JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 21, 309-326 (2005)

Scheduling Precedence Constrained Parallel Tasks on Multiprocessors Using the Harmonic System Partitioning Scheme

KEQIN LI
Department of Computer Science
State University of New York
New Paltz, New York 12561, U.S.A.
E-mail: lik@newpaltz.edu

We present an algorithm for scheduling precedence constrained parallel tasks on multiprocessors with noncontiguous processor allocation. The algorithm is called LLH_m (Level-by-level and List scheduling using the Harmonic system partitioning scheme), where $m \geq 1$ is a positive integer, which is a parameter for the harmonic system partitioning scheme. Three basic techniques are employed in algorithm LLH_m. First, a task graph is divided into levels, and tasks are scheduled level by level to follow the precedence constraints. Second, tasks in the same level are scheduled using algorithm H_m developed in [16] for scheduling independent parallel tasks. The list scheduling method is used to implement algorithm H_m. Third, the harmonic system partitioning scheme is used for processor allocation. It is shown here that for wide task graphs and some common task size distributions, as the size of a computation and m increase, and as the task sizes become smaller, the average-case performance ratio of algorithm LLH_m approaches one.

Keywords: asymptotic average-case performance ratio, harmonic system partitioning scheme, parallel task, precedence constraint, task scheduling, wide task graph

1. INTRODUCTION

A parallel computation that consists of precedence constrained parallel tasks executed on a multiprocessor can be specified as a quintuple $C = \langle P, T, \prec, \pi, \tau \rangle$. P is the number of processors available; that is, P identical processors are allocated to the parallel computation C. $T = \{T_1, T_2, \ldots, T_n\}$ is a set of parallel tasks. $\prec$ is a partial order (or a set of precedence constraints) on T; i.e., if $T_i \prec T_j$, then task T_j cannot start to execute until task T_i finishes. $\pi : T \rightarrow [1 \ldots P]$ gives the processor requirements of the tasks; i.e., $\pi(T_i)$ is the number of processors needed to execute task T_i, where $\pi(T_i)$ is also called the size of T_i. $\tau : T \rightarrow (0, +\infty)$ describes the execution times of the tasks; i.e., $\tau(T_i)$ is the execution time of task T_i. Given a specification of a parallel computation $C = \langle P, T, \prec, \pi, \tau \rangle$, the problem of *scheduling precedence constrained parallel tasks on multiprocessors* is to find a nonpreemptive schedule of the tasks in T on a multiprocessor system with P processors, such that the schedule length (i.e., the completion time of the n tasks) is minimized. Notice that to execute task T_i, any $\pi(T_i)$ of the P processors can be allocated to T_i. This scheduling problem has applications in parallel computing systems, such as sym-

Received March 25, 2003; revised May 21, 2004; accepted July 9, 2004.
Communicated by Chu-Sing Yang.

metric shared memory multiprocessors, and even in distributed computing systems, such as bus-connected networks of workstations. In these systems, a processor allocation mechanism is independent of the topology of an interconnection network. To make our scheduling model simple and manageable, we assume that intertask communication times, synchronization costs, and the effect of shared resource contention are already included in task execution times.

As indicated in [17], the above scheduling problem includes many well known problems in operations research and computer science. When each task requires only one processor and executes in unit time, i.e., $\pi(T_i) = 1$ and $\tau(T_i) = 1$ for all $1 \le i \le n$, the problem becomes the *precedence constrained scheduling* problem [13, 21]. When there is no precedence constraint, i.e., $\prec$ is empty, and each task requires only one processor, the problem reduces to the classic *multiprocessor scheduling* problem for sequential tasks [6, 9]. When there is no precedence constraint and all tasks execute in unit time, the problem reduces to the *one dimensional bin packing* problem [11, 12]. All these problems are NP-hard [5] and have been studied extensively in the literature (see [2, 7, 8] for surveys and more references on these and related problems). These three problems imply that our scheduling problem is substantially more difficult than other similar scheduling problems. This difficulty is due to precedence constraints among tasks, inherent difficulty in scheduling (even independent) tasks, and processor allocation in multiprocessors. Any one of the three issues alone makes our problem NP-hard. In other words, we are facing a scheduling problem with both precedence and processor constraints. An efficient heuristic algorithm for tackling this problem should employ good strategies for handling all three difficult issues.

Notice that our problem is different from the ones studied in [20, 22], where tasks are malleable, i.e., each task may be executed with various numbers of processors, and execution times are adjusted accordingly to reflect various speedup assumptions. The problems involved in scheduling malleable parallel tasks have been investigated by many researchers in recent years, and a large body of literature exists. Our problem is also different from the one studied in [18], where it is required that $\pi(T_i)$ contiguous processors be allocated to task T_i. Such a requirement makes the problem even more difficult to solve. The special case for scheduling precedence constrained sequential tasks has been studied extensively [6, 9, 10, 14]. The special case for scheduling independent parallel tasks has recently been investigated in [16].

A feasible and effective way to solve NP-hard problems is to use heuristic (or approximation) algorithms that produce near-optimal solutions. Let $A(C)$ be the makespan of the schedule generated by an algorithm A for a parallel computation C, and let $\mathrm{OPT}(C)$ be the makespan of an optimal schedule of C. The quantity $R_A = \sup_C(A(C)/\mathrm{OPT}(C))$ is called the *absolute worst-case performance ratio* of algorithm A. The quantity

$$R_A^\infty = \lim_{Z\to\infty}\left(\sup_{\mathrm{OPT}(C)/\tau^*=Z}\left(\frac{A(C)}{\mathrm{OPT}(C)}\right)\right)$$

is called the *asymptotic worst-case performance ratio* of algorithm A, where τ^* is the longest execution time of the n tasks. Our definition implies that we are interested in the asymptotic behaviour of an algorithm when the individual task execution times become

less and less significant compared to the overall execution time of a parallel computation as the size of the parallel computation increases. If there exist two constants α and β such that for all C, $A(C) \le \alpha \cdot \mathrm{OPT}(C) + \beta\tau^*$, then $R_A^\infty \le \alpha$, and α is called an *asymptotic worst-case performance bound* of algorithm A. Moreover, if for any small $\varepsilon > 0$ and all large $Z > 0$, there exists C, such that $\mathrm{OPT}(C)/\tau^* \ge Z$ and $A(C) \ge (\alpha - \varepsilon)\mathrm{OPT}(C)$, then the bound α is called tight, i.e., $R_A^\infty = \alpha$. When task sizes and execution times are random variables, both $A(C)$ and $\mathrm{OPT}(C)$ become random variables, and $\bar{R}_A^n = E(A(C))/E(\mathrm{OPT}(C))$ is called the *average-case performance ratio* of algorithm A for n tasks, where $E(\cdot)$ stands for the expectation of a random variable. The quantity

$$\bar{R}_A^\infty = \lim_{n\to\infty}\left(\frac{E(A(C))}{E(\mathrm{OPT}(C))}\right)$$

is called the *asymptotic average-case performance ratio* of algorithm A. Of course, $\bar{R}_A^\infty$ depends on the probability distributions of the task sizes and execution times. If there exists a constant γ such that for all C, $E(A(C)) \le \gamma \cdot E(\mathrm{OPT}(C))$ as $n \to \infty$, then $\bar{R}_A^\infty \le \gamma$, and γ is called an *asymptotic average-case performance bound* of algorithm A.

It is still not clear whether there exists an approximation algorithm A which has a finite asymptotic worst-case performance ratio R_A^∞ or finite asymptotic average-case performance ratio $\bar{R}_A^\infty$ for scheduling precedence constrained parallel tasks on multiprocessors. In [15], the author showed that for many heuristic choices of the initial priority list, the list scheduling (LS) algorithm has unbounded R_{LS}. However, it was also shown that when task sizes are bounded from above by a fraction of P, the list scheduling algorithm has finite R_{LS}. In particular, if $\pi(T_i) \le qP$ for all $1 \le i \le n$, where $1/P \le q \le 1$ is a constant, then we have

$$R_{\mathrm{LS}} \le \frac{(2-q)P}{(1-q)P+1}.$$

A similar bound of $(2 - q)/(1 - q)$ was also obtained in [3]. Nevertheless, list scheduling algorithms are fundamentally limited due to their inability to handle precedence constraints, task execution times, and task sizes simultaneously.

The problem of *scheduling precedence constrained parallel tasks on multicomputers with contiguous processor allocation* has recently been considered in [18]. In this problem, there are P identical processors, say, $M_1, M_2, \ldots, M_P$, allocated to a parallel computation C. It is required that $\pi(T_i)$ contiguous processors, i.e., $M_k, M_{k+1}, M_{k+2}, \ldots, M_{k+\pi(T_i)-1}$, for some k, be allocated to a task T_i. The contiguous processor allocation requirement is based on the fact that processors in a multicomputer system are connected by a certain network, e.g., a linear array, which is the simplest topology. Therefore, the processors allocated to a task should be able to form a subsystem that has the same topology as the original system. In [18], an algorithm called LLB (*L*evel-by-level and *L*argest-task-first scheduling with a *B*inary system partitioning scheme) was proposed. Three basic techniques are employed in algorithm LLB, namely, level by level scheduling for handling the precedence constraints, the largest-task-first strategy for scheduling

independent parallel tasks on each level, and the binary system partitioning scheme for processor allocation and to support implementation of the largest-task-first scheduling algorithm.

While algorithm LLB can certainly be used to solve the problem considered in this paper, that is, scheduling precedence constrained parallel tasks on multiprocessors with noncontiguous processor allocation, it has several limitations. For example, the binary system partitioning scheme requires that the system size P be a power of 2. Strictly speaking, algorithm LLB only solves a special case of our scheduling problem. Since the binary system partitioning scheme is mainly designed for contiguous processor allocation and since we only consider noncontiguous processor allocation, there are opportunities to improve the performance by using more efficient system partitioning and processor allocation schemes. The main limitation of algorithm LLB is that its average-case performance does not improve when task sizes are small, since the amount of internal fragmentation caused by the binary system partitioning scheme does not decrease as tasks become small. Intuitively, better heuristics exist that are able to produce better schedules for small tasks.

In this paper, we present an algorithm for scheduling precedence constrained parallel tasks on multiprocessors with noncontiguous processor allocation. The algorithm is called LLH_m (*L*evel-by-level and *L*ist scheduling using the *H*armonic system partitioning scheme), where $m \geq 1$ is a positive integer, which is a parameter for the harmonic system partitioning scheme. Three basic techniques are employed in algorithm LLH_m to handle three difficult issues in our scheduling problem.

- First, similar to algorithm LLB, a task graph is divided into levels, and tasks are scheduled level by level to follow the precedence constraints.
- Second, tasks in the same level are scheduled using algorithm H_m, which was developed in [16] for scheduling independent parallel tasks. The list scheduling method is used to implement algorithm H_m.
- Third, algorithm H_m uses the harmonic system partitioning scheme as the processor allocation strategy since it is more efficient than the binary system partitioning scheme.

It will be shown that for wide task graphs and some common task size distributions, as the size of a computation n and m increase and the task sizes become smaller, the average-case performance ratio $\bar{R}^n_{\mathrm{LLH}_m}$ of algorithm LLH_m approaches one. That is, if $\pi(T_i) \leq P/r$ for all $1 \leq i \leq n$, where $r \geq 1$ is an integer, then we have $\lim_{m\to\infty} \bar{R}^\infty_{\mathrm{LLH}_m} = 1$ as $r \to \infty$.

It is worth noting that for unit-time tasks, an asymptotic worst-case performance bound of 2.7 can be achieved [4]. It should also be noted that the level-by-level scheduling method was used in [1] to schedule parallel tasks with identical execution times.

We review algorithm H_m and its performance in section 2. Algorithm LLH_m is presented and its average-case performance is analyzed in section 3. We give some example task size distributions in section 4 and show the quality of the performance bounds. In section 5, we demonstrate by numerical data the performance of algorithm LLH_m in scheduling several typical wide task graphs. Section 6 concludes the paper.

2. ALGORITHM H_m AND ITS PERFORMANCE

To schedule a list $L = (T_1, T_2, \ldots, T_n)$ of n independent parallel tasks, algorithm H_m divides L into m sublists $L_1, L_2, \ldots, L_m$ according to the task sizes (i.e., the numbers of processors requested by tasks), where $m \geq 1$ is a positive integer. For $1 \leq k \leq m - 1$, we define $L_k = \{T_i \in L \mid P/(k + 1) < \pi(T_i) \leq P/k\}$; i.e., L_k contains all the tasks in L that have sizes in the interval $I_k = (P/(k + 1), P/k]$. We define $L_m = \{T_i \in L \mid 0 < \pi(T_i) \leq P/m\}$; i.e., L_m contains all the tasks whose sizes are in the range $I_m = (0, P/m]$. The idea behind the harmonic system partitioning scheme is to schedule tasks of similar sizes together. The similarity is defined by the intervals $I_1, I_2, \ldots, I_k, \ldots, I_m$. For tasks in L_k, processor utilization is higher than $k/(k + 1)$. As k increases, the similarity among tasks in L_k increases, and processor utilization also increases. Hence, the harmonic system partitioning scheme is very good at handling small tasks.

Algorithm H_m produces schedules of L_k's sequentially and separately. To process tasks in L_k, where $1 \leq k \leq m - 1$, the P processors are partitioned into k groups, $G_1, G_2, \ldots, G_k$, each of which contains P/k processors. Each group G_j of processors is treated as a unit and is assigned to a task in L_k. This is basically the harmonic system partitioning and processor allocation scheme. Such an allocation can be implemented using, for example, the list scheduling algorithm [6]. Suppose $L_k = (T_1^k, T_2^k, \ldots, T_{n_k}^k)$, where n_k is the number of tasks in L_k. Initially, group G_j is assigned to T_j^k, where $1 \leq j \leq k$, and $T_1^k, T_2^k, \ldots, T_k^k$ are removed from L_k. Upon completion of a task T_j^k, the first unscheduled task in L_k, i.e., T_{k+1}^k, is removed from L_k and scheduled to execute on G_j. This process is repeated until all the tasks in L_k are finished. Then, algorithm H_m begins scheduling the next sublist L_{k+1}.

For L_m, there is no need to divide the P processors. The list scheduling algorithm is again employed here. Let $L_m = (T_1^m, T_2^m, \ldots, T_{n_m}^m)$. Initially, as many tasks in L_m are scheduled as possible; i.e., tasks $T_1^m, T_2^m, \ldots, T_s^m$ start to execute, where s is defined in such a way that the total size of $T_1^m, T_2^m, \ldots, T_s^m$ is no larger than P, but the total size of $T_1^m, T_2^m, \ldots, T_{s+1}^m$ exceeds P. When a task finishes, the next task in L_m begins to execute, provided that there are enough idle processors. Notice that the scheduling of tasks in L_m takes advantage of noncontiguous processor allocation.

Let $H_m(L)$ be the makespan of the schedule produced by algorithm H_m for L, and let $\mathrm{OPT}(L)$ be the makespan of an optimal schedule of L. The worst-case performance of algorithm H_m is given by the following theorem [16].

Theorem 1 For any list L of n tasks and $m \geq 3$, we have

$$\mathrm{H}_m(L) \leq 1\frac{13}{18}\mathrm{OPT}(L) + (m-1)\tau^*.$$

Furthermore, for $m \geq 6$ and any large $Z > 0$, there exists L, such that $\mathrm{OPT}(L)/\tau^* \geq Z$ and $\mathrm{H}_m(L)/\mathrm{OPT}(L) = 1\frac{2}{3}$. Therefore, for all $m \geq 6$, we have $1.666\ldots \leq R_{\mathrm{H}_m}^{\infty} \leq 1.722\ldots$.

For tasks with small sizes, algorithm H_m exhibits much better performance due to increased processor utilization in the harmonic system partitioning scheme, as claimed by the following theorem [16].

Theorem 2 For any list L of n tasks, such that $\pi(T_i) \le P/r$ for all $1 \le i \le n$, where $r > 1$ is an integer, we have

$$\mathrm{H}_m(L) \le \left(1+\frac{1}{r}\right)\mathrm{OPT}(L)+(m-r+1)\tau^*.$$

Furthermore, for $m \ge r + 1$ and any large $Z > 0$, there exists L, such that $\mathrm{OPT}(L)/\tau^* \ge Z$ and $\mathrm{H}_m(L)/\mathrm{OPT}(L) = 1 + 1/(r + 1)$. Therefore, we have $1 + 1/(r + 1) \le R^{\infty}_{\mathrm{H}_m} \le 1 + 1/r$.

The asymptotic worst-case performance ratio of algorithm H_m is better than that of the largest-task-first (LTF) scheduling algorithm supported by the binary system partitioning scheme [18] in scheduling independent parallel tasks. Due to internal fragmentation in the binary system partitioning scheme, about half of the allocated processors are wasted in the worst case. Hence, the asymptotic worst-case performance ratio of algorithm LTF is $R^{\infty}_{\mathrm{LTF}} = 2$ and does not improve when task sizes are small. However, as task sizes become smaller, the asymptotic worst-case performance ratio of algorithm H_m gets better and eventually approaches one.

Now, let us consider the average-case performance of algorithm H_m. For the purpose of probabilistic analysis, we make the following assumptions.

(A1) We assume that the task sizes are normalized such that $0 < \pi(T_i) \le 1$, and that the $\pi(T_i)$'s are independent and identically distributed (i.i.d.) random variables with a common probability density function $f(x)$ in the range (0, 1].

(A2) Our assumption about the task execution times is quite general; i.e., the $\tau(T_i)$'s are i.i.d. random variables with mean μ and variance σ^2, where μ and σ are any finite numbers independent of n. Let $c = \sigma/\mu$ denote the coefficient of variation.

(A3) The probability distributions of task sizes and execution times are independent of each other.

The average-case performance of algorithm H_m is characterized by

$$E(\mathrm{OPT}(L)) \ge \max\left(\int_0^1 xf(x)dx,\ \int_{\frac{1}{2}}^1 f(x)dx\right)n\mu, \tag{1}$$

$$\begin{aligned} E(\mathrm{H}_m(L)) \le & \left(\sum_{k=1}^{m-1}\frac{1}{k}\int_{\frac{1}{k+1}}^{\frac{1}{k}} f(x)dx+\frac{m}{m-1}\int_0^{\frac{1}{m}} xf(x)dx+\frac{m-1}{n}\right)n\mu \\ & +\left(\frac{\sqrt{2}}{3}m^{1.5}+\sqrt{\frac{n}{2}}\right)\sigma, \end{aligned} \tag{2}$$

and the following theorem [16].

Theorem 3 We have the following asymptotic average-case performance bound for algorithm H_m:

$$\bar{R}_{\mathrm{H}_m}^{\infty} = \lim_{n\to\infty}\left(\frac{E(\mathrm{H}_m(L))}{E(\mathrm{OPT}(L))}\right) \le \frac{\displaystyle\sum_{k=1}^{m-1}\frac{1}{k}\int_{\frac{1}{k+1}}^{\frac{1}{k}} f(x)dx + \frac{m}{m-1}\int_0^{\frac{1}{m}} xf(x)dx}{\max\left(\displaystyle\int_0^1 xf(x)dx,\ \int_{\frac{1}{2}}^1 f(x)dx\right)}.$$

(Note that the bound only depends on $f(x)$.)

3. ALGORITHM LLH$_m$ AND ITS AVERAGE-CASE PERFORMANCE

The structure of a parallel computation $C = \langle P, T, \prec, \pi, \tau\rangle$ can be represented by a directed acyclic graph $G = (T, \prec)$, where nodes stand for tasks in T, and arcs stand for precedence constraints in $\prec$. Strictly speaking, since the size n of C is a variable, G represents a family of graphs. A directed acyclic task graph can be decomposed into levels, which are denoted by $V_1, V_2, \ldots, V_L$. Tasks with no predecessors (called initial tasks) constitute level 1. Generally, a task T_i is in level V_l if the number of nodes on the longest path from some initial task to T_i is l. Note that all the tasks in the same level are independent of each other; hence, they can be scheduled using algorithm H_m. Let L be the number of levels in G, and let $n_l = |V_l|$ be the number of tasks in V_l, where $1 \le l \le L$. Define

$$\beta_n = \frac{L}{n}$$

and

$$\eta_n = \frac{1}{n}\left(\sqrt{\frac{n_1}{2}} + \sqrt{\frac{n_2}{2}} + \ \ldots\ + \sqrt{\frac{n_L}{2}}\right).$$

A task graph (actually, a family of graphs) G is said to be *wide* if

$$\beta_\infty = \lim_{n\to\infty}\frac{L}{n} = 0.$$

The above condition also implies that

$$\eta_\infty = \lim_{n\to\infty}\frac{1}{n}\left(\sqrt{\frac{n_1}{2}} + \sqrt{\frac{n_2}{2}} + \ \ldots\ + \sqrt{\frac{n_L}{2}}\right) \le \lim_{n\to\infty}\frac{L}{n}\sqrt{\frac{n}{2L}} = \sqrt{\frac{\beta_\infty}{2}} = 0.$$

A task graph is *narrow* if the above condition is not satisfied. Informally, wide task graphs exhibit large degree of parallelism.

Algorithm LLH$_m$ schedules tasks in T level by level. A task graph is processed in the order $V_1, V_2, \ldots, V_L$. Tasks in V_{l+1} cannot start to execute until all the tasks in V_l are finished. For level V_l, we use algorithm H_m to generate its schedule.

Let $\bar{\pi}$ be the mean task size, i.e.,

$$\bar{\pi} = \int_0^1 xf(x)dx,$$

and let b be the probability that the size of a task exceeds 1/2, i.e.,

$$b = \int_{\frac{1}{2}}^1 f(x)dx.$$

Define

$$D = \max(\bar{\pi}, b) = \max\left(\int_0^1 xf(x)dx,\ \int_{\frac{1}{2}}^1 f(x)dx \right).$$

Now, we will present the main result of the paper.

Theorem 4 Under assumptions (A1)-(A3) about task sizes and execution times, we have the following average-case performance bound for algorithm LLH_m:

$$\begin{aligned}\bar{R}^n_{\text{LLH}_m} &= \frac{E(\text{LLH}_m(C))}{E(\text{OPT}(C))} \\ &\le \frac{1}{D}\left(\sum_{k=1}^{m-1} \frac{1}{k} \int_{\frac{1}{k+1}}^{\frac{1}{k}} f(x)dx + \frac{m}{m-1} \int_0^{\frac{1}{m}} xf(x)dx + \beta_n (m-1) + \beta_n \frac{\sqrt{2}}{3} m^{1.5} c + \eta_n c \right).\end{aligned}$$

Remark: The above bound depends on three factors, namely, $f(x)$: the probability distribution of task sizes; μ and σ: parameters of task execution times; β_n and η_n: characterizations of a task graph.

Proof: Let the area of a task T_i be defined as $\pi(T_i)\tau(T_i)$, i.e., the product of its size and execution time. It is clear that OPT(C) is bounded from below by the total area of the tasks in T, that is, $E(\text{OPT}(C)) \ge n\bar{\pi}\mu$. Since the tasks with sizes exceeding 1/2 have to be executed sequentially, we have $E(\text{OPT}(C)) \ge bn\mu$. Thus, Eq. (1) can be easily extended to

$$E(\text{OPT}(C)) \ge \max(n\bar{\pi}\mu,\ bn\mu) = Dn\mu. \tag{3}$$

It is clear that the level-by-level scheduling method yields

$$E(\text{LLH}_m(C)) = E(\text{H}_m(V_1)) + E(\text{H}_m(V_2)) + \ldots + E(\text{H}_m(V_L)),$$

where, by Eq. (2),

$$E(\text{H}_m(V_l)) \le \left(\sum_{k=1}^{m-1} \frac{1}{k} \int_{\frac{1}{k+1}}^{\frac{1}{k}} f(x)dx + \frac{m}{m-1} \int_0^{\frac{1}{m}} xf(x)dx \right) n_l \mu + (m-1)\mu + \left(\frac{\sqrt{2}}{3} m^{1.5} + \sqrt{\frac{n_l}{2}} \right) \sigma.$$

Hence, we obtain the following bound for $E(\text{LLH}_m(C))$:

$$E(\text{LLH}_m(C)) \le \left(\sum_{k=1}^{m-1} \frac{1}{k} \int_{\frac{1}{k+1}}^{\frac{1}{k}} f(x)dx + \frac{m}{m-1} \int_0^{\frac{1}{m}} xf(x)dx \right) n\mu + L(m-1)\mu$$
$$+ L\frac{\sqrt{2}}{3} m^{1.5} \sigma + \left(\sum_{l=1}^{L} \sqrt{\frac{n_l}{2}} \right) \sigma. \tag{4}$$

Combining Eqs. (3) and (4), we get

$$\frac{E(\text{LLH}_m(C))}{E(\text{OPT}(C))} \le \frac{1}{D} \left(\sum_{k=1}^{m-1} \frac{1}{k} \int_{\frac{1}{k+1}}^{\frac{1}{k}} f(x)dx + \frac{m}{m-1} \int_0^{\frac{1}{m}} xf(x)dx + \frac{L}{n}(m-1) \right.$$
$$\left. + \left(\frac{L}{n} \right) \frac{\sqrt{2}}{3} m^{1.5} \left(\frac{\sigma}{\mu} \right) + \frac{1}{n} \left(\sum_{l=1}^{L} \sqrt{\frac{n_l}{2}} \right) \frac{\sigma}{\mu} \right).$$

This proves the theorem. ❑

It is clear that as $n \to \infty$, we get the following asymptotic average-case performance bound according to the definition of wide task graphs.

Corollary 1 For wide task graphs, we have the following asymptotic average-case performance bound for algorithm LLH_m:

$$\bar{R}_{\text{LLH}_m}^{\infty} = \lim_{n \to \infty} \left(\frac{E(\text{LLH}_m(C))}{E(\text{OPT}(C))} \right) \le \frac{\sum_{k=1}^{m-1} \frac{1}{k} \int_{\frac{1}{k+1}}^{\frac{1}{k}} f(x)dx + \frac{m}{m-1} \int_0^{\frac{1}{m}} xf(x)dx}{\max\left(\int_0^1 xf(x)dx,\ \int_{\frac{1}{2}}^1 f(x)dx \right)}.$$

(Note that the bound only depends on $f(x)$.)

The bound in Corollary 1 is exactly the same as that in Theorem 3. This means that as a wide task graph gets larger, there is more and more parallelism, and the performance of algorithm LLH_m in scheduling precedence constrained parallel tasks approaches that of algorithm H_m in scheduling independent parallel tasks.

4. EXAMPLE DISTRIBUTIONS

As an example, let us consider uniform distributions; that is, the $\pi(T_i)$'s are i.i.d. random variables uniformly distributed in the range $(0, 1/r]$, where $r \ge 1$ is a positive integer. That is, $f(x) = r$ for $0 < x \le 1/r$. (Notice that when P is sufficiently large, a discrete uniform distribution on $\{1, 2, \ldots, P/r\}$ can be treated as a continuous uniform distribution on $(0, 1/r]$.) The following result is a consequence of Theorem 4 and Corollary 1 by direct manipulations.

Corollary 2 If the $\pi(T_i)$'s are i.i.d. random variables uniformly distributed in the range $(0, 1/r]$, we have $\bar{R}_{\text{LLH}_m}^{\infty} \le B_r$, that is,

$$E(\text{LLH}_m(C)) \le B_r E(\text{OPT}(C))$$

as $n \to \infty$, where

$$B_r = 2r^2 \left[\frac{\pi^2}{6} - \frac{1}{r} - \left(1 + \frac{1}{2^2} + \cdots + \frac{1}{(r-1)^2} \right) \right]$$

as $m \to \infty$.

To show the quality of the average-case performance bound B_r in Corollary 2, we give the following numerical data:

$B_1 = 1.2898680\ldots$
$B_2 = 1.1594720\ldots$
$B_3 = 1.1088121\ldots$
$B_4 = 1.0823327\ldots$
$B_5 = 1.0661449\ldots$
$B_6 = 1.0552487\ldots$
$B_7 = 1.0474219\ldots$
$B_8 = 1.0415306\ldots$
$B_9 = 1.0369372\ldots$
$B_{10} = 1.0332559\ldots$

It is clear that $B_r < 1 + 1/r$ for all $r > 1$; i.e., B_r is less than the asymptotic worst-case performance bound in Theorem 2.

Though closed form solutions are not available, the average-case performance bounds of algorithm LLH_m could be calculated using Theorem 4 and Corollary 1 numerically for an arbitrary probability distribution of task sizes. For instance, let us consider a truncated exponential distribution, i.e.,

$$f(x) = \frac{\lambda e^{-\lambda x}}{1 - e^{-\lambda}},\ 0 < x \le 1.$$

Corollary 3 If the $\pi(T_i)$'s are i.i.d. random variables exponentially distributed in the range (0, 1], we have $\bar{R}^{\infty}_{\text{LLH}_m} \le B_\lambda$, that is,

$$E(\text{LLH}_m(C)) \le B_\lambda E(\text{OPT}(C))$$

as $n \to \infty$, where

$$B_\lambda = \frac{\displaystyle\sum_{k=1}^{\infty} \frac{1}{k} \cdot \frac{e^{-\lambda/(k+1)} - e^{-\lambda/k}}{1 - e^{-\lambda}}}{\displaystyle\frac{1}{\lambda} - \frac{e^{-\lambda}}{1 - e^{-\lambda}}}$$

as $m \to \infty$.

To show the average-case performance bound B_λ in Corollary 3, we choose λ in such a way that the mean task size

$$\bar{\pi} = \frac{1}{\lambda} - \frac{e^{-\lambda}}{1-e^{-\lambda}}$$

takes the values $1/(2r)$ for $r = 1, 2, \ldots, 10$, so that a comparison can be made between performance bounds of LLH$_m$ under the uniform and exponential distributions:

$\lambda = 0.0001813\ldots$, $B_\lambda = 1.2898305\ldots$
$\lambda = 3.5935119\ldots$, $B_\lambda = 1.2731064\ldots$
$\lambda = 5.9030000\ldots$, $B_\lambda = 1.2145755\ldots$
$\lambda = 7.9781077\ldots$, $B_\lambda = 1.1640016\ldots$
$\lambda = 9.9954411\ldots$, $B_\lambda = 1.1273400\ldots$
$\lambda = 11.9991145\ldots$, $B_\lambda = 1.1021181\ldots$
$\lambda = 13.9998370\ldots$, $B_\lambda = 1.0846745\ldots$
$\lambda = 15.9999711\ldots$, $B_\lambda = 1.0722076\ldots$
$\lambda = 17.9999950\ldots$, $B_\lambda = 1.0629307\ldots$
$\lambda = 19.9999991\ldots$, $B_\lambda = 1.0557653\ldots$

5. EXAMPLE TASK GRAPHS

In this section, we provide several example wide task graphs to demonstrate our bound in Theorem 4 for the average-case performance ratio of algorithm LLH$_m$. These task graphs have been extensively used to study precedence constrained parallel tasks [14, 18, 19, 23].

Example 1: Iterative Computations. An iterative computation is organized as a series of alternating sequential and parallel phases (see Fig. 1). There is a master task that is active in sequential phases, which represent synchronization, communication, collection, and distribution of partial results. There are s slave tasks that are active during parallel phases, which represent actual computations. There are numerous applications that fall into this category, such as relaxation methods for solving partial differential equations, iterative algorithms for solving linear and nonlinear systems of equations, searches for extreme values of functions, smoothing in image processing, etc. In an iterative computation with s slave tasks and t repetitions, we have $L = 2t + 1$, $n_l = 1$ for $l = 1, 3, 5, \ldots$, $n_l = s \geq 2$ for $l = 2, 4, 6, \ldots$, $n = (s + 1)t + 1$, and

$$\beta_n = \frac{2t+1}{(s+1)t+1},$$

$$\eta_n = \frac{1}{(s+1)t+1}\left(\frac{t+1}{\sqrt{2}} + t \cdot \sqrt{\frac{s}{2}}\right).$$

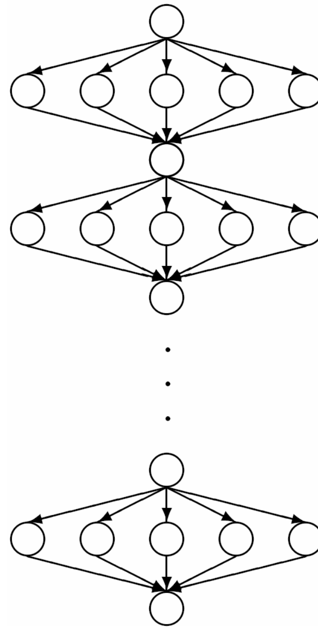

Fig. 1. An iterative computation with $s = 5$.

If we fix s and use t, the number of repetitions, as an indication of the size of a computation, then

$$\beta_\infty = \lim_{t\to\infty} \frac{2t+1}{(s+1)t+1} = \frac{2}{s+1} > 0$$

and

$$\eta_\infty = \lim_{t\to\infty} \frac{(\sqrt{s}+1)t+1}{\sqrt{2}((s+1)t+1)} = \frac{\sqrt{s}+1}{\sqrt{2}(s+1)} > 0;$$

that is, an iterative computation gives a narrow task graph. However, it is easy to see that if we fix t and use s to indicate the size of a computation, then we have $\beta_\infty = \eta_\infty = 0$; that is, an iterative computation has a wide task graph.

In Table 1, we show the bound for $\overline{R}^n_{\mathrm{LLH}_m}$ ($m = 20$) calculated using Theorem 4 in iterative computations with a uniform task size distribution in the range $(0, 1/r]$, where $r = 1, 2, \ldots, 10$. We assume that the coefficient of variation of task execution times is $c = \sigma/\mu = 1$. The number of iterations is $t = 100$, and the number of slave tasks is $s = 10000 + 100000z$ for $z = 0, 1, 2, \ldots, 9$. The following observations are made.

(O1) The performance bound is a decreasing function of the size of the computation. This means that the average-case performance of algorithm LLH_m improves when there is more parallelism.

Table 1. Bound for $\bar{R}^n_{\mathrm{LLH}_m}$ in scheduling iterative computations.

r	$z = 0$	$z = 1$	$z = 2$	$z = 3$	$z = 4$	$z = 5$	$z = 6$	$z = 7$	$z = 8$	$z = 9$
1	1.3288	1.2965	1.2942	1.2933	1.2928	1.2924	1.2922	1.2920	1.2918	1.2917
2	1.2376	1.1729	1.1684	1.1665	1.1655	1.1648	1.1643	1.1639	1.1636	1.1633
3	1.2262	1.1292	1.1224	1.1196	1.1181	1.1170	1.1163	1.1157	1.1153	1.1149
4	1.2392	1.1098	1.1008	1.0971	1.0950	1.0936	1.0926	1.0919	1.0913	1.0908
5	1.2627	1.1010	1.0897	1.0851	1.0825	1.0807	1.0795	1.0785	1.0778	1.0772
6	1.2917	1.0976	1.0841	1.0785	1.0754	1.0733	1.0718	1.0707	1.0698	1.0690
7	1.3239	1.0974	1.0817	1.0752	1.0715	1.0691	1.0673	1.0660	1.0650	1.0641
8	1.3582	1.0994	1.0814	1.0740	1.0698	1.0670	1.0650	1.0635	1.0623	1.0613
9	1.3941	1.1028	1.0826	1.0743	1.0695	1.0664	1.0642	1.0625	1.0611	1.0600
10	1.4309	1.1074	1.0849	1.0756	1.0704	1.0669	1.0644	1.0625	1.0610	1.0598

(O2) Since m is fixed at a relatively small value, the performance bound is not a decreasing function of r in some columns of the table.

(O3) When the size of the computation is sufficiently large, the performance bound approaches a value very close to B_r given in Corollary 2, and B_r is a decreasing function of r.

(O4) As the amount of parallelism and m increase, and as the task sizes become smaller, the average-case performance ratio of algorithm LLH_m approaches one.

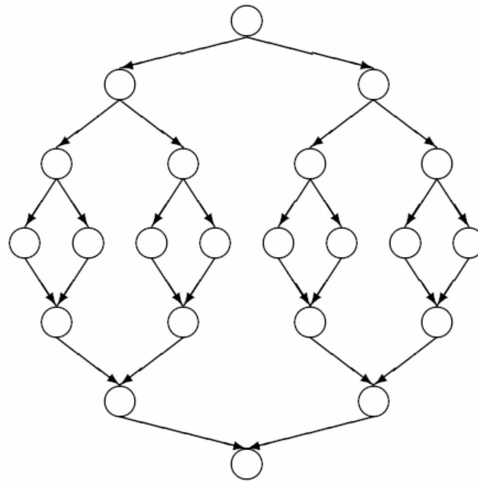

Fig. 2. A partitioning algorithm with $b = 2$ and $h = 3$.

Example 2: Partitioning Algorithms. A partitioning algorithm performs a divide-and-conquer computation. In a partitioning algorithm with branching factor $b \geq 2$ and height $h \geq 0$ (see Fig. 2, where $b = 2$ and $h = 3$), we have $L = 2h + 1$, $n_l = b^{l-1}$ for $l = 1, 2, 3, \ldots, h + 1$, and $n_l = b^{2h+1-l}$ for $l = h + 2, h + 3, \ldots, 2h + 1$. The height h indicates the size of a

computation. Levels V_1, V_2, …, V_h contain tasks which partition large problems into small subproblems. Level V_{h+1} includes tasks which solve base case problems. Levels V_{h+2}, V_{h+3}, …, V_{2h+1} have tasks which combine subsolutions into a grand solution. The parameters for the structure of the task graph are

$$n = b^0 + b^1 + b^2 + \cdots + b^{h-1} + b^h + b^{h-1} + \cdots + b^0 = \frac{b^{h+1} + b^h - 2}{b-1},$$

$$\beta_n = \frac{(2h+1)(b-1)}{b^{h+1} + b^h - 2},$$

$$\eta_n = \frac{1}{n}\left[2\left(\sqrt{\frac{1}{2}} + \sqrt{\frac{b}{2}} + \sqrt{\frac{b^2}{2}} + \cdots + \sqrt{\frac{b^{h-1}}{2}}\right) + \sqrt{\frac{b^h}{2}}\right]$$
$$= \left(\frac{b-1}{b^{h+1} + b^h - 2}\right)\left(\sqrt{2}\left(\frac{\sqrt{b^h} - 1}{\sqrt{b} - 1}\right) + \sqrt{\frac{b^h}{2}}\right)$$
$$= \left(\frac{\sqrt{b} + 1}{b^{h+1} + b^h - 2}\right)\left(\frac{\sqrt{b^{h+1}} + \sqrt{b^h} - 2}{\sqrt{2}}\right).$$

Since $\beta_\infty = \eta_\infty = 0$, a partitioning algorithm has a wide task graph.

In Table 2, we show the bound for $\bar{R}^n_{\mathrm{LLH}_m}$ (m = 20) calculated using Theorem 4 in partitioning algorithms with a uniform task size distribution in the range (0, $1/r$], where r = 1, 2, …, 10. We assume that the coefficient of variation of task execution times is $c = \sigma/\mu = 1$. The branching factor is b = 2, and the height is h = 16, 17, …, 25. It is clear that observations (O1)-(O4) of algorithm LLH_m in scheduling iterative computations also hold for partitioning algorithms.

Table 2. Bound for $\bar{R}^n_{\mathrm{LLH}_m}$ in scheduling partitioning algorithms.

r	h = 16	h = 17	h = 18	h = 19	h = 20	h = 21	h = 22	h = 23	h = 24	h = 25
1	1.3212	1.3084	1.3011	1.2968	1.2942	1.2927	1.2917	1.2911	1.2907	1.2905
2	1.2223	1.1968	1.1821	1.1735	1.1684	1.1653	1.1634	1.1622	1.1614	1.1609
3	1.2033	1.1650	1.1430	1.1301	1.1224	1.1178	1.1150	1.1132	1.1120	1.1112
4	1.2087	1.1579	1.1282	1.1111	1.1009	1.0947	1.0909	1.0885	1.0869	1.0859
5	1.2245	1.1607	1.1240	1.1025	1.0898	1.0821	1.0773	1.0743	1.0723	1.0711
6	1.2459	1.1693	1.1252	1.0994	1.0841	1.0749	1.0692	1.0656	1.0632	1.0617
7	1.2704	1.1811	1.1296	1.0996	1.0818	1.0710	1.0643	1.0601	1.0574	1.0556
8	1.2971	1.1950	1.1362	1.1019	1.0815	1.0692	1.0615	1.0567	1.0536	1.0516
9	1.3253	1.2104	1.1442	1.1056	1.0827	1.0688	1.0602	1.0548	1.0513	1.0491
10	1.3546	1.2269	1.1534	1.1105	1.0850	1.0696	1.0600	1.0540	1.0502	1.0476

Example 3: Linear Algebra Task Graphs. A linear algebra task graph with L levels (see Fig. 3, where $L = 5$) has $n_l = L - l + 1$ for $l = 1, 2, \ldots, L$, $n = L(L + 1)/2$, and

$$\beta_n = \frac{2}{L+1},$$

$$\eta_n = \frac{1}{n}\left(\sqrt{\frac{1}{2}} + \sqrt{\frac{2}{2}} + \sqrt{\frac{3}{2}} + \cdots + \sqrt{\frac{L}{2}}\right).$$

It is clear that a linear algebra task graph is wide. In Table 3, we show the bound for $\bar{R}^n_{\mathrm{LLH}_m}$ ($m = 20$) calculated using Theorem 4 in linear algebra task graphs with a uniform task size distribution in the range $(0, 1/r]$, where $r = 1, 2, \ldots, 10$. We assume that the coefficient of variation of task execution times is $c = \sigma/\mu = 1$. The number of levels is $L = 50000z$ for $z = 1, 2, \ldots, 10$.

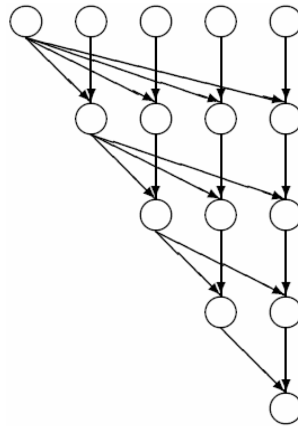

Fig. 3. A linear algebra task graph with $L = 5$.

Table 3. Bound for $\bar{R}^n_{\mathrm{LLH}_m}$ in scheduling linear algebra task graphs.

r	$z = 1$	$z = 2$	$z = 3$	$z = 4$	$z = 5$	$z = 6$	$z = 7$	$z = 8$	$z = 9$	$z = 10$
1	1.3033	1.2984	1.2965	1.2954	1.2947	1.2942	1.2938	1.2936	1.2933	1.2931
2	1.1865	1.1767	1.1728	1.1707	1.1693	1.1683	1.1676	1.1670	1.1665	1.1661
3	1.1496	1.1349	1.1291	1.1259	1.1239	1.1224	1.1213	1.1204	1.1197	1.1191
4	1.1371	1.1174	1.1098	1.1055	1.1028	1.1008	1.0993	1.0981	1.0972	1.0964
5	1.1350	1.1104	1.1009	1.0956	1.0921	1.0897	1.0878	1.0864	1.0852	1.0842
6	1.1384	1.1089	1.0975	1.0911	1.0870	1.0840	1.0818	1.0800	1.0786	1.0774
7	1.1451	1.1107	1.0973	1.0899	1.0851	1.0816	1.0790	1.0770	1.0753	1.0739
8	1.1539	1.1146	1.0993	1.0908	1.0853	1.0814	1.0784	1.0760	1.0741	1.0725
9	1.1642	1.1199	1.1027	1.0932	1.0870	1.0826	1.0792	1.0766	1.0744	1.0726
10	1.1755	1.1264	1.1073	1.0967	1.0898	1.0848	1.0811	1.0782	1.0758	1.0738

6. CONCLUDING REMARKS

We have investigated the problem of scheduling precedence constrained parallel tasks on multiprocessors. We observe that if a task graph is scheduled level by level, then the scheduling performance is determined by the algorithm for scheduling independent parallel tasks. The performance of algorithm H_m, which uses the harmonic system partitioning scheme for noncontiguous processor allocation, is superior to that of the largest-task-first algorithm, which uses the binary system partitioning scheme for contiguous processor allocation. In particular, $R^{\infty}_{H_m}$ approaches one when task sizes become smaller and smaller. Therefore, we have been able to develop an efficient algorithm LLH_m for scheduling precedence constrained parallel tasks in multiprocessors with noncontiguous processor allocation. It has been shown that for wide task graphs and some common task size distributions, as the size of a computation and m increase, and as the task sizes become smaller, the average-case performance ratio of algorithm LLH_m approaches one.

As a further research direction, it will be of great interest to see how our algorithm performs in scheduling real parallel computations on real parallel machines.

ACKNOWLEDGMENTS

The author wishes to thank two anonymous reviewers for their constructive comments in revising the paper. This material is based upon work supported by the US National Science Foundation under Grant No. CCR-0091719. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation. A preliminary version of the paper was presented on International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, Nevada, June 23-26, 2003.

REFERENCES

1. S. Bischof and E. W. Mayr, "On-line scheduling of parallel jobs with runtime restrictions," in *Proceedings of the 9th International Symposium on Algorithms and Computation*, LNCS, Vol. 1533, 1998, pp. 119-128.
2. M. Drozdowski, "Scheduling multiprocessor tasks − an overview," *European Journal of Operational Research*, Vol. 94, 1996, pp. 215-230.
3. A. Feldmann, M. Y. Kao, J. Sgall, and S. H. Teng, "Optimal online scheduling of parallel jobs with dependencies," in *Proceedings of the 25th ACM Symposium on Theory of Computing*, 1993, pp. 642-651.
4. M. R. Garey, R. L. Graham, D. S. Johnson, and A. C. C. Yao, "Resource constrained scheduling as generalized bin packing," *Journal of Combinatorial Theory* (*A*), Vol. 21, 1976, pp. 257-298.
5. M. R. Garey and D. S. Johnson, *Computers and Intractability − A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, 1979.
6. R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM Journal on Applied Mathematics*, Vol. 2, 1969, pp. 416-429.

7. R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," *Annals of Discrete Mathematics*, Vol. 5, 1979, pp. 287-326.
8. D. S. Hochbaum (ed.), *Approximation Algorithms for NP-Hard Problems*, PWS Publishing, Boston, Massachusetts, 1997.
9. D. S. Hochbaum and D. B. Shmoys, "Using dual approximation algorithms for scheduling problems: practical and theoretical results," *Journal of the ACM*, Vol. 34, 1987, pp. 144-162.
10. K. K. Jain and V. Rajaraman, "Lower and upper bounds on time for multiprocessor optimal schedules," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, 1994, pp. 879-886.
11. D. S. Johnson *et al.*, "Worst-case performance bounds for simple one-dimensional packing algorithms," *SIAM Journal on Computing*, Vol. 3, 1974, pp. 299-325.
12. N. Karmarkar and R. M. Karp, "An efficient approximation scheme for the one-dimensional bin packing problem," in *Proceedings of the 23rd Symposium on Foundations of Computer Science*, 1982, pp. 312-320.
13. J. K. Lenstra and A. H. G. R. Kan, "Complexity of scheduling under precedence constraints," *Operations Research*, Vol. 26, 1978, pp. 22-25.
14. K. Li, "Stochastic bounds for parallel program execution times with processor constraints," *IEEE Transactions on Computers*, Vol. 46, 1997, pp. 630-636.
15. K. Li, "Analysis of the list scheduling algorithm for precedence constrained parallel tasks," *Journal of Combinatorial Optimization*, Vol. 3, 1999, pp. 73-88.
16. K. Li, "Analysis of an approximation algorithm for scheduling independent parallel tasks," *Discrete Mathematics and Theoretical Computer Science*, Vol. 3, 1999, pp. 155-166.
17. K. Li and Y. Pan, "On scheduling precedence constrained parallel tasks on multiprocessors," in *Proceedings of International Conference on Modelling and Simulation*, 1996, pp. 63-75.
18. K. Li and Y. Pan, "Probabilistic analysis of scheduling precedence constrained parallel tasks on multicomputers with contiguous processor allocation," *IEEE Transactions on Computers*, Vol. 49, 2000, pp. 1021-1030.
19. S. Madala and J. B. Sinclair, "Performance of synchronous parallel algorithms with regular structures," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 2, 1991, pp. 105-116.
20. J. Turek *et al.*, "Scheduling parallelizable tasks to minimize average response time," in *Proceedings of the 6th ACM Symposium on Parallel Algorithms and Architectures*, 1994, pp. 200-209.
21. J. D. Ullman, "NP-complete scheduling problems," *Journal of Computer and System Science*, Vol. 10, 1975, pp. 384-393.
22. Q. Wang and K. H. Cheng, "A heuristic of scheduling parallel tasks and its analysis," *SIAM Journal on Computing*, Vol. 21, 1992, pp. 281-294.
23. N. Yazici-Pekergin and J. M. Vincent, "Stochastic bounds on execution times of parallel programs," *IEEE Transactions on Software Engineering*, Vol. 17, 1991, pp. 1005-1012.

Keqin Li (李克勤) is a full Professor of Computer Science in State University of New York at New Paltz. His research interests are mainly in design and analysis of algorithms, parallel and distributed computing, and computer networking, with particular interests in approximation algorithms, parallel algorithms, job scheduling, task dispatching, load balancing, performance evaluation, dynamic tree embedding, scalability analysis, parallel computing using optical interconnects, optical networks, and wireless networks. He has 190 research publications and has received several best paper awards.