Energy-Efficient and High-Performance Processing of Large-Scale Parallel Applications in Data Centers

Keqin Li*
Department of Computer Science
State University of New York
New Paltz, New York 12561, USA
Email: lik@newpaltz.edu

Abstract – *When a multicore processor in a data center for cloud computing is shared by a large number of parallel tasks of a large-scale parallel application simultaneously, we are facing the problem of allocating the cores to the tasks and schedule the tasks, such that the system performance is optimized or the energy consumption is minimized. The motivation of the present paper is to investigate energy-efficient and high-performance processing of large-scale parallel applications on multicore processors in data centers. In particular, we address scheduling precedence constrained parallel tasks on multicore processors with dynamically variable voltage and speed as combinatorial optimization problems. We point out that our scheduling problems contain four nontrivial subproblems, namely, precedence constraining, system partitioning, task scheduling, and power supplying. We describe our methods to deal with precedence constraints, system partitioning, and task scheduling. We develop our optimal four-level energy/time/power allocation scheme for minimizing schedule length and minimizing energy consumption, analyze the performance of our heuristic algorithms, and derive accurate performance bounds. We demonstrate simulation data, which validate our analytical results.*

Keywords: Data center, energy-efficient scheduling, high-performance computing, multicore processor, parallel task, performance analysis, precedence constraint.

*The author can be reached at phone: (845) 257-3534, fax: (845) 257-3996.

Contents

1 Introduction

1.1 Motivation

Next generation supercomputers require drastically better energy efficiency to allow these systems to scale to exaflop computing levels. Virtually all major processor vendors and companies such as AMD, Intel, and IBM are developing high-performance and highly energy-efficient multicore processors and dedicating their current and future development and manufacturing to multicore products. It is conceivable that future multicore architectures can hold dozens or even hundreds of cores on a single die [3]. For instance, Adapteva's Epiphany scalable manycore architecture consists of hundreds and thousands of RISC microprocessors, all sharing a single flat and unobstructed memory hierarchy, which allows cores to communicate with each other very efficiently with low core-to-core communication overhead. The number of cores in this new type of massively parallel multicore architecture can be up to 4096 [1]. The Epiphany manycore architecture has been designed to maximize floating point computing power with the lowest possible energy consumption, aiming to deliver 100 and more gigaflops of performance at under 2 watts of power [4].

Multicore processors provide an ultimate solution to power management and performance optimization in current and future high-performance computing. A multicore processor contains multiple independent processors, called cores, integrated onto a single circuit die (known an a chip multiprocessor or CMP). An m-core processor achieves the same performance of a single-core processor whose clock frequency is m times faster, but consumes only $1/m^{\phi-1}$ ($\phi \geq 3$) of the energy of the single-core processor. The performance gain from a multicore processor is mainly from parallelism, i.e., multiple cores' working together to achieve the performance of a single faster and more energy-consuming processor. A multicore processor implements multiprocessing in a single physical package. It can implement parallel architectures such as superscalar, multithreading, VLIW, vector processing, SIMD, and MIMD. Intercore communications are supported by message passing or shared memory. The degree of parallelism can increase together with the number m of cores. When m is large, a multicore processor is also called a manycore or a massively multicore processor.

Modern information technology is developed into the era of cloud computing, which has received considerable attention in recent years and is widely accepted as a promising and ultimate way of managing and improving the utilization of data and computing resources and delivering various computing and communication services. However, enterprise data centers will spend several times as much on energy costs as on hardware and server management and administrative costs. Furthermore, many data centers are realizing that even if they are willing to pay for more power consumption, capacity constraints on the electricity grid mean that additional power is unavailable. Energy efficiency is one of the most important issues for large-scale computing systems in current and future data centers. Cloud computing can be an inherently energy-efficient technology, due to centralized energy management of computations on large-scale computing systems, instead of distributed and individualized applications without efficient energy consumption control [10]. Moreover, such potential for significant energy savings can be fully explored with balanced consideration of system performance and energy consumption.

As in all computing systems, increasing the utilization of a multicore processor becomes a critical issue, as the number of cores increases and as multicore processors are more and more widely employed in data centers. One effective way of increasing the utilization is to take the approach of multitasking, i.e., allowing multiple tasks to be executed simultaneously in a multicore processor. Such sharing of computing resources not only improves system utilization, but also improves system performance, because more users' requests can be processed in the same among of time. Such performance enhancement is very important in optimizing the quality of service in a data center for cloud computing, where multicore processors are employed as servers. Partitioning and sharing of a large multicore processor among multiple tasks is particularly important for large-scale scientific computations and business applications, where each computation or application consists of a large number of parallel tasks, and each parallel task requires several cores simultaneously for its execution.

When a multicore processor in a data center for cloud computing is shared by a large number of parallel tasks of a large-scale parallel application simultaneously, we are facing the problem of allocating the cores to the tasks and schedule 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. Such optimization problems need to be formulated and efficient algorithms need to be developed and their performance need to be analyzed and evaluated. The motivation of the present paper is to investigate energy-efficient and high-performance processing of large-scale parallel applications on multicore processors in data centers. In particular, we study low-power scheduling of precedence constrained parallel tasks on multicore processors. Our approach is to define combinatorial optimization problems, develop heuristic algorithms, analyze their performance, and validate our analytical results by simulations.

1.2 Our Contributions

In this paper, we address scheduling precedence constrained parallel tasks on multicore processors with dynamically variable voltage and speed as combinatorial optimization problems. In particular, we define the problem of minimizing schedule length with energy consumption constraint and the problem of minimizing energy consumption with schedule length constraint on multicore processors. Our scheduling problems are defined in such a way that the energy-delay product is optimized by fixing one factor and minimizing the other. The first problem emphasizes energy efficiency, while the second problem emphasizes high performance.

We notice that energy-efficient and high-performance scheduling of parallel tasks with precedence constraints has not been investigated before as combinatorial optimization problems. Furthermore, all existing studies are on scheduling sequential tasks which require one processor to execute, or independent tasks which have no precedence constraint. Our study in this paper makes some initial attempt to energy-efficient and high-performance scheduling of parallel tasks with precedence constraints on multicore processors with dynamic voltage and speed.

Our scheduling problems contain four nontrivial subproblems, namely, precedence constraining, system partitioning, task scheduling, and power supplying. Each subproblem should be solved efficiently, so that heuristic algorithms with overall good performance can be developed. These subproblems and our strategies to solve them are described as follows.

- *Precedence Constraining* – Precedence constraints make design and analysis of heuristic algorithms more difficult. We propose to use level-by-level scheduling algorithms to deal with precedence constraints. Since tasks in the same level are independent of each other, they can be scheduled by any of the efficient algorithms previously developed for scheduling independent tasks. Such decomposition of scheduling precedence constrained tasks into scheduling levels of independent tasks makes analysis of level-by-level scheduling algorithms much easier and clearer than analysis of other algorithms.

- *System Partitioning* – Since each parallel task requests for multiple cores for its execution, a multicore processor should be partitioned into clusters of cores to be assigned to the tasks. We use the harmonic system partitioning and core allocation scheme, which divides a multicore processor into clusters of equal sizes and schedules tasks of similar sizes together to increase core utilization.

- *Task Scheduling* – Parallel tasks are scheduled together with system partitioning and precedence constraining, and it is NP-hard even scheduling independent sequential tasks without system partitioning and precedence constraint. Our approach is to divide a list (i.e., a level) of tasks into sublists, such that each sublist contains tasks of similar sizes which are scheduled on clusters of equal sizes. Scheduling such parallel tasks on clusters is no more difficult than scheduling sequential tasks and can be performed by list scheduling algorithms.

- *Power Supplying* – Tasks should be supplied with appropriate powers and execution speeds, such that the schedule length is minimized by consuming given amount of energy or the energy consumed is minimized without missing a given deadline. We adopt a four-level energy/time/power allocation scheme for a given schedule, namely, optimal energy/time allocation among levels of tasks (Theorems 6 and 10), optimal energy/time allocation among sublists of tasks in the same level (Theorems 5 and 9), optimal energy allocation among groups of tasks in the same sublist (Theorems 4 and 8), and optimal power supplies to tasks in the same group (Theorems 3 and 7).

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

The remainder of the paper is organized as follows. In Section 2, we review related research in the literature. In Section 3, we present background information, including the power and task models, definitions of our problems, and lower bounds for optimal solutions. In Section 4, we describe our methods to deal with precedence constraints, system partitioning, and task scheduling. In Section 5, we develop our optimal four-level energy/time/power allocation scheme for minimizing schedule length and minimizing energy consumption, analyze the performance of our heuristic algorithms, and derive accurate performance bounds. In Section 6, we demonstrate simulation data, which validate our analytical results. In Section 7, we summarize the paper and give further research directions.

2 Related Work

Increased energy consumption causes severe economic, ecological, and technical problems. 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 [15, 16, 17]. There exists an explosive body of literature on power-aware computing and communication. The reader is referred to [5, 9, 45, 46] for comprehensive surveys.

Software techniques for power reduction are supported by a mechanism called *dynamic voltage scaling* [2]. 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 [44]. In a pioneering paper [47], the authors first proposed the approach to energy saving by using fine grain control of CPU speed by an operating system scheduler. In a subsequent work [49], the authors analyzed offline and online algorithms for scheduling tasks with arrival times and deadlines on a uniprocessor computer with minimum energy consumption. These research have been extended in [7, 12, 25, 33, 34, 35, 50] and inspired substantial further investigation, much of which focus on real-time applications. In [6, 20, 21, 24, 27, 36, 37, 38, 39, 40, 42, 43, 48, 52, 53, 54, 55] and many other related work, the authors addressed the problem of scheduling independent or precedence constrained tasks on uniprocessor or multiprocessor computers where the actual execution time of a task may be less than the estimated worst-case execution time. The main issue is energy reduction by slack time reclamation.

There are two considerations in dealing with the energy-delay tradeoff. On the one hand, in high-performance computing systems, power-aware design techniques and algorithms attempt to maximize performance under certain energy consumption constraints. On the other hand, low-power and energy-efficient design techniques and algorithms aim to minimize energy consumption while still meeting certain performance goals. In [8], the author studied the problems of minimizing the expected execution time given a hard energy budget and minimizing the expected energy expenditure given a hard execution deadline for a single task with randomized execution requirement. In [11], the author considered scheduling jobs with equal requirements on multiprocessors. In [14], the authors studied the relationship among parallelization, performance, and energy consumption, and the problem of minimizing energy-delay product. In [18], the authors addressed joint minimization of carbon emission and maximization of profit. In [23, 26], the authors attempted joint minimization of energy consumption and task execution time. In [41], the authors investigated the problem of system value maximization subject to both time and energy constraints. In [56], the authors considered task scheduling on clusters with significant communication costs.

In [28, 29, 30, 31, 32], 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. In [28, 31], we studied the problems of scheduling independent sequential tasks. In [29, 32], we studied the problems of scheduling independent parallel tasks. In [30], we studied the problems of scheduling precedence constrained sequential tasks. In this paper, we study the problems of scheduling precedence constrained parallel tasks.

3 Preliminaries

In this section, we present background information, including the power and task models, definitions of our problems, and lower bounds for optimal solutions.

3.1 Power and Task Models

Power dissipation and circuit delay in digital CMOS circuits can be accurately modeled by simple equations, even for complex microprocessor circuits. CMOS circuits have dynamic, static, and short-circuit power dissipation; however, the dominant component in a well designed circuit is dynamic power consumption p (i.e., the switching component of power), which is approximately $p = aCV^2 f$, where a is an activity factor, C is the loading capacitance, V is the supply voltage, and f is the clock frequency [13]. In the ideal case, the supply voltage and the clock frequency are related in such a way that $V \propto f^{\phi}$ for some constant $\phi > 0$ [51]. The processor execution speed s is usually linearly proportional to the clock frequency, namely, $s \propto f$. For ease of discussion, we will assume that $V = bf^{\phi}$ and $s = cf$, where b and c are some constants. Hence, we know that power consumption is $p = aCV^2 f = ab^2Cf^{2\phi+1} = (ab^2C/c^{2\phi+1})s^{2\phi+1} = \xi s^{\alpha}$, where $\xi = ab^2C/c^{2\phi+1}$ and $\alpha = 2\phi + 1$. For instance, by setting $b = 1.16$, $aC = 7.0$, $c = 1.0$, $\phi = 0.5$, $\alpha = 2\phi + 1 = 2.0$, and $\xi = ab^2C/c^{\alpha} = 9.4192$, the value of p calculated by the equation $p = aCV^2 f = \xi s^{\alpha}$ is reasonably close to that in [22] for the Intel Pentium M processor.

Assume that we are given a parallel computation or application with a set of n precedence constrained parallel tasks. The precedence constraints can be specified as a partial order $\prec$ over the set of tasks $\{1, 2, ..., n\}$, or a task graph $G = (V, E)$, where $V = \{1, 2, ..., n\}$ 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. A parallel task i, where $1 \le i \le n$, 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.

We are also given a multicore processor with m homogeneous and identical cores. To execute a task i, any π_i of the m cores of the multicore processor can be allocated to task i. Several tasks can be executed simultaneously on the multicore processor, with the restriction that the total number

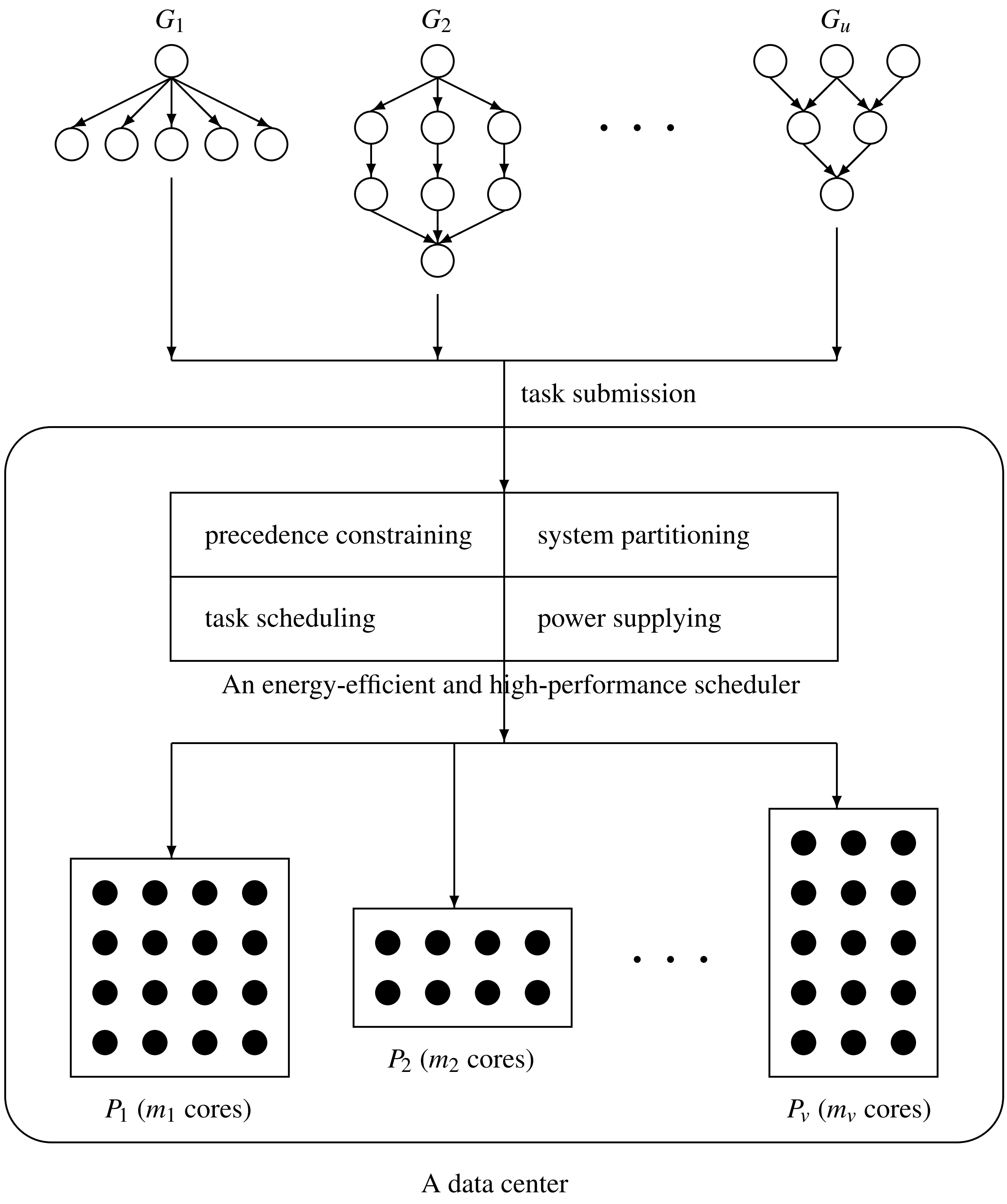

Figure 1: Processing of parallel applications in a data center.

of active cores (i.e., cores allocated to tasks being executed) at any moment cannot exceed m.

In a more general setting, we can consider scheduling u parallel applications represented by task graphs $G_1, G_2, ..., G_u$ respectively, on v multicore processors $P_1, P_2, ..., P_v$ in a data center with $m_1, m_2, ..., m_v$ cores respectively (see Figure 1). Notice that multiple task graphs can be viewed as a single task graph with disconnected components. Therefore, our task model can accommodate multiple parallel applications. However, scheduling on multiple multicore processors is significantly different from scheduling on a single multicore processor. In this paper, we focus on scheduling parallel applications on a single multicore processor, and leave the study of scheduling parallel applications on multiple multicore processors as a further research topic.

We use p_i to represent the power supplied to task i and s_i to represent the speed to execute task i. It is noticed that the constant ξ in $p_i = \xi s_i^{\alpha}$ only linearly scales the value of p_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.

3.2 Problems

Our combinatorial optimization problems to be solved in this paper are formally defined as follows.

Given n parallel tasks with precedence constraints $\prec$, task sizes π_1, π_2, ..., π_n, and task execution requirements r_1, r_2, ..., r_n, the problem of *minimizing schedule length with energy consumption constraint* E on an m-core processor is to find the power supplies p_1, p_2, ..., p_n (equivalently, the task execution speeds s_1, s_2, ..., s_n) and a nonpreemptive schedule of the n tasks on the m-core processor, such that the schedule length is minimized and that the total energy consumed does not exceed E. This problem aims at achieving energy-efficient processing of large-scale parallel applications with the best possible performance.

Given n parallel tasks with precedence constraints $\prec$, task sizes π_1, π_2, ..., π_n, and task execution requirements r_1, r_2, ..., r_n, the problem of *minimizing energy consumption with schedule length constraint* T on an m-core processor is to find the power supplies p_1, p_2, ..., p_n (equivalently, the task execution speeds s_1, s_2, ..., s_n) and a nonpreemptive schedule of the n tasks on the m-core processor, such that the total energy consumption is minimized and that the schedule length does not exceed T. This problem aims at achieving high-performance processing of large-scale parallel applications with the lowest possible energy consumption.

The above two problems are NP-hard even when the tasks are independent (i.e., $\prec = \emptyset$) and sequential (i.e., $\pi_i = 1$ for all $1 \le i \le n$) [28]. Thus, we will seek fast heuristic algorithms with near-optimal performance.

3.3 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. We define T^* to be the length of an optimal schedule, and E^* to

Table 1. *Summary of Our Methods to Solve the Subproblems.*

Subproblem	Method
precedence constraining	level-by-level scheduling algorithms
system partitioning	harmonic system partitioning and core allocation scheme
task scheduling	list scheduling algorithms
power supplying	four-level energy/time/power allocation scheme

be the minimum amount of energy consumed by an optimal schedule.

The following theorem gives a lower bound for the optimal schedule length T^* for the problem of minimizing schedule length with energy consumption constraint.

Theorem 1 *For the problem of minimizing schedule length with energy consumption constraint in scheduling parallel tasks, we have the following lower bound,*

$$T^* \ge \left(\frac{m}{E} \left(\frac{W}{m} \right)^{\alpha} \right)^{1/(\alpha-1)}$$

for the optimal schedule length.

The following theorem gives a lower bound for the minimum energy consumption E^* for the problem of minimizing energy consumption with schedule length constraint.

Theorem 2 *For the problem of minimizing energy consumption with schedule length constraint in scheduling parallel tasks, we have the following lower bound,*

$$E^* \ge m \left(\frac{W}{m} \right)^{\alpha} \frac{1}{T^{\alpha-1}}$$

for the minimum energy consumption.

The above lower bound theorems were proved for independent parallel tasks [29], and therefore, are also applicable to precedence constrained parallel tasks. The significance of these lower bounds is that they can be used to evaluate the performance of heuristic algorithms when their solutions are compared with optimal solutions (see Subsections 5.1.4 and 5.2.4).

4 Heuristic Algorithms

In this section, we describe our methods to deal with precedence constraints, system partitioning, and task scheduling, i.e., our methods to solve the first three subproblems. Table 1 gives a summary of our strategies to solve the subproblems.

4.1 Precedence Constraining

Recall that a set of n parallel tasks with precedence constraints can be represented by a partial order $\prec$ on the tasks, i.e., for two tasks i and j, if $i \prec j$, then task j cannot start its execution until task i finishes. It is clear that the n tasks and the partial order $\prec$ can be represented by a directed task graph, in which, there are n vertices for the n tasks and (i, j) is an arc if and only if $i \prec j$. We call j a successor of i and i a predecessor of j. Furthermore, such a task graph must be a *directed acyclic graph* (dag). An arc (i, j) is redundant if there exists k such that (i,k) and (k, j) are also arcs in the task graph. We assume that there is no redundant arc in the task graph.

A dag can be decomposed into levels, with v being the number of levels. Tasks with no predecessors (called initial tasks) constitute level 1. Generally, a task i is in level l if the number of nodes on the longest path from some initial task to task i is l, where $1 \le l \le v$. Note that all tasks in the same level are independent of each other, and hence, they can be scheduled by any of the algorithms (e.g., those from [29, 32]) for scheduling independent parallel tasks. Algorithm LL-H_c-A, where A is a list scheduling algorithm, standing for *level-by-level* scheduling with algorithm H_c-A, schedules the n tasks level by level in the order level 1, level 2, ..., level v. Tasks in level $l+1$ cannot start their execution until all tasks in level l are completed. For each level l, where $1 \le l \le v$, we use algorithm H_c-A developed in [29] to generate its schedule (see Figure 2).

The details of algorithm H_c-A is given in the next two subsections.

4.2 System Partitioning

Our algorithms for scheduling independent parallel tasks are called H_c-A, where “H_c" stands for the *harmonic* system partitioning scheme with parameter c to be presented below, and A is a list scheduling algorithm to be presented in the next subsection.

To schedule a list of independent parallel tasks in level l, algorithm H_c-A divides the list into c sublists $(l,1),(l,2),...,(l,c)$ according to task sizes (i.e., numbers of cores requested by tasks), where $c \ge 1$ is a positive integer constant. For $1 \le j \le c-1$, we define sublist (l, j) to be the sublist of tasks with

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

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

Algorithm H_c-A produces schedules of the sublists sequentially and separately (see Figure 2). To schedule tasks in sublist (l, j), where $1 \le j \le c$, the m cores are partitioned into j *clusters* and each cluster contains m/j cores. Each cluster of cores is treated as one unit to be allocated to one task in sublist (l, j). This is basically the harmonic system partitioning and core allocation scheme.

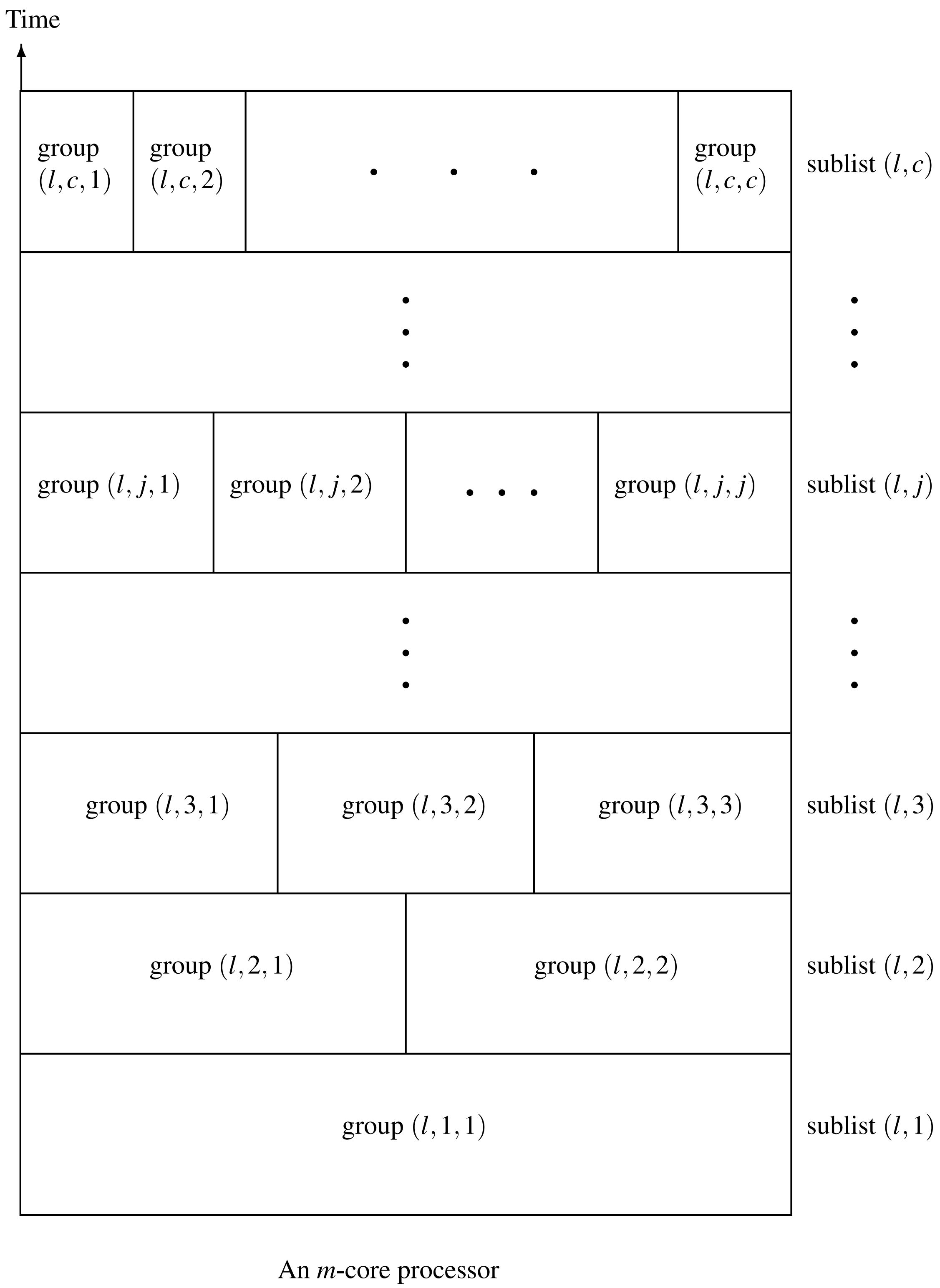

Figure 2: Scheduling of level l.

The justification of the scheme is from the observation that there can be at most j parallel tasks from sublist (l, j) to be executed simultaneously. Therefore, scheduling parallel tasks in sublist (l, j) on the j clusters, where each task i has core requirement π_i and execution requirement r_i, is equivalent to scheduling a list of sequential tasks on j processors where each task i has execution requirement r_i. It is clear that scheduling of a list of sequential tasks on j processors (i.e., scheduling of a sublist (l, j) of parallel tasks on j clusters) can be accomplished by using algorithm A, where A is a list scheduling algorithm to be elaborated in the next subsection.

4.3 Task Scheduling

When a multicore processor with m cores is partitioned into $j \geq 1$ clusters, scheduling tasks in sublist (l, j) is essentially dividing sublist (l, j) into j groups $(l, j, 1), (l, j, 2), ..., (l, j, j)$ of tasks, such that each group of tasks are executed on one cluster (see Figure 2). Such a partition of sublist (l, j) into j groups is essentially a schedule of the tasks in sublist (l, j) on m cores with j clusters. Once a partition (i.e., a schedule) is determined, we can use the methods in the next section to find optimal energy/time allocation and power supplies.

We propose to use the list scheduling algorithm and its variations to solve the task scheduling problem. Tasks in sublist (l, j) are scheduled on j clusters by using the classic *list scheduling* algorithm [19] and by ignoring the issue of power supplies and execution speeds. In other words, the task execution times are simply the task execution requirements r_1, r_2, ..., r_n, and tasks are assigned to the j clusters (i.e., groups) by using the list scheduling algorithm, which works as follows to schedule a list of tasks 1, 2, 3 ....

- List Scheduling (LS): Initially, task k is scheduled on cluster (or group) k, where $1 \leq k \leq j$, and tasks 1, 2, ..., j are removed from the list. Upon the completion of a task k, the first unscheduled task in the list, i.e., task $j + 1$, is removed from the list and scheduled to be executed on cluster k. This process repeats until all tasks in the list are finished.

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

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

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

- Largest Size First (LSF): This algorithm is the same as the LS algorithm, except that the tasks are arranged such that $\pi_1 \geq \pi_2 \geq \cdots \geq \pi_n$.

- Smallest Size First (SSF): This algorithm is the same as the LS algorithm, except that the tasks are arranged such that $\pi_1 \leq \pi_2 \leq \cdots \leq \pi_n$.

- Largest Task First (LTF): This algorithm is the same as the LS algorithm, except that the tasks are arranged such that $\pi_1^{1/\alpha} r_1 \geq \pi_2^{1/\alpha} r_2 \geq \cdots \geq \pi_n^{1/\alpha} r_n$.

Table 2. *Overview of the Optimal Energy/Time/Power Allocation Scheme.*

Level	Method	Theorems
1	optimal power supplies to tasks in the same group	3 and 7
2	optimal energy allocation among groups of tasks in the same sublist	4 and 8
3	optimal energy/time allocation among sublists of tasks in the same level	5 and 9
4	optimal energy/time allocation among levels of tasks	6 and 10

- Smallest Task First (STF): This algorithm is the same as the LS algorithm, except that the tasks are arranged such that $\pi_1^{1/\alpha} r_1 \le \pi_2^{1/\alpha} r_2 \le \cdots \le \pi_n^{1/\alpha} r_n$.

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

5 Optimal Energy/Time/Power Allocation

In this section, we develop our optimal four-level energy/time/power allocation scheme for minimizing schedule length and minimizing energy consumption, i.e., our method to solve the last subproblem. We also analyze the performance of our heuristic algorithms and derive accurate performance bounds.

Once the n precedence constrained parallel tasks are decomposed into v levels, 1, 2, ..., v, and tasks in each level l are divided into c sublists $(l,1),(l,2),...,(l,c)$, and tasks in each sublist (l,j) are further partitioned into j groups $(l,j,1),(l,j,2),...,(l,j,j)$, power supplies to the tasks which minimize the schedule length within energy consumption constraint or the energy consumption within schedule length constraint can be determined. We adopt a four-level energy/time/power allocation scheme for a given schedule, namely,

- Level 1 – optimal power supplies to tasks in the same group (l,j,k) (Theorems 3 and 7);
- Level 2 – optimal energy allocation among groups $(l,j,1),(l,j,2),...,(l,j,j)$ of tasks in the same sublist (l,j) (Theorems 4 and 8);
- Level 3 – optimal energy/time allocation among sublists $(l,1),(l,2),...,(l,c)$ of tasks in the same level l (Theorems 5 and 9);
- Level 4 – optimal energy/time allocation among levels 1, 2, ..., l of tasks of a parallel application (Theorems 6 and 10).

Table 2 gives an overview of our energy/time/power allocation scheme. We will give the details of the above optimal four-level energy/time/power allocation scheme for the two optimization problems separately.

5.1 Minimizing Schedule Length

5.1.1 Level 1

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

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

The following result [29] gives the optimal power supplies when the n parallel tasks are scheduled sequentially.

Theorem 3 *When n parallel tasks are scheduled sequentially, the schedule length is minimized when task i is supplied with power* $p_i = (E/M)^{\alpha/(\alpha-1)}/\pi_i$, *where* $1 \le i \le n$. *The optimal schedule length is* $T = M^{\alpha/(\alpha-1)}/E^{1/(\alpha-1)}$.

5.1.2 Level 2

Now, we consider optimal energy allocation among groups of tasks in the same sublist. Again, we discuss group level energy allocation in a more general case, i.e., scheduling n parallel tasks on m cores, where $\pi_i \le m/j$ for all $1 \le i \le n$ with $j \ge 1$. In this case, the m cores can be partitioned into j clusters, such that each cluster contains m/j cores. Each cluster of cores are treated as one unit to be allocated to one task. Assume that the set of n tasks is partitioned into j groups, such that all the tasks in group k are executed on cluster k, where $1 \le k \le j$. Let M_k denote the total $\pi_i^{1/\alpha} r_i$ of the tasks in group k. For a given partition of the n tasks into j groups, we are seeking an optimal energy allocation and power supplies that minimize the schedule length. Let E_k be the energy consumed by all the tasks in group k. The following result [29] characterizes the optimal energy allocation and power supplies.

Theorem 4 *For a given partition* M_1, M_2, ..., M_j *of n parallel tasks into j groups on a multicore processor partitioned into j clusters, the schedule length is minimized when task i in group k is supplied with power* $p_i = (E_k/M_k)^{\alpha/(\alpha-1)}/\pi_i$, *where*

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

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

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

for the above energy allocation and power supplies.

5.1.3 Level 3

To use algorithm H$_c$-A to solve the problem of minimizing schedule length with energy consumption constraint E, we need to allocate the available energy E to the c sublists. We use E_1, E_2, ..., E_c to represent an energy allocation to the c sublists, where sublist j consumes energy E_j, and $E_1+E_2+\cdots+E_c=E$. By using any of the list scheduling algorithms to schedule tasks in sublist j, we get a partition of the tasks in sublist j into j groups. Let R_j be the total execution requirement of tasks in sublist j, and $R_{j,k}$ be the total execution requirement of tasks in group k, and $M_{j,k}$ be the total $\pi_i^{1/\alpha}r_i$ of tasks in group k, where $1\le k\le j$. Theorem 5 [29] provides optimal energy allocation to the c sublists for minimizing schedule length with energy consumption constraint in scheduling parallel tasks by using scheduling algorithms H$_c$-A, where A is a list scheduling algorithm.

Theorem 5 *For a given partition $M_{j,1}$, $M_{j,2}$, ..., $M_{j,j}$ of the tasks in sublist j into j groups produced by a list scheduling algorithm A, where $1\le j\le c$, and an energy allocation E_1, E_2, ..., E_c to the c sublists, the length of the schedule produced by algorithm H_c-A is*

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

The energy allocation E_1, E_2, ..., E_c which minimizes T is

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

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

$$T=\frac{(N_1^{1/\alpha}+N_2^{1/\alpha}+\cdots+N_c^{1/\alpha})^{\alpha/(\alpha-1)}}{E^{1/(\alpha-1)}},$$

by using the above energy allocation.

5.1.4 Level 4

To use a level-by-level scheduling algorithm to solve the problem of minimizing schedule length with energy consumption constraint E, we need to allocate the available energy E to the v levels. We use E_1, E_2, ..., E_v to represent an energy allocation to the v levels, where level l consumes energy E_l, and $E_1+E_2+\cdots+E_v=E$.

Let $R_{l,j,k}$ be the total execution requirement of tasks in group (l,j,k), i.e., group k of sublist (l,j) of level l, and $R_{l,j}$ be the total execution requirement of tasks in sublist (l,j) of level l, and R_j be the total execution requirement of tasks in sublist (l,j) of all levels, and $M_{l,j,k}$ be the total $\pi_i^{1/\alpha}r_i$ of tasks in group (l,j,k), where $1\le l\le v$ and $1\le j\le c$ and $1\le k\le j$.

By Theorem 5, for a given partition $M_{l,j,1}$, $M_{l,j,2}$, ..., $M_{l,j,j}$ of the tasks in sublist (l, j) of level l into j groups produced by a list scheduling algorithm A, where $1 \le l \le v$ and $1 \le j \le c$, and an energy allocation $E_{l,1}$, $E_{l,2}$, ..., $E_{l,c}$ to the c sublists of level l, where

$$E_{l,j} = \left(\frac{N_{l,j}^{1/\alpha}}{N_{l,1}^{1/\alpha} + N_{l,2}^{1/\alpha} + \cdots + N_{l,c}^{1/\alpha}} \right) E_l,$$

with $N_{l,j} = M_{l,j,1}^{\alpha} + M_{l,j,2}^{\alpha} + \cdots + M_{l,j,j}^{\alpha}$, for all $1 \le l \le v$ and $1 \le j \le c$, the scheduling algorithm H_c-A produces schedule length

$$T_l = \frac{(N_{l,1}^{1/\alpha} + N_{l,2}^{1/\alpha} + \cdots + N_{l,c}^{1/\alpha})^{\alpha/(\alpha-1)}}{E_l^{1/(\alpha-1)}},$$

for tasks in level l, where $1 \le l \le v$. Since the level-by-level scheduling algorithm produces schedule length $T = T_1 + T_2 + \cdots + T_v$, we have

$$T = \sum_{l=1}^{v} \frac{(N_{l,1}^{1/\alpha} + N_{l,2}^{1/\alpha} + \cdots + N_{l,c}^{1/\alpha})^{\alpha/(\alpha-1)}}{E_l^{1/(\alpha-1)}}.$$

Let $S_l = (N_{l,1}^{1/\alpha} + N_{l,2}^{1/\alpha} + \cdots + N_{l,c}^{1/\alpha})^{\alpha}$, for all $1 \le l \le v$. By the definition of S_l, we obtain

$$T = \left(\frac{S_1}{E_1} \right)^{1/(\alpha-1)} + \left(\frac{S_2}{E_2} \right)^{1/(\alpha-1)} + \cdots + \left(\frac{S_v}{E_v} \right)^{1/(\alpha-1)}.$$

To minimize T with the constraint $F(E_1, E_2, ..., E_v) = E_1 + E_2 + \cdots + E_v = E$, we use the Lagrange multiplier system

$$\nabla T(E_1, E_2, ..., E_v) = \lambda \nabla F(E_1, E_2, ..., E_v),$$

where λ is the Lagrange multiplier. Since $\partial T / \partial E_l = \lambda \partial F / \partial E_l$, that is,

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

$1 \le l \le v$, we get

$$E_l = S_l^{1/\alpha} \left(\frac{1}{\lambda(1-\alpha)} \right)^{(\alpha-1)/\alpha},$$

which implies that

$$E = (S_1^{1/\alpha} + S_2^{1/\alpha} + \cdots + S_v^{1/\alpha}) \left(\frac{1}{\lambda(1-\alpha)} \right)^{(\alpha-1)/\alpha},$$

and

$$E_l = \left(\frac{S_l^{1/\alpha}}{S_1^{1/\alpha} + S_2^{1/\alpha} + \cdots + S_v^{1/\alpha}} \right) E,$$

for all $1 \le l \le v$. By using the above energy allocation, we have

$$\begin{aligned} T &= \sum_{l=1}^{v} \left(\frac{S_l}{E_l} \right)^{1/(\alpha-1)} \\ &= \sum_{l=1}^{v} \frac{S_l^{1/(\alpha-1)}}{\left(\left(\frac{S_l^{1/\alpha}}{S_1^{1/\alpha} + S_2^{1/\alpha} + \cdots + S_v^{1/\alpha}} \right) E \right)^{1/(\alpha-1)}} \\ &= \sum_{l=1}^{v} \frac{S_l^{1/\alpha} (S_1^{1/\alpha} + S_2^{1/\alpha} + \cdots + S_v^{1/\alpha})^{1/(\alpha-1)}}{E^{1/(\alpha-1)}} \\ &= \frac{(S_1^{1/\alpha} + S_2^{1/\alpha} + \cdots + S_v^{1/\alpha})^{\alpha/(\alpha-1)}}{E^{1/(\alpha-1)}}. \end{aligned}$$

For any list scheduling algorithm A, we have $R_{l,j,k} \le R_{l,j}/j + r^*$, for all $1 \le l \le v$ and $1 \le j \le c$ and $1 \le k \le j$, where $r^* = \max(r_1, r_2, ..., r_n)$ is the maximum task execution requirement. Since $\pi_i \le m/j$ for every task i in group (l, j, k) of sublist (l, j) of level l, we get

$$M_{l,j,k} \le \left(\frac{m}{j} \right)^{1/\alpha} R_{l,j,k} \le \left(\frac{m}{j} \right)^{1/\alpha} \left(\frac{R_{l,j}}{j} + r^* \right).$$

Therefore,

$$N_{l,j} \le m \left(\frac{R_{l,j}}{j} + r^* \right)^{\alpha},$$

and

$$N_{l,j}^{1/\alpha} \le m^{1/\alpha} \left(\frac{R_{l,j}}{j} + r^* \right),$$

and

$$N_{l,1}^{1/\alpha} + N_{l,2}^{1/\alpha} + \cdots + N_{l,c}^{1/\alpha} \le m^{1/\alpha} \left(\left(\sum_{j=1}^{c} \frac{R_{l,j}}{j} \right) + cr^* \right).$$

Consequently,

$$S_l \le m \left(\left(\sum_{j=1}^{c} \frac{R_{l,j}}{j} \right) + cr^* \right)^{\alpha},$$

and

$$S_l^{1/\alpha} \le m^{1/\alpha} \left(\left(\sum_{j=1}^{c} \frac{R_{l,j}}{j} \right) + cr^* \right),$$

and

$$S_1^{1/\alpha}+S_2^{1/\alpha}+\cdots+S_v^{1/\alpha}\le m^{1/\alpha}\left(\left(\sum_{j=1}^{c}\frac{R_j}{j}\right)+cvr^*\right),$$

which implies that

$$T\le m^{1/(\alpha-1)}\left(\left(\sum_{j=1}^{c}\frac{R_j}{j}\right)+cvr^*\right)^{\alpha/(\alpha-1)}\frac{1}{E^{1/(\alpha-1)}}.$$

We define the *performance ratio* as $\beta=T/T^*$ for heuristic algorithms that solve the problem of minimizing schedule length with energy consumption constraint on a multicore processor. By Theorem 1, we get

$$\beta=\frac{T}{T^*}\le\left(\left(\left(\sum_{j=1}^{c}\frac{R_j}{j}\right)+cvr^*\right)\Big/\left(\frac{W}{m}\right)\right)^{\alpha/(\alpha-1)}.$$

Theorem 6 provides optimal energy allocation to the v levels for minimizing schedule length with energy consumption constraint in scheduling precedence constrained parallel tasks by using level-by-level scheduling algorithms LL-H$_c$-A, where A is a list scheduling algorithm.

Theorem 6 *For a given partition $M_{l,j,1}$, $M_{l,j,2}$, ..., $M_{l,j,j}$ of the tasks in sublist (l,j) of level l into j groups produced by a list scheduling algorithm A, where $1\le l\le v$ and $1\le j\le c$, and an energy allocation E_1, E_2, ..., E_v to the v levels, the level-by-level scheduling algorithm LL-H$_c$-A produces schedule length*

$$T=\sum_{l=1}^{v}\frac{(N_{l,1}^{1/\alpha}+N_{l,2}^{1/\alpha}+\cdots+N_{l,c}^{1/\alpha})^{\alpha/(\alpha-1)}}{E_l^{1/(\alpha-1)}},$$

where $N_{l,j}=M_{l,j,1}^{\alpha}+M_{l,j,2}^{\alpha}+\cdots+M_{l,j,j}^{\alpha}$, for all $1\le l\le v$ and $1\le j\le c$. The energy allocation E_1, E_2, ..., E_v which minimizes T is

$$E_l=\left(\frac{S_l^{1/\alpha}}{S_1^{1/\alpha}+S_2^{1/\alpha}+\cdots+S_v^{1/\alpha}}\right)E,$$

where $S_l=(N_{l,1}^{1/\alpha}+N_{l,2}^{1/\alpha}+\cdots+N_{l,c}^{1/\alpha})^{\alpha}$, for all $1\le l\le v$, and the minimized schedule length is

$$T=\frac{(S_1^{1/\alpha}+S_2^{1/\alpha}+\cdots+S_v^{1/\alpha})^{\alpha/(\alpha-1)}}{E^{1/(\alpha-1)}},$$

by using the above energy allocation. The performance ratio is

$$\beta\le\left(\left(\left(\sum_{j=1}^{c}\frac{R_j}{j}\right)+cvr^*\right)\Big/\left(\frac{W}{m}\right)\right)^{\alpha/(\alpha-1)},$$

where $r^*=\max(r_1,r_2,...,r_n)$ is the maximum task execution requirement.

Theorems 4 and 5 and 6 give the power supply to the task i in group (l,j,k) as

$$\frac{1}{\pi_i}\left(\frac{E_{l,j,k}}{M_{l,j,k}}\right)^{\alpha/(\alpha-1)} = \frac{1}{\pi_i}\left(\left(\frac{M^{\alpha}_{l,j,k}}{M^{\alpha}_{l,j,1}+M^{\alpha}_{l,j,2}+\cdots+M^{\alpha}_{l,j,j}}\right)\right.$$

$$\left.\left(\frac{N^{1/\alpha}_{l,j}}{N^{1/\alpha}_{l,1}+N^{1/\alpha}_{l,2}+\cdots+N^{1/\alpha}_{l,c}}\right)\left(\frac{S^{1/\alpha}_{l}}{S^{1/\alpha}_{1}+S^{1/\alpha}_{2}+\cdots+S^{1/\alpha}_{v}}\right)\frac{E}{M_{l,j,k}}\right)^{\alpha/(\alpha-1)},$$

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

We notice that the performance bound given in Theorem 6 is loose and pessimistic mainly due to the overestimation of the π_i's in sublist (l,j) to m/j. One possible remedy is to use the value of $(m/(j+1)+m/j)/2$ as an approximation to π_i. Also, as the number of tasks gets large, the term cvr^* may be removed. This gives rise to the following performance bound for β:

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

Our simulation shows that the modified bound in (1) is more accurate than the performance bound given in Theorem 6.

5.2 Minimizing Energy Consumption

5.2.1 Level 1

The following result [29] gives the optimal power supplies when n parallel tasks are scheduled sequentially.

Theorem 7 *When n parallel tasks are scheduled sequentially, the total energy consumption is minimized when task i is supplied with power* $p_i = (M/T)^{\alpha}/\pi_i$, *where* $1 \le i \le n$. *The minimum energy consumption is* $E = M^{\alpha}/T^{\alpha-1}$.

5.2.2 Level 2

The following result [29] gives the optimal energy allocation and power supplies that minimize energy consumption for a given partition of n tasks into j groups on a multicore processor.

Theorem 8 *For a given partition* M_1, M_2, ..., M_j *of n parallel tasks into j groups on a multicore processor partitioned into j clusters, the total energy consumption is minimized when task i in group k is executed with power* $p_i = (M_k/T)^{\alpha}/\pi_i$, *where* $1 \le k \le j$. *The minimum energy consumption is*

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

for the above energy allocation and power supplies.

5.2.3 Level 3

To use algorithm H_c-A to solve the problem of minimizing energy consumption with schedule length constraint T, we need to allocate the time T to the c sublists. We use T_1, T_2, ..., T_c to represent a time allocation to the c sublists, where tasks in sublist sublist j are executed within deadline T_j, and $T_1+T_2+\cdots+T_c=T$. Theorem 9 [29] provides optimal time allocation to the c sublists for minimizing energy consumption with schedule length constraint in scheduling parallel tasks by using scheduling algorithms H_c-A, where A is a list scheduling algorithm.

Theorem 9 *For a given partition $M_{j,1}$, $M_{j,2}$, ..., $M_{j,j}$ of the tasks in sublist j into j groups produced by a list scheduling algorithm A, where $1\le j\le c$, and a time allocation T_1, T_2, ..., T_c to the c sublists, the amount of energy consumed by algorithm H_c-A is*

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

The time allocation T_1, T_2, ..., T_c which minimizes E is

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

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

$$E=\frac{(N_1^{1/\alpha}+N_2^{1/\alpha}+\cdots+N_c^{1/\alpha})^{\alpha}}{T^{\alpha-1}},$$

by using the above time allocation.

5.2.4 Level 4

To use a level-by-level scheduling algorithm to solve the problem of minimizing energy consumption with schedule length constraint T, we need to allocate the time T to the v levels. We use T_1, T_2, ..., T_v to represent a time allocation to the v levels, where tasks in level l are executed within deadline T_l, and $T_1+T_2+\cdots+T_v=T$.

By Theorem 9, for a given partition $M_{l,j,1}, M_{l,j,2}$, ..., $M_{l,j,j}$ of the tasks in sublist (l,j) of level l into j groups produced by a list scheduling algorithm A, where $1\le l\le v$ and $1\le j\le c$, and a time allocation $T_{l,1}$, $T_{l,2}$, ..., $T_{l,c}$ to the c sublists of level l, where

$$T_{l,j}=\left(\frac{N_{l,j}^{1/\alpha}}{N_{l,1}^{1/\alpha}+N_{l,2}^{1/\alpha}+\cdots+N_{l,c}^{1/\alpha}}\right)T_l,$$

with $N_{l,j}=M_{l,j,1}^{\alpha}+M_{l,j,2}^{\alpha}+\cdots+M_{l,j,j}^{\alpha}$, for all $1\le l\le v$ and $1\le j\le c$, the scheduling algorithm H_c-A consumes energy

$$E_l=\frac{(N_{l,1}^{1/\alpha}+N_{l,2}^{1/\alpha}+\cdots+N_{l,c}^{1/\alpha})^{\alpha}}{T_l^{\alpha-1}},$$

for tasks in level l, where $1 \le l \le v$. Since the level-by-level scheduling algorithm consumes energy $E = E_1 + E_2 + \cdots + E_v$, we have

$$E = \sum_{l=1}^{v} \frac{(N_{l,1}^{1/\alpha} + N_{l,2}^{1/\alpha} + \cdots + N_{l,c}^{1/\alpha})^{\alpha}}{T_l^{\alpha-1}}.$$

By the definition of S_l, we obtain

$$E = \frac{S_1}{T_1^{\alpha-1}} + \frac{S_2}{T_2^{\alpha-1}} + \cdots + \frac{S_v}{T_v^{\alpha-1}}.$$

To minimize E with the constraint $F(T_1, T_2, ..., T_v) = T_1 + T_2 + \cdots + T_v = T$, we use the Lagrange multiplier system

$$\nabla E(T_1, T_2, ..., T_v) = \lambda \nabla F(T_1, T_2, ..., T_v),$$

where λ is the Lagrange multiplier. Since $\partial E / \partial T_l = \lambda \partial F / \partial T_l$, that is,

$$S_l \left(\frac{1-\alpha}{T_l^{\alpha}} \right) = \lambda,$$

$1 \le l \le v$, we get

$$T_l = S_l^{1/\alpha} \left(\frac{1-\alpha}{\lambda} \right)^{1/\alpha},$$

which implies that

$$T = (S_1^{1/\alpha} + S_2^{1/\alpha} + \cdots + S_v^{1/\alpha}) \left(\frac{1-\alpha}{\lambda} \right)^{1/\alpha},$$

and

$$T_l = \left(\frac{S_l^{1/\alpha}}{S_1^{1/\alpha} + S_2^{1/\alpha} + \cdots + S_v^{1/\alpha}} \right) T,$$

for all $1 \le l \le v$. By using the above time allocation, we have

$$\begin{aligned}
E &= \sum_{l=1}^{v} \frac{S_l}{T_l^{\alpha-1}} \\
&= \sum_{l=1}^{v} \frac{S_l}{\left(\left(\frac{S_l^{1/\alpha}}{S_1^{1/\alpha} + S_2^{1/\alpha} + \cdots + S_v^{1/\alpha}} \right) T \right)^{\alpha-1}} \\
&= \sum_{l=1}^{v} \frac{S_l^{1/\alpha} (S_1^{1/\alpha} + S_2^{1/\alpha} + \cdots + S_v^{1/\alpha})^{\alpha-1}}{T^{\alpha-1}} \\
&= \frac{(S_1^{1/\alpha} + S_2^{1/\alpha} + \cdots + S_v^{1/\alpha})^{\alpha}}{T^{\alpha-1}}.
\end{aligned}$$

Similar to the derivation in Subsection 5.1.4, we have

$$S_1^{1/\alpha}+S_2^{1/\alpha}+\cdots+S_v^{1/\alpha}\le m^{1/\alpha}\left(\left(\sum_{j=1}^{c}\frac{R_j}{j}\right)+cvr^*\right),$$

which implies that

$$E\le m\left(\left(\sum_{j=1}^{c}\frac{R_j}{j}\right)+cvr^*\right)^{\alpha}\frac{1}{T^{\alpha-1}}.$$

We define the *performance ratio* as $\beta=E/E^*$ for heuristic algorithms that solve the problem of minimizing energy consumption with schedule length constraint on a multicore processor. By Theorem 2, we get

$$\beta=\frac{E}{E^*}\le\left(\left(\left(\sum_{j=1}^{c}\frac{R_j}{j}\right)+cvr^*\right)\Big/\left(\frac{W}{m}\right)\right)^{\alpha}.$$

Theorem 10 provides optimal time allocation to the v levels for minimizing energy consumption with schedule length constraint in scheduling precedence constrained parallel tasks by using level-by-level scheduling algorithms LL-H$_c$-A, where A is a list scheduling algorithm.

Theorem 10 *For a given partition $M_{l,j,1}$, $M_{l,j,2}$, ..., $M_{l,j,j}$ of the tasks in sublist (l,j) of level l into j groups produced by a list scheduling algorithm A, where $1\le l\le v$ and $1\le j\le c$, and a time allocation T_1, T_2, ..., T_v to the v levels, the level-by-level scheduling algorithm LL-H$_c$-A consumes energy*

$$E=\sum_{l=1}^{v}\frac{(N_{l,1}^{1/\alpha}+N_{l,2}^{1/\alpha}+\cdots+N_{l,c}^{1/\alpha})^{\alpha}}{T_l^{\alpha-1}},$$

where $N_{l,j}=M_{l,j,1}^{\alpha}+M_{l,j,2}^{\alpha}+\cdots+M_{l,j,j}^{\alpha}$, for all $1\le l\le v$ and $1\le j\le c$. The time allocation T_1, T_2, ..., T_v which minimizes E is

$$T_l=\left(\frac{S_l^{1/\alpha}}{S_1^{1/\alpha}+S_2^{1/\alpha}+\cdots+S_v^{1/\alpha}}\right)T,$$

where $S_l=(N_{l,1}^{1/\alpha}+N_{l,2}^{1/\alpha}+\cdots+N_{l,c}^{1/\alpha})^{\alpha}$, for all $1\le l\le v$, and the minimized energy consumption is

$$E=\frac{(S_1^{1/\alpha}+S_2^{1/\alpha}+\cdots+S_v^{1/\alpha})^{\alpha}}{T^{\alpha-1}},$$

by using the above time allocation. The performance ratio is

$$\beta\le\left(\left(\left(\sum_{j=1}^{c}\frac{R_j}{j}\right)+cvr^*\right)\Big/\left(\frac{W}{m}\right)\right)^{\alpha},$$

where $r^*=\max(r_1,r_2,...,r_n)$ is the maximum task execution requirement.

Theorems 8 and 9 and 10 give the power supply to the task i in group (l, j, k) as

$$\frac{1}{\pi_i}\left(\frac{M_{l,j,k}}{T_{l,j}}\right)^{\alpha} = \frac{1}{\pi_i}\left(\left(\frac{N_{l,1}^{1/\alpha}+N_{l,2}^{1/\alpha}+\cdots+N_{l,c}^{1/\alpha}}{N_{l,j}^{1/\alpha}}\right)\left(\frac{S_1^{1/\alpha}+S_2^{1/\alpha}+\cdots+S_v^{1/\alpha}}{S_l^{1/\alpha}}\right)\frac{M_{l,j,k}}{T}\right)^{\alpha},$$

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

Again, we adjust the performance bound given in Theorem 10 to

$$\left(\left(\sum_{j=1}^{c}\frac{R_j}{j}\left(\frac{2j+1}{2j+2}\right)^{1/\alpha}\right)\Bigg/\left(\frac{W}{m}\right)\right)^{\alpha}. \tag{2}$$

Our simulation shows that the modified bound in (2) is more accurate than the performance bound given in Theorem 10.

6 Simulation Data

To validate our analytical results, extensive simulations have been conducted. In this section, we demonstrate some numerical and experimental data for several example task graphs. The following task graphs are considered in our experiments.

- *Tree-Structured Computations.* Many computations are tree-structured, including backtracking search, branch-and-bound computations, game-tree evaluation, functional and logical programming, and various numeric computations. For simplicity, we consider $\mathrm{CT}(b,h)$, i.e., complete b-ary trees of height h (see Figure 3 where $b = 2$ and $h = 4$). It is easy to see that there are $v = h+1$ levels numbered as 0, 1, 2, ..., h, and $n_l = b^l$ for $0 \le l \le h$, and $n = (b^{h+1}-1)/(b-1)$.

- *Partitioning Algorithms.* A partitioning algorithm $\mathrm{PA}(b,h)$ represents a divide-and-conquer computation with branching factor b and height (i.e., depth of recursion) h (see Figure 4 where $b = 2$ and $h = 3$). The dag of $\mathrm{PA}(b,h)$ has $v = 2h+1$ levels numbered as 0, 1, 2, ..., $2h$. A partitioning algorithm proceeds in three stages. In levels $0, 1, ..., h-1$, each task is divided into b subtasks. Then, in level h, subproblems of small sizes are solved directly. Finally, in levels $h+1, h+2, ..., 2h$, solutions to subproblems are combined to form the solution to the original problem. Clearly, $n_l = n_{2h-l} = b^l$, for all $0 \le l \le h-1$, $n_h = b^h$, and $n = (b^{h+1}+b^h-2)/(b-1)$.

- *Linear Algebra Task Graphs.* A linear algebra task graph $\mathrm{LA}(v)$ with v levels (see Figure 5 where $v = 5$) has $n_l = v-l+1$ for $l = 1, 2, ..., v$, and $n = v(v+1)/2$.

- *Diamond Dags.* A diamond dag $\mathrm{DD}(d)$ (see Figure 6 where $d = 4$) contains $v = 2d-1$ levels numbered as 1, 2, ..., $2d-1$. It is clear that $n_l = n_{2d-l} = l$, for all $1 \le l \le d-1$, $n_d = d$, and $n = d^2$.

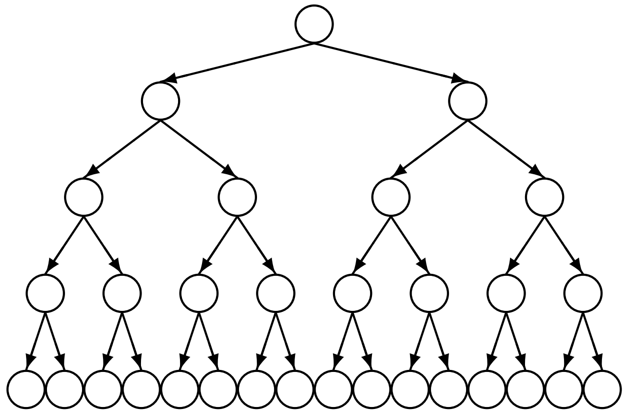

Figure 3: $\mathrm{CT}(b,h)$: a complete binary tree with $b=2$ and $h=4$.

Table 3A. *Simulation Data for Expected NSL on CT(2,12).*

	uniform		binomial		geometric	
$\bar{\pi}$	simulation	analysis	simulation	analysis	simulation	analysis
10	1.1772602	1.1850145	1.1127903	1.0635657	1.2695944	1.3183482
20	1.1609754	1.1485746	1.1046696	1.0817685	1.2527372	1.2739448
30	1.2032217	1.2026955	1.1401395	1.1407631	1.2827662	1.3051035
40	1.3783493	1.4501456	1.2111586	1.2364135	1.2959831	1.3174113
50	1.3977418	1.4592250	1.2498124	1.2784298	1.2998132	1.3175610
60	1.3278814	1.3437082	1.2799084	1.3180794	1.3030358	1.3200509

(99% confidence interval $\pm 0.365\%$)

Table 3B. *Simulation Data for Expected NEC on CT(2,12).*

	uniform		binomial		geometric	
$\bar{\pi}$	simulation	analysis	simulation	analysis	simulation	analysis
10	1.3816853	1.4002241	1.2386909	1.1314678	1.6180743	1.7403012
20	1.3471473	1.3204301	1.2223807	1.1720051	1.5698000	1.6194065
30	1.4504859	1.4461415	1.2989038	1.2983591	1.6412385	1.6968020
40	1.9023971	2.1084568	1.4683900	1.5308593	1.6805737	1.7387274
50	1.9592480	2.1352965	1.5604366	1.6323378	1.6883269	1.7364845
60	1.7623788	1.8044903	1.6405732	1.7409541	1.6957874	1.7386959

(99% confidence interval $\pm 0.687\%$)

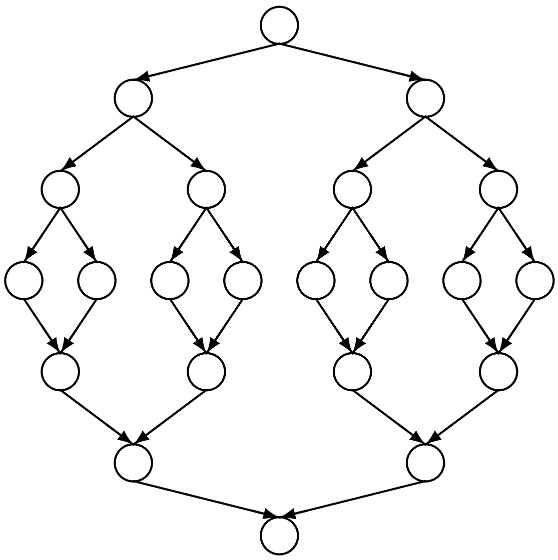

Figure 4: PA(b,h): a partitioning algorithm with $b = 2$ and $h = 3$.

Table 4A. *Simulation Data for Expected NSL on PA(2,12).*

	uniform		binomial		geometric	
$\bar{\pi}$	simulation	analysis	simulation	analysis	simulation	analysis
10	1.1940250	1.1841913	1.1287074	1.0635894	1.2918262	1.3185661
20	1.1710935	1.1489358	1.1120907	1.0822820	1.2628233	1.2735483
30	1.2121712	1.2032254	1.1414699	1.1396784	1.2893692	1.3044971
40	1.3838241	1.4505296	1.2130609	1.2377678	1.3006607	1.3152063
50	1.4034276	1.4608829	1.2497254	1.2777187	1.3052527	1.3182187
60	1.3319146	1.3448578	1.2799201	1.3177687	1.3067475	1.3179615

(99% confidence interval $\pm 0.284\%$)

Table 4B. *Simulation Data for Expected NEC on PA(2,12).*

	uniform		binomial		geometric	
$\bar{\pi}$	simulation	analysis	simulation	analysis	simulation	analysis
10	1.4280855	1.4053089	1.2756771	1.1309478	1.6643757	1.7374005
20	1.3687912	1.3196764	1.2362757	1.1716339	1.5959196	1.6185853
30	1.4680717	1.4464946	1.3037462	1.3007006	1.6629560	1.7012833
40	1.9143602	2.1021764	1.4697836	1.5294041	1.6933298	1.7328875
50	1.9717267	2.1383667	1.5614395	1.6318344	1.7026727	1.7361106
60	1.7748939	1.8095803	1.6402284	1.7397315	1.7084739	1.7376521

(99% confidence interval $\pm 0.565\%$)

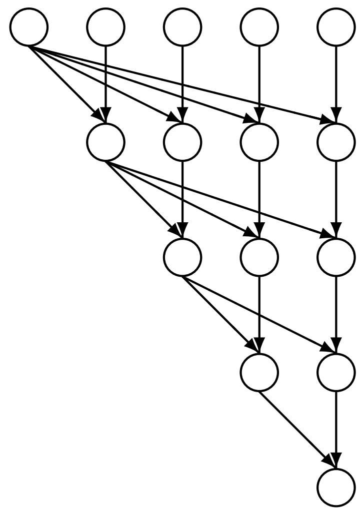

Figure 5: LA(v): a linear algebra task graph with $v = 5$.

Table 5A. *Simulation Data for Expected NSL on LA(2000).*

	uniform		binomial		geometric	
$\bar{\pi}$	simulation	analysis	simulation	analysis	simulation	analysis
10	1.1392509	1.1841096	1.0771624	1.0638363	1.2300726	1.3179978
20	1.1430859	1.1491148	1.0989144	1.0823187	1.2321125	1.2722681
30	1.1954796	1.2028781	1.1372623	1.1399934	1.2686012	1.3032303
40	1.3729227	1.4497884	1.2109722	1.2375699	1.2858406	1.3161030
50	1.3964647	1.4610101	1.2488649	1.2779096	1.2930727	1.3191233
60	1.3272967	1.3445859	1.2802743	1.3187192	1.2959390	1.3182489

(99% confidence interval $\pm 0.085\%$)

Table 5B. *Simulation Data for Expected NEC on LA(2000).*

	uniform		binomial		geometric	
$\bar{\pi}$	simulation	analysis	simulation	analysis	simulation	analysis
10	1.2974381	1.4020482	1.1602487	1.1313969	1.5137571	1.7379887
20	1.3062497	1.3200333	1.2076518	1.1715685	1.5175999	1.6178453
30	1.4292225	1.4470430	1.2933014	1.2994524	1.6099920	1.6995260
40	1.8847470	2.1014650	1.4664142	1.5315937	1.6530311	1.7317472
50	1.9501571	2.1348479	1.5596494	1.6330611	1.6715971	1.7392715
60	1.7624447	1.8088376	1.6389275	1.7388263	1.6797186	1.7382355

(99% confidence interval $\pm 0.204\%$)

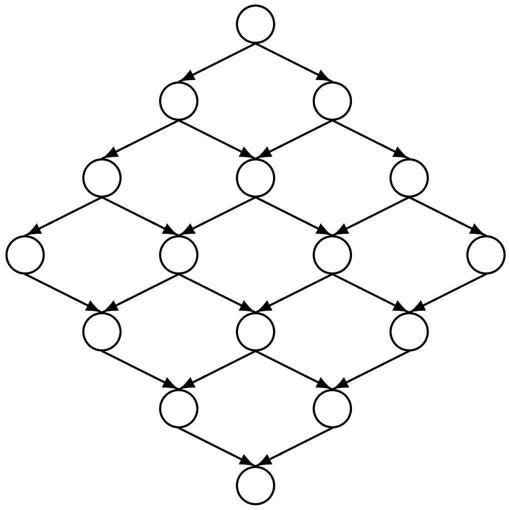

Figure 6: $\mathrm{DD}(d)$: a diamond dag with $d = 4$.

Table 6A. *Simulation Data for Expected NSL on DD(2000).*

	uniform		binomial		geometric	
$\bar{\pi}$	simulation	analysis	simulation	analysis	simulation	analysis
10	1.1393071	1.1842982	1.0770276	1.0636933	1.2303693	1.3183983
20	1.1429980	1.1490295	1.0989960	1.0822466	1.2316570	1.2714949
30	1.1955924	1.2030593	1.1372779	1.1400176	1.2690205	1.3039776
40	1.3726198	1.4493161	1.2109189	1.2375156	1.2859527	1.3162776
50	1.3962951	1.4607530	1.2487413	1.2777190	1.2932855	1.3193741
60	1.3274819	1.3447974	1.2803877	1.3189128	1.2962310	1.3186892

(99% confidence interval $\pm 0.054\%$)

Table 6B. *Simulation Data for Expected NEC on DD(2000).*

	uniform		binomial		geometric	
$\bar{\pi}$	simulation	analysis	simulation	analysis	simulation	analysis
10	1.2978774	1.4023671	1.1597583	1.1313744	1.5144683	1.7391638
20	1.3063526	1.3202184	1.2076968	1.1715103	1.5179540	1.6182936
30	1.4292362	1.4470899	1.2934523	1.2996875	1.6099667	1.6996302
40	1.8840943	2.1007925	1.4659063	1.5308111	1.6536717	1.7325694
50	1.9501477	2.1345382	1.5596254	1.6330039	1.6719013	1.7398729
60	1.7625789	1.8090184	1.6405736	1.7412621	1.6799813	1.7386383

(99% confidence interval $\pm 0.155\%$)

Since each task graph has at least one parameter, we are actually dealing with classes of task graphs.

We define the *normalized schedule length* (NSL) as

$$\mathrm{NSL} = \frac{T}{\left(\dfrac{m}{E}\left(\dfrac{W}{m}\right)^{\alpha}\right)^{1/(\alpha-1)}}.$$

When T is the schedule length produced by a heuristic algorithm LL-H$_c$-A according to Theorem 6, the normalized schedule length is

$$\mathrm{NSL} = \left(\frac{(S_1^{1/\alpha} + S_2^{1/\alpha} + \cdots + S_v^{1/\alpha})^{\alpha}}{m\left(\dfrac{W}{m}\right)^{\alpha}}\right)^{1/(\alpha-1)}.$$

NSL is an upper bound for the performance ratio $\beta = T/T^*$ for the problem of minimizing schedule length with energy consumption constraint on a multicore processor. When the π_i's and the r_i's are random variables, T, T^*, β, and NSL all become random variables. It is clear that for the problem of minimizing schedule length with energy consumption constraint, we have $\bar{\beta} \le \overline{\mathrm{NSL}}$, i.e., the expected performance ratio is no larger than the expected normalized schedule length. (We use $\bar{x}$ to represent the expectation of a random variable x.)

We define the *normalized energy consumption* (NEC) as

$$\mathrm{NEC} = \frac{E}{m\left(\dfrac{W}{m}\right)^{\alpha}\dfrac{1}{T^{\alpha-1}}}.$$

When E is the energy consumed by a heuristic algorithm LL-H$_c$-A according to Theorem 10, the normalized energy consumption is

$$\mathrm{NEC} = \frac{(S_1^{1/\alpha} + S_2^{1/\alpha} + \cdots + S_v^{1/\alpha})^{\alpha}}{m\left(\dfrac{W}{m}\right)^{\alpha}}.$$

NEC is an upper bound for the performance ratio $\beta = E/E^*$ for the problem of minimizing energy consumption with schedule length constraint on a multicore processor. For the problem of minimizing energy consumption with schedule length constraint, we have $\bar{\beta} \le \overline{\mathrm{NEC}}$.

Notice that for a given task graph, the expected normalized schedule length $\overline{\mathrm{NSL}}$ and the expected normalized energy consumption $\overline{\mathrm{NEC}}$ are determined by A, c, m, α, and the probability distributions of the π_i's and the r_i's. In our simulations, the algorithm A is chosen as LS; the parameter c is set as 20; the number of cores is set as $m = 128$; and the parameter α is set as 3. The particular choices of these values do not affect our general observations and conclusions. For convenience, the r_i's are treated as independent and identically distributed (i.i.d.) continuous random variables uniformly distributed in $[0,1)$. The π_i's are i.i.d. discrete random variables. We consider three types of probability distributions of task sizes with about the same expected task size $\bar{\pi}$. Let a_b be the probability that $\pi_i = b$, where $b \ge 1$.

- Uniform distributions in the range $[1..u]$, i.e., $a_b = 1/u$ 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 $[1..m]$, i.e.,

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

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

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

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

- Geometric distributions in the range $[1..m]$, i.e.,

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

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

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

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

In Tables 3–6, we show and compare the analytical results with simulation data. For each task graph in { CT(2,12), PA(2,12), LA(2000), DD(2000) }, and each $\bar{\pi}$ in the range 10, 20, ..., 60, and each probability distribution of task sizes, we generate *rep* sets of tasks, produce their schedules by using algorithm LL-H$_c$-LS, calculate their NSL (or NEC) and the bound (1) (or bound (2)), report the average of NSL (or NEC) which is the experimental value of $\overline{\text{NSL}}$ (or $\overline{\text{NEC}}$), and report the average of bound (1) (or bound (2)) which is the numerical value of analytical results. The number *rep* is large enough to ensure high quality experimental data. The 99% confidence interval of all the data in the same table is also given.

We have the following observations from our simulations.

- $\overline{\text{NSL}}$ is less than 1.41 and $\overline{\text{NEC}}$ is less than 1.98. Therefore, our algorithms produce solutions reasonably close to optimum. In fact, $\overline{\text{NSL}}$ and $\overline{\text{NEC}}$ reported here are very close to those for independent parallel tasks reported in [29].

- The performance of algorithm LL-H$_c$-A for A other than LS is very close (within $\pm 1\%$) to the performance of algorithm LL-H$_c$-LS. Since these data do not provide further insight, they are not shown here.

- The performance bound (1) is very close to $\overline{\text{NSL}}$ and the performance bound (2) is very close to $\overline{\text{NEC}}$.

7 Summary and Future Research

We have emphasized the significance of investigating energy-efficient and high-performance processing of large-scale parallel applications on multicore processors in data centers. We addressed scheduling precedence constrained parallel tasks on multicore processors with dynamically variable voltage and speed as combinatorial optimization problems. We pointed out that our scheduling problems contain four nontrivial subproblems, namely, precedence constraining, system partitioning, task scheduling, and power supplying. We described our methods to deal with precedence constraints, system partitioning, and task scheduling, and developed our optimal four-level energy/time/power allocation scheme for minimizing schedule length and minimizing energy consumption. We also analyzed the performance of our heuristic algorithms, and derived accurate performance bounds. We demonstrated simulation data, which validate our analytical results.

Further research can be directed toward employing more effective and efficient algorithms to deal with independent tasks in the same level. Notice that the approach in this paper (i.e., algorithm LL-H_c-A) belongs to the class of post-power-determination algorithms. Such an algorithm first generates a schedule, and then determines power supplies [31, 32]. The classes of pre-power-determination and hybrid algorithms are worth of investigation [30]. Our study in this paper can also be extended to multiple multicore/manycore processors in data centers and discrete speed levels.

References

[1] http://en.wikipedia.org/wiki/Adapteva

[2] http://en.wikipedia.org/wiki/Dynamic_voltage_scaling

[3] http://www.intel.com/multicore/

[4] http://www.multicoreinfo.com/2011/10/adapteva-2/

[5] S. Albers, "Energy-efficient algorithms," *Communications of the ACM*, vol. 53, no. 5, pp. 86-96, 2010.

[6] H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alvarez, "Power-aware scheduling for periodic real-time tasks," *IEEE Transactions on Computers*, vol. 53, no. 5, pp. 584-600, 2004.

[7] N. Bansal, T. Kimbrel, and K. Pruhs, "Dynamic speed scaling to manage energy and temperature," *Proceedings of the 45th IEEE Symposium on Foundation of Computer Science*, pp. 520-529, 2004.

[8] J. A. Barnett, "Dynamic task-level voltage scheduling optimizations," *IEEE Transactions on Computers*, vol. 54, no. 5, pp. 508-520, 2005.

[9] L. Benini, A. Bogliolo, and G. De Micheli, “A survey of design techniques for system-level dynamic power management," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 3, pp. 299-316, 2000.

[10] A. Berl, E. Gelenbe, M. Di Girolamo, G. Giuliani, H. De Meer, M. Q. Dang, and K. Pentikousis, “Energy-efficient cloud computing," *The Computer Journal*, vol. 53, no. 7, pp. 1045-1051, 2010.

[11] D. P. Bunde, “Power-aware scheduling for makespan and flow," *Proceedings of the 18th ACM Symposium on Parallelism in Algorithms and Architectures*, pp. 190-196, 2006.

[12] H.-L. Chan, W.-T. Chan, T.-W. Lam, L.-K. Lee, K.-S. Mak, and P. W. H. Wong, “Energy efficient online deadline scheduling," *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms*, pp. 795-804, 2007.

[13] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, “Low-power CMOS digital design," *IEEE Journal on Solid-State Circuits*, vol. 27, no. 4, pp. 473-484, 1992.

[14] S. Cho and R. G. Melhem, “On the interplay of parallelization, program performance, and energy consumption," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 3, pp. 342-353, 2010.

[15] D. Donofrio, L. Oliker, J. Shalf, M. F. Wehner, C. Rowen, J. Krueger, S. Kamil, and M. Mohiyuddin, “Energy-efficient computing for extreme-scale science," *Computer*, vol. 42, no. 11, pp. 62-71, 2009.

[16] W.-c. Feng and K. W. Cameron, “The green500 list: encouraging sustainable supercomputing," *Computer*, vol. 40, no. 12, pp. 50-55, 2007.

[17] V. W. Freeh, D. K. Lowenthal, F. Pan, N. Kappiah, R. Springer, B. L. Rountree, and M. E. Femal, “Analyzing the energy-time trade-off in high-performance computing applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 6, pp. 835-848, 2007.

[18] S. K. Garg, C. S. Yeo, A. Anandasivam, and R. Buyya, “Environment-conscious scheduling of HPC applications on distributed cloud-oriented data centers," *Journal of Parallel Distributed Computing*, vol. 71, no. 6, pp. 732-749, 2011.

[19] R. L. Graham, “Bounds on multiprocessing timing anomalies," *SIAM J. Appl. Math.*, vol. 2, pp. 416-429, 1969.

[20] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. B. Srivastava, “Power optimization of variable-voltage core-based systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 12, pp. 1702-1714, 1999.

[21] C. Im, S. Ha, and H. Kim, “Dynamic voltage scheduling with buffers in low-power multimedia applications," *ACM Transactions on Embedded Computing Systems*, vol. 3, no. 4, pp. 686-705, 2004.

[22] Intel, *Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor – White Paper*, March 2004.

[23] S. U. Khan and I. Ahmad, “A cooperative game theoretical technique for joint optimization of energy consumption and response time in computational grids," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 3, pp. 346-360, 2009.

[24] C. M. Krishna and Y.-H. Lee, “Voltage-clock-scaling adaptive scheduling techniques for low power in hard real-time systems," *IEEE Transactions on Computers*, vol. 52, no. 12, pp. 1586-1593, 2003.

[25] W.-C. Kwon and T. Kim, “Optimal voltage allocation techniques for dynamically variable voltage processors," *ACM Transactions on Embedded Computing Systems*, vol. 4, no. 1, pp. 211-230, 2005.

[26] Y. C. Lee and A. Y. Zomaya, “Energy conscious scheduling for distributed computing systems under different operating conditions," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 8, pp. 1374-1381, 2011.

[27] Y.-H. Lee and C. M. Krishna, “Voltage-clock scaling for low energy consumption in fixed-priority real-time systems," *Real-Time Systems*, vol. 24, no. 3, pp. 303-317, 2003.

[28] K. Li, “Performance analysis of power-aware task scheduling algorithms on multiprocessor computers with dynamic voltage and speed," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 11, pp. 1484-1497, 2008.

[29] K. Li, “Energy efficient scheduling of parallel tasks on multiprocessor computers," *Journal of Supercomputing*, vol. 60, no. 2, pp. 223-247, 2012.

[30] K. Li, “Scheduling precedence constrained tasks with reduced processor energy on multiprocessor computers," *IEEE Transactions on Computers*, vol. 61, no. 12, pp. 1668-1681, 2012.

[31] K. Li, “Power allocation and task scheduling on multiprocessor computers with energy and time constraints," *Energy-Efficient Distributed Computing Systems*, A. Y. Zomaya and Y. C. Lee, eds., Chapter 1, pp. 1-37, John Wiley & Sons, 2012.

[32] K. Li, “Algorithms and analysis of energy-efficient scheduling of parallel tasks," *Handbook of Energy-Aware and Green Computing*, Vol. 1 (Chapter 15), I. Ahmad and S. Ranka, eds., pp. 331-360, CRC Press/Taylor & Francis Group, 2012.

[33] M. Li, B. J. Liu, and F. F. Yao, “Min-energy voltage allocation for tree-structured tasks," *Journal of Combinatorial Optimization*, vol. 11, pp. 305-319, 2006.

[34] M. Li, A. C. Yao, and F. F. Yao, “Discrete and continuous min-energy schedules for variable voltage processors," *Proceedings of the National Academy of Sciences USA*, vol. 103, no. 11, pp. 3983-3987, 2006.

[35] M. Li and F. F. Yao, "An efficient algorithm for computing optimal discrete voltage schedules," *SIAM Journal on Computing*, vol. 35, no. 3, pp. 658-671, 2006.

[36] J. R. Lorch and A. J. Smith, "PACE: a new approach to dynamic voltage scaling," *IEEE Transactions on Computers*, vol. 53, no. 7, pp. 856-869, 2004.

[37] R. N. Mahapatra and W. Zhao, "An energy-efficient slack distribution technique for multimode distributed real-time embedded systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 7, pp. 650-662, 2005.

[38] B. C. Mochocki, X. S. Hu, and G. Quan, "A unified approach to variable voltage scheduling for nonideal DVS processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 9, pp. 1370-1377, 2004.

[39] G. Quan and X. S. Hu, "Energy efficient DVS schedule for fixed-priority real-time systems," *ACM Transactions on Embedded Computing Systems*, vol. 6, no. 4, Article no. 29, 2007.

[40] N. B. Rizvandi, J. Taheri, and A. Y. Zomaya, "Some observations on optimal frequency selection in DVFS-based energy consumption minimization," *Journal of Parallel Distributed Computing*, vol. 71, no. 8, pp. 1154-1164, 2011.

[41] C. Rusu, R. Melhem, D. Mossé, "Maximizing the system value while satisfying time and energy constraints," *Proceedings of the 23rd IEEE Real-Time Systems Symposium*, pp. 256-265, 2002.

[42] D. Shin and J. Kim, "Power-aware scheduling of conditional task graphs in real-time multiprocessor systems," *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 408-413, 2003.

[43] D. Shin, J. Kim, and S. Lee, "Intra-task voltage scheduling for low-energy hard real-time applications," *IEEE Design & Test of Computers*, vol. 18, no. 2, pp. 20-30, 2001.

[44] M. R. Stan and K. Skadron, "Guest editors' introduction: power-aware computing," *IEEE Computer*, vol. 36, no. 12, pp. 35-38, 2003.

[45] O. S. Unsal and I. Koren, "System-level power-aware design techniques in real-time systems," *Proceedings of the IEEE*, vol. 91, no. 7, pp. 1055-1069, 2003.

[46] V. Venkatachalam and M. Franz, "Power reduction techniques for microprocessor systems," *ACM Computing Surveys*, vol. 37, no. 3, pp. 195-237, 2005.

[47] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," *Proceedings of the 1st USENIX Symposium on Operating Systems Design and Implementation*, pp. 13-23, 1994.

[48] P. Yang, C. Wong, P. Marchal, F. Catthoor, D. Desmet, D. Verkest, and R. Lauwereins, “Energy-aware runtime scheduling for embedded-multiprocessor SOCs," *IEEE Design & Test of Computers*, vol. 18, no. 5, pp. 46-58, 2001.

[49] F. Yao, A. Demers, and S. Shenker, “A scheduling model for reduced CPU energy," *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science*, pp. 374-382, 1995.

[50] H.-S. Yun and J. Kim, “On energy-optimal voltage scheduling for fixed-priority hard real-time systems," *ACM Transactions on Embedded Computing Systems*, vol. 2, no. 3, pp. 393-430, 2003.

[51] B. Zhai, D. Blaauw, D. Sylvester, and K. Flautner, “Theoretical and practical limits of dynamic voltage scaling," *Proceedings of the 41st Design Automation Conference*, pp. 868-873, 2004.

[52] X. Zhong and C.-Z. Xu, “Energy-aware modeling and scheduling for dynamic voltage scaling with statistical real-time guarantee," *IEEE Transactions on Computers*, vol. 56, no. 3, pp. 358-372, 2007.

[53] D. Zhu, R. Melhem, and B. R. Childers, “Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 7, pp. 686-700, 2003.

[54] D. Zhu, D. Mossé, and R. Melhem, “Power-aware scheduling for AND/OR graphs in real-time systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 9, pp. 849-864, 2004.

[55] J. Zhuo and C. Chakrabarti, “Energy-efficient dynamic task scheduling algorithms for DVS systems," *ACM Transactions on Embedded Computing Systems*, vol. 7, no. 2, Article no. 17, 2008.

[56] Z. Zong, A. Manzanares, X. Ruan, and X. Qin, “EAD and PEBD: two energy-aware duplication scheduling algorithms for parallel tasks on homogeneous clusters," *IEEE Transactions on Computers*, vol. 60, no. 3, pp. 360-374, 2011.