allocation $T_1, T_2, \ldots, T_c$ to the c sublists, the amount of energy consumed by algorithm H_c-A is*

$$E_{H_c\text{-}A} = \sum_{j=1}^{c} \left(\frac{M_{j,1}^{\alpha} + M_{j,2}^{\alpha} + \cdots + M_{j,j}^{\alpha}}{T_j^{\alpha-1}} \right).$$

The time allocation $T_1, T_2, \ldots, T_c$ which minimizes $E_{H_c\text{-}A}$ is

$$T_j = \left(\frac{N_j^{1/\alpha}}{N_1^{1/\alpha} + N_2^{1/\alpha} + \cdots + N_c^{1/\alpha}} \right) T,$$

where $N_j = M_{j,1}^{\alpha} + M_{j,2}^{\alpha} + \cdots + M_{j,j}^{\alpha}$, for all $1 \le j \le c$, and the minimized energy consumption is

$$E_{H_c\text{-}A} = \frac{\left(N_1^{1/\alpha} + N_2^{1/\alpha} + \cdots + N_c^{1/\alpha} \right)^{\alpha}}{T^{\alpha-1}},$$

by using the previous time allocation. The performance ratio is $\gamma_{H_c\text{-}A} \le C_{H_c\text{-}A}$, where

$$C_{H_c\text{-}A} = \frac{\left(N_1^{1/\alpha} + N_2^{1/\alpha} + \cdots + N_c^{1/\alpha} \right)^{\alpha}}{m(W/m)^{\alpha}}.$$

Furthermore, we have

$$C_{H_c\text{-}A} \approx \left(\left(\sum_{j=1}^{c} \frac{R_j}{j} \left(\frac{2j+1}{2j+2} \right)^{1/\alpha} \right) \Big/ \left(\frac{W}{m} \right) \right)^{\alpha}.$$

Theorems 15.13 and 15.14 give the power supply to task *i* in group *k* of sublist *j* as

$$\frac{1}{\pi_i} \left(\frac{M_{j,k}}{T_j} \right)^{\alpha} = \frac{1}{\pi_i} \left(\frac{M_{j,k} \left(N_1^{1/\alpha} + N_2^{1/\alpha} + \cdots + N_c^{1/\alpha} \right)}{N_j^{1/\alpha} T} \right)^{\alpha},$$

for all $1 \le j \le c$ and $1 \le k \le j$.

15.4.5 Performance Data

In this section, we demonstrate numerical and simulation data for the average-case performance bounds derived for pre-power-determination algorithms in theorems 15.11 and 15.14.

All parameters are set in the same way as pre-power-determination algorithms.

In Table 15.19, we show the average-case performance-bound $\overline{B}_{H_c\text{-LS}}$. For each combination of the expected task size $\bar{\pi} = 10, 20, 30, 40, 50, 60$ and the three probability distributions of task sizes, we show $\overline{B}_{H_c\text{-LS}}$ obtained by random sampling as follows. We generate 200 sets of *n* parallel tasks, find $B_{H_c\text{-LS}}$ by using theorem 15.11, and report the average of the 200 values of $B_{H_c\text{-LS}}$, which is the experimental value of $\overline{B}_{H_c\text{-LS}}$. In Table 15.20, we show our approximation of the average-case performance-bound $\overline{B}_{H_c\text{-LS}}$. These data are obtained by using the same method of Table 15.19.

In Table 15.21, we show the average-case performance-bound $\overline{C}_{H_c\text{-LS}}$. For each combination of the expected task size $\bar{\pi} = 10, 20, 30, 40, 50, 60$ and the three probability distributions of task sizes, we show $\overline{C}_{H_c\text{-LS}}$ obtained by random sampling as follows. We generate 200 sets of *n* parallel tasks, find $C_{H_c\text{-LS}}$ by

TABLE 15.19 Simulation Data for $\overline{B}_{H_c\text{-LS}}$

Average Task Size	Uniform Distribution	Binomial Distribution	Geometric Distribution
10	1.1310152	1.0710001	1.2195685
20	1.1374080	1.0998380	1.2254590
30	1.1921506	1.1376307	1.2639101
40	1.3722257	1.2113868	1.2818346
50	1.3957714	1.2486290	1.2907563
60	1.3266285	1.2804629	1.2935460

TABLE 15.20 Simulation Data for the Approximation of $\overline{B}_{H_c\text{-LS}}$

Average Task Size	Uniform Distribution	Binomial Distribution	Geometric Distribution
10	1.1850743	1.0637377	1.3192655
20	1.1495195	1.0823137	1.2730873
30	1.2030697	1.1397329	1.3026445
40	1.4479539	1.2382998	1.3166660
50	1.4619769	1.2776203	1.3207317
60	1.3441788	1.3188035	1.3190466

TABLE 15.21 Simulation Data for $\overline{C}_{H_c\text{-LS}}$

Average Task Size	Uniform Distribution	Binomial Distribution	Geometric Distribution
10	1.2775988	1.1481701	1.4868035
20	1.2942669	1.2084384	1.5021708
30	1.4215784	1.2940830	1.5991783
40	1.8829298	1.4667344	1.6447990
50	1.9496348	1.5596231	1.6669842
60	1.7650820	1.6392893	1.6759810

TABLE 15.22 Simulation Data for the Approximation of $\overline{C}_{H_c\text{-LS}}$

Average Task Size	Uniform Distribution	Binomial Distribution	Geometric Distribution
10	1.4023173	1.1313445	1.7373603
20	1.3197428	1.1711625	1.6161178
30	1.4463168	1.2997302	1.7008003
40	2.0986717	1.5306295	1.7312586
50	2.1356210	1.6311186	1.7435018
60	1.8069105	1.7392907	1.7393141

using theorem 15.14, and report the average of the 200 values of $C_{H_c\text{-LS}}$, which is the experimental value of $\overline{C}_{H_c\text{-LS}}$. In Table 15.22, we show our approximation of the average-case performance-bound $\overline{C}_{H_c\text{-LS}}$. These data are obtained by using the same method of Table 15.21.

We would like to mention that the 99% confidence interval of all the data in the same table is no more than $\pm 0.6\%$.

We have the following observations from our simulations.

- $\overline{B}_{H_c\text{-LS}}$ is less than 1.4 and $\overline{C}_{H_c\text{-LS}}$ is less than 1.95. Since $\overline{B}_{H_c\text{-LS}}$ and $\overline{C}_{H_c\text{-LS}}$ only give upper bonds for the average-case performance ratios, the performance of our heuristic algorithms is even better and our heuristic algorithms are able to produce solutions very close to optimum.
- The performance of algorithm H_c-A for A other than LS (i.e., LRF, SRF, LSF, SSF, LTF, STF) is very close (within $\pm 1\%$) to the performance of algorithm H_c-LS. Since these data do not provide further insight, they are not shown here.
- Our approximate performance bounds are very accurate.

It is interesting to compare the performance of pre-power-determination algorithms and post-power-determination algorithms. We find that algorithm H_c-LS performs better than ET-SIMPLE, ET-GREEDY, EE-SIMPLE, EE-GREEDY, and is comparable with ES-SIMPLE; however, it performs worse than ES-GREEDY. Although post-power-determination provides an optimal power allocation for a given schedule, the performance of H_c-LS is still worse than ES-GREEDY due to the inefficiency of the harmonic system partition scheme.

15.5 Summary

We have developed a number of pre-power-determination algorithms and post-power-determination algorithms for energy-aware scheduling of parallel tasks. Our best heuristic algorithms (ES-GREEDY and ES-GREEDY*) are able to produce solutions very close to optimum.

It is of great interest to extend our algorithms to other task models, processor models, and scheduling models described in [27].

Acknowledgment

The material presented in this chapter is based in part upon the author's work in [26].

References

1. S. Albers, Energy-efficient algorithms, *Communications of the ACM*, 53(5), 86–96, 2010.
2. H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alvarez, Power-aware scheduling for periodic real-time tasks, *IEEE Transactions on Computers*, 53(5), 584–600, 2004.
3. N. Bansal, T. Kimbrel, and K. Pruhs, Dynamic speed scaling to manage energy and temperature, *Proceedings of the 45th IEEE Symposium on Foundation of Computer Science*, Rome, Italy, 520–529, 2004.
4. J. A. Barnett, Dynamic task-level voltage scheduling optimizations, *IEEE Transactions on Computers*, 54(5), 508–520, 2005.
5. L. Benini, A. Bogliolo, and G. De Micheli, A survey of design techniques for system-level dynamic power management, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(3), 299–316, 2000.
6. D. P. Bunde, Power-aware scheduling for makespan and flow, *Proceedings of the 18th ACM Symposium on Parallelism in Algorithms and Architectures*, Cambridge, MA, p. 190–196, 2006.
7. H.-L. Chan, W.-T. Chan, T.-W. Lam, L.-K. Lee, K.-S. Mak, and P. W. H. Wong, Energy efficient online deadline scheduling, *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms*, New Orleans, LA, p. 795–804, 2007.
8. A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, Low-power CMOS digital design, *IEEE Journal on Solid-State Circuits*, 27(4), 473–484, 1992.
9. S. Cho and R. G. Melhem, On the interplay of parallelization, program performance, and energy consumption, *IEEE Transactions on Parallel and Distributed Systems*, 21(3), 342–353, 2010.
10. E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson, Approximation algorithms for bin packing: A survey, in *Approximation Algorithms for NP-Hard Problems*, D. S. Hochbaum, ed., PWS Publishing Company, Boston, MA, 1997.
11. W.-C. Feng, The importance of being low power in high performance computing, *Cyber Infrastructure Watch Quarterly*, 1(3), Los Alamos National Laboratory, August 2005.
12. W.-C. Feng and K. W. Cameron, The Green500 list: Encouraging sustainable supercomputing, *Computer*, 40(12), 50–55, 2007.

13. A. Gara, et al., Overview of the Blue Gene/L system architecture, *IBM Journal of Research and Development*, 49(2/3), 195–212, 2005.
14. M. R. Garey and D. S. Johnson, *Computers and Intractability—A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, 1979.
15. R. L. Graham, Bounds on multiprocessing timing anomalies, *SIAM Journal of Applied Mathematics*, 2, 416–429, 1969.
16. S. L. Graham, M. Snir, and C. A. Patterson, eds., Getting up to speed: The future of supercomputing, Committee on the Future of Supercomputing, National Research Council, National Academies Press, Washington, DC, 2005.
17. I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. B. Srivastava, Power optimization of variable-voltage core-based systems, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(12), 1702–1714, 1999.
18. C. Im, S. Ha, and H. Kim, Dynamic voltage scheduling with buffers in low-power multimedia applications, *ACM Transactions on Embedded Computing Systems*, 3(4), 686–705, 2004.
19. S. U. Khan and I. Ahmad, A cooperative game theoretical technique for joint optimization of energy consumption and response time in computational grids, *IEEE Transactions on Parallel and Distributed Systems*, 20(3), 346–360, 2009.
20. C. M. Krishna and Y.-H. Lee, Voltage-clock-scaling adaptive scheduling techniques for low power in hard real-time systems, *IEEE Transactions on Computers*, 52(12), 1586–1593, 2003.
21. W.-C. Kwon and T. Kim, Optimal voltage allocation techniques for dynamically variable voltage processors, *ACM Transactions on Embedded Computing Systems*, 4(1), 211–230, 2005.
22. Y. C. Lee and A. Y. Zomaya, Energy conscious scheduling for distributed computing systems under different operating conditions, *IEEE Transactions on Parallel and Distributed Systems*, 272–279, 1999.
23. Y.-H. Lee and C. M. Krishna, Voltage-clock scaling for low energy consumption in fixed-priority real-time systems, *Real-Time Systems*, 24(3), 303–317, 2003.
24. K. Li, An average-case analysis of online non-clairvoyant scheduling of independent parallel tasks, *Journal of Parallel and Distributed Computing*, 66(5), 617–625, 2006.
25. K. Li, Performance analysis of power-aware task scheduling algorithms on multiprocessor computers with dynamic voltage and speed, *IEEE Transactions on Parallel and Distributed Systems*, 19(11), 1484–1497, 2008.
26. K. Li, Energy efficient scheduling of parallel tasks on multiprocessor computers, *Journal of Supercomputing*, 2011.
AQ1
27. K. Li, Power allocation and task scheduling on multiprocessor computers with energy and time constraints, *Energy Aware Distributed Computing Systems*, Y.-C. Lee and A. Zomaya, eds., Wiley Series on Parallel and Distributed Computing, 2011.
28. M. Li, B. J. Liu, and F. F. Yao, Min-energy voltage allocation for tree-structured tasks, *Journal of Combinatorial Optimization*, 11, 305–319, 2006.
29. M. Li, A. C. Yao, and F. F. Yao, Discrete and continuous min-energy schedules for variable voltage processors, *Proceedings of the National Academy of Sciences USA*, 103(11), pp. 3983–3987, 2006.
30. M. Li and F. F. Yao, An efficient algorithm for computing optimal discrete voltage schedules, *SIAM Journal on Computing*, 35(3), 658–671, 2006.
31. J. R. Lorch and A. J. Smith, PACE: A new approach to dynamic voltage scaling, *IEEE Transactions on Computers*, 53(7), 856–869, 2004.
32. R. N. Mahapatra and W. Zhao, An energy-efficient slack distribution technique for multimode distributed real-time embedded systems, *IEEE Transactions on Parallel and Distributed Systems*, 16(7), 650–662, 2005.
33. A. J. Martin, Towards an energy complexity of computation, *Information Processing Letters*, 77, 181–187, 2001.
34. G. Quan and X. S. Hu, Energy efficient DVS schedule for fixed-priority real-time systems, *ACM Transactions on Embedded Computing Systems*, 6(4), Article no. 29, 2007.

35. C. Rusu, R. Melhem, D. Mossé, Maximizing the system value while satisfying time and energy constraints, *Proceedings of the 23rd IEEE Real-Time Systems Symposium*, Austin, TX, pp. 256–265, 2002.
36. D. Shin and J. Kim, Power-aware scheduling of conditional task graphs in real-time multiprocessor systems, *Proceedings of the International Symposium on Low Power Electronics and Design*, Fukuoka, Japan, pp. 408–413, 2003.
37. D. Shin, J. Kim, and S. Lee, Intra-task voltage scheduling for low-energy hard real-time applications, *IEEE Design & Test of Computers*, 18(2), 20–30, 2001.
38. M. R. Stan and K. Skadron, Guest editors' introduction: Power-aware computing, *IEEE Computer*, 36(12) 35–38, 2003.
39. O. S. Unsal and I. Koren, System-level power-aware design techniques in real-time systems, *Proceedings of the IEEE*, 91(7), 1055–1069, 2003.
40. V. Venkatachalam and M. Franz, Power reduction techniques for microprocessor systems, *ACM Computing Surveys*, 37(3), 195–237, 2005.
41. M. Weiser, B. Welch, A. Demers, and S. Shenker, Scheduling for reduced CPU energy, *Proceedings of the First USENIX Symposium on Operating Systems Design and Implementation*, San Diego, CA, pp. 13–23, 1994.
42. P. Yang, C. Wong, P. Marchal, F. Catthoor, D. Desmet, D. Verkest, and R. Lauwereins, Energy-aware runtime scheduling for embedded-multiprocessor SOCs, *IEEE Design & Test of Computers*, 18(5), 46–58, 2001.
43. F. Yao, A. Demers, and S. Shenker, A scheduling model for reduced CPU energy, *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science*, Washington DC, pp. 374–382, 1995.
44. H.-S. Yun and J. Kim, On energy-optimal voltage scheduling for fixed-priority hard real-time systems, *ACM Transactions on Embedded Computing Systems*, 2(3), 393–430, 2003.
45. B. Zhai, D. Blaauw, D. Sylvester, and K. Flautner, Theoretical and practical limits of dynamic voltage scaling, *Proceedings of the 41st Design Automation Conference*, San Diego, CA, pp. 868–873, 2004.
46. X. Zhong and C.-Z. Xu, Energy-aware modeling and scheduling for dynamic voltage scaling with statistical real-time guarantee, *IEEE Transactions on Computers*, 56(3), 358–372, 2007.
47. D. Zhu, R. Melhem, and B. R. Childers, Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems, *IEEE Transactions on Parallel and Distributed Systems*, 14(7), 686–700, 2003.
48. D. Zhu, D. Mossé, and R. Melhem, Power-aware scheduling for AND/OR graphs in real-time systems, *IEEE Transactions on Parallel and Distributed Systems*, 15(9), 849–864, 2004.
49. J. Zhuo and C. Chakrabarti, Energy-efficient dynamic task scheduling algorithms for DVS systems, *ACM Transactions on Embedded Computing Systems*, 7(2), Article no. 17, 2008.

AUTHOR QUERY

[AQ1] Please provide volume number and page numbers for the Ref. [26]