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.

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.

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

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
requirements necessary to house the computational power at the scale of 10^{16} 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 EpC4 (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 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.

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.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>The number of parallel tasks</td>
</tr>
<tr>
<td>( n_j )</td>
<td>The number of parallel tasks scheduled on ( M_j )</td>
</tr>
<tr>
<td>( \pi_i )</td>
<td>The number of cores requested by task ( i )</td>
</tr>
<tr>
<td>( r_i )</td>
<td>The maximum execution requirement on the cores executing task ( i )</td>
</tr>
<tr>
<td>( w_i )</td>
<td>The work of task ( i )</td>
</tr>
<tr>
<td>( p_i )</td>
<td>The power supplied to execute task ( i )</td>
</tr>
<tr>
<td>( s_i )</td>
<td>The execution speed of task ( i )</td>
</tr>
<tr>
<td>( t_i )</td>
<td>The execution time of task ( i )</td>
</tr>
<tr>
<td>( e_i )</td>
<td>The energy consumed to execute task ( i )</td>
</tr>
<tr>
<td>( m )</td>
<td>The number of manycore processors</td>
</tr>
<tr>
<td>( M_j )</td>
<td>The name as well as the number of cores in the ( j )-th manycore processor</td>
</tr>
<tr>
<td>( \alpha )</td>
<td>Exponent of power consumption</td>
</tr>
<tr>
<td>( S_k )</td>
<td>Discrete level of core speed</td>
</tr>
<tr>
<td>( \phi_k )</td>
<td>( S_k )/( S_k )</td>
</tr>
<tr>
<td>( E )</td>
<td>Energy constraint</td>
</tr>
<tr>
<td>( E_i )</td>
<td>Energy allocated to ( M_i )</td>
</tr>
<tr>
<td>( E_k )</td>
<td>( W_{S_k}^{-1} )</td>
</tr>
<tr>
<td>( E_{OPT} )</td>
<td>The total amount of work to be performed for the ( i )-th parallel tasks</td>
</tr>
<tr>
<td>( W )</td>
<td>The total amount of work to be performed for the ( n )-th parallel tasks</td>
</tr>
<tr>
<td>( A )</td>
<td>An algorithm</td>
</tr>
<tr>
<td>( A_{opt} )</td>
<td>( A_{opt}(\pi_1, \pi_2, \ldots, \pi_n) )</td>
</tr>
<tr>
<td>( T )</td>
<td>The length of a schedule produced by algorithm ( A )</td>
</tr>
<tr>
<td>( T_{OPT} )</td>
<td>The shortest length of an optimal schedule</td>
</tr>
<tr>
<td>( \gamma )</td>
<td>The performance ratio of an algorithm ( A ) for time constrained scheduling</td>
</tr>
<tr>
<td>( W_{OPT} )</td>
<td>The minimum amount of energy consumed by an optimal schedule</td>
</tr>
<tr>
<td>( \beta_k )</td>
<td>The performance ratio of an algorithm ( A ) for energy constrained scheduling</td>
</tr>
<tr>
<td>( \gamma_k )</td>
<td>The performance ratio of an algorithm ( A ) for time constrained scheduling</td>
</tr>
</tbody>
</table>

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 \( \pi_i \) and \( r_i \) explained below. The integer \( \pi_i \) 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 \( \pi_i \) 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 \( \pi_i \) 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 \( \preceq \) 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 \( i \preceq j \), 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 \( j \)th manycore processor, where \( 1 \leq j \leq m \).

To execute a task \( i \), any \( \pi_i \) cores of the \( j \)th manycore processor can be allocated to task \( i \). However, core allocation cannot be performed across different manycore processors, i.e., all the \( \pi_i \) 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 \( s_i^\alpha \), 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 \( \pi_i \) cores allocated to task \( i \) have the same speed \( s_i \) for duration \( t_i \), although some of the \( \pi_i \) cores may be idle for some time. The energy consumed to execute task \( i \) is \( e_i = \pi_i r_i p_i = \pi_i r_i p_i^{1/\alpha} = \pi_i r_i s_i^{\alpha-1} = w_i s_i^{\alpha-1} \), 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 assumed 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 assumed 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 \( \phi_d = S_{k+1}/S_k \) for all \( 1 \leq k \leq d - 1 \). The quantity \( \phi_d \) 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 \( \beta_A = T_A/T_{OPT} \). 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 \( \gamma_A = E_A/E_{OPT} \). 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, \( T_A, T_{OPT}, \beta_A, B, E_A, E_{OPT}, \gamma_A, \) and \( C \) are all random variables. Let \( x \) be the expectation of a random variable \( x \).

**Definition 3.** If \( \beta_A \leq B \), then \( \bar{x}_A \leq \bar{y} \), where \( \bar{y} \) is an average-case performance bound of algorithm \( A \).

**Definition 4.** If \( \gamma_A \leq C \), then \( \gamma_A \leq \bar{C} \), where \( \bar{C} \) is an average-case performance bound of algorithm \( A \).
2.4. Lower bounds

Let \( W = w_1 + w_2 + \cdots + w_n \) 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 \( T_{\text{OPT}} \).

**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 \( M_i \) cores simulating an \( M_i \)-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 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 \( E_{\text{OPT}} \).

**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 [79].
- **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 \( A_1 \rightarrow A_2 \), 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 \( A(t_1, t_2, \ldots, t_n) = A(t_1, t_2, \ldots, t_n)/s \) 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

\[ \beta_{ES-A} \leq \frac{A(r_1, r_2, \ldots, r_n)}{W/M}. \]  

**Proof.** By using pre-power-determination algorithm ES-A to solve the energy constrained scheduling problem, we have \( \beta_1 = \beta_2 = \cdots = \beta_n = \beta \), where \( \beta \) is the same execution speed set for all tasks. The energy consumed by task \( i \) is

\[ e_i = \pi_i r_i p_i^{1-1/\alpha} = w_i s_i^{1-1/\alpha} = w_i \bar{s}_i^{-1} = w_i \bar{s}^{-1}, \]

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

\[ \sum_{i=1}^{n} e_i = s^{-1} \sum_{i=1}^{n} w_i = Ws^{-1} = E, \]

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

\[ T_{ES-A} = A(t_1, t_2, \ldots, t_n) = \frac{A(r_1, r_2, \ldots, r_n)}{s} = \frac{A(r_1, r_2, \ldots, r_n)}{W} \left( \frac{W}{E} \right)^{1/(\alpha-1)}. \]

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

\[ \beta_{ES-A} = \frac{T_{ES-A}}{T_{OPT}} \leq \frac{A(r_1, r_2, \ldots, r_n)(W/E)^{1/(\alpha-1)}}{((M/E)(W/M)^{\alpha})^{1/(\alpha-1)}} = \frac{A(r_1, r_2, \ldots, r_n)}{W/M}. \]

The theorem is thus proved. \( \square \)

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

\[ \gamma_{ES-A} \leq \left( \frac{A(r_1, r_2, \ldots, r_n)}{W/M} \right)^{a-1} . \]  

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

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

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

\[ E_{ES-A} = \left( \frac{A(r_1, r_2, \ldots, r_n)}{T} \right)^{a-1} W. \]

The execution speed is

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

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

\[ \gamma_{ES-A} = \frac{E_{ES-A}}{E_{OPT}} \leq \frac{A(r_1, r_2, \ldots, r_n)(T/\alpha-1)(W/M)^{\alpha}}{W/M} \]

This proves the theorem. \( \square \)

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 \( < \) 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 list \( L \), 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 \( \pi_i \) idle cores. If so, task \( i \) is allocated \( \pi_i \) 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 any incoming arcs constitute level 1. In general, a node \( i \) is in level \( j \) if the longest path from a node in level \( j \) 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 ≤ j ≤ h. The total number of nodes is n = \sum b^j \leq b^h \leq 2h. The precedence constraints are i < bi - b + 2, i < bi - b + 3, . . . , i < bi + 1, for all 1 ≤ i ≤ (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 ≤ j ≤ h, and b^h - j for h + 1 ≤ j ≤ 2h. The total number of nodes is n = \sum_{j=0}^{2h} b^j = (2b^h + b^h + b^h + \ldots + b^h) = (b^{h+1} - 1)/(b - 1). The precedence constraints are i < bi - b + 2, i < bi - b + 3, . . . , i < bi + 1, for all 1 ≤ i ≤ (b^h - 1)/(b - 1) (i.e., nodes in levels 0 to h - 1), and i < j for (n - 1 - i)/b for all (b^h - 1)/(b - 1) + 1 ≤ i ≤ (b^{h+1} - 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 nj = v - j + 1, where 1 ≤ j ≤ v. The total number of nodes is n = v(v + 1)/2. Let aj = nj + nj + 1 + \ldots + nj - 1 be the leading index of the nodes in level j, i.e., nodes in level j are aj, aj + 1, . . . , aj + nj - 1, where 1 ≤ j ≤ v. The precedence constraints are aj < aj+1+k, for all 1 ≤ j ≤ v - 1 and 0 ≤ k ≤ nj - 1, and aj+k < aj+1+k, for all 1 ≤ j ≤ v - 1 and 0 ≤ k ≤ nj - 2. We set v = 200 in our simulations.
• Diamond dags (DD(d))—A dag with n = d^2 nodes. The number of levels is \lfloor \log_d n \rfloor. The number of nodes in level j is nj = j for 1 ≤ j ≤ d, and nj = d - j + 1 for d + 1 ≤ j ≤ 2d - 1. Let aj = nj + nj + 1 + \ldots + nj - 1 be the leading index of the nodes in level j, i.e., nodes in level j are aj, aj + 1, . . . , aj + nj - 1, where 1 ≤ j ≤ d. The precedence constraints are aj < aj+1+k and aj+k < aj+1+k, for all 1 ≤ j ≤ d - 1 and 0 ≤ k ≤ nj - 1, aj < aj+1+k+1 and aj+k+1 < aj+1+k+1, for all d ≤ j ≤ 2d - 2, aj+k+1 < aj+1+k+1 and aj+k+1 < aj+1+k+1, for all 1 ≤ j ≤ 2d - 2 and 1 ≤ k ≤ nj - 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 ≤ i < j ≤ 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 ≤ i ≤ n. If p = 2/n, then it is in the range [0, 1]. We set n = 2000 and p = 0.01 in our simulations.

The task sizes \pi_1, \pi_2, . . . , \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 \xi_b be the probability that \pi_i = b, where \text{b} ≥ 1.
• Uniform distributions in the range [1...u], i.e., \xi_b = 1/u for all 1 ≤ b ≤ 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_n), i.e.,
\xi_b = \frac{1}{1 - (1 - p)^D} \left( \frac{D}{b} \right) p^b (1 - p)^{D - b},
for all 1 ≤ b ≤ D, where p is chosen such that Dp = \bar{\pi}, i.e., p = \bar{\pi}/D. However, the actual expectation of task sizes is
\bar{\pi} = \frac{1}{1 - (1 - p)^D} = \frac{\bar{\pi}}{1 - (1 - \bar{\pi})^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 = ±0.543883].

<table>
<thead>
<tr>
<th>Average task size</th>
<th>Uniform distribution</th>
<th>Binomial distribution</th>
<th>Geometric distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1.2245384</td>
<td>1.2233637</td>
<td>1.2157317</td>
</tr>
<tr>
<td>20</td>
<td>1.1701406</td>
<td>1.1146973</td>
<td>1.1302962</td>
</tr>
<tr>
<td>30</td>
<td>1.0936097</td>
<td>1.0834506</td>
<td>1.1064624</td>
</tr>
<tr>
<td>40</td>
<td>1.0856006</td>
<td>1.0721085</td>
<td>1.0989122</td>
</tr>
<tr>
<td>50</td>
<td>1.0940931</td>
<td>1.0859161</td>
<td>1.0977620</td>
</tr>
<tr>
<td>60</td>
<td>1.0957334</td>
<td>1.0936939</td>
<td>1.0939319</td>
</tr>
</tbody>
</table>

### Table 1B
Simulation data for the performance bound (2) [IT(500), CI = ±0.100843].

<table>
<thead>
<tr>
<th>Average task size</th>
<th>Uniform distribution</th>
<th>Binomial distribution</th>
<th>Geometric distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1.5010664</td>
<td>1.4974894</td>
<td>1.4743129</td>
</tr>
<tr>
<td>20</td>
<td>1.2463026</td>
<td>1.2419159</td>
<td>1.2756722</td>
</tr>
<tr>
<td>30</td>
<td>1.1965773</td>
<td>1.1750458</td>
<td>1.2284814</td>
</tr>
<tr>
<td>40</td>
<td>1.1785307</td>
<td>1.1477416</td>
<td>1.2110139</td>
</tr>
<tr>
<td>50</td>
<td>1.1973015</td>
<td>1.1794252</td>
<td>1.2055510</td>
</tr>
<tr>
<td>60</td>
<td>1.1990831</td>
<td>1.1956248</td>
<td>1.1955979</td>
</tr>
</tbody>
</table>

• Geometric distributions in the range [1...D], where D = \min(M_1, M_2, \ldots, M_n), i.e.,
\xi_b = \frac{q(1 - q)^{b-1}}{1 - (1 - q)^D},
for all 1 ≤ b ≤ D, where q is chosen such that \pi = \bar{\pi}, i.e., q = 1/\bar{\pi}. However, the actual expectation of task sizes is
\bar{\pi} - (\bar{\pi} + D)(1 - 1/\bar{\pi})^{-1} = \frac{\bar{\pi} - (\bar{\pi} + D)(1 - 1/\bar{\pi})^{-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, . . . , 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_n) = (128, 128, 196, 256, 256).

The parameter \alpha 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_{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.

In Table 1B, we show simulation data for the performance bound of \gamma_{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%).

<table>
<thead>
<tr>
<th>Average task size</th>
<th>Uniform distribution</th>
<th>Binomial distribution</th>
<th>Geometric distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1.2935307</td>
<td>1.3144250</td>
<td>1.3744934</td>
</tr>
<tr>
<td>20</td>
<td>1.1340361</td>
<td>1.1307032</td>
<td>1.164624</td>
</tr>
<tr>
<td>30</td>
<td>1.0920498</td>
<td>1.0903256</td>
<td>1.105078</td>
</tr>
<tr>
<td>40</td>
<td>1.0707470</td>
<td>1.0687952</td>
<td>1.057613</td>
</tr>
<tr>
<td>50</td>
<td>1.0678596</td>
<td>1.0772388</td>
<td>1.088394</td>
</tr>
<tr>
<td>60</td>
<td>1.0687400</td>
<td>1.0896199</td>
<td>1.0843028</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Average task size</th>
<th>Uniform distribution</th>
<th>Binomial distribution</th>
<th>Geometric distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1.6705812</td>
<td>1.7356443</td>
<td>1.882202</td>
</tr>
<tr>
<td>20</td>
<td>1.2810853</td>
<td>1.2806652</td>
<td>1.348139</td>
</tr>
<tr>
<td>30</td>
<td>1.1846435</td>
<td>1.1770298</td>
<td>1.2386948</td>
</tr>
<tr>
<td>40</td>
<td>1.1501013</td>
<td>1.1365295</td>
<td>1.2044014</td>
</tr>
<tr>
<td>50</td>
<td>1.1379861</td>
<td>1.1595916</td>
<td>1.1861233</td>
</tr>
<tr>
<td>60</td>
<td>1.1429223</td>
<td>1.185100</td>
<td>1.1769446</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Average task size</th>
<th>Uniform distribution</th>
<th>Binomial distribution</th>
<th>Geometric distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1.4736047</td>
<td>1.4673924</td>
<td>1.5293757</td>
</tr>
<tr>
<td>20</td>
<td>1.1691393</td>
<td>1.1913076</td>
<td>1.2417406</td>
</tr>
<tr>
<td>30</td>
<td>1.1191124</td>
<td>1.1156335</td>
<td>1.1534792</td>
</tr>
<tr>
<td>40</td>
<td>1.0895668</td>
<td>1.0807251</td>
<td>1.1218429</td>
</tr>
<tr>
<td>50</td>
<td>1.0703738</td>
<td>1.0925717</td>
<td>1.1087258</td>
</tr>
<tr>
<td>60</td>
<td>1.0741365</td>
<td>1.1003475</td>
<td>1.1007152</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Average task size</th>
<th>Uniform distribution</th>
<th>Binomial distribution</th>
<th>Geometric distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2.0610655</td>
<td>2.1835053</td>
<td>2.3687051</td>
</tr>
<tr>
<td>20</td>
<td>1.4136206</td>
<td>1.4257227</td>
<td>1.5356184</td>
</tr>
<tr>
<td>30</td>
<td>1.2505442</td>
<td>1.2452238</td>
<td>1.3269139</td>
</tr>
<tr>
<td>40</td>
<td>1.1857568</td>
<td>1.1840410</td>
<td>1.2604947</td>
</tr>
<tr>
<td>50</td>
<td>1.1600027</td>
<td>1.1920766</td>
<td>1.2270179</td>
</tr>
<tr>
<td>60</td>
<td>1.1511566</td>
<td>1.2114935</td>
<td>1.2078904</td>
</tr>
</tbody>
</table>

Table 3A
Simulation data for the performance bound (1) (IA(200), CI = ±1.54235%).

<table>
<thead>
<tr>
<th>Average task size</th>
<th>Uniform distribution</th>
<th>Binomial distribution</th>
<th>Geometric distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1.4734815</td>
<td>1.4865587</td>
<td>1.5310269</td>
</tr>
<tr>
<td>20</td>
<td>1.1279051</td>
<td>1.1283578</td>
<td>1.1886043</td>
</tr>
<tr>
<td>30</td>
<td>1.0722037</td>
<td>1.0680042</td>
<td>1.1152157</td>
</tr>
<tr>
<td>40</td>
<td>1.0566575</td>
<td>1.0535742</td>
<td>1.0898886</td>
</tr>
<tr>
<td>50</td>
<td>1.0570322</td>
<td>1.0718280</td>
<td>1.0789807</td>
</tr>
<tr>
<td>60</td>
<td>1.0607582</td>
<td>1.0756554</td>
<td>1.0744901</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Average task size</th>
<th>Uniform distribution</th>
<th>Binomial distribution</th>
<th>Geometric distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2.1903773</td>
<td>2.1833820</td>
<td>2.3235177</td>
</tr>
<tr>
<td>20</td>
<td>1.2741338</td>
<td>1.2678296</td>
<td>1.4177314</td>
</tr>
<tr>
<td>30</td>
<td>1.1478139</td>
<td>1.1395352</td>
<td>1.2425587</td>
</tr>
<tr>
<td>40</td>
<td>1.1172385</td>
<td>1.1079037</td>
<td>1.1842388</td>
</tr>
<tr>
<td>50</td>
<td>1.1169127</td>
<td>1.1487781</td>
<td>1.1638758</td>
</tr>
<tr>
<td>60</td>
<td>1.1265809</td>
<td>1.1585084</td>
<td>1.1536938</td>
</tr>
</tbody>
</table>

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.

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 = M1 + M2 + M3 + M4 + M5 = 960, the number of tasks that can be executed simultaneously is roughly M/\(\bar{\tau}\), which is 96, 48, 32, 24, 19, 16, for \(\bar{\tau} = 10, 20, 30, 40, 50, 60\) 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(200,0.001). 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(200,0.001).
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 $t_{j,1}, t_{j,2}, \ldots, t_{j,m}$ are the execution requirements of the $n_j$ tasks scheduled on $M_j$, where $1 \leq j \leq m$. Let $T_j = \{t_{j,1}, t_{j,2}, \ldots, t_{j,n_j}\}$ denote the length of the schedule produced by algorithm $A$ for scheduling the $n_j$ independent parallel tasks with execution times $t_{j,1}, t_{j,2}, \ldots, t_{j,m}$ on $M_j$. It is clear that if all tasks are executed with the same speed $s_j$ on $M_j$, we have

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

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

Let $W_j = \pi_1 t_{j,1} + \pi_2 t_{j,2} + \cdots + \pi_m t_{j,m}$ 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 $1 \leq j \leq m$.

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 $1 \leq j \leq m$. The schedule length is

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

The performance ratio is

$$\beta_A ES = \left(\frac{A_1^{\alpha-1} W_1 + A_2^{\alpha-1} W_2 + \cdots + A_m^{\alpha-1} W_m}{W}\right)^{1/(\alpha-1)} \frac{1}{W/M}.$$

**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

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

Since

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

we have

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

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

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

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

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

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 $1 \leq j \leq m$. By **Proposition 1**, the performance ratio of algorithm $A$-ES is

$$\beta_A ES = \frac{T_A ES}{T_{OPT}} \leq \left(\frac{(1/E)(A_1^{\alpha-1} W_1 + A_2^{\alpha-1} W_2 + \cdots + A_m^{\alpha-1} W_m))^{1/(\alpha-1)}}{(M/E)(W/M)^{\alpha})^{1/(\alpha-1)}}\right)^{1/(\alpha-1)}$$

$$= \left(\frac{A_1^{\alpha-1} W_1 + A_2^{\alpha-1} W_2 + \cdots + A_m^{\alpha-1} W_m}{W^{\alpha/M}}\right)^{1/(\alpha-1)} \frac{1}{W/M}.$$
Notice that \( A(r_1, r_2, \ldots, r_n) = \max\{A_1, A_2, \ldots, A_m\} \). Hence, we have
\[
\left( A_1^{a-1}W_1 + A_2^{a-1}W_2 + \cdots + A_m^{a-1}W_m \right) \leq A(r_1, r_2, \ldots, r_n).
\]
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
\[
E_j = \frac{A_j^{a-1}W_j}{T^{a-1}},
\]
for all \( 1 \leq j \leq m \). The total energy consumed is
\[
E_{A-ES} = \frac{1}{T^{a-1}} \sum_{j=1}^{m} A_j^{a-1}W_j.
\]

The performance ratio is
\[
\gamma_{A-ES} = \left( A_1^{a-1}W_1 + A_2^{a-1}W_2 + \cdots + A_m^{a-1}W_m \right) \frac{1}{(W/M)^{a-1}}. \tag{4}
\]

**Proof.** By using a post-power-determination algorithm A-ES to solve the time constrained scheduling problem, we need
\[
T_j = A_j \left( \frac{W_j}{E_j} \right)^{1/(a-1)} = T,
\]
that is,
\[
E_j = \frac{A_j^{a-1}W_j}{T^{a-1}},
\]
for all \( 1 \leq j \leq m \). In other words, we need to provide enough energy \( E_{A-ES} \), so that the deadline \( T \) is met, i.e.,
\[
T_{A-ES} = \left( \frac{1}{E_{A-ES}} \sum_{j=1}^{m} A_j^{a-1}W_j \right)^{1/(a-1)} = T.
\]
The above equation implies that the total energy \( E_{A-ES} \) consumed by algorithm A-ES is
\[
E_{A-ES} = \frac{1}{T^{a-1}} \sum_{j=1}^{m} A_j^{a-1}W_j.
\]
By Proposition 2, the performance ratio of algorithm A-ES is
\[
\gamma_{A-ES} = \frac{E_{A-ES}}{E_{OPT}} \leq \frac{1}{(T^{a-1}) \left( (M/T^{a-1}) (W/M)^{a-1} \right)} \leq \frac{1}{(W/M)^{a-1}}.
\]
This proves the theorem. ■

### Table 7A

<table>
<thead>
<tr>
<th>Average task size</th>
<th>Uniform distribution</th>
<th>Binomial distribution</th>
<th>Geometric distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1.1738552</td>
<td>1.1788602</td>
<td>1.1450335</td>
</tr>
<tr>
<td>20</td>
<td>1.08343691</td>
<td>1.0858256</td>
<td>1.0776934</td>
</tr>
<tr>
<td>30</td>
<td>1.0640175</td>
<td>1.0595913</td>
<td>1.0725996</td>
</tr>
<tr>
<td>40</td>
<td>1.0609201</td>
<td>1.0516637</td>
<td>1.0690041</td>
</tr>
<tr>
<td>50</td>
<td>1.0714196</td>
<td>1.0689405</td>
<td>1.0677635</td>
</tr>
<tr>
<td>60</td>
<td>1.0756109</td>
<td>1.0784248</td>
<td>1.0689723</td>
</tr>
</tbody>
</table>

### Table 7B

<table>
<thead>
<tr>
<th>Average task size</th>
<th>Uniform distribution</th>
<th>Binomial distribution</th>
<th>Geometric distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1.17357196</td>
<td>1.3953765</td>
<td>3.149472</td>
</tr>
<tr>
<td>20</td>
<td>1.1793046</td>
<td>1.771137</td>
<td>1.163004</td>
</tr>
<tr>
<td>30</td>
<td>1.1334744</td>
<td>1.1203476</td>
<td>1.1468589</td>
</tr>
<tr>
<td>40</td>
<td>1.1253395</td>
<td>1.1055719</td>
<td>1.1447847</td>
</tr>
<tr>
<td>50</td>
<td>1.1494851</td>
<td>1.1456666</td>
<td>1.1427224</td>
</tr>
<tr>
<td>60</td>
<td>1.1617398</td>
<td>1.1636299</td>
<td>1.1437753</td>
</tr>
</tbody>
</table>

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 a 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 \( E_k = W S_k^{a-1} \), 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
\[
s = \left( \frac{E}{W} \right)^{(a-1)}.
\]
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
\[
\left( \sum_{j \in I} w_j \right) S_k^{a-1} + \left( \sum_{i \notin I} w_i \right) S_{k+1}^{a-1} \leq E,
\]
and that the left-hand side of the above inequality is as large as possible. The last inequality implies that

$$\sum_{i \in \mathcal{I}} w_i \leq \frac{E - W S_k^{-1}}{\phi_k^{-1} \Delta - S_k^{-1}} = \frac{E - E_k}{\phi_k^{-1} \Delta - S_k^{-1}}.$$ 

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

$$\sum_{i \in \mathcal{I}} w_i \leq K,$$

where

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

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

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

where $t_i = n_i/S_k$ for $i \not\in I$, and $t_i = n_i/S_{k+1}$ for $i \in I$. By Proposition 1, the performance ratio of algorithm $ES-A$ is

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

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

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

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 $T_k = A(t_1, t_2, \ldots, t_n)/S_k$, where $1 \leq k \leq d$. Assume that $T_k > T \geq T_{k+1}$ and $S_k < s \leq S_{k+1}$, where $T$ is the time constraint, and

$$s = \frac{A(t_1, t_2, \ldots, t_n)}{T}$$

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 $0 \leq \Delta \leq 1 - 1/\phi_k$.

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

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

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

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

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

$$\gamma_{ES-A} = \frac{E_{ES-A}}{E_{OPT}} \leq \left(\frac{w_1 + w_2 + \cdots + w_b S_k^{-1} + (w_{b+1} + w_{b+2} + \cdots + w_n) S_{k+1}^{-1}}{(M/T)^{1-\alpha}(W/M)^{\alpha}}\right). \quad (6)$$

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

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

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.
The simulation data for the performance bound (5) (DAG(2000, 0.001), CI = ±0.1). Table 13A

Table 13B
Simulation data for the performance bound (6) (DAG(2000, 0.001), CI = ±0.13405%).

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

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

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

\[
    = \left( \frac{A_j}{\phi_k^{\alpha_k - 1}} W_1 \phi_1^{\alpha - 1} W_2 \phi_2^{\alpha - 1} \cdots W_m \phi_m^{\alpha - 1} \right)^{1/(\alpha - 1)},
\]

where \( 1 \leq j \leq m \). Let \( E_{j,k} = W_j S_k^{-1} \), where \( 1 \leq j \leq m \) and \( 1 \leq k \leq d \). Assume that \( E_{j,k} < E_{j,k+1} \) and \( S_k \leq s_{k+1} \), 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
\[
\phi^{k_j} \leq \left( \frac{E_j}{W_j} \right)^{1/(\alpha - 1)} < \phi^{k_j+1},
\]
which implies that
\[
k_j = \left[ \frac{1}{\alpha - 1} \log_\phi \frac{E_j}{W_j} \right],
\]
for all $1 \leq j \leq m$.

Let the tasks executed on $M_i$ be named as $(j, i)$, where $1 \leq i \leq n$. Our strategy is to find a set of indices $I_j = \{1, 2, \ldots, n\}$, such that tasks $(j, i), i \in I_j$, are executed with speed $S_{k_j}$ and tasks $(j, i), i \notin I_j$, are executed with speed $S_k$, and that
\[
\left( \sum_{i \in I_j} w_{j,i} \right) S_{k_j}^{\alpha - 1} + \left( \sum_{i \notin I_j} w_{j,i} \right) S_{k_j}^{\alpha - 1} \leq E_j,
\]
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 $0 \leq \Delta_j < \phi^{\alpha - 1} - 1$, we get
\[
\sum_{i \in I_j} w_{j,i} \leq K_j,
\]
where
\[
K_j = \left( \frac{\Delta_j}{\phi^{\alpha - 1} - 1} \right) W_j.
\]

It is clear that we have
\[
T_{AES} = \max(T_1, T_2, \ldots, T_m) = \max_{1 \leq j \leq m} \left( A_j(t_{j,1}, t_{j,2}, \ldots, t_{j,n}) \right),
\]
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 $T_j = A_j(t_{j,1}, t_{j,2}, \ldots, t_{j,n})$ denotes the schedule length produced by algorithm $A$ for tasks with execution times $t_{j,1}, t_{j,2}, \ldots, t_{j,n}$ on $M_j$. By Proposition 1, the performance ratio of algorithm $A$-ES is
\[
\beta_{AES} = \frac{T_{AES}}{T_{OPT}} \leq \max_{1 \leq j \leq m} \left( \frac{A_j(t_{j,1}, t_{j,2}, \ldots, t_{j,n})}{(M/E)(W/M)^{\alpha}} \right).
\]

The above discussion can be summarized in the following theorem.

Theorem 7. By using a post-power-determination algorithm $A$-ES to solve the time constrained scheduling problem, the performance ratio is
\[
\beta_{AES} \leq \max_{1 \leq j \leq m} \left( \frac{A_j(t_{j,1}, t_{j,2}, \ldots, t_{j,n})}{(M/E)(W/M)^{\alpha}} \right).
\]

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
\[
s_j = \left( \frac{E_j}{W_j} \right)^{1/(\alpha - 1)} = \frac{A_j(t_{j,1}, t_{j,2}, \ldots, t_{j,n})}{T}.
\]

where $1 \leq j \leq m$. Let $T_{j,k} = A_j/S_k = A_j(r_{j,1}, r_{j,2}, \ldots, r_{j,n})/S_k$, where $1 \leq j \leq m$ and $1 \leq k \leq d$. Assume that $T_{j,k_i} > T \geq T_{j,k_{i+1}}$ and $S_{k_i} < S_j \leq S_{k_{i+1}}$. If $S_k = \phi^k$ for all $k$, then we have
\[
\phi^{k_j} < \frac{A_j}{T} \leq \phi^{k_{j+1}},
\]
which implies that
\[
k_j = \left[ \log_\phi \frac{A_j}{T} \right] - 1,
\]
for all $1 \leq j \leq m$.

Our strategy is to arrange the tasks executed on $M_j$ in certain order, say, $(j, 1), (j, 2), \ldots, (j, n)$, and then find an index $b_j$, such that tasks $(j, 1), (j, 2), \ldots, (j, b_j)$ are executed with speed $S_k$, and tasks $(j, b_j + 1), (j, b_j + 2), \ldots, (j, n)$ are executed with speed $S_{k_{j+1}}$, and that
\[
T_j = A_j(t_{j,1}, t_{j,2}, \ldots, t_{j,n}) \leq T,
\]
where $t_{j,i} = r_{j,i}/S_k$ for $1 \leq i \leq b_j$, and $t_{j,i} = r_{j,i}/S_{k_{j+1}}$ for $b_j + 1 \leq i \leq n$, and $b_j$ is as large as possible. Then, we have
\[
E_{A-ES} = \sum_{j=1}^{m} \left( \left( \sum_{i=1}^{b_j} w_{j,i} \right) S_{k_j}^{\alpha - 1} + \left( \sum_{i=b_j+1}^{n} w_{j,i} \right) S_{k_{j+1}}^{\alpha - 1} \right).
\]

By Proposition 2, the performance ratio of algorithm $A$-ES is
\[
\gamma_{AES} = \frac{E_{AES}}{E_{OPT}} \leq \frac{1}{(M/T^{\alpha - 1})(W/M)^{\alpha}} \times \sum_{j=1}^{m} \left( \left( \sum_{i=1}^{b_j} w_{j,i} \right) S_{k_j}^{\alpha - 1} + \left( \sum_{i=b_j+1}^{n} w_{j,i} \right) S_{k_{j+1}}^{\alpha - 1} \right).
\]

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
\[
\gamma_{AES} \leq \frac{1}{(M/T^{\alpha - 1})(W/M)^{\alpha}} \times \sum_{j=1}^{m} \left( \left( \sum_{i=1}^{b_j} w_{j,i} \right) S_{k_j}^{\alpha - 1} + \left( \sum_{i=b_j+1}^{n} w_{j,i} \right) S_{k_{j+1}}^{\alpha - 1} \right).
\]
Table 14A
Simulation data for the performance bound (7) (100, CI = ±3.79952%).

<table>
<thead>
<tr>
<th>Average task size</th>
<th>Uniform distribution</th>
<th>Binomial distribution</th>
<th>Geometric distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1.6658728</td>
<td>1.6558267</td>
<td>1.5970324</td>
</tr>
<tr>
<td>20</td>
<td>1.4908681</td>
<td>1.4966154</td>
<td>1.4934169</td>
</tr>
<tr>
<td>30</td>
<td>1.4908525</td>
<td>1.4362307</td>
<td>1.4830170</td>
</tr>
<tr>
<td>40</td>
<td>1.4941999</td>
<td>1.4002700</td>
<td>1.476269</td>
</tr>
<tr>
<td>50</td>
<td>1.4620843</td>
<td>1.4290061</td>
<td>1.5039383</td>
</tr>
<tr>
<td>60</td>
<td>1.4796898</td>
<td>1.4449650</td>
<td>1.5159271</td>
</tr>
</tbody>
</table>

Table 14B
Simulation data for the performance bound (8) (100, CI = ±1.88312%).

<table>
<thead>
<tr>
<th>Average task size</th>
<th>Uniform distribution</th>
<th>Binomial distribution</th>
<th>Geometric distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1.673675</td>
<td>1.6489660</td>
<td>1.6107650</td>
</tr>
<tr>
<td>20</td>
<td>1.487165</td>
<td>1.4374588</td>
<td>1.4599286</td>
</tr>
<tr>
<td>30</td>
<td>1.455915</td>
<td>1.4097659</td>
<td>1.4603438</td>
</tr>
<tr>
<td>40</td>
<td>1.420428</td>
<td>1.3921292</td>
<td>1.4203996</td>
</tr>
<tr>
<td>50</td>
<td>1.461258</td>
<td>1.4371667</td>
<td>1.464019</td>
</tr>
<tr>
<td>60</td>
<td>1.4704655</td>
<td>1.4951573</td>
<td>1.4405544</td>
</tr>
</tbody>
</table>

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., $S_0$ and $S_{n+1}$ in Theorem 6), while the post-power-determination algorithm allows 2m speeds (i.e., $S_0$ and $S_{n+1}$, 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