Future Generation Computer Systems 82 (2018) 591–605

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Scheduling parallel tasks with energy and time constraints on multiple manycore processors in a cloud computing environment

Keqin Li *

Department of Computer Science, State University of New York, New Paltz, New York 12561, USA

HIGHLIGHTS

- Addressing energy and time constrained scheduling of precedence constrained parallel tasks.
- Deriving lower bounds for optimal solutions.
- Developing pre-power-determination algorithms and post-power-determination algorithms.
- Considering multiple manycore processors with continuous or discrete speed levels.
- Evaluating the performance of these algorithms analytically and experimentally.

ARTICLE INFO

Article history:
Received 29 February 2016
Received in revised form
28 December 2016
Accepted 7 January 2017
Available online 11 January 2017

Keywords:
Cloud computing
Energy constrained scheduling
Manycore processor
Parallel tasks
Performance evaluation
Precedence constraint
Simulation
Time constrained scheduling

ABSTRACT

When multiple manycore processors in a data center for cloud computing are shared by a large number parallel tasks simultaneously, we are facing the problem of allocating the cores to the tasks and scheduling the tasks, such that the system performance is optimized or the energy consumption is minimized. Furthermore, such core allocation and task scheduling should be conducted with energy constraints or performance constraints. The problems of energy and time constrained scheduling of precedence constrained parallel tasks on multiple manycore processors in a cloud computing environment are defined as optimization problems. Lower bounds for optimal solutions are generalized from a single parallel computing system to multiple parallel computing systems. Our approach in this paper is to design and analyze the performance of heuristic algorithms that employ the equal-speed method. Pre-power-determination algorithms and post-power-determination algorithms are developed for both energy and time constrained scheduling of precedence constrained parallel tasks on multiple manycore processors with continuous or discrete speed levels. The performance of these algorithms are evaluated analytically and experimentally. Our main strategy is to embed the equal-speed method into our algorithms, which not only makes our analysis possible, but also yields good performance of our algorithms.

1. Introduction

1.1. Motivation

Exascale computing is the next major goal of high-performance computing (HPC), which is very important to modern economy and businesses. The Exascale Computing Project (ECP) of US Department of Energy will enable a new era of computational capabilities by advancing HPC on an exponential scale. The creation of a capable exascale ecosystem will have profound effects on the lives of Americans, improving the nation's economic competitiveness, scientific capabilities, and national security. The Council on Competitiveness Study of US Industrial HPC Users, sponsored by the Defense Advanced Research Projects Agency (DARPA) and conducted by market research firm IDC, found that 97% of the US companies surveyed could not exist or could not compete effectively, without the use of HPC [1, p. 83]. Achieving energy-efficient exascale computing is a major technical challenge, since there is a consensus that an exascale computing system should not consume more than 20 megawatts of power [2]. Unfortunately, according to Green500, which ranks the most energy-efficient supercomputers in the world, energy efficiency is improving exponentially for the top 25% of datacenters, but the median is only slightly improving, meaning a lot of datacenters are lagging in efficiency. Achieving the landmark goal 50,000 MFLOPS per watt is driving datacenters to meet the power and cooling

* Fax: +1 845 257 3996.
E-mail address: lik@newpaltz.edu.

http://dx.doi.org/10.1016/j.future.2017.01.010
0167-739X/

requirements necessary to house the computational power at the scale of 10^{18} FLOPS.

The energy consumption of data centers is a hot topic that has received growing attention (see [3] for a recent survey), given that data centers currently account for 1%–2% of global electricity use. However, cloud computing holds great potential to reduce data center energy demand, due to large reduction in total servers through consolidation and large increase in facility efficiency compared to traditional local data centers. The high energy efficiency in cloud computing is realized by replacing a large number of servers with low utilization and efficiency, and inefficient and small-scale cooling systems without in-house professionals by a small number of cloud servers with high utilization and efficiency, and advanced and continuously optimized and highly efficient cooling systems. Therefore, it has been widely recognized that cloud-based computing cannot only reduce IT capital costs, reduce labor costs, and enhance productivity, but also provide a promising and ultimate way of managing and improving the resource utilization and energy efficiency of current and future data and computing centers [4].

A growing body of evidence reveals that the cloud is remarkably energy-efficient. Based on a recent Google analysis, a typical company or organization that migrates to the cloud can save 68%–87% in energy cost for office computing and reduce similar amount of carbon emission. Replicating these savings across the entire economy would have a profound impact. According to a recent study by the Carbon Disclosure Project, by migrating to cloud computing, large US companies could achieve annual energy saving of 12.3 billion USD and carbon reduction of 85.7 million metric tonnes by 2020 (equivalent to the annual emissions of over 16.8 million passenger vehicles) [5]. In another recent study, it was reported that if all US business users shift their email, productivity software, and customer relationship management software to the cloud, the energy consumption of these software applications might be reduced by as much as 87%, or 326 Petajoules, enough energy to generate the electricity used by the City of Los Angeles each year (23 billion kilowatt-hours) [6].

One effective way of achieving high energy efficiency is to employ multicore and manycore processors. For instance, Adapteva's main product family is the Epiphany scalable multicore MIMD architecture. The Epiphany architecture could accommodate chips with up to 4096 RISC out-of-order microprocessors, all sharing a single 32-bit flat memory space. Each RISC processor in the Epiphany architecture is superscalar with 64 32-bit unified register file (integer or single precision) microprocessor operating up to 1 GHz and capable of 2 GFLOPS (single precision). The Epiphany multicore coprocessor is a scalable shared memory architecture, featuring up to 4096 processors on a single chip connected through a high-bandwidth on-chip network. Each Epiphany processor core includes a tiny high-performance floating point RISC processor for multicore processing, a high bandwidth local memory system, and an extensive set of built in hardware features for multicore communication. The peak energy efficiency of ExG4 ($x = 16, 64, 256$, 1k, 4k) can be as high as 70 GFLOPS per watt, higher than the expected 50 GFLOPS per watt in exascale computing.

When multiple manycore processors in a data center for cloud computing are shared by a large number parallel tasks simultaneously, we are facing the problem of allocating the cores to the tasks and scheduling the tasks, such that the system performance is optimized or the energy consumption is minimized. Furthermore, such core allocation and task scheduling should be conducted with energy constraints or performance constraints. The motivation of this paper is to study the problems of energy and time constrained scheduling of precedence constrained parallel tasks on multiple manycore processors in a cloud computing environment as optimization problems. As mentioned above, such investigation is of significant importance in taking advantage of the energy efficiency potential of cloud computing and manycore processors.

1.2. Related research

According to Moore's law, power consumption in computer systems has increased at an exponential speed for decades. Power density in high-performance computer systems will soon reach that of a nuclear reactor [7]. Such increased energy consumption causes severe economic, ecological, and technical problems [8–11]. Power conservation is critical in many computation and communication environments and has attracted extensive research activities. 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 [12–14]. There exists an explosive body of literature on power-aware computing and communication. For comprehensive surveys, the reader is referred to [15,16,7] for research on system-level power-aware design and power reduction techniques, [17–19] for research on green data centers, cloud computing systems, and distributed systems, and [20–22] for research on energy-efficient scheduling algorithms and scheduling techniques.

Power consumption in computing systems can be reduced by thermal-aware hardware and software design at various levels. Software techniques for power reduction are supported by a mechanism called *dynamic voltage scaling* (equivalently, dynamic frequency scaling, dynamic speed scaling, dynamic power scaling). 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 (see, e.g., [23]). This paper will employ the technique of dynamic voltage scaling for energy and time constrained scheduling of parallel tasks on multiple manycore processors.

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 [24]. Since the pioneering work in [25,26], power-aware task scheduling on processors with variable voltages and speeds has been extensively studied, including scheduling tasks with arrival times and deadlines on a uniprocessor computer with minimum energy consumption [27–36], scheduling independent or precedence constrained tasks on uniprocessor or multiprocessor computers in real-time applications [37–55], dealing with the energy-delay tradeoff [56–65], developing high-performance and energy-efficient computing systems [4,12–14], improving system level power management [66–70], and conducting other studies [71–73]. In [74,23,75–79], we addressed energy and time constrained power allocation and task scheduling on multiprocessors with dynamically variable voltage and frequency and speed and power as combinatorial optimization problems. However, all the above research only considered a single system. Energy-efficient scheduling of precedence constrained parallel tasks on multiple parallel computing systems has rarely been studied before, and this will be the main focus of this paper. (We have noticed that system-wide power management has been considered in [80–82].)

Much existing research assume that a task can be supplied with any power and a processor can be set at any speed, that is, clock frequency and supply voltage and execution speed and power supply can be changed continuously in any range. However, the currently available processors have only a few *discrete* clock frequency and supply voltage and execution speed and power levels [83,84]. Task scheduling on processors with discrete speed levels has been investigated by a number of researchers. For instances, it was shown that an optimal preemptive schedule with minimum energy consumption on a uniprocessor computer can

be found in polynomial time [32,34,35]. Processors with discrete speed levels were also considered in real-time multiprocessor systems [85,53]. Furthermore, energy and time constrained nonpreemptive task scheduling on multiprocessor computers with discrete speed levels has been studied analytically [86].

1.3. Our contributions

In this paper, we investigate the problems of energy and time constrained scheduling of precedence constrained parallel tasks on multiple manycore processors in a cloud computing environment. The main contributions of the paper are summarized as follows.

- Lower bounds for optimal solutions are generalized from a single parallel computing system to multiple parallel computing systems.
- Pre-power-determination algorithms and post-power-determination algorithms are devised for both energy and time constrained scheduling of precedence constrained parallel tasks on multiple manycore processors with continuous or discrete speed levels.
- The performance of these algorithms are evaluated analytically and experimentally. Our main strategy is to embed the equal-speed method into our algorithms, which not only makes our analysis possible, but also yields good performance of our algorithms.

Our approach in this paper is to design and analyze the performance of heuristic algorithms that employ the equal-speed method, which means that tasks scheduled on the same manycore processor are assigned the same power and executed with the same speed (for processors with continuous speed levels) or roughly the same speed (for processors with discrete speed levels). Of course, tasks scheduled on different manycore processors may still be executed with different speeds. We will provide justification of the equal-speed method in Section 2.5.

Our investigation is significant in a number of ways.

- First, we extend the equal-speed method from one parallel computing system to multiple parallel computing systems.
- Second, the equal-speed method can be incorporated into both pre-power-determination algorithms and post-power-determination algorithms.
- Third, the equal-speed method can be used for both energy constrained scheduling and time constrained scheduling.
- Fourth, such extension has been implemented and applied successfully to processors with continuous speed levels or discrete speed levels.

The equal-speed method not only makes our analysis possible, but also yields good performance of our algorithms. Notice that our algorithms and their analysis are also applicable to

- multiple multiprocessor and multicomputer systems for parallel computing, supercomputing, and high-performance computing;
- multiple clusters of homogeneous nodes in a distributed computing environment;
- multiple computing systems with identical cores in a grid computing environment;

in addition to multiple manycore processors in a cloud computing environment. To the best of the author's knowledge, power-aware scheduling of precedence constrained parallel tasks on multiple parallel and distributed computing systems has rarely been studied before.

The rest of the paper is organized as follows. In Section 2, we provide necessary background information, including the task model, the processor model, and the power model used in this paper, definitions of our optimization problems and performance measures, and lower bounds for performance analysis and evaluation of our heuristic algorithms. In Section 3, we develop and analyze pre-power-determination algorithms and post-power-determination algorithms for energy and time constrained scheduling of precedence constrained parallel tasks on multiple manycore processors with continuous speed levels. In Section 4, we develop and evaluate pre-power-determination algorithms and post-power-determination algorithms for energy and time constrained scheduling of precedence constrained parallel tasks on multiple manycore processors with discrete speed levels. In Section 5, we give a summary and conclude the paper.

Table 0
Symbols and definitions.

Symbol	Definition
n	The number of parallel tasks
n_j	The number of parallel tasks scheduled on M_j
π_i	The number of cores requested by task i
r_i	The maximum execution requirement on the cores executing task i
w_i	The work of task i
p_i	The power supplied to execute task i
s_i	The execution speed of task i
t_i	The execution time of task i
e_i	The energy consumed to execute task i
m	The number of manycore processors
M_j	The name as well as the number of cores in the jth manycore processor
α	Exponent of power consumption
S_k	Discrete level of core speed
ϕ_k	S_{k+1}/S_k
E	Energy constraint
E_j	Energy allocated to M_j
E_k	$WS_k^{\alpha-1}$
$E_{j,k}$	$W_jS_k^{\alpha-1}$
T	Time constraint
T_j	$A_j(t_{j,1}, t_{j,2}, \ldots, t_{j,n_j})$
T_k	$A(r_1, r_2, \ldots, r_n)/S_k$
$T_{j,k}$	$A_j(r_{j,1}, r_{j,2}, \ldots, r_{j,n_j})/S_k$
A	An algorithm
A_j	$A_j(r_{j,1}, r_{j,2}, \ldots, r_{j,n_j})$
T_A	The length of a schedule produced by algorithm A
T_{OPT}	The shortest length of an optimal schedule
E_A	The amount of energy consumed by algorithm A
E_{OPT}	The minimum amount of energy consumed by an optimal schedule
β_A	The performance ratio of an algorithm A for energy constrained scheduling
γ_A	The performance ratio of an algorithm A for time constrained scheduling
W	The total amount of work to be performed for the n parallel tasks
W_j	The total amount of work to be performed for the n_j parallel tasks scheduled on M_j
A_1–A_2	A pre-power-determination algorithm (A_1: power allocation, A_2: task scheduling)
A_1–A_2	A post-power-determination algorithm (A_1: task scheduling, A_2: power allocation)

2. Preliminaries

2.1. The models

We first describe the task model, the processor model, and the power model used in this paper. Table 0 provides a list of symbols and their definitions used in this paper.

2.1.1. The task model

Assume that we are given a set of n parallel tasks. A parallel task i, where $1 \leq i \leq n$, is specified by π_i and r_i explained below. The integer π_i is the number of cores requested by task i, i.e., the *size* of task i. It is possible that in executing task i, the π_i cores may have different execution requirements (i.e., the numbers of core cycles or the numbers of instructions executed on the cores)

due to imbalanced load distribution. Let r_i represent the maximum execution requirement on the π_i cores executing task i. The product $w_i = \pi_i r_i$ is called the *work* of task i.

The n parallel tasks can be either independent or precedence constrained. A set of precedence constrained parallel tasks can be specified as a partial order $\prec$ over the set of tasks $\{1, 2, \ldots, n\}$, or a task graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$ is the set of tasks and E is a set of arcs representing the precedence constraints. The relationship $i \prec j$, or an arc (i, j) from i to j, means that task i must be executed before task j, i.e., task j cannot be executed until task i is completed. We call j a successor of i and i a predecessor of j.

2.1.2. The processor model

We are also given m manycore processors with $M_1, M_2, \ldots, M_m$ cores respectively. All the $M = M_1 + M_2 + \cdots + M_m$ cores in the m manycore processors are identical. We use M_j to represent the name as well as the number of cores in the jth manycore processor, where $1 \leq j \leq m$.

To execute a task i, any π_i of the M_j cores of the jth manycore processor can be allocated to task i. However, core allocation cannot be performed across different manycore processors, i.e., all the π_i cores allocated to task i must reside in the same manycore processor. Several tasks can be executed simultaneously on M_j, with the restriction that the total number of active cores (i.e., cores allocated to tasks being executed) at any moment cannot exceed M_j, for all $1 \leq j \leq m$.

Notice that inter-processor communications take considerably more time than intra-processor communications. If a task is executed by several manycore processors, the execution time is no longer its execution requirement divided by the execution speed. The specification of the increased execution time is not clear. The increased execution time depends on many factors, such as (1) the communication network bandwidth; (2) the type of algorithm; (3) the number of manycore processors involved; (4) the number of cores allocated on each manycore processor. These hardware and software and management dependent factors make the specification of parallel task execution times very complicated when cores are allocated across several manycore processors, and eventually make energy and time constrained scheduling analytically intractable. Further investigation along this direction is beyond the scope of this paper and should be considered as a future research direction.

2.1.3. The power model

Power dissipation and circuit delay in digital CMOS (complementary metal–oxide-semiconductor) 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 [87]. Since $s \propto f$, where s is the processor speed, and $f \propto V^\gamma$ with $0 < \gamma \leq 1$ [88], which implies that $V \propto f^{1/\gamma}$, we know that power consumption is $p \propto f^\alpha$ and $p \propto s^\alpha$, where $\alpha = 1 + 2/\gamma \geq 3$.

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 cores allocated to task i have the same speed s_i for duration t_i, although some of the π_i cores 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 will consider processors with continuous or discrete clock frequency and supply voltage and execution speed and power levels.

- *Continuous speed levels*—In this model, it is assume that a task can be supplied with any power and a core can be set at any execution speed, that is, power and speed can be changed continuously and unboundedly.
- *Discrete speed levels*—In this model, it is assume that there are only d discrete levels of power supply $P_1, P_2, \ldots, P_d$, and d discrete levels of core speed $S_1, S_2, \ldots, S_d$, that is, power and speed can be set with only a few options.

Throughout the paper, we assume that $S_1 < S_2 < \cdots < S_d$, where $d \geq 2$. We also define $\phi_k = S_{k+1}/S_k$, for all $1 \leq k \leq d-1$. The quantity ϕ_k measures the relative gap between two successive speed levels S_k and S_{k+1}. Although d is a finite number, we assume that the number d of speed levels is large enough to accommodate the needs of our algorithms.

2.2. The problems

The power-aware scheduling problems considered in this paper are formally defined as follows.

Problem 1 (*Energy Constrained Scheduling*)**.**
Input: A set of n precedence constrained parallel tasks with task sizes $\pi_1, \pi_2, \ldots, \pi_n$ and task execution requirements $r_1, r_2, \ldots, r_n$, m manycore processors with $M_1, M_2, \ldots, M_m$ identical cores, and energy constraint E.
Output: Power supplies $p_1, p_2, \ldots, p_n$ to the n tasks and a nonpreemptive schedule of the n parallel tasks on the m manycore processors, such that the schedule length is minimized and the total energy consumed does not exceed E.

Problem 2 (*Time Constrained Scheduling*)**.**
Input: A set of n precedence constrained parallel tasks with task sizes $\pi_1, \pi_2, \ldots, \pi_n$ and task execution requirements $r_1, r_2, \ldots, r_n$, m manycore processors with $M_1, M_2, \ldots, M_m$ identical cores, and time constraint T.
Output: Power supplies $p_1, p_2, \ldots, p_n$ to the n tasks and a nonpreemptive schedule of the n parallel tasks on the m manycore processors, such that the total energy consumed is minimized and the schedule length does not exceed T.

Notice that allocating power to a task is equivalent to determining the execution speed of the task.

2.3. Performance measures

Let T_A denote the length of a schedule produced by algorithm A, and T_{OPT} denote the shortest length of an optimal schedule. Similarly, let E_A denote the 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 energy and time constrained scheduling algorithms.

Definition 1. The *performance ratio* of an algorithm A that solves the energy constrained scheduling problem 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 2. The *performance ratio* of an algorithm A that solves the time constrained scheduling problem 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 and/or random precedence constraints, 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 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 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.

2.4. Lower bounds

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.

2.4.1. Energy constrained scheduling

For the energy constrained scheduling problem, we have the following proposition which gives a lower bound for the optimal schedule length T_{OPT}.

Proposition 1. *For the energy constrained scheduling problem, 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.

Proof. Let us consider one manycore processor with M cores. Furthermore, the processor is divided into m subsystems with $M_1, M_2, \ldots, M_m$ cores respectively. It is clear that any schedule on the m manycore processors $M_1, M_2, \ldots, M_m$ can be implemented on the single M-core processor, with each subsystem of M_j cores simulating an M_j-core processor. However, the reverse is not true, i.e., a schedule on the M-core processor may not be implemented on the m manycore processors, since cores in the M-core processor can be allocated across the boundaries of the m subsystems, which is not allowed for the m manycore processors. In other words, a single M-core processor is more flexible in core allocation than m manycore processors. This implies that the length of an optimal schedule on the m manycore processors is no shorter than the length of an optimal schedule on the M-core processor. From [77], it is already known that the length of an optimal schedule on the M-core processor is at least

$$\left(\frac{M}{E} \left(\frac{W}{M} \right)^{\alpha} \right)^{1/(\alpha-1)}.$$

Hence, the length of an optimal schedule on the m manycore processors is also at least the above bound. ■

Notice that the above lower bound is applicable to manycore processors with continuous speed levels, and of course, also to discrete speed levels.

2.4.2. Time constrained scheduling

For the time constrained scheduling problem, we have the following result which gives a lower bound for the minimum energy consumption E_{OPT}.

Proposition 2. *For the time constrained scheduling problem, 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.

Proof. The proof follows the same reasoning as that in the proof of Proposition 1. The total energy consumption of an optimal schedule on the m manycore processors is no less than the total energy consumption of an optimal schedule on the M-core processor. From [77], it is already known that the total energy consumption of an optimal schedule on the M-core processor is at least

$$M \left(\frac{W}{M} \right)^{\alpha} \frac{1}{T^{\alpha-1}}.$$

Hence, the total energy consumption of an optimal schedule on the m manycore processors is also at least the above bound. ■

Again, the above lower bound is applicable to manycore processors with continuous or discrete speed levels.

2.5. Overview of our method

There are three basic strategies to power allocation (i.e., execution speed determination) and task scheduling.

- Post-power-determination algorithms—First, a schedule of the tasks is produced based on the task execution requirements. Then, the execution speeds of the tasks are determined [78].
- Hybrid algorithms—The task execution speeds are determined together with task scheduling, i.e., power allocation and task scheduling are interleaved [79].
- Pre-power-determination algorithms—First, the execution speed of each task is determined. Then, a schedule of the tasks is generated based on the known task execution times [89].

It has been shown that the class of pre-power-determination algorithms have the best performance in power-aware scheduling of precedence constrained parallel tasks.

Generally speaking, even for a set of independent parallel tasks on one parallel computing system, after a schedule of the tasks is produced based on the task execution requirements, it is hard to set the task execution speeds, except the case where all tasks are executed with the same speed. The reason is that if tasks are executed with different speeds, processor allocation and task scheduling need to be changed. To make post-power-determination algorithms and hybrid algorithms possible, special processor allocation methods (e.g., the harmonic processor allocation scheme) and task scheduling algorithms (e.g., the level-by-level scheduling method) need to be developed [78, 79]. However, these processor allocation methods and task scheduling algorithms introduce inefficiency in dealing with precedence constrained parallel tasks. This is the primary reason that pre-power-determination algorithms adopting the equal-speed method perform better than post-power-determination algorithms and hybrid algorithms [89].

Our approach in this paper is to extend the equal-speed method from one parallel computing system to multiple parallel computing systems. The equal-speed method can be applied to both pre-power-determination algorithms and post-power-determination algorithms and to both energy constrained scheduling and time constrained scheduling. Furthermore, such extension has been successful for processors with continuous speed levels or discrete speed levels.

3. Continuous speed levels

3.1. Pre-power-determination algorithms

A pre-power-determination algorithm can be represented as A_1–A_2, where A_1 is a power allocation algorithm, and A_2 is a task scheduling algorithm. The power allocation algorithm in this section is *equal speed* (ES), i.e., all tasks are allocated the same power and executed with the same speed. Once the task execution speeds and times are available, any efficient task scheduling algorithm A_2 can be used to produce a schedule. The pre-power-determination algorithm in this section is called ES-A, where A is any efficient task scheduling algorithm.

For n precedence constrained parallel tasks with execution times $t_1, t_2, \ldots, t_n$, we use the notation $A(t_1, t_2, \ldots, t_n)$ to denote the length of the schedule produced by algorithm A on m manycore processors $M_1, M_2, \ldots, M_m$. It is clear that if all tasks are executed with the same speed s, we have $A(t_1, t_2, \ldots, t_n) = A(r_1, r_2, \ldots, r_n)/s$, for all algorithm A.

3.1.1. Energy constrained scheduling

The following theorem gives a performance bound of the pre-power-determination algorithm ES-*A* in solving the energy constrained scheduling problem.

Theorem 1. *By using a pre-power-determination algorithm ES-A to solve the energy constrained scheduling problem, the performance ratio is*

$$\beta_{\text{ES-}A} \leq \frac{A(r_1, r_2, \ldots, r_n)}{W/M}. \tag{1}$$

Proof. By using pre-power-determination algorithm ES-*A* to solve the energy constrained scheduling problem, we have $s_1 = s_2 = \cdots = s_n = s$, where s is the same execution speed set for all tasks. The energy consumed by task i is

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

for all $1 \leq i \leq n$. Since the total energy consumed is constrained by

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

we obtain the identical task execution speed, which is

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

and the execution time of task i, which is

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

for all $1 \leq i \leq n$. Therefore, the length of the schedule produced by algorithm ES-*A* is

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

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

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

The theorem is thus proved. ■

3.1.2. Time constrained scheduling

The following theorem gives a performance bound of the pre-power-determination algorithm ES-*A* in solving the time constrained scheduling problem.

Theorem 2. *By using a pre-power-determination algorithm ES-A to solve the time constrained scheduling problem, the performance ratio is*

$$\gamma_{\text{ES-}A} \leq \left(\frac{A(r_1, r_2, \ldots, r_n)}{W/M}\right)^{\alpha-1}. \tag{2}$$

Proof. By using a pre-power-determination algorithm ES-*A* to solve the time constrained scheduling problem, we need to provide enough energy $E_{\text{ES-}A}$, so that the deadline T is met, i.e.,

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

The above equation implies that the total energy $E_{\text{ES-}A}$ consumed by algorithm ES-*A* is

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

The execution speed is

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

which is obvious, since we need $A(r_1, r_2, \ldots, r_n)/s = T$. By Proposition 2, the performance ratio of algorithm ES-*A* is

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

This proves the theorem. ■

3.1.3. Simulation results

We now show the expectation of the performance bounds in (1)–(2) through simulation. (Note: All simulation results in this paper are obtained by a simulation program written in the C++ programming language and running in a Linux environment. All parameter settings will be described in detail.)

The task scheduling algorithm *A* is the well known *list scheduling* (LS) algorithm, originally proposed in [90] for scheduling sequential tasks that demand for only one processor, i.e., $\pi_i = 1$, for all $1 \leq i \leq N$. A list schedule is based on an initial ordering of the tasks $L = (i_1, i_2, \ldots, i_n)$, called a priority list. Initially, at time zero, the scheduler instantaneously scans list L from the beginning, searching for tasks that are ready to be executed, i.e., which have no predecessors under $\prec$ still waiting in L. The first ready task in L is removed from L and sent to an idle core for processing. Such a search is repeated until there is no ready task or there is no more core available. In general, whenever a core completes a task, the scheduler immediately scans L, looking for the first ready task to be executed. If such a ready task is not found, the core becomes idle and waits for the next finished task. As running tasks complete, more precedence constraints are removed and more tasks will be ready.

The extension of the LS algorithm to scheduling parallel tasks is straightforward [91]. When the scheduler finds a ready task i, the scheduler checks whether there are at least π_i idle cores. If so, task i is allocated π_i cores and executed nonpreemptively on these cores. Otherwise, the ready task i still needs to wait in L until other running tasks complete. Therefore, initially and later whenever a task is completed, each of the remaining tasks not scheduled yet is examined to see whether it can be scheduled for execution, i.e., whether all its predecessors are completed and there are enough cores for the task. The above algorithm can be further extended to multiple manycore processors initially and later whenever a task is completed, each of the remaining tasks not scheduled yet is examined to see whether it can be scheduled for execution on one of the m manycore processors, i.e., every manycore processor will be examined.

It is clear that a task graph is a directed acyclic graph (dag). A dag can be divided into levels, such that tasks in the same level are independent of each other. Nodes without ant incoming arcs constitute level 1. In general, a node i is in level j if the longest path from a node in level 1 to node i has j nodes. Let l be the number of levels in a dag and n_j denote the number of tasks in level j, where $1 \leq j \leq l$. An arc (i, j) is redundant if there exists a path from i to j which goes through other nodes. Assume that there is no redundant arc. Then, arcs only connect nodes in adjacent levels. We consider six types of task graphs.

- *Independent tasks* (IT(n))—A dag representing n independent tasks, i.e., a dag which has no arc and one level. We set $n = 500$ in our simulations.
- *Complete trees* (CT(b, h))—A dag representing a search tree with branching factor b and height h. The number of levels is $l = h + 1$. The number of nodes in level j is b^j, where $0 \le j \le h$. The total number of nodes is $n = b^0 + b^1 + b^2 + \cdots + b^h = (b^{h+1} - 1)/(b - 1)$. The precedence constraints are $i \prec bi - b + 2, i \prec bi - b + 3, \ldots, i \prec bi + 1$, for all $1 \le i \le (b^h - 1)/(b - 1)$ (i.e., nodes in levels 0 to $h - 1$). We set $b = 2$ and $h = 10$ in our simulations.
- *Partitioning algorithms* (PA(b, h))—A dag representing a divide-and-conquer algorithm with b subproblems and depth of recursion h. The number of levels is $l = 2h + 1$. The number of nodes in level j is b^j for $0 \le j \le h$, and b^{2h-j} for $h + 1 \le j \le 2h$. The total number of nodes is $n = 2(b^0 + b^1 + b^2 + \cdots + b^{h-1}) + b^h = (b^{h+1} + b^h - 2)/(b - 1)$. The precedence constraints are $i \prec bi - b + 2, i \prec bi - b + 3, \ldots, i \prec bi + 1$, for all $1 \le i \le (b^h - 1)/(b - 1)$ (i.e., nodes in levels 0 to $h - 1$), and $i \prec n - (n - 1 - i)/b$ for all $(b^h - 1)/(b - 1) + 1 \le i \le (b^{h+1} + b^h - 2)/(b - 1) - 1$ (i.e., nodes in levels h to $2h - 1$). We set $b = 2$ and $h = 10$ in our simulations.
- *Linear algebra task graphs* (LA(v))—A dag representing a linear algebra algorithm with $l = v$ levels. The number of nodes in level j is $n_j = v - j + 1$, where $1 \le j \le v$. The total number of nodes is $n = v(v + 1)/2$. Let $a_j = n_1 + n_2 + \cdots + n_{j-1} + 1$ be the leading index of the nodes in level j, i.e., nodes in level j are $a_j, a_j + 1, \ldots, a_j + n_j - 1$, where $1 \le j \le v$. The precedence constraints are $a_j \prec a_{j+1} + k$, for all $1 \le j \le v - 1$ and $0 \le k \le n_j - 1$, and $a_j + k \prec a_{j+1} + k - 1$, for all $1 \le k \le n_j - 1$. We set $v = 200$ in our simulations.
- *Diamond dags* (DD(d))—A dag with $n = d^2$ nodes. The number of levels is $l = 2d - 1$. The number of nodes in level j is $n_j = j$ for $1 \le j \le d$, and $n_j = d - j + 1$ for $d + 1 \le j \le 2d - 1$. Let $a_j = n_1 + n_2 + \cdots + n_{j-1} + 1$ be the leading index of the nodes in level j, i.e., nodes in level j are $a_j, a_j + 1, \ldots, a_j + n_j - 1$, where $1 \le j \le v$. The precedence constraints are $a_j + k \prec a_{j+1} + k$ and $a_j + k \prec a_{j+1} + k + 1$, for all $1 \le j \le d - 1$ and $0 \le k \le n_j - 1$, $a_j \prec a_{j+1}$ and $a_j + n_j - 1 \prec a_{j+1} + n_{j+1} - 1$, for all $d \le j \le 2d - 2$, $a_j + k \prec a_{j+1} + k - 1$ and $a_j + k \prec a_{j+1} + k$, for all $d \le j \le 2d - 2$ and $1 \le k \le n_j - 2$. We set $d = 200$ in our simulations.
- *Random dags* (DAG(n, p))—A random dag with n nodes and arc probability p. For each pair of i and j, where $1 \le i < j \le n$, there is an arc (i, j) with probability p, which is independent of the probability of other arcs. The expected number of successors of task i is $(n - i)p$, where $1 \le i \le n$. If $p = 2/n$, then it is in the range $[0, 2)$. We set $n = 2000$ and $p = 0.001$ in our simulations.

The task sizes $\pi_1, \pi_2, \ldots, \pi_n$ are i.i.d. discrete random variables in $[1..D]$. We consider three types of probability distributions of task sizes with about the same expected task size $\bar{\pi}$ [23,77–79]. Let ξ_b be the probability that $\pi_i = b$, where $b \ge 1$.

- Uniform distributions in the range $[1..u]$, i.e., $\xi_b = 1/u$ for all $1 \le b \le u$, where u is chosen such that $(u + 1)/2 = \bar{\pi}$, i.e., $u = 2\bar{\pi} - 1$.
- Binomial distributions in the range $[1..D]$, where $D = \min\{M_1, M_2, \ldots, M_m\}$, i.e.,

$$\xi_b = \frac{1}{1 - (1 - p)^D} \binom{D}{b} p^b (1 - p)^{D-b},$$

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

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

which is slightly greater than $\bar{\pi}$, especially when $\bar{\pi}$ is small.

Table 1A
Simulation data for the performance bound (1) (IT(500), CI = ±0.54386%).

Average task size	Uniform distribution	Binomial distribution	Geometric distribution
10	1.2245384	1.2233637	1.2157317
20	1.1170146	1.1146973	1.1302962
30	1.0926097	1.0835406	1.1096424
40	1.0865066	1.0721085	1.0989122
50	1.0940931	1.0859161	1.0977620
60	1.0957334	1.0936939	1.0939319

Table 1B
Simulation data for the performance bound (2) (IT(500), CI = ±1.00848%).

Average task size	Uniform distribution	Binomial distribution	Geometric distribution
10	1.5010664	1.4974894	1.4743129
20	1.2463026	1.2419159	1.2756722
30	1.1965773	1.1750458	1.2284184
40	1.1785307	1.1477416	1.2110139
50	1.1973015	1.1794252	1.2055510
60	1.1990831	1.1956248	1.1955979

- Geometric distributions in the range $[1..D]$, where $D = \min\{M_1, M_2, \ldots, M_m\}$, i.e.,

$$\xi_b = \frac{q(1 - q)^{b-1}}{1 - (1 - q)^D},$$

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

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

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

The task execution requirements $r_1, r_2, \ldots, r_n$ are treated as i.i.d. continuous random variables uniformly distributed in $[0, 1)$.

We consider a cloud computing environment with $m = 5$ manycore processors, where the numbers of cores are $(M_1, M_2, M_3, M_4, M_5) = (128, 128, 196, 256, 256)$.

The parameter α is set as 3.

Our simulation data for the performance bounds of the pre-power-determination algorithm ES-LS are given in Tables 1–6, for the six types of task graphs respectively.

In Table 1A, we show simulation data for the performance bound of $\beta_{\text{ES-LS}}$ of IT(500) in (1) for $\bar{\pi} = 10, 20, 30, 40, 50, 60$ and the three types of task size distributions. For each combination of average task size and task size distribution, we generate certain number *rep* of sets of parallel tasks with random task sizes and execution requirements, apply the ES-LS algorithm, obtain the schedule length, and record the performance bound. The average value of the recorded performance bounds is reported as the expectation of the performance bound in (1), i.e., the average-case performance bound for energy-constrained scheduling. The number *rep* is chosen such that the 99% confidence interval (CI) is within a reasonable range.

In Table 1B, we show simulation data for the performance bound of $\gamma_{\text{ES-LS}}$ of IT(500) in (2) for $\bar{\pi} = 10, 20, 30, 40, 50, 60$ and the three types of task size distributions. For each combination of average task size and task size distribution, we generate certain number *rep* of sets of parallel tasks with random task sizes and execution requirements, apply the ES-LS algorithm, obtain the total energy consumption, and record the performance bound. The average value of the recorded performance bounds is reported as the expectation of the performance bound in (2), i.e., the average-case performance bound for time-constrained scheduling.

In Tables 2–6, we repeat the same work for other task graphs, i.e., CT(2,10), PA(2,10), LA(200), DD(200), and DAG(2000,0.001).

Table 2A
Simulation data for the performance bound (1) (CT(2,10), CI = ±0.90249%).

Average task size	Uniform distribution	Binomial distribution	Geometric distribution
10	1.2935307	1.3142450	1.3744934
20	1.1340361	1.1307032	1.1644624
30	1.0902498	1.0835326	1.1105078
40	1.0707470	1.0687952	1.0976130
50	1.0678596	1.0772388	1.0883946
60	1.0687400	1.0896199	1.0843028

Table 2B
Simulation data for the performance bound (2) (CT(2,10), CI = ±1.98097%).

Average task size	Uniform distribution	Binomial distribution	Geometric distribution
10	1.6705812	1.7356443	1.8822022
20	1.2810853	1.2806652	1.3481393
30	1.1846435	1.1770298	1.2386948
40	1.1501013	1.1365295	1.2044014
50	1.1379861	1.1595916	1.1861223
60	1.1429223	1.1885100	1.1769446

Table 3A
Simulation data for the performance bound (1) (PA(2,10), CI = ±0.67899%).

Average task size	Uniform distribution	Binomial distribution	Geometric distribution
10	1.4376047	1.4673924	1.5293757
20	1.1861939	1.1913076	1.2417406
30	1.1191124	1.1156335	1.1534792
40	1.0895668	1.0870271	1.1218429
50	1.0770378	1.0925717	1.1087258
60	1.0741365	1.1003475	1.1007152

Table 3B
Simulation data for the performance bound (2) (PA(2,10), CI = ±1.47089%).

Average task size	Uniform distribution	Binomial distribution	Geometric distribution
10	2.0610655	2.1835053	2.3687051
20	1.4136206	1.4257727	1.5356184
30	1.2505442	1.2452238	1.3296139
40	1.1857568	1.1840140	1.2624047
50	1.1600027	1.1920766	1.2270179
60	1.1511566	1.2114935	1.2078904

Table 4A
Simulation data for the performance bound (1) (LA(200), CI = ±1.54235%).

Average task size	Uniform distribution	Binomial distribution	Geometric distribution
10	1.4734815	1.4866587	1.5330269
20	1.1279051	1.1285387	1.1886043
30	1.0722037	1.0680042	1.1152157
40	1.0566575	1.0535742	1.0898886
50	1.0570322	1.0718280	1.0789907
60	1.0607582	1.0756554	1.0744501

Table 4B
Simulation data for the performance bound (2) (LA(200), CI = ±2.52869%).

Average task size	Uniform distribution	Binomial distribution	Geometric distribution
10	2.1903773	2.1833820	2.3235177
20	1.2741338	1.2678296	1.4177314
30	1.1478139	1.1395352	1.2422587
40	1.1172385	1.1079037	1.1842388
50	1.1169127	1.1487781	1.1638758
60	1.1265809	1.1585084	1.1536938

Notice that in Tables 6A–6B, for each combination of average task size and task size distribution, we generate certain number *rep* of sets of parallel tasks with random task sizes and execution requirements, and for each set of parallel tasks, we also generate a random dag.

Table 5A
Simulation data for the performance bound (1) (DD(200), CI = ±0.91345%).

Average task size	Uniform distribution	Binomial distribution	Geometric distribution
10	1.4631074	1.4649221	1.4958565
20	1.1238616	1.1210457	1.1704593
30	1.0704894	1.0665019	1.1108245
40	1.0567077	1.0525773	1.0892356
50	1.0568041	1.0716766	1.0807973
60	1.0613479	1.0756088	1.0752173

Table 5B
Simulation data for the performance bound (2) (DD(200), CI = ±1.85918%).

Average task size	Uniform distribution	Binomial distribution	Geometric distribution
10	2.1495022	2.1576704	2.2434231
20	1.2676168	1.2584933	1.3704508
30	1.1469711	1.1372444	1.2336513
40	1.1165582	1.1083230	1.1875610
50	1.1155978	1.1485951	1.1664790
60	1.1255280	1.1567379	1.1550887

Table 6A
Simulation data for the performance bound (1) (DAG(2000,0.001), CI = ±0.88592%).

Average task size	Uniform distribution	Binomial distribution	Geometric distribution
10	1.0958950	1.1076803	1.1839701
20	1.0437175	1.0423292	1.0838451
30	1.0366603	1.0305382	1.0529580
40	1.0363596	1.0310143	1.0480047
50	1.0431013	1.0522974	1.0469290
60	1.0511509	1.0698291	1.0466739

Table 6B
Simulation data for the performance bound (2) (DAG(2000,0.001), CI = ±1.82052%).

Average task size	Uniform distribution	Binomial distribution	Geometric distribution
10	1.1982456	1.2090995	1.4015087
20	1.0892417	1.0874801	1.1696480
30	1.0735307	1.0616709	1.1057308
40	1.0751805	1.0624417	1.1000261
50	1.0865988	1.1069430	1.0965371
60	1.1047828	1.1422357	1.0964218

We observe that the average-case performance bounds for both energy-constrained and time-constrained scheduling is determined by the number of processors, the numbers of cores on the processors, the type of task graph, the number of tasks, the average task size, the distribution of task sizes, the average execution requirement, the distribution of execution requirements, and quality of the scheduling algorithm *A*. Since M is as large as $M_1 + M_2 + M_3 + M_4 + M_5 = 960$, the number of tasks that can be executed simultaneously is roughly $M/\bar{\pi}$, which is 96, 48, 32, 24, 19, 16, for $\bar{\pi} = 10, 20, 30, 40, 50, 60$, respectively. There must be enough work to be done. A good indication is the average task size and the number of tasks, which is 500 for IT(500), 2047 for CT(2,10), 3070 for PA(2,10), 20 100 for LA(200), 40 000 for DD(200), and 2000 for DAG(2000,0.001). There must also be sufficient independent tasks which can be executed simultaneously. A good indication is the *width* of a task graph, i.e., $\max\{n_1, n_2, \ldots, n_l\}$, the largest number of tasks in a level. The width of the six task graphs is 500 for IT(500), 1024 for CT(2,10), 1024 for PA(2,10), 200 for LA(200), 200 for DD(200), and 864 for DAG(2000,0.001).

All our simulation data demonstrate that the pre-power-determination algorithm ES-LS exhibit very good performance in energy-constrained and time-constrained scheduling of precedence constrained parallel tasks on multiple manycore processors. This is based on the observation that the performance bounds are very close to one, which means that the performance of our algorithm is very close to that of the optimal algorithm. It is clear that increasing the number of tasks and especially the number of independent tasks will further reduce the average-case performance bounds, because the quality of the LS algorithm will further improve. For instance, CT(2,10) exhibits better performance than PA(2,10) due to relatively more independent tasks. Furthermore, for some probability distributions of task sizes (e.g., the geometric distribution), increasing the average task size reduces the average-case performance bound.

3.2. Post-power-determination algorithms

A post-power-determination algorithm can also be represented as A_1–A_2, where A_1 is a task scheduling algorithm, and A_2 is a power allocation algorithm. The power allocation algorithm in this section is also ES, i.e. all tasks scheduled on the same manycore processor are allocated the same power and executed with the same speed. However, tasks on different manycore processors may be executed with different speeds. It is clear that for independent parallel tasks, such a power allocation method is feasible. Since power allocation can be done in such a way that the overall schedule length or energy consumption is minimized, there might be performance improvement. The post-power-determination algorithm in this section is called A-ES, where A is any efficient task scheduling algorithm.

Assume that $r_{j,1}, r_{j,2}, \ldots, r_{j,n_j}$ are the execution requirements of the n_j tasks scheduled on M_j, where $1 \le j \le m$. Let $T_j = A_j(t_{j,1}, t_{j,2}, \ldots, t_{j,n_j})$ denote the length of the schedule produced by algorithm A for scheduling the n_j independent parallel tasks with execution times $t_{j,1}, t_{j,2}, \ldots, t_{j,n_j}$ on M_j. It is clear that if all tasks are executed with the same speed s_j on M_j, we have

$$\begin{aligned} T_j &= A_j(t_{j,1}, t_{j,2}, \ldots, t_{j,n_j}) = A_j(r_{j,1}, r_{j,2}, \ldots, r_{j,n_j})/s_j \\ &= A_j/s_j, \end{aligned}$$

where $A_j = A_j(r_{j,1}, r_{j,2}, \ldots, r_{j,n_j})$, for all $1 \le j \le m$, and all algorithm A.

Let $W_j = \pi_{j,1}r_{j,1} + \pi_{j,2}r_{j,2} + \cdots + \pi_{j,n_j}r_{j,n_j}$ be the total amount of work to be performed for the n_j parallel tasks scheduled on M_j. Assume that E is divided into $E_1, E_2, \ldots, E_m$, where E_j is allocated to M_j, and $E = E_1 + E_2 + \cdots + E_m$. Since the execution speed of M_j is

$$s_j = \left(\frac{E_j}{W_j}\right)^{1/(\alpha-1)},$$

we have

$$T_j = A_j \left(\frac{W_j}{E_j}\right)^{1/(\alpha-1)},$$

for all $1 \le j \le m$.

3.2.1. Energy constrained scheduling

The following theorem gives a performance bound of the post-power-determination algorithm A-ES in solving the energy constrained scheduling problem.

Theorem 3. *By using a post-power-determination algorithm A-ES to solve the energy constrained scheduling problem, the optimal energy allocation is*

$$E_j = \left(\frac{A_j^{\alpha-1}W_j}{A_1^{\alpha-1}W_1 + A_2^{\alpha-1}W_2 + \cdots + A_m^{\alpha-1}W_m}\right)E,$$

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

$$T_{A\text{-ES}} = \left(\frac{1}{E}\sum_{j=1}^{m} A_j^{\alpha-1}W_j\right)^{1/(\alpha-1)}.$$

The performance ratio is

$$\beta_{A\text{-ES}} \le \left(\frac{A_1^{\alpha-1}W_1 + A_2^{\alpha-1}W_2 + \cdots + A_m^{\alpha-1}W_m}{W}\right)^{1/(\alpha-1)} \frac{1}{W/M}. \tag{3}$$

Proof. It is clear that E should be divided and allocated to the m manycore processors in such a way that all processors complete their work at the same time, i.e., $T_1 = T_2 = \cdots = T_m = T$. In other words, we need

$$\begin{aligned} A_1\left(\frac{W_1}{E_1}\right)^{1/(\alpha-1)} &= A_2\left(\frac{W_2}{E_2}\right)^{1/(\alpha-1)} = \cdots \\ &= A_m\left(\frac{W_m}{E_m}\right)^{1/(\alpha-1)} = T. \end{aligned}$$

Since

$$A_j\left(\frac{W_j}{E_j}\right)^{1/(\alpha-1)} = T,$$

we have

$$E_j = \frac{A_j^{\alpha-1}W_j}{T^{\alpha-1}},$$

for all $1 \le j \le m$. By the condition that

$$E = \sum_{j=1}^{m} E_j = \sum_{j=1}^{m} \frac{A_j^{\alpha-1}W_j}{T^{\alpha-1}} = \frac{1}{T^{\alpha-1}}\sum_{j=1}^{m} A_j^{\alpha-1}W_j,$$

we get the schedule length of algorithm A-ES, which is

$$T_{A\text{-ES}} = T = \left(\frac{1}{E}\sum_{j=1}^{m} A_j^{\alpha-1}W_j\right)^{1/(\alpha-1)}.$$

Furthermore, we obtain the optimal energy allocation, i.e.,

$$E_j = \left(\frac{A_j^{\alpha-1}W_j}{A_1^{\alpha-1}W_1 + A_2^{\alpha-1}W_2 + \cdots + A_m^{\alpha-1}W_m}\right)E,$$

for all $1 \le j \le m$. By Proposition 1, the performance ratio of algorithm A-ES is

$$\begin{aligned} \beta_{A\text{-ES}} &= \frac{T_{A\text{-ES}}}{T_{\text{OPT}}} \\ &\le \frac{((1/E)(A_1^{\alpha-1}W_1 + A_2^{\alpha-1}W_2 + \cdots + A_m^{\alpha-1}W_m))^{1/(\alpha-1)}}{((M/E)(W/M)^{\alpha})^{1/(\alpha-1)}} \\ &= \left(\frac{A_1^{\alpha-1}W_1 + A_2^{\alpha-1}W_2 + \cdots + A_m^{\alpha-1}W_m}{W^{\alpha}/M^{\alpha-1}}\right)^{1/(\alpha-1)} \\ &= \left(\frac{A_1^{\alpha-1}W_1 + A_2^{\alpha-1}W_2 + \cdots + A_m^{\alpha-1}W_m}{W}\right)^{1/(\alpha-1)} \frac{1}{W/M}. \end{aligned}$$

The theorem is thus proved. ■

Notice that $A(r_1, r_2, \ldots, r_n) = \max\{A_1, A_2, \ldots, A_m\}$. Hence, we have

$$\left(\frac{A_1^{\alpha-1}W_1 + A_2^{\alpha-1}W_2 + \cdots + A_m^{\alpha-1}W_m}{W}\right)^{1/(\alpha-1)} \leq A(r_1, r_2, \ldots, r_n).$$

That is, the performance bound of the post-power-determination algorithm A-ES is no greater than the performance bound of the pre-power-determination algorithm ES-A in solving the energy constrained scheduling problem.

3.2.2. Time constrained scheduling

The following theorem gives a performance bound of the post-power-determination algorithm A-ES in solving the time constrained scheduling problem.

Theorem 4. *By using a post-power-determination algorithm A-ES to solve the time constrained scheduling problem, the energy allocation is*

$$E_j = \frac{A_j^{\alpha-1}W_j}{T^{\alpha-1}},$$

for all $1 \leq j \leq m$. *The total energy consumed is*

$$E_{A\text{-ES}} = \frac{1}{T^{\alpha-1}}\sum_{j=1}^{m} A_j^{\alpha-1}W_j.$$

The performance ratio is

$$\gamma_{A\text{-ES}} \leq \left(\frac{A_1^{\alpha-1}W_1 + A_2^{\alpha-1}W_2 + \cdots + A_m^{\alpha-1}W_m}{W}\right)\frac{1}{(W/M)^{\alpha-1}}. \quad (4)$$

Proof. By using a post-power-determination algorithm A-ES to solve the time constrained scheduling problem, we need

$$T_j = A_j\left(\frac{W_j}{E_j}\right)^{1/(\alpha-1)} = T,$$

that is,

$$E_j = \frac{A_j^{\alpha-1}W_j}{T^{\alpha-1}},$$

for all $1 \leq j \leq m$. In other words, we need to provide enough energy $E_{A\text{-ES}}$, so that the deadline T is met, i.e.,

$$T_{A\text{-ES}} = \left(\frac{1}{E_{A\text{-ES}}}\sum_{j=1}^{m} A_j^{\alpha-1}W_j\right)^{1/(\alpha-1)} = T.$$

The above equation implies that the total energy $E_{A\text{-ES}}$ consumed by algorithm A-ES is

$$E_{A\text{-ES}} = \frac{1}{T^{\alpha-1}}\sum_{j=1}^{m} A_j^{\alpha-1}W_j.$$

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

$$\begin{aligned}\gamma_{A\text{-ES}} &= \frac{E_{A\text{-ES}}}{E_{\text{OPT}}} \\ &\leq \frac{(1/T)^{\alpha-1}(A_1^{\alpha-1}W_1 + A_2^{\alpha-1}W_2 + \cdots + A_m^{\alpha-1}W_m)}{(M/T^{\alpha-1})(W/M)^{\alpha}} \\ &= \left(\frac{A_1^{\alpha-1}W_1 + A_2^{\alpha-1}W_2 + \cdots + A_m^{\alpha-1}W_m}{W}\right)\frac{1}{(W/M)^{\alpha-1}}.\end{aligned}$$

This proves the theorem. ■

Table 7A
Simulation data for the performance bound (3) (IT(500), CI = ±0.34104%).

Average task size	Uniform distribution	Binomial distribution	Geometric distribution
10	1.1738552	1.1788602	1.1450335
20	1.0833691	1.0858256	1.0776934
30	1.0640175	1.0595913	1.0722596
40	1.0609201	1.0516637	1.0690041
50	1.0714196	1.0689405	1.0677635
60	1.0756109	1.0784248	1.0689723

Table 7B
Simulation data for the performance bound (4) (IT(500), CI = ±0.68379%).

Average task size	Uniform distribution	Binomial distribution	Geometric distribution
10	1.3757196	1.3953765	1.3149472
20	1.1739046	1.1771137	1.1630049
30	1.1334744	1.1203476	1.1468589
40	1.1253395	1.1055719	1.1447847
50	1.1494851	1.1455666	1.1427324
60	1.1617398	1.1636299	1.1437753

It is easy to see that the performance bound of the post-power-determination algorithm A-ES is no greater than the performance bound of the pre-power-determination algorithm ES-A in solving the time constrained scheduling problem.

3.2.3. Simulation results

We now show the expectation of the performance bounds in (3)–(4) through simulation.

Our simulation data for the performance bounds of the post-power-determination algorithm LS-ES are given in Tables 7A–7B for IT(500).

All parameters are identical to those set for Tables 1A–1B. As proven by Theorems 3–4, the post-power-determination algorithm LS-ES performs better than the pre-power-determination algorithm ES-LS for each combination of average task size and task size distribution.

4. Discrete speed levels

Assume that the available speed levels are $S_1 < S_2 < \cdots < S_d$. Furthermore, let $\phi_k = S_{k+1}/S_k$, where $1 \leq k \leq d-1$. With only finite number of speed levels, the challenge is how to generate a schedule which is close to that generated with continuous speed levels. We will show how the algorithms in Section 3 can be adapted to discrete speed levels without significant loss of performance.

4.1. Pre-power-determination algorithms

4.1.1. Energy constrained scheduling

Our main idea is to generate a schedule which is an approximation of the schedule produced in Theorem 1 by using the available discrete speed levels.

Let $E_k = WS_k^{\alpha-1}$, where $1 \leq k \leq d$. Assume that $E_k \leq E < E_{k+1}$ and $S_k \leq s < S_{k+1}$, where E is the energy constraint, and

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

is the execution speed obtained in the proof of Theorem 1.

Our strategy is to find a set of indices $I \subseteq \{1, 2, \ldots, n\}$, such that tasks i, $i \in I$, are executed with speed S_{k+1} and tasks i, $i \notin I$, are executed with speed S_k, and that

$$\left(\sum_{i\notin I} w_i\right)S_k^{\alpha-1} + \left(\sum_{i\in I} w_i\right)S_{k+1}^{\alpha-1} \leq E,$$

and that the left-hand side of the above inequality is as large as possible. The last inequality implies that

$$\sum_{i\in I} w_i \le \frac{E - WS_k^{\alpha-1}}{S_{k+1}^{\alpha-1} - S_k^{\alpha-1}} = \frac{E - E_k}{S_{k+1}^{\alpha-1} - S_k^{\alpha-1}}.$$

If we assume that $E = E_k(1+\Delta) = WS_k^{\alpha-1}(1+\Delta)$, where $0 \le \Delta < \phi_k^{\alpha-1} - 1$, we get

$$\sum_{i\in I} w_i \le K,$$

where

$$K = \frac{WS_k^{\alpha-1}\Delta}{S_{k+1}^{\alpha-1} - S_k^{\alpha-1}} = \left(\frac{\Delta}{\phi_k^{\alpha-1} - 1}\right) W.$$

The problem of finding a set of indices I is actually to choose a subset of objects from n objects of sizes $w_1, w_2, \ldots, w_n$, such that these chosen objects can be placed into a bag with capacity K, and that the total size of the selected objects in the bag is as large as possible. The set I can be determined by using a simple greedy algorithm called *list placement* (LP) [92], which works as follows. Initially, the available space of the bag is K. We scan the list of objects one after another. For each object i, we put the object into the bag if w_i is no greater than the currently available space of the bag. After object i is packed into the bag, the available space of the bag is reduced by w_i.

It is clear that once the execution speed of each task is determined, we have

$$T_{\text{ES-}A} = A(t_1, t_2, \ldots, t_n),$$

where $t_i = r_i/S_k$ for $i \notin I$, and $t_i = r_i/S_{k+1}$ for $i \in I$. By Proposition 1, the performance ratio of algorithm ES-A is

$$\beta_{\text{ES-}A} = \frac{T_{\text{ES-}A}}{T_{\text{OPT}}} \le \frac{A(t_1, t_2, \ldots, t_n)}{((M/E)(W/M)^{\alpha})^{1/(\alpha-1)}}.$$

The above discussion can be summarized in the following theorem.

Theorem 5. *By using a pre-power-determination algorithm ES-A to solve the energy constrained scheduling problem, the performance ratio is*

$$\beta_{\text{ES-}A} \le \frac{A(t_1, t_2, \ldots, t_n)}{((M/E)(W/M)^{\alpha})^{1/(\alpha-1)}}. \tag{5}$$

4.1.2. Time constrained scheduling

Our main idea is to generate a schedule which is an approximation of the schedule produced in Theorem 2 by using the available discrete speed levels.

Let $T_k = A(r_1, r_2, \ldots, r_n)/S_k$, where $1 \le k \le d$. Assume that $T_k > T \ge T_{k+1}$ and $S_k < s \le S_{k+1}$, where T is the time constraint, and

$$s = \frac{A(r_1, r_2, \ldots, r_n)}{T}$$

is the execution speed obtained in the proof of Theorem 2. Furthermore, we assume that the time constraint is $T = T_k(1-\Delta)$, where $0 < \Delta \le 1 - 1/\phi_k$.

Our strategy is to arrange the tasks in certain order, and then find an index b, such that tasks $1, 2, \ldots, b$ are executed with speed S_k and tasks $b+1, b+2, \ldots, n$ are executed with speed S_{k+1}, and that

$$A(t_1, t_2, \ldots, t_n) \le T,$$

where $t_i = r_i/S_k$ for $1 \le i \le b$, and $t_i = r_i/S_{k+1}$ for $b+1 \le i \le n$, and b is as large as possible. Then, we have

$$E_{\text{ES-}A} = \left(\sum_{i=1}^{b} w_i\right) S_k^{\alpha-1} + \left(\sum_{i=b+1}^{n} w_i\right) S_{k+1}^{\alpha-1}.$$

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

$$\gamma_{\text{ES-}A} = \frac{E_{\text{ES-}A}}{E_{\text{OPT}}} \le \frac{(w_1 + w_2 + \cdots + w_b)S_k^{\alpha-1} + (w_{b+1} + w_{b+2} + \cdots + w_n)S_{k+1}^{\alpha-1}}{(M/T^{\alpha-1})(W/M)^{\alpha}}.$$

The above discussion can be summarized in the following theorem.

Theorem 6. *By using a pre-power-determination algorithm ES-A to solve the time constrained scheduling problem, the performance ratio is*

$$\gamma_{\text{ES-}A} \le \frac{(w_1 + w_2 + \cdots + w_b)S_k^{\alpha-1} + (w_{b+1} + w_{b+2} + \cdots + w_n)S_{k+1}^{\alpha-1}}{(M/T^{\alpha-1})(W/M)^{\alpha}}. \tag{6}$$

The index b can be found by using the binary search algorithm in an interval $[lb, ub]$. It is clear that $A(t_1, t_2, \ldots, t_n)$ depends on b. Initially, $lb = 1$ and $ub = n$. We set $b = (lb + ub)/2$. If $A(t_1, t_2, \ldots, t_n) < T$, we set $lb = (lb + ub)/2 + 1$; otherwise, we set $ub = (lb + ub)/2$. The search is completed if $lb = ub$. Finally, we set $b = (lb + ub)/2 - 1$.

4.1.3. Simulation results

We now show the expectation of the performance bounds in (5)–(6) through simulation.

Our simulation data for the performance bounds of the pre-power-determination algorithm ES-LS are given in Tables 8–13, for the six types of task graphs respectively.

Table 8A
Simulation data for the performance bound (5) (IT(500), CI = ±1.31418%).

Average task size	Uniform distribution	Binomial distribution	Geometric distribution
10	1.4802000	1.4815759	1.4675007
20	1.3088172	1.3085285	1.3123301
30	1.2587699	1.2574060	1.2717605
40	1.2431246	1.2346972	1.2632977
50	1.2461323	1.2410009	1.2527744
60	1.2482941	1.2442779	1.2488359

Table 8B
Simulation data for the performance bound (6) (IT(500), CI = ±2.82929%).

Average task size	Uniform distribution	Binomial distribution	Geometric distribution
10	1.7722774	1.8148113	1.9029352
20	1.4889217	1.4995786	1.5994512
30	1.4600634	1.4433426	1.5208194
40	1.4565743	1.4160428	1.5092297
50	1.4811762	1.4767136	1.4953507
60	1.5248528	1.4872187	1.4786129

Table 9A
Simulation data for the performance bound (5) (CT(2,10), CI = ±1.04101%).

Average task size	Uniform distribution	Binomial distribution	Geometric distribution
10	1.4011281	1.4114901	1.4475286
20	1.2599358	1.2459119	1.2725152
30	1.2105799	1.2026877	1.2411670
40	1.1952755	1.1900901	1.2259406
50	1.1998833	1.2073815	1.2237489
60	1.1983067	1.2201052	1.2160760

Table 9B
Simulation data for the performance bound (6) (CT(2,10), CI = ±3.38677%).

Average task size	Uniform distribution	Binomial distribution	Geometric distribution
10	2.2733281	2.4176649	2.7652964
20	1.6340871	1.6880994	1.8174636
30	1.5349267	1.5332805	1.5998362
40	1.4685468	1.4692488	1.5619782
50	1.4652617	1.4923328	1.5407591
60	1.4459194	1.5270916	1.5086720

Table 10A
Simulation data for the performance bound (5) (PA(2,10), CI = ±1.39962%).

Average task size	Uniform distribution	Binomial distribution	Geometric distribution
10	1.6869191	1.7102639	1.7663974
20	1.3618047	1.3569517	1.4295905
30	1.2806196	1.2785914	1.3216469
40	1.2415204	1.2347142	1.2760383
50	1.2259741	1.2447799	1.2578893
60	1.2164613	1.2438438	1.2515346

Table 10B
Simulation data for the performance bound (6) (PA(2,10), CI = ±2.92578%).

Average task size	Uniform distribution	Binomial distribution	Geometric distribution
10	2.3769129	2.5905027	2.8151365
20	1.7078194	1.7178979	1.8952732
30	1.5456078	1.5275878	1.6199307
40	1.4885049	1.4737307	1.5568187
50	1.4608128	1.4902875	1.5349947
60	1.4565319	1.5208104	1.5145033

Table 11A
Simulation data for the performance bound (5) (LA(200), CI = ±2.01333%).

Average task size	Uniform distribution	Binomial distribution	Geometric distribution
10	1.7878766	1.8252803	1.9026930
20	1.3194789	1.3111903	1.3871795
30	1.2263830	1.2199219	1.2782206
40	1.1977777	1.1972839	1.2565644
50	1.1974917	1.2115625	1.2270764
60	1.2091115	1.2223213	1.2175088

Table 11B
Simulation data for the performance bound (6) (LA(200), CI = ±2.78964%).

Average task size	Uniform distribution	Binomial distribution	Geometric distribution
10	2.2819040	2.2617298	2.4914600
20	1.5108635	1.4785284	1.6713468
30	1.4033391	1.3944339	1.5041119
40	1.3831826	1.3702492	1.4553787
50	1.4149842	1.4706011	1.4539490
60	1.4010601	1.4666081	1.4348255

Table 12A
Simulation data for the performance bound (5) (DD(200), CI = ±1.63712%).

Average task size	Uniform distribution	Binomial distribution	Geometric distribution
10	1.6878659	1.6868576	1.7264586
20	1.2836456	1.2701128	1.3203330
30	1.2114772	1.1965342	1.2506925
40	1.1842532	1.1863488	1.2261895
50	1.1874710	1.2106041	1.2253910
60	1.2013503	1.2073089	1.2170532

All parameters used in Tables 1–6 remain the same. We set $\phi_k = 2$ for all $1 \le k \le d-1$, where d is large enough and not very related. The energy constraint is $E = E_k(1+\Delta)$, where k is some integer and Δ is uniformly distributed in $[0, \phi_k^{\alpha-1} - 1)$. The time constraint is $T = T_k(1-\Delta)$, where k is some integer and Δ is uniformly distributed in $(0, 1 - 1/\phi_k]$.

Table 12B
Simulation data for the performance bound (6) (DD(200), CI = ±3.64169%).

Average task size	Uniform distribution	Binomial distribution	Geometric distribution
10	2.5825475	2.5688181	2.7436398
20	1.5881836	1.5637508	1.6967545
30	1.4420455	1.4157434	1.5346161
40	1.4115704	1.4104472	1.4599536
50	1.4103420	1.4452986	1.4717989
60	1.4270322	1.4603983	1.4615142

Table 13A
Simulation data for the performance bound (5) (DAG(2000,0.001), CI = ±1.33405%).

Average task size	Uniform distribution	Binomial distribution	Geometric distribution
10	1.2783476	1.2654799	1.3461753
20	1.1909535	1.1857032	1.2222676
30	1.1742890	1.1681412	1.1959972
40	1.1781274	1.1614185	1.1878143
50	1.1793475	1.1818928	1.1878168
60	1.1850562	1.2097923	1.1880627

Table 13B
Simulation data for the performance bound (6) (DAG(2000,0.001), CI = ±2.83384%).

Average task size	Uniform distribution	Binomial distribution	Geometric distribution
10	1.4501086	1.4947344	1.7786323
20	1.3446787	1.3416277	1.4682050
30	1.3598515	1.3415398	1.3866931
40	1.3450735	1.3323213	1.3586393
50	1.3510648	1.4051232	1.3529095
60	1.3946523	1.4530305	1.3726073

As expected, the performance bounds of ES-LS for discrete speed levels perform worse than for continuous speed levels. However, one should notice that the lower bounds in Propositions 1 and 2 are overly optimistic for manycore processors with discrete speed levels. In other words, an optimal solution should be worse than the one predicted by our lower bounds. Therefore, the actual performance ratios should be better than the data in Tables 8–13.

4.2. Post-power-determination algorithms

4.2.1. Energy constrained scheduling

Our main idea is to generate a schedule which is an approximation of the schedule produced in Theorem 3 by using the available discrete speed levels.

From the proof of Theorem 3, we know that the execution speed of M_j is

$$s_j = \left(\frac{E_j}{W_j}\right)^{1/(\alpha-1)} = \left(\frac{A_j}{(A_1^{\alpha-1}W_1 + A_2^{\alpha-1}W_2 + \cdots + A_m^{\alpha-1}W_m)^{1/(\alpha-1)}}\right) E^{1/(\alpha-1)},$$

where $1 \le j \le m$. Let $E_{j,k} = W_j S_k^{\alpha-1}$, where $1 \le j \le m$ and $1 \le k \le d$. Assume that $E_{j,k_j} \le E_j < E_{j,k_j+1}$ and $S_{k_j} \le s_j < S_{k_j+1}$, where E_j is the amount of energy allocated to M_j in the proof of Theorem 3.

If $S_k = \phi^k$ for all k, then we have

$$\phi^{k_j} \le \left(\frac{E_j}{W_j}\right)^{1/(\alpha-1)} < \phi^{k_j+1},$$

which implies that

$$k_j = \left\lfloor \frac{1}{\alpha-1} \log_\phi \frac{E_j}{W_j} \right\rfloor,$$

for all $1 \le j \le m$.

Let the tasks executed on M_j be named as (j, i), where $1 \le i \le n_j$. Our strategy is to find a set of indices $I_j \subseteq \{1, 2, \ldots, n_j\}$, such that tasks (j, i), $i \in I_j$, are executed with speed S_{k_j+1} and tasks (j, i), $i \notin I_j$, are executed with speed S_{k_j}, and that

$$\left(\sum_{i \notin I_j} w_{j,i}\right) S_{k_j}^{\alpha-1} + \left(\sum_{i \in I_j} w_{j,i}\right) S_{k_j+1}^{\alpha-1} \le E_j,$$

and that the left-hand side of the above inequality is as large as possible. If we assume that $E_j = E_{j,k_j}(1 + \Delta_j) = W_j S_{k_j}^{\alpha-1}(1 + \Delta_j)$, where $0 \le \Delta_j < \phi_{k_j}^{\alpha-1} - 1$, we get

$$\sum_{i \in I_j} w_{j,i} \le K_j,$$

where

$$K_j = \left(\frac{\Delta_j}{\phi_{k_j}^{\alpha-1} - 1}\right) W_j.$$

It is clear that we have

$$T_{\text{A-ES}} = \max(T_1, T_2, \ldots, T_m) = \max_{1 \le j \le m} \left(A_j(t_{j,1}, t_{j,2}, \ldots, t_{j,n_j})\right),$$

where $t_{j,i} = r_{j,i}/S_{k_j}$ for $i \notin I_j$, and $t_{j,i} = r_{j,i}/S_{k_j+1}$ for $i \in I_j$, and $T_j = A_j(t_{j,1}, t_{j,2}, \ldots, t_{j,n_j})$ denotes the schedule length produced by algorithm A for tasks with execution times $t_{j,1}, t_{j,2}, \ldots, t_{j,n_j}$ on M_j. By Proposition 1, the performance ratio of algorithm A-ES is

$$\begin{aligned}\beta_{\text{A-ES}} &= \frac{T_{\text{A-ES}}}{T_{\text{OPT}}} \le \frac{\max(T_1, T_2, \ldots, T_m)}{((M/E)(W/M)^\alpha)^{1/(\alpha-1)}} \\ &= \frac{\max_{1 \le j \le m} \left(A_j(t_{j,1}, t_{j,2}, \ldots, t_{j,n_j})\right)}{((M/E)(W/M)^\alpha)^{1/(\alpha-1)}}.\end{aligned}$$

The above discussion can be summarized in the following theorem.

Theorem 7. *By using a post-power-determination algorithm A-ES to solve the energy constrained scheduling problem, the performance ratio is*

$$\beta_{\text{A-ES}} \le \frac{\max_{1 \le j \le m} \left(A_j(t_{j,1}, t_{j,2}, \ldots, t_{j,n_j})\right)}{((M/E)(W/M)^\alpha)^{1/(\alpha-1)}}. \tag{7}$$

The set I_j can be determined by using a list placement algorithm described in Section 4.1.1.

4.2.2. Time constrained scheduling

Our main idea is to generate a schedule which is an approximation of the schedule produced in Theorem 4 by using the available discrete speed levels.

From the proof of Theorem 4, we know that the execution speed of M_j is

$$s_j = \left(\frac{E_j}{W_j}\right)^{1/(\alpha-1)} = \frac{A_j}{T} = \frac{A_j(r_{j,1}, r_{j,2}, \ldots, r_{j,n_j})}{T},$$

where $1 \le j \le m$. Let $T_{j,k} = A_j/S_k = A_j(r_{j,1}, r_{j,2}, \ldots, r_{j,n_j})/S_k$, where $1 \le j \le m$ and $1 \le k \le d$. Assume that $T_{j,k_j} > T \ge T_{j,k_j+1}$ and $S_{k_j} < s_j \le S_{k_j+1}$.

If $S_k = \phi^k$ for all k, then we have

$$\phi^{k_j} < \frac{A_j}{T} \le \phi^{k_j+1},$$

which implies that

$$k_j = \left\lceil \log_\phi \frac{A_j}{T} \right\rceil - 1,$$

for all $1 \le j \le m$.

Our strategy is to arrange the tasks executed on M_j in certain order, say, $(j, 1)$, $(j, 2)$, …, (j, n_j), and then find an index b_j, such that tasks $(j, 1)$, $(j, 2)$, …, (j, b_j) are executed with speed S_{k_j} and tasks $(j, b_j + 1)$, $(j, b_j + 2)$, …, (j, n_j) are executed with speed S_{k_j+1}, and that

$$T_j = A_j(t_{j,1}, t_{j,2}, \ldots, t_{j,n_j}) \le T,$$

where $t_{j,i} = r_{j,i}/S_{k_j}$ for $1 \le i \le b_j$, and $t_{j,i} = r_{j,i}/S_{k_j+1}$ for $b_j + 1 \le i \le n_j$, and b_j is as large as possible. Then, we have

$$E_{\text{A-ES}} = \sum_{j=1}^{m} \left(\left(\sum_{i=1}^{b_j} w_{j,i}\right) S_{k_j}^{\alpha-1} + \left(\sum_{i=b_j+1}^{n_j} w_{j,i}\right) S_{k_j+1}^{\alpha-1} \right).$$

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

$$\begin{aligned}\gamma_{\text{A-ES}} = \frac{E_{\text{A-ES}}}{E_{\text{OPT}}} &\le \frac{1}{(M/T^{\alpha-1})(W/M)^\alpha} \\ &\times \sum_{j=1}^{m} \left(\left(\sum_{i=1}^{b_j} w_{j,i}\right) S_{k_j}^{\alpha-1} + \left(\sum_{i=b_j+1}^{n_j} w_{j,i}\right) S_{k_j+1}^{\alpha-1} \right).\end{aligned}$$

The above discussion can be summarized in the following theorem.

Theorem 8. *By using a post-power-determination algorithm A-ES to solve the time constrained scheduling problem, the performance ratio is*

$$\begin{aligned}\gamma_{\text{A-ES}} &\le \frac{1}{(M/T^{\alpha-1})(W/M)^\alpha} \\ &\times \sum_{j=1}^{m} \left(\left(\sum_{i=1}^{b_j} w_{j,i}\right) S_{k_j}^{\alpha-1} + \left(\sum_{i=b_j+1}^{n_j} w_{j,i}\right) S_{k_j+1}^{\alpha-1} \right).\end{aligned} \tag{8}$$

The index b_j can be found by using the binary search algorithm described in Section 4.1.2.

4.2.3. Simulation results

We now show the expectation of the performance bounds in (7)–(8) through simulation.

Our simulation data for the performance bounds of the post-power-determination algorithm LS-ES are given in Tables 14A–14B for IT(500).

All parameters are identical to those set for Tables 8A–8B.

We observe that for energy constrained scheduling, the post-power-determination algorithm LS-ES performs worse than the pre-power-determination algorithm ES-LS for each combination of average task size and task size distribution. The reason is that when E_j is allocated to tasks in M_j, i.e., finding the set I_j, there is some leftover of E_j, for all $1 \le j \le m$. Hence, when the LP algorithm is applied m times on the m manycore processors separately by the post-power-determination algorithm LS-ES, there will be more wasted energy than a single application of the LP algorithm for all the n tasks by the pre-power-determination algorithm ES-LS.

Table 14A
Simulation data for the performance bound (7) (IT(500), CI = ±3.79952%).

Average task size	Uniform distribution	Binomial distribution	Geometric distribution
10	1.6568728	1.6558267	1.5970324
20	1.4908681	1.4966154	1.4934169
30	1.4560852	1.4362307	1.4850170
40	1.4894199	1.4002700	1.4756269
50	1.4620843	1.4290061	1.5093383
60	1.4679689	1.4449650	1.5159271

Table 14B
Simulation data for the performance bound (8) (IT(500), CI = ±1.88312%).

Average task size	Uniform distribution	Binomial distribution	Geometric distribution
10	1.6373675	1.6489660	1.6107650
20	1.4487165	1.4374588	1.4599286
30	1.4345915	1.4097659	1.4564348
40	1.4259428	1.3921292	1.4623996
50	1.4461258	1.4371667	1.4664019
60	1.4704655	1.4951573	1.4405544

However, for time constrained scheduling, the post-power-determination algorithm LS-ES performs better than the pre-power-determination algorithm ES-LS for each combination of average task size and task size distribution. In other words, m binary searches on the m manycore processors separately yields less energy consumption than a single binary search on the entire list of n tasks. The reason is that the pre-power-determination only allows two speeds (i.e., S_k and S_{k+1} in Theorem 6), while the post-power-determination algorithm allows $2m$ speeds (i.e., S_{k_j} and S_{k_j+1}, for $1 \leq j \leq m$, in Theorem 8, two speeds for each of the m manycore processors).

5. Summary

We have defined the problems of energy and time constrained scheduling of precedence constrained parallel tasks on multiple manycore processors in a cloud computing environment. We extended our lower bounds from a single parallel computing system to multiple parallel computing systems. We developed pre-power-determination algorithms and post-power-determination algorithms for both energy and time constrained scheduling of precedence constrained parallel tasks on multiple manycore processors with continuous or discrete speed levels. We evaluated the performance of these algorithms analytically and experimentally. Our main strategy is to embed the ES method into our algorithms. The ES method not only makes our analysis possible, but also yields good performance of our algorithms.

We would like to emphasize two significant aspects of our research. First, our algorithms developed in this paper are applicable to a variety of environments, including parallel, distributed, cluster, grid, and cloud computing environments, where manycore processors are employed. Second, the performance of our algorithms have been compared with optimal algorithms, such that the algorithms in this paper have rigorous performance guarantee and practical implication.

Acknowledgments

The author would like to express his gratitude to three anonymous reviewers for their criticism and comments on improving the quality of the manuscript.

References

[1] M.V. Zelkowitz (Ed.), Advances in Computers, Vol. 72, Academic Press, London, UK, 2008.
[2] J. Shalf, S. Dosanjh, J. Morrison, Exascale computing technology challenges, Lecture Notes in Comput. Sci. 6449 (2011) 1–25.
[3] T. Mastelic, A. Oleksiak, H. Claussen, I. Brandic, J.-M. Pierson, A.V. Vasilakos, Cloud computing: survey on energy efficiency, ACM Comput. Surv. 47 (2) (2015) article no. 33.
[4] A. Berl, E. Gelenbe, M. Di Girolamo, G. Giuliani, H. De Meer, M.Q. Dang, K. Pentikousis, Energy-efficient cloud computing, Comput. J. 53 (7) (2010) 1045–1051.
[5] Google, Google apps: energy efficiency in the cloud, Whitepaper, 2012. https://static.googleusercontent.com/media/www.google.com/en//green/pdf/google-apps.pdf.
[6] E. Masanet, A. Shehabi, L. Ramakrishnan, J. Liang, X. Ma, B. Walker, V. Hendrix, P. Mantha, The energy efficiency potential of cloud-based software: A US case study, Lawrence Berkeley National Laboratory, Berkeley, California, 2013.
[7] V. Venkatachalam, M. Franz, Power reduction techniques for microprocessor systems, ACM Comput. Surv. 37 (3) (2005) 195–237.
[8] W.-C. Feng, The importance of being low power in high performance computing, CTWatch Quart. 1 (3) (2005) Los Alamos National Laboratory.
[9] A. Gara, et al., Overview of the Blue Gene/L system architecture, IBM J. Res. Dev. 49 (2/3) (2005) 195–212.
[10] 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, 2005.
[11] M.B. Srivastava, A.P. Chandrakasan, R.W. Rroderson, Predictive system shutdown and other architectural techniques for energy efficient programmable computation, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 4 (1) (1996) 42–55.
[12] D. Donofrio, L. Oliker, J. Shalf, M.F. Wehner, C. Rowen, J. Krueger, S. Kamil, M. Mohiyuddin, Energy-efficient computing for extreme-scale science, Computer 42 (11) (2009) 62–71.
[13] W.-c. Feng, K.W. Cameron, The green500 list: encouraging sustainable supercomputing, Computer 40 (12) (2007) 50–55.
[14] V.W. Freeh, D.K. Lowenthal, F. Pan, N. Kappiah, R. Springer, B.L. Rountree, M.E. Femal, Analyzing the energy–time trade-off in high-performance computing applications, IEEE Trans. Parallel Distrib. Syst. 18 (6) (2007) 835–848.
[15] L. Benini, A. Bogliolo, G. De Micheli, A survey of design techniques for system-level dynamic power management, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 8 (3) (2000) 299–316.
[16] O.S. Unsal, I. Koren, System-level power-aware design techniques in real-time systems, Proc. IEEE 91 (7) (2003) 1055–1069.
[17] A. Beloglazov, R. Buyya, Y.C. Lee, A. Zomaya, A taxonomy and survey of energy-efficient data centers and cloud computing systems, Adv. Comput. 82 (2011) 47–111.
[18] F. Kong, X. Liu, A survey on green-energy-aware power management for datacenters, ACM Comput. Surv. 47 (2) (2014) article 30.
[19] A.-C. Orgerie, M.D. de Assuncao, L. Lefevre, A survey on techniques for improving the energy efficiency of large-scale distributed systems, ACM Comput. Surv. 46 (4) (2014) article 47.
[20] S. Albers, Energy-efficient algorithms, Commun. ACM 53 (5) (2010) 86–96.
[21] G.L. Valentini, W. Lassonde, S.U. Khan, N. Min-Allah, S.A. Madani, J. Li, L. Zhang, L. Wang, N. Ghani, J. Kolodziej, H. Li, A.Y. Zomaya, C.-Z. Xu, P. Balaji, A. Vishnu, F. Pinel, J.E. Pecero, D. Kliazovich, P. Bouvry, An overview of energy efficiency techniques in cluster computing systems, Cluster Comput. 16 (1) (2013) 3–15.
[22] S. Zhuravlev, J.C. Saez, S. Blagodurov, A. Fedorova, M. Prieto, Survey of energy-cognizant scheduling techniques, IEEE Trans. Parallel Distrib. Syst. 24 (7) (2013) 1447–1464.
[23] K. Li, Energy efficient scheduling of parallel tasks on multiprocessor computers, J. Supercomput. 60 (2) (2012) 223–247.
[24] M.R. Stan, K. Skadron, Guest editors' introduction: power-aware computing, IEEE Comput. 36 (12) (2003) 35–38.
[25] M. Weiser, B. Welch, A. Demers, S. Shenker, Scheduling for reduced CPU energy, in: Proceedings of the 1st USENIX Symposium on Operating Systems Design and Implementation, 1994, pp. 13–23.
[26] F. Yao, A. Demers, S. Shenker, A scheduling model for reduced CPU energy, in: Proceedings of the 36th IEEE Symposium on Foundations of Computer Science, 1995, pp. 374–382.
[27] E. Bampis, C. Dürr, F. Kacem, I. Milis, Speed scaling with power down scheduling for agreeable deadlines, Sustain. Comput.: Inform. Syst. 2 (4) (2012) 184–189.
[28] N. Bansal, T. Kimbrel, K. Pruhs, Speed scaling to manage energy and temperature, J. ACM 54 (1) (2007) article no. 3.
[29] P. Baptiste, M. Chrobak, C. Dürr, Polynomial-time algorithms for minimum energy scheduling, ACM Trans. Algorithms 8 (3) (2012) article no. 26.
[30] H.-L. Chan, W.-T. Chan, T.-W. Lam, L.-K. Lee, K.-S. Mak, P.W.H. Wong, Energy efficient online deadline scheduling, in: Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms, 2007, pp. 795–804.
[31] S. Irani, S. Shukla, R. Gupta, Algorithms for power savings, ACM Trans. Algorithms 3 (4) (2007) article no. 41.
[32] W.-C. Kwon, T. Kim, Optimal voltage allocation techniques for dynamically variable voltage processors, ACM Trans. Embedded Comput. Syst. 4 (1) (2005) 211–230.
[33] M. Li, B.J. Liu, F.F. Yao, Min-energy voltage allocation for tree-structured tasks, J. Comb. Optim. 11 (2006) 305–319.
[34] M. Li, A.C. Yao, F.F. Yao, Discrete and continuous min-energy schedules for variable voltage processors, Proc. Natl. Acad. Sci. USA 103 (11) (2006) 3983–3987.
[35] M. Li, F.F. Yao, An efficient algorithm for computing optimal discrete voltage schedules, SIAM J. Comput. 35 (3) (2006) 658–671.
[36] H.-S. Yun, J. Kim, On energy-optimal voltage scheduling for fixed-priority hard real-time systems, ACM Trans. Embedded Comput. Syst. 2 (3) (2003) 393–430.

[37] H. Aydin, R. Melhem, D. Mossé, P. Mejía-Alvarez, Power-aware scheduling for periodic real-time tasks, IEEE Trans. Comput. 53 (5) (2004) 584–600.
[38] M.E.T. Gerards, Algorithmic Power Management – Energy Minimization under Real-Time Constraints (Ph.D. thesis), University of Twente, Netherlands, 2014.
[39] J.-J. Han, X. Wu, D. Zhu, H. Jin, L.T. Yang, J.-L. Gaudiot, Synchronization-aware energy management for VFI-based multicore real-time systems, IEEE Trans. Comput. 61 (12) (2012) 1682–1696.
[40] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, M.B. Srivastava, Power optimization of variable-voltage core-based systems, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 18 (12) (1999) 1702–1714.
[41] C. Im, S. Ha, H. Kim, Dynamic voltage scheduling with buffers in low-power multimedia applications, ACM Trans. Embedded Comput. Syst. 3 (4) (2004) 686–705.
[42] C.M. Krishna, Y.-H. Lee, Voltage-clock-scaling adaptive scheduling techniques for low power in hard real-time systems, IEEE Trans. Comput. 52 (12) (2003) 1586–1593.
[43] Y.-H. Lee, C.M. Krishna, Voltage-clock scaling for low energy consumption in fixed-priority real-time systems, Real-Time Syst. 24 (3) (2003) 303–317.
[44] J.R. Lorch, A.J. Smith, PACE: a new approach to dynamic voltage scaling, IEEE Trans. Comput. 53 (7) (2004) 856–869.
[45] R.N. Mahapatra, W. Zhao, An energy-efficient slack distribution technique for multimode distributed real-time embedded systems, IEEE Trans. Parallel Distrib. Syst. 16 (7) (2005) 650–662.
[46] B.C. Mochocki, X.S. Hu, G. Quan, A unified approach to variable voltage scheduling for nonideal DVS processors, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 23 (9) (2004) 1370–1377.
[47] G. Quan, X.S. Hu, Energy efficient DVS schedule for fixed-priority real-time systems, ACM Trans. Embedded Comput. Syst. 6 (4) (2007) Article no. 29.
[48] N.B. Rizvandi, J. Taheri, A.Y. Zomaya, Some observations on optimal frequency selection in DVFS-based energy consumption minimization, J. Parallel Distrib. Comput. 71 (8) (2011) 1154–1164.
[49] D. Shin, J. Kim, Power-aware scheduling of conditional task graphs in real-time multiprocessor systems, in: Proceedings of the International Symposium on Low Power Electronics and Design, 2003, pp. 408–413.
[50] D. Shin, J. Kim, S. Lee, Intra-task voltage scheduling for low-energy hard real-time applications, IEEE Des. Test Comput. 18 (2) (2001) 20–30.
[51] P. Yang, C. Wong, P. Marchal, F. Catthoor, D. Desmet, D. Verkest, R. Lauwereins, Energy-aware runtime scheduling for embedded-multiprocessor SOCs, IEEE Des. Test Comput. 18 (5) (2001) 46–58.
[52] X. Zhong, C.-Z. Xu, Energy-aware modeling and scheduling for dynamic voltage scaling with statistical real-time guarantee, IEEE Trans. Comput. 56 (3) (2007) 358–372.
[53] D. Zhu, R. Melhem, B.R. Childers, Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems, IEEE Trans. Parallel Distrib. Syst. 14 (7) (2003) 686–700.
[54] D. Zhu, D. Mossé, R. Melhem, Power-aware scheduling for AND/OR graphs in real-time systems, IEEE Trans. Parallel Distrib. Syst. 15 (9) (2004) 849–864.
[55] J. Zhuo, C. Chakrabarti, Energy-efficient dynamic task scheduling algorithms for DVS systems, ACM Trans. Embedded Comput. Syst. 7 (2) (2008) Article no. 17.
[56] J.A. Barnett, Dynamic task-level voltage scheduling optimizations, IEEE Trans. Comput. 54 (5) (2005) 508–520.
[57] D.P. Bunde, Power-aware scheduling for makespan and flow, in: Proceedings of the 18th ACM Symposium on Parallelism in Algorithms and Architectures, 2006, pp. 190–196.
[58] S. Cho, R.G. Melhem, On the interplay of parallelization, program performance, and energy consumption, IEEE Trans. Parallel Distrib. Syst. 21 (3) (2010) 342–353.
[59] S.K. Garg, C.S. Yeo, A. Anandasivam, R. Buyya, Environment-conscious scheduling of HPC applications on distributed cloud-oriented data centers, J. Parallel Distrib. Comput. 71 (6) (2011) 732–749.
[60] S.U. Khan, I. Ahmad, A cooperative game theoretical technique for joint optimization of energy consumption and response time in computational grids, IEEE Trans. Parallel Distrib. Syst. 20 (3) (2009) 346–360.
[61] Y.C. Lee, A.Y. Zomaya, Energy conscious scheduling for distributed computing systems under different operating conditions, IEEE Trans. Parallel Distrib. Syst. 22 (8) (2011) 1374–1381.
[62] K. Li, X. Tang, K. Li, Energy-efficient stochastic task scheduling on heterogeneous computing systems, IEEE Trans. Parallel Distrib. Syst. 25 (11) (2014) 2867–2876.
[63] C. Rusu, R. Melhem, D. Mossé, Maximizing rewards for real-time applications with energy constraints, ACM Trans. Embedded Comput. Syst. 2 (4) (2003) 537–559.
[64] L. Zhang, K. Li, Y. Xu, F. Zhang, K. Li, Maximizing reliability with energy conservation for parallel task scheduling in a heterogeneous cluster, Inform. Sci. 319 (2015) 113–131.
[65] Z. Zong, A. Manzanares, X. Ruan, X. Qin, EAD and PEBD: two energy-aware duplication scheduling algorithms for parallel tasks on homogeneous clusters, IEEE Trans. Comput. 60 (3) (2011) 360–374.
[66] V. Devadas, H. Aydin, On the interplay of voltage/frequency scaling and device power management for frame-based real-time embedded applications, IEEE Trans. Comput. 61 (1) (2012) 31–44.
[67] F. Hu, J.J. Evans, Power and environment aware control of Beowulf clusters, Cluster Comput. 12 (3) (2009) 299–308.
[68] W.-K. Lee, S.-W. Lee, W.-O. Siew, Hybrid model for dynamic power management, IEEE Trans. Consum. Electron. 55 (2) (2009) 656–664.
[69] G. Lovász, F. Niedermeier, H. de Meer, Performance tradeoffs of energy-aware virtual machine consolidation, Cluster Comput. 16 (3) (2013) 481–496.
[70] V.A. Patil, V. Chaudhary, Rack aware scheduling in HPC data centers: an energy conservation strategy, Cluster Comput. 16 (3) (2013) 559–573.
[71] J. Augustine, S. Irani, C. Swamy, Optimal power-down strategies, SIAM J. Comput. 37 (5) (2008) 1499–1516.
[72] J. Mei, K. Li, K. Li, Energy-aware task scheduling in heterogeneous computing environments, Cluster Comput. 17 (2) (2014) 537–550.
[73] L.M. Zhang, K. Li, D.C.-T. Lo, Y. Zhang, Energy-efficient task scheduling algorithms on heterogeneous computers with continuous and discrete speeds, Sustain. Comput.: Inform. Syst. 3 (2) (2013) 109–118.
[74] K. Li, Performance analysis of power-aware task scheduling algorithms on multiprocessor computers with dynamic voltage and speed, IEEE Trans. Parallel Distrib. Syst. 19 (11) (2008) 1484–1497.
[75] K. Li, Scheduling precedence constrained tasks with reduced processor energy on multiprocessor computers, IEEE Trans. Comput. 61 (12) (2012) 1668–1681.
[76] K. Li, Power allocation and task scheduling on multiprocessor computers with energy and time constraints, in: A.Y. Zomaya, Y.C. Lee (Eds.), Energy-Efficient Distributed Computing Systems, John Wiley & Sons, 2012, pp. 1–37. (Chapter 1).
[77] K. Li, Algorithms and analysis of energy-efficient scheduling of parallel tasks, in: I. Ahmad, S. Ranka (Eds.), Handbook of Energy-Aware and Green Computing, Vol. 1, CRC Press/Taylor & Francis Group, 2012, pp. 331–360. (Chapter 15).
[78] K. Li, Energy-efficient and high-performance processing of large-scale parallel applications in data centers, in: S.U. Khan, A.Y. Zomaya (Eds.), Data Centers, Springer, 2015, pp. 1–33. (Chapter 1).
[79] K. Li, Power and performance management for parallel computations in clouds and data centers, J. Comput. System Sci. 82 (2016) 174–190.
[80] D.A. Ellsworth, A.D. Malony, B. Rountree, M. Schulz, POW: system-wide dynamic reallocation of limited power in HPC, in: Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing, 2015, pp. 145–148.
[81] D.A. Ellsworth, A.D. Malony, B. Rountree, M. Schulz, Dynamic power sharing for higher job throughput, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Article No. 80, 2015.
[82] T. Patki, D.K. Lowenthal, A. Sasidharan, M. Maiterth, B.L. Rountree, M. Schulz, B.R. de Supins, Practical resource management in power-constrained, high performance computing, in: Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing, 2015, pp. 121–132.
[83] Intel, Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor – White Paper, March 2004.
[84] G. Qu, 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.
[85] M. Marinoni, G. Buttazzo, Elastic DVS management in processors with discrete voltage/frequency modes, IEEE Trans. Ind. Inf. 3 (1) (2007) 51–62.
[86] K. Li, Energy and time constrained task scheduling on multiprocessor computers with discrete speed levels, J. Parallel Distrib. Comput. 95 (2016) 15–28.
[87] A.P. Chandrakasan, S. Sheng, R.W. Brodersen, Low-power CMOS digital design, IEEE J. Solid-State Circuits 27 (4) (1992) 473–484.
[88] B. Zhai, D. Blaauw, D. Sylvester, K. Flautner, Theoretical and practical limits of dynamic voltage scaling, in: Proceedings of the 41st Design Automation Conference, 2004, pp. 868–873.
[89] K. Li, Private communication available on results of pre-power-determination algorithms.
[90] R.L. Graham, Bounds on multiprocessing timing anomalies, SIAM J. Appl. Math. 2 (1969) 416–429.
[91] K. Li, Analysis of the list scheduling algorithm for precedence constrained parallel tasks, J. Comb. Optim. 3 (1999) 73–88.
[92] K. Li, Average-case performance analysis of an approximation algorithm for maximum subset sum using recurrence relations, Comput. Math. Appl. 36 (6) (1998) 63–75.

Keqin Li is a SUNY Distinguished Professor of computer science in the State University of New York. He is also a Distinguished Professor of Chinese National Recruitment Program of Global Experts (1000 Plan) at Hunan University, China. He was an Intellectual Ventures endowed visiting chair professor at the National Laboratory for Information Science and Technology, Tsinghua University, Beijing, China, during 2011–2014. His current research interests include parallel computing and high-performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU–GPU hybrid and cooperative computing, multicore computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of things and cyber–physical systems. He has published over 460 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently or has served on the editorial boards of *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Services Computing*, and *IEEE Transactions on Sustainable Computing*. He is an IEEE Fellow.