Journal of Parallel and Distributed Computing 133 (2019) 210–220

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

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

Non-clairvoyant scheduling of independent parallel tasks on single and multiple multicore processors

Keqin Li

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

HIGHLIGHTS

- Worst-case performance bound for offline scheduling on a single multicore processor.
- Worst-case performance bound for offline scheduling on multiple multicore processors.
- Average-case performance bound for online scheduling on multiple multicore processors.

ARTICLE INFO

Article history:
Received 25 September 2017
Received in revised form 3 April 2018
Accepted 1 June 2018
Available online 23 June 2018

Keywords:
Multicore processor
Non-clairvoyant scheduling
Online scheduling
Parallel task
Performance bound
Task scheduling

ABSTRACT

We investigate the problem of non-clairvoyant scheduling of independent parallel tasks on single and multiple multicore processors. For a single multicore processor, we derive an asymptotic worst-case performance bound for a non-clairvoyant offline scheduling algorithm called *largest task first* (LTF). The result improves our previous result on a single parallel computing system. For multiple multicore processors, we derive an asymptotic worst-case performance bound for the LTF algorithm. To the best of our knowledge, there has been little result on scheduling parallel tasks on multiple parallel computing systems. For multiple multicore processors, we also derive an asymptotic average-case performance bound for a non-clairvoyant online scheduling algorithm called *random task first* (RTF). The result extends our earlier result on a single parallel computing system. Extensive simulation results are also demonstrated.

1. Introduction

1.1. Motivation

Since power consumption in a microprocessor is in the order of the clock frequency raised to the power of $\phi \geq 3$, increasing clock frequency and execution speed has led to the power density (Watts/cm^2) of a microprocessor comparative to that of a nuclear reactor and a rocket nozzle. Furthermore, the exponential growth of transistors per chip increases the capacitance and power consumption of a chip. We are putting more transistors on a chip than we can afford to turn them on. Power consumption has been the major limitation of further performance improvement of a single processor.

Multicore processors provide an ultimate solution to power management and performance optimization in current and future microprocessors. A multicore processor contains multiple independent processors, called cores, integrated onto a single circuit die (known as 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 $1/M^{\phi-1}$ of the energy of the single-core processor. All major microprocessor companies and vendors are producing multicore chips and dedicating their current and future development and manufacturing to multicore products.

The power and performance gain in a multicore processor is mainly from parallelism, i.e., multiple slower but less energy-consuming cores' working together to achieve the performance of a single faster but more energy-consuming processor. A multicore processor implements multiprocessing in a single physical package. It can implement various parallel computing 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, which is typically tens and even hundreds in the current technology. When M is large, a multicore processor is also called a many-core or a massively multicore processor.

As in all computing systems, increasing the utilization of a multicore processor becomes a critical issue as the number of cores

E-mail address: lik@newpaltz.edu.

https://doi.org/10.1016/j.jpdc.2018.06.001
0743-7315/

increases. 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 amount of time. Such performance enhancement is very important in optimizing the quality of service in a datacenter for cloud computing, where multicore processors are employed as servers.

When multicore processors are shared by a large number of users who submit independent applications and tasks, we are facing the problem of allocating the cores to the users and schedule the tasks such that the system performance is optimized. Such an optimization problem needs to be formulated and efficient algorithms need to be developed and their performance needs to be analyzed and evaluated.

1.2. Background information

In this paper, we investigate the problem of non-clairvoyant scheduling of independent parallel tasks on single and multiple multicore processors and servers.

Assume that we are given a list of n independent parallel tasks $L = (T_1, T_2, \ldots, T_n)$. A parallel task T_i is specified as $T_i = (s_i, t_i)$, where s_i is the number of cores requested by T_i, i.e., the *size* of T_i, and t_i is the execution time of T_i. We are also given a multicore processor with M identical cores. For such a single multicore processor, the problem is to find a nonpreemptive and non-clairvoyant schedule of the n independent parallel tasks on the M cores, such that the total execution time of the n tasks is minimized. To execute a task T_i, any s_i of the M cores can be allocated to T_i. Several tasks can be executed simultaneously, with the restriction that the total number of active cores (i.e., cores allocated to tasks being executed) at any moment cannot exceed M.

In a more general case, we are given m multicore processors with $M_1, M_2, \ldots, M_m$ cores respectively. All the $M = M_1 + M_2 + \cdots + M_m$ cores in the m multicore processors are identical. To execute a task T_i, any s_i of the M_j cores of the jth multicore processor can be allocated to T_i, and the task execution time of T_i is always t_i. However, core allocation cannot be performed across different multicore processors, i.e., all the s_i cores allocated to T_i must reside in the same multicore processor. (Notice that inter-processor communications take considerably more time than intra-processor communications. If a task T_i is executed by several multicore processors, the execution time is no longer t_i. The specification of the increased execution time is not clear.) For such multiple multicore processors, the problem is to find a nonpreemptive and non-clairvoyant schedule of the n independent parallel tasks on the m multicore processors with $M_1, M_2, \ldots, M_m$ cores, such that the total execution time of the n tasks is minimized.

In many applications, the execution time of a task is unavailable until the task is executed and completed. Although task execution times are included in the definition of our problem, we assume in this paper that for each task T_i, we only know its size s_i, i.e., the number of cores requested to execute the task. However, the execution time t_i of the task is unknown, due to its unpredictable input and other factors. A scheduling algorithm is *clairvoyant* if the algorithm knows the execution times of all tasks, and *non-clairvoyant* if the algorithm does not have any information of task execution times. Furthermore, in many applications, not all the tasks in a list L of tasks are available when a scheduling algorithm generates a schedule. A scheduling algorithm is *offline* if the algorithm knows the complete list of tasks, i.e., tasks can be executed in any order; and *online* if the algorithm must generate a schedule in the order of $T_1, T_2, \ldots, T_n$, i.e., when task T_i is scheduled, the algorithm does not have any information of future tasks $T_{i+1}, T_{i+2}, \ldots, T_n$.

The parallel task scheduling problem defined above is considered as a type of resource constrained scheduling problem [6,9], where the resource is a set of cores. The problem is NP-hard, since it includes two subproblems of core allocation and task scheduling. Each of these two subproblems alone makes the problem NP-hard. When all tasks are sequential, i.e., $s_i = 1$ for all $1 \le i \le n$, the problem becomes the classic multiprocessor scheduling problem [11]. When all tasks have identical execution time, i.e., $t_i = t$ for some t and for all $1 \le i \le n$, the problem becomes the classic bin packing problem [13]. Both of these two problems are well known NP-hard problems.

To solve an NP-hard optimization problem with realistically useful algorithms, one effective way is to develop fast algorithms that are able to produce near-optimal solutions. Let $A(L)$ denote the length of the schedule produced by an algorithm A for a list L of n independent parallel tasks, and $\mathrm{OPT}(L)$ the length of an optimal schedule with the minimum length. We define $t^* = \max(t_1, t_2, \ldots, t_n)$ to be the longest execution time of the parallel tasks in L. If

$$A(L) \le \alpha \mathrm{OPT}(L)$$

for all L, we call α an *absolute worst-case performance bound* of A. If

$$A(L) \le \alpha \mathrm{OPT}(L) + \gamma t^*$$

for all L, we call α an *asymptotic worst-case performance bound* of A. When task sizes and execution times are random variables, if

$$\mathbf{E}(A(L)) \le \beta \mathbf{E}(\mathrm{OPT}(L))$$

for all L with $n \to \infty$, where $\mathbf{E}(x)$ represents the expectation of a random variable x, we call β an *asymptotic average-case performance bound* of A.

1.3. Our contributions

Our main results in this paper are summarized as follows.

- For a single M-core processor, there is a non-clairvoyant offline scheduling algorithm, called the *largest task first* (LTF) algorithm, which has an asymptotic worst-case performance bound of
$$\alpha_k = 1 + \frac{(k+2)(k+3)+1}{(k+1)(k+2)^2},$$
where k is the largest integer such that all task sizes do not exceed M/k.
- For multiple multicore processors, the LTF algorithm has an asymptotic worst-case performance bound of
$$\alpha_{k,r} = \frac{k+1}{k-r+1},$$
where k is the largest integer such that all task sizes do not exceed M^*/k; M^* is the smallest number of cores; and r is the ratio of M^* to the average number of cores.
- For m multicore processors whose total number of cores is M, there is a non-clairvoyant online scheduling algorithm, called the *random task first* (RTF) algorithm, which has an asymptotic average-case performance bound of
$$\beta = \frac{M}{M - m((3-(1+1/D)^{D+1})D+1)},$$
where we assume that task sizes are uniformly distributed in the range $[1..D]$.

Our result of the LTF algorithm on a single multicore processor improves our previous result on a single parallel computing system [15]. To the best of our knowledge, there has been little result on scheduling parallel tasks on multiple parallel computing

systems [16]. Our result of the RTF algorithm on multiple multicore processors extends our earlier result on a single parallel computing system [17].

The remainder of the paper is organized as follows. In Section 2, we review related studies. In Section 3, we analyze the worst-case performance of the LTF algorithm on a single multicore processor. In Section 4, we analyze the worst-case performance of the LTF algorithm on multiple multicore processors. In Section 5, we analyze the average-case performance of the RTF algorithm on multiple multicore processors. Simulation results are also provided in Sections 3–5. We conclude the paper in Section 6.

2. Related work

Scheduling parallel tasks has been studied by a number of researchers (see [1] for a comprehensive survey and many recent studies [21–24]). In addition to the many variations to the problem, including preemptive and nonpreemptive schedules, dedicated and parallel processors, fixed and variable system sizes, malleable and non-malleable task sizes [12], we are more interested in the following cases.

Clairvoyant Offline Scheduling. When all the task execution times are available to a scheduling algorithm, the parallel task scheduling problem is very similar to the classic two-dimensional packing problem, which has been extensively studied [4,5,10], and an asymptotic worst-case performance bound as low as 1.25 can be obtained.

Non-clairvoyant Offline Scheduling. It has been known that no non-clairvoyant offline scheduling algorithm can have an absolute worst-case performance bound less than $2-1/M$ [19], and a simple non-clairvoyant offline scheduling algorithm called GREEDY can achieve an absolute worst-case performance bound of $2-1/M$ [8]. There is a non-clairvoyant offline scheduling algorithm called H_m, which can achieve an asymptotic worst-case performance bound of 1.7222 [15]. Furthermore, when task sizes are no greater than M/k for some integer $k \geq 2$, algorithm H_m can achieve an asymptotic worst-case performance bound of $1 + 1/k$. Notice that for algorithm LTF, we have $\alpha_1 = 1.7222$ and $\alpha_k < 1 + 1/k$ for all $k \geq 2$ (Theorem 1). Although algorithm LTF has the same asymptotic worst-case performance bound as that of algorithm H_m in the general case, LTF has an improved asymptotic worst-case performance bound when task sizes are small.

Non-clairvoyant Online Scheduling. It is easy to show that no clairvoyant or non-clairvoyant online parallel task scheduling algorithm can have a finite absolute or asymptotic worst-case performance bound. Therefore, research attention has been focused on the analysis of average-case performance of online scheduling algorithms. In [17], it was shown that there is a non-clairvoyant online scheduling algorithm called SIMPLE, which can achieve an asymptotic average-case performance bound of

$$\frac{M}{M-(3-(1+1/D)^{D+1})D-1},$$

assuming that task sizes are uniformly distributed in the range $[1..D]$. It is clear that the above result for a single system is extended to multiple systems (Theorem 3) in this paper.

Scheduling on Multiple Systems. Scheduling parallel tasks on multiple parallel computing systems has rarely been investigated before, primarily due to the sophistication of the problem. A similar but different scheduling problem in a grid computing environment was considered in [16], where execution of a parallel task across several systems is allowed. However, in this paper, each task must be executed on one multicore processor. Therefore, our result of the LTF algorithm on multiple multicore processors (Theorem 2) is the first attempt in this direction.

Scheduling on Multicore Processors. For multicore processors, research so far has been focused on scheduling independent or precedence constrained sequential tasks [2,3,14,20]. Little investigation has been done for parallel tasks.

Other Recent Studies. A hybrid multicore architecture combining CPUs and GPUs was considered in [7], where tasks are independent and sequential, and each task can be processed on either CPU or GPU with different execution times. An adaptive multicore architecture was considered in [18], which allows tasks to be malleable, i.e., the number of cores allocated per application is not fixed and can change during execution through preemption.

3. Scheduling on a single multicore processor

3.1. The LTF scheduling algorithm

The scheduling algorithm considered in this section is called *largest task first* (LTF). Algorithm LTF does not need the information of task execution times, i.e., $t_1, t_2, \ldots, t_n$, thus satisfying the requirement of non-clairvoyant scheduling. However, algorithm LTF needs all the information of task sizes, i.e., $s_1, s_2, \ldots, s_n$. Hence, it is an offline scheduling algorithm.

To schedule a list $L = (T_1, T_2, \ldots, T_n)$ of parallel tasks on a single multicore processor with M cores, algorithm LTF first sorts the n tasks in a nonincreasing order of task sizes. Without loss of generality, let us assume that $s_1 \geq s_2 \geq \cdots \geq s_n$. Algorithm LTF then schedules the n tasks in the order of $T_1, T_2, \ldots, T_n$. Initially, tasks in the beginning of L, say $T_1, T_2, \ldots, T_i$, are scheduled for execution, where i is as large as possible, i.e., the total size of $T_1, T_2, \ldots, T_i$ does not exceed M,

$$s_1 + s_2 + \cdots + s_i \leq M,$$

but the total size of $T_1, T_2, \ldots, T_i, T_{i+1}$ exceeds M,

$$s_1 + s_2 + \cdots + s_i + s_{i+1} > M.$$

Whenever a task is completed, algorithm LTF checks whether there are enough available cores (i.e., inactive and free cores not allocated to any task) for task T_{i+1}. If so, task T_i is scheduled for execution; otherwise, algorithm LTF waits for the next completion of a task. The above procedure is repeated until all tasks are scheduled.

Notice that task $T_{i'}$ cannot be scheduled earlier than T_i if $i' > i$. This restriction can be removed to improve the performance of algorithm LTF. In algorithm LTF*, initially and whenever a task is completed, algorithm LTF* examines all the remaining tasks and schedules all tasks that can be scheduled (i.e., there are enough available cores to accommodate these tasks).

3.2. Worst-case analysis

In this section, we analyze the asymptotic worst-case performance of algorithm LTF for non-clairvoyant offline scheduling of independent parallel tasks on a single multicore processor with M cores.

We use the notation LTF(L) to stand for the length of the schedule produced by algorithm LTF for a list L of n independent parallel tasks. Let $s^* = \max(s_1, s_2, \ldots, s_n)$ be the largest size of the n tasks. Recall that $t^* = \max(t_1, t_2, \ldots, t_n)$ is the longest execution time of the n tasks.

Our main result of this section is the following theorem.

Theorem 1. *To schedule any list L of independent parallel tasks on a single multicore processor with M cores by using the LTF algorithm, we have*

$$LTF(L) \leq \alpha_k OPT(L) + 4t^*,$$

where the asymptotic worst-case performance bound is

$$\alpha_k = 1 + \frac{(k+2)(k+3)+1}{(k+1)(k+2)^2},$$

and $k \geq 1$ is some integer such that $M/(k+1) < s^* \leq M/k$, i.e., $k = \lfloor M/s^* \rfloor$.

Proof. First, let us examine LTF(L). Assume that $M/(k+1) < s^* \leq M/k$ for some integer $k \geq 1$. We also assume that the execution of the n tasks starts at time 0, and all the tasks are executed during time interval $I = [0, \text{LTF}(L))$. We divide L into four sublists L_k, L_{k+1}, L_{k+2}, and L_{k+3}, where

- L_k includes all tasks T_i with $M/(k+1) < s_i \leq M/k$;
- L_{k+1} includes all tasks T_i with $M/(k+2) < s_i \leq M/(k+1)$;
- L_{k+2} includes all tasks T_i with $M/(k+3) < s_i \leq M/(k+2)$;
- L_{k+3} includes all tasks T_i with $0 < s_i \leq M/(k+3)$.

The list L is a simple concatenation of the four sublists, i.e., $L = L_k L_{k+1} L_{k+2} L_{k+3}$. Notice that some of the above sublists may be empty, depending on the values of M, k, and the s_i's.

We define the moments b_k, c_k, b_{k+1}, c_{k+1}, b_{k+2}, c_{k+2}, b_{k+3}, c_{k+3} as follows:

- b_k: the moment when the first task in L_k is scheduled for execution;
- c_k: the moment when all tasks in L_k complete their executions;
- b_{k+1}: the moment when the first task in L_{k+1} is scheduled for execution;
- c_{k+1}: the moment when all tasks in L_{k+1} complete their executions;
- b_{k+2}: the moment when the first task in L_{k+2} is scheduled for execution;
- c_{k+2}: the moment when all tasks in L_{k+2} complete their executions;
- b_{k+3}: the moment when the first task in L_{k+3} is scheduled for execution;
- c'_{k+3}: the moment when the task which completes last is scheduled for execution;
- c_{k+3}: the moment when all tasks in L_{k+3} complete their executions.

Therefore, tasks in L_k are executed during the time interval $[b_k, c_k)$; tasks in L_{k+1} are executed during the time interval $[b_{k+1}, c_{k+1})$; tasks in L_{k+2} are executed during the time interval $[b_{k+2}, c_{k+2})$; and tasks in L_{k+3} are executed during the time interval $[b_{k+3}, c_{k+3})$.

In a "normal" case, i.e., when each of L_k, L_{k+1}, L_{k+2}, L_{k+3} contains a large number of tasks, we have

$$b_k < b_{k+1} < c_k < b_{k+2} < c_{k+1} < b_{k+3} < c_{k+2} < c'_{k+3} < c_{k+3}.$$

The above moments divide the time interval $I = [0, \text{LTF}(L))$ into eight subintervals I_k, I'_k, I_{k+1}, I'_{k+1}, I_{k+2}, I'_{k+2}, I_{k+3}, I'_{k+3}, where $I_k = [b_k, b_{k+1})$, $I'_k = [b_{k+1}, c_k)$, $I_{k+1} = [c_k, b_{k+2})$, $I'_{k+1} = [b_{k+2}, c_{k+1})$, $I_{k+2} = [c_{k+1}, b_{k+3})$, $I'_{k+2} = [b_{k+3}, c_{k+2})$, $I_{k+3} = [c_{k+2}, c'_{k+3})$, $I'_{k+3} = [c'_{k+3}, c_{k+3})$. Let τ_k, τ'_k, τ_{k+1}, τ'_{k+1}, τ_{k+2}, τ'_{k+2}, τ_{k+3}, τ'_{k+3} denote the lengths of the subintervals I_k, I'_k, I_{k+1}, I'_{k+1}, I_{k+2}, I'_{k+2}, I_{k+3}, I'_{k+3}. Then, we have

$$\text{LTF}(L) = \tau_k + \tau'_k + \tau_{k+1} + \tau'_{k+1} + \tau_{k+2} + \tau'_{k+2} + \tau_{k+3} + \tau'_{k+3}.$$

We observe that at b_{k+1}, there are some tasks of L_k which are still in execution. The remaining execution time of these tasks cannot exceed t^*. Therefore, we have $\tau'_k \leq t^*$. Similarly, we have $\tau'_{k+1} \leq t^*$ and $\tau'_{k+2} \leq t^*$. The length of the subinterval I'_{k+3}, i.e., the execution time of the task in L_{k+3} which completes last, is also no longer than t^*, i.e., $\tau'_{k+3} \leq t^*$. Hence, we get

$$\text{LTF}(L) \leq \tau_k + \tau_{k+1} + \tau_{k+2} + \tau_{k+3} + 4t^*.$$

Notice that

- during time interval I_k, only tasks in L_k are in execution.
- during time interval I_{k+1}, only tasks in L_{k+1} are in execution;
- during time interval I_{k+2}, only tasks in L_{k+2} are in execution;
- during time interval I_{k+3}, only tasks in L_{k+3} are in execution.

In the above discussion, it is assumed that the four subintervals I'_k, I'_{k+1}, I'_{k+2}, I'_{k+3} do not overlap. If I'_k and I'_{k+1} overlap, then $\tau_{k+1} = 0$. If I'_{k+1} and I'_{k+2} overlap, then $\tau_{k+2} = 0$. If I'_{k+2} and I'_{k+3} overlap, then $\tau_{k+3} = 0$. It is also possible that $\tau_k = 0$.

Now, let us examine OPT(L). We have the following observations.

- Notice that during time interval I_k, there are constantly k tasks of L_k in execution. Hence, the total execution time of tasks in L_k is at least $k\tau_k$. Since the system can only accommodate k tasks of L_k simultaneously, we must have

  $$\text{OPT}(L) \geq k\tau_k/k = \tau_k,$$

  just to schedule tasks in L_k.
- Furthermore, during time interval I_{k+1}, there are constantly $k+1$ tasks of L_{k+1} in execution. Hence, the total execution time of tasks in L_k and L_{k+1} is at least $k\tau_k + (k+1)\tau_{k+1}$. Since the system can only accommodate $k+1$ tasks of L_k and L_{k+1} simultaneously, we must have

  $$\text{OPT}(L) \geq \frac{k\tau_k + (k+1)\tau_{k+1}}{k+1} = \left(\frac{k}{k+1}\right)\tau_k + \tau_{k+1}.$$

- Let us call the product $w_i = s_i t_i$ the amount of *work* to be done for task T_i. We define

  $$W_k = \sum_{T_i \in L_k} w_i$$

  to be the total amount of work to be done for tasks in L_k. Similarly, we define

  $$W_{k+1} = \sum_{T_i \in L_{k+1}} w_i,$$

  and

  $$W_{k+2} = \sum_{T_i \in L_{k+2}} w_i,$$

  and

  $$W_{k+3} = \sum_{T_i \in L_{k+3}} w_i.$$

  It is clear that

  $$\text{OPT}(L) \geq \frac{W_k + W_{k+1} + W_{k+2} + W_{k+3}}{M},$$

  namely, the optimal schedule length is at least the total amount of work divided by the number of cores. Since there are constantly k tasks of L_k in execution during time interval I_k and the size of each task in L_k is at least $M/(k+1)$, we have

  $$W_k \geq \left(\frac{k}{k+1}\right) M\tau_k.$$

  Similarly, we get

  $$W_{k+1} \geq \left(\frac{k+1}{k+2}\right) M\tau_{k+1},$$

  and

  $$W_{k+2} \geq \left(\frac{k+2}{k+3}\right) M\tau_{k+2}.$$

  As for W_{k+3}, we notice that during time interval I_{k+3}, the number of idle cores cannot exceed $M/(k+3)$; otherwise, at least one task in L_{k+3}, say, the task that completes last, would be scheduled earlier. Therefore, we get

  $$W_{k+3} \geq \left(1 - \frac{1}{k+3}\right) M\tau_{k+3} = \left(\frac{k+2}{k+3}\right) M\tau_{k+3}.$$

Based on the last four inequalities, we obtain

$$\mathrm{OPT}(L) \geq \left(\frac{k}{k+1}\right)\tau_k + \left(\frac{k+1}{k+2}\right)\tau_{k+1} + \left(\frac{k+2}{k+3}\right)x,$$

where $x = \tau_{k+2} + \tau_{k+3}$. Summarizing the above discussion on $\mathrm{OPT}(L)$, we have

$$\mathrm{OPT}(L) \geq \max\left(\tau_k, \left(\frac{k}{k+1}\right)\tau_k + \tau_{k+1}, \left(\frac{k}{k+1}\right)\tau_k + \left(\frac{k+1}{k+2}\right)\tau_{k+1} + \left(\frac{k+2}{k+3}\right)x\right).$$

Finally, we study the ratio

$$R = \frac{\tau_k + \tau_{k+1} + x}{\max\left(\tau_k, \left(\frac{k}{k+1}\right)\tau_k + \tau_{k+1}, \left(\frac{k}{k+1}\right)\tau_k + \left(\frac{k+1}{k+2}\right)\tau_{k+1} + \left(\frac{k+2}{k+3}\right)x\right)}.$$

We consider three cases, depending on which term in the three terms of the max operator is the maximum.

Case 1. (The first term is the maximum.) In this case, we have

$$\tau_k \geq \left(\frac{k}{k+1}\right)\tau_k + \tau_{k+1},$$

which implies that

$$\frac{\tau_{k+1}}{\tau_k} \leq \frac{1}{k+1}.$$

Furthermore, since

$$\tau_k \geq \left(\frac{k}{k+1}\right)\tau_k + \left(\frac{k+1}{k+2}\right)\tau_{k+1} + \left(\frac{k+2}{k+3}\right)x,$$

we get

$$\frac{x}{\tau_k} \leq \frac{k+3}{k+2}\left(\frac{1}{k+1} - \left(\frac{k+1}{k+2}\right)\frac{\tau_{k+1}}{\tau_k}\right).$$

Consequently, we get

$$\begin{aligned} R &= \frac{\tau_k + \tau_{k+1} + x}{\tau_k} \\ &= 1 + \frac{\tau_{k+1}}{\tau_k} + \frac{x}{\tau_k} \\ &\leq 1 + \frac{\tau_{k+1}}{\tau_k} + \frac{k+3}{k+2}\left(\frac{1}{k+1} - \left(\frac{k+1}{k+2}\right)\frac{\tau_{k+1}}{\tau_k}\right) \\ &= 1 + \frac{k+3}{(k+1)(k+2)} + \left(1 - \frac{(k+1)(k+3)}{(k+2)^2}\right)\frac{\tau_{k+1}}{\tau_k} \\ &= 1 + \frac{k+3}{(k+1)(k+2)} + \frac{1}{(k+1)(k+2)^2} \\ &= 1 + \frac{(k+2)(k+3)+1}{(k+1)(k+2)^2}. \end{aligned}$$

Case 2. (The second term is the maximum.) In this case, we have

$$\tau_k \leq \left(\frac{k}{k+1}\right)\tau_k + \tau_{k+1},$$

which implies that

$$\tau_k \leq (k+1)\tau_{k+1}.$$

Furthermore, since

$$\left(\frac{k}{k+1}\right)\tau_k + \tau_{k+1} \geq \left(\frac{k}{k+1}\right)\tau_k + \left(\frac{k+1}{k+2}\right)\tau_{k+1} + \left(\frac{k+2}{k+3}\right)x,$$

we get

$$x \leq \left(\frac{k+3}{(k+2)^2}\right)\tau_{k+1}.$$

Hence, we get

$$\begin{aligned} R &= \frac{\tau_k + \tau_{k+1} + x}{\left(\frac{k}{k+1}\right)\tau_k + \tau_{k+1}} \\ &\leq \frac{\tau_k + \tau_{k+1} + \left(\frac{k+3}{(k+2)^2}\right)\tau_{k+1}}{\left(\frac{k}{k+1}\right)\tau_k + \tau_{k+1}} \\ &= \frac{\tau_k + \left(\frac{k^2+5k+7}{(k+2)^2}\right)\tau_{k+1}}{\left(\frac{k}{k+1}\right)\tau_k + \tau_{k+1}}. \end{aligned}$$

To proceed, we need the following simple fact, namely, if $ad - bc > 0$ (< 0, respectively), the function

$$f(y) = \frac{ay+b}{cy+d}$$

is an increasing (decreasing, respectively) function of y, where $a, b, c, d, y > 0$. Hence, the last ratio is an increasing function of τ_k and achieves its maximum value when $\tau_k = (k+1)\tau_{k+1}$, which is

$$\frac{(k+1) + \frac{k^2+5k+7}{(k+2)^2}}{k+1} = 1 + \frac{k^2+5k+7}{(k+1)(k+2)^2},$$

which is the same as that of Case 1.

Case 3. (The third term is the maximum.) In this case, we have

$$\left(\frac{k}{k+1}\right)\tau_k + \tau_{k+1} \leq \left(\frac{k}{k+1}\right)\tau_k + \left(\frac{k+1}{k+2}\right)\tau_{k+1} + \left(\frac{k+2}{k+3}\right)x,$$

which implies that

$$x \geq \left(\frac{k+3}{(k+2)^2}\right)\tau_{k+1}.$$

We observe that

$$R = \frac{x + \tau_k + \tau_{k+1}}{\left(\frac{k+2}{k+3}\right)x + \left(\frac{k}{k+1}\right)\tau_k + \left(\frac{k+1}{k+2}\right)\tau_{k+1}}$$

is a decreasing function of x and achieves its maximum value when

$$x = \left(\frac{k+3}{(k+2)^2}\right)\tau_{k+1},$$

that is,

$$\begin{aligned} R &\leq \frac{\left(\frac{k+3}{(k+2)^2}\right)\tau_{k+1} + \tau_k + \tau_{k+1}}{\left(\frac{k+2}{k+3}\right)\left(\frac{k+3}{(k+2)^2}\right)\tau_{k+1} + \left(\frac{k}{k+1}\right)\tau_k + \left(\frac{k+1}{k+2}\right)\tau_{k+1}} \\ &= \frac{\tau_k + \left(\frac{k^2+5k+7}{(k+2)^2}\right)\tau_{k+1}}{\left(\frac{k}{k+1}\right)\tau_k + \tau_{k+1}}, \end{aligned}$$

which is an increasing function of τ_k. Since

$$\tau_k \leq \left(\frac{k}{k+1}\right)\tau_k + \left(\frac{k+1}{k+2}\right)\tau_{k+1} + \left(\frac{k+2}{k+3}\right)x,$$

with

$$x = \left(\frac{k+3}{(k+2)^2}\right)\tau_{k+1},$$

we get

$$\tau_k \leq (k+1)\tau_{k+1}.$$

Hence, the R achieves its maximum value when $\tau_k = (k+1)\tau_{k+1}$, which is the same as that of Cases 1 and 2.

Summarizing the above discussion, we obtain

$$\mathrm{LTF}(L) \leq \left(1 + \frac{(k+2)(k+3)+1}{(k+1)(k+2)^2}\right)\mathrm{OPT}(L) + 4t^*.$$

The theorem is proven. □

Table 1
The performance bound of Theorem 1.

k	α_k
1	1.7222222
2	1.4375000
3	1.3100000
4	1.2388889
5	1.1938776
6	1.1629464
7	1.1404321
8	1.1233333
9	1.1099174
10	1.0991162

In Table 1, we show the numerical data of the asymptotic worst-case performance bound of algorithm LTF in Theorem 1, i.e.,

$$\alpha_k = 1 + \frac{(k+2)(k+3)+1}{(k+1)(k+2)^2},$$

for $k = 1, 2, \ldots, 10$.

3.3. Simulation results

In this section, we demonstrate simulation results on the average-case performance of both algorithms LTF and LTF* for non-clairvoyant offline scheduling of independent parallel tasks on a single multicore processor with M cores.

Assume that task sizes are independent and identically distributed (i.i.d.) random variables with a common probability distribution $(p_1, p_2, \ldots, p_D)$ in the range $[1..D]$ with $D \le M$, where p_s is the probability that $s_i = s$, for all $1 \le s \le D$. We consider three types of probability distributions of task sizes with about the same expected task size $\mathbf{E}(s_i)$.

- Uniform distributions in the range $[1..D]$, i.e., $p_s = 1/D$ for all $1 \le s \le D$. The average task size is $\mathbf{E}(s_i) = (D+1)/2$.
- Binomial distributions in the range $[1..D]$, i.e.,

  $$p_s = \frac{1}{1-(1-q)^D}\binom{D}{s} q^s (1-q)^{D-s},$$

  for all $1 \le s \le D$, where q is chosen such that $Dq = (D+1)/2$, i.e., $q = (D+1)/(2D)$. However, the actual average task size is

  $$\mathbf{E}(s_i) = \frac{Dq}{1-(1-q)^D},$$

  which is slightly greater than $(D+1)/2$.
- Geometric distributions in the range $[1..D]$, i.e.,

  $$p_s = \frac{q(1-q)^{s-1}}{1-(1-q)^D},$$

  for all $1 \le s \le D$, where q is chosen such that $1/q = (D+1)/2$, i.e., $q = 2/(D+1)$. However, the actual average task size is

  $$\mathbf{E}(s_i) = \frac{1/q - (1/q+D)(1-q)^D}{1-(1-q)^D},$$

  which is less than $(D+1)/2$.

In Table 2, we show our experimental data on the average-case performance of algorithms LTF and LTF*. The number of cores is set as $M = 128$. For each combination of k, where $1 \le k \le 10$, and each of the three probability distributions, and the LTF and LTF* algorithms, we generate a list L of $n = 5000$ random tasks. Task sizes are i.i.d. random variable from a given probability distribution, where we set $D = \lfloor M/k \rfloor$. Task execution times are i.i.d. random variables uniformly distributed in the range (0, 10). For each list of random tasks, we apply algorithm LTF or LTF*, and get the schedule length LTF(L) or LTF*(L). As for OPT(L), we simply use the lower bound

$$\mathrm{OPT}(L) \ge W/M = \frac{1}{M}\sum_{i=1}^{n} s_i t_i.$$

The ratio of LTF(L)/OPT(L) or LTF*(L)/OPT(L) is the result of an experiment. The experiment is repeated for 100 times, and the average of the 100 values is considered as $\mathbf{E}$(LTF(L)/OPT(L)) or $\mathbf{E}$(LTF*(L)/OPT(L)), which is displayed in the table. The maximum 99% confidence interval of all the data in the table is ±0.3253353%.

We have the following observations.

- The average-case performance of algorithm LTF in Table 2 is much better than the worst-case performance in Table 1.
- The performance of algorithm LTF* is noticeably better than that of algorithm LTF. The performance of LTF* is very close to optimal and practically very useful.

4. Scheduling on multiple multicore processors

4.1. The extended LTF scheduling algorithm

The LTF algorithm can be easily extended to multiple multicore processors with $M_1, M_2, \ldots, M_m$ cores. Again, tasks are scheduled in a nonincreasing order of task sizes. Initially, tasks $T_{i_{j-1}+1}, T_{i_{j-1}+2}, \ldots, T_{i_j}$ are scheduled for execution on the jth multicore processor, where $i_0 = 0$ and the indices $i_1, i_2, \ldots, i_m$ are defined in such a way that the total size of $T_{i_{j-1}+1}, T_{i_{j-1}+2}, \ldots, T_{i_j}$ does not exceed M_j,

$$s_{i_{j-1}+1} + s_{i_{j-1}+2} + \cdots + s_{i_j} \le M_j,$$

but the total size of $T_{i_{j-1}+1}, T_{i_{j-1}+2}, \ldots, T_{i_j}, T_{i_j+1}$ exceeds M_j,

$$s_{i_{j-1}+1} + s_{i_{j-1}+2} + \cdots + s_{i_j} + s_{i_j+1} > M_j,$$

for all $1 \le j \le m$. Whenever a task is completed on a multicore processor, say, the jth multicore processor, algorithm LTF checks whether there are enough available cores for task T_{i_m+1}. If so, task T_{i_m+1} is scheduled for execution; otherwise, algorithm LTF waits for the next completion of a task.

Algorithm LTF* can be extended to multiple multicore processors accordingly.

4.2. Worst-case analysis

In this section, we analyze the asymptotic worst-case performance of algorithm LTF for non-clairvoyant offline scheduling of independent parallel tasks on multiple multicore processors with $M_1, M_2, \ldots, M_m$ cores.

We define $M = M_1 + M_2 + \cdots + M_m$ to be the total number of cores of the m multicore processors, and $M^* = \min(M_1, M_2, \ldots, M_m)$ to be the minimum size of the m multicore processors. Furthermore, let $r = mM^*/M = M^*/(M/m)$, which is the ratio of the minimum processor size to the average processor size.

It is assumed in this section that $s^* \le M^*$, i.e., the maximum task size is no greater than the minimum processor size. In other words, any task can be executed on any multicore processor.

Our main result of this section is the following theorem.

Theorem 2. *To schedule any list L of independent parallel tasks on m multicore processors with M cores by using the LTF algorithm, we have*

$$LTF(L) \le \alpha_{k,r} OPT(L) + t^*,$$

Table 2
Simulation results of LTF and LTF* on a single multicore processor.

k	Uniform		Binomial		Geometric	
	LTF	LTF*	LTF	LTF*	LTF	LTF*
1	1.2836656	1.0086699	1.4897393	1.0631060	1.2904968	1.0048925
2	1.1559544	1.0030697	1.1446964	1.0704966	1.1412308	1.0065995
3	1.1026384	1.0040161	1.0917927	1.0367635	1.0891142	1.0050399
4	1.0773207	1.0033769	1.0576074	1.0293787	1.0668444	1.0039470
5	1.0608234	1.0031505	1.0545063	1.0118485	1.0518679	1.0040560
6	1.0488520	1.0033442	1.0544935	1.0055328	1.0430472	1.0046057
7	1.0420992	1.0036175	1.0455553	1.0059473	1.0378420	1.0050309
8	1.0406718	1.0040414	1.0355460	1.0064859	1.0355423	1.0054756
9	1.0424241	1.0043263	1.0258349	1.0065875	1.0341625	1.0058136
10	1.0380658	1.0047071	1.0201550	1.0059795	1.0298851	1.0063139

where the asymptotic worst-case performance bound is

$$\alpha_{k,r} = 1 + \frac{r}{k - r + 1},$$

and $M^*/(k+1) < s^* \leq M^*/k$, *or,* $k = \lfloor M^*/s^* \rfloor$.

Proof. We assume that the execution of the n tasks starts at time 0, and all the tasks are executed during time interval $[0, \text{LTF}(L))$. Assume that task T_l is the task which is completed last, and T_l is scheduled at time τ. Then, we have $t_l = \text{LTF}(L) - \tau$, i.e.,

$$\text{LTF}(L) = \tau + t_l \leq \tau + t^*.$$

Let us consider any moment during the time interval $[0, \tau)$ and any multicore processor j. By the definition of k, we know that the number of tasks in execution at that moment on the jth multicore processor is at least k, because $ks^* \leq M^* \leq M_j$, i.e., the jth multicore processor can accommodate at least k tasks simultaneously, for all $1 \leq j \leq m$. A key observation is that at any moment, the number of idle cores in the jth multicore processor is less than the sizes of the tasks in execution on the jth multicore processor. To show this, we assume that the tasks in execution on the jth multicore processor at certain moment are $T_{i_1}, T_{i_2}, \ldots, T_{i_p}$, where $p \geq k$. Without loss of generality, let us assume that $s_{i_1} \geq s_{i_2} \geq \cdots \geq s_{i_p}$. If there are d idle cores, then we must have $d < s_{i_p}$; otherwise, since $s_l \leq s_{i_p} \leq d$, some task (say, task T_l) would be scheduled earlier. This implies that

$$d < \frac{M_j}{p+1} \leq \frac{M_j}{k+1},$$

for all $1 \leq j \leq m$. In particular, we have

$$d < \frac{M^*}{k+1},$$

that is, at any moment during the time interval $[0, \tau)$, the number of idle cores in each of the m multicore processors is less than $M^*/(k+1)$.

Notice that the total amount of work to be performed for all the n tasks is

$$W = \sum_{i=1}^{n} s_i t_i.$$

It is clear that W is at least

$$W \geq \sum_{j=1}^{m} \left(M_j - \frac{M^*}{k+1} \right) \tau = \left(M - \frac{mM^*}{k+1} \right) \tau.$$

Hence, we get

$$\text{OPT}(L) \geq \frac{W}{M} \geq \left(1 - \frac{1}{k+1} \cdot \frac{mM^*}{M} \right) \tau.$$

By the definition of r, we obtain

$$\text{OPT}(L) \geq \left(1 - \frac{r}{k+1} \right) \tau,$$

which implies that

$$\tau \leq \left(\frac{k+1}{k-r+1} \right) \text{OPT}(L).$$

Summarizing the above discussion, we obtain

$$\text{LTF}(L) \leq \left(\frac{k+1}{k-r+1} \right) \text{OPT}(L) + t^*.$$

This proves the theorem. □

It is easy to see that since $r \leq 1$, we have

$$\frac{k+1}{k-r+1} \leq 1 + \frac{1}{k},$$

where the equality is achieved when $r = 1$, i.e., $M_1 = M_2 = \cdots = M_m$.

In Table 3, we show the numerical data of the asymptotic worst-case performance bound of algorithm LTF in Theorem 2, i.e.,

$$\alpha_{k,r} = 1 + \frac{r}{k+1-r},$$

for $k = 1, 2, \ldots, 10$ and $r = 1.00, 0.75, 0.50, 0.25$.

Table 3
The performance bound of Theorem 2.

k	$r = 1.00$	$r = 0.75$	$r = 0.50$	$r = 0.25$
1	2.0000000	1.6000000	1.3333333	1.1428571
2	1.5000000	1.3333333	1.2000000	1.0909091
3	1.3333333	1.2307692	1.1428571	1.0666667
4	1.2500000	1.1764706	1.1111111	1.0526316
5	1.2000000	1.1428571	1.0909091	1.0434783
6	1.1666667	1.1200000	1.0769231	1.0370370
7	1.1428571	1.1034483	1.0666667	1.0322581
8	1.1250000	1.0909091	1.0588235	1.0285714
9	1.1111111	1.0810811	1.0526316	1.0256410
10	1.1000000	1.0731707	1.0476190	1.0232558

4.3. Simulation results

In this section, we demonstrate simulation results on the average-case performance of both algorithms LTF and LTF* for non-clairvoyant offline scheduling of independent parallel tasks on multiple multicore processors.

In Table 4, we show our experimental data on the average-case performance of algorithms LTF and LTF*. The number of multicore processors is $m = 5$. The numbers of cores are set as $(M_1, M_2, M_3, M_4, M_5) = (128, 128, 192, 256, 256)$. For each combination of k, where $1 \leq k \leq 10$, and each of the three probability distributions, and the LTF and LTF* algorithms, we generate a list L of $n = 5000$ random tasks. Task sizes are i.i.d. random variable from a given probability distribution, where we set $D = \lfloor M^*/k \rfloor$. Task execution times are i.i.d. random variables uniformly distributed in the range $(0, 10)$. For each list of random

Table 4
Simulation results of LTF and LTF* on multiple multicore processors.

k	Uniform		Binomial		Geometric	
	LTF	LTF*	LTF	LTF*	LTF	LTF*
1	1.2165348	1.0076198	1.2422061	1.0620983	1.1966610	1.0108762
2	1.1070906	1.0103615	1.0937155	1.0439797	1.0977763	1.0136766
3	1.0806176	1.0134536	1.0645665	1.0308059	1.0732611	1.0175264
4	1.0625377	1.0163439	1.0473301	1.0275828	1.0613371	1.0220800
5	1.0565701	1.0198179	1.0478568	1.0242367	1.0584516	1.0271077
6	1.0520925	1.0226328	1.0501519	1.0257317	1.0574081	1.0320433
7	1.0499784	1.0259279	1.0477142	1.0283908	1.0586942	1.0366524
8	1.0505199	1.0289361	1.0463114	1.0307059	1.0590290	1.0407761
9	1.0552177	1.0324511	1.0465785	1.0341991	1.0626428	1.0456997
10	1.0554874	1.0373198	1.0493552	1.0382159	1.0666131	1.0522816

tasks, we apply algorithm LTF or LTF*, and get the schedule length LTF(L) or LTF*(L). As for OPT(L), we simply use the lower bound

$$\mathrm{OPT}(L) \geq W/M = \frac{1}{M}\sum_{i=1}^{n} s_i t_i.$$

The ratio of LTF(L)/OPT(L) or LTF*(L)/OPT(L) is the result of an experiment. The experiment is repeated for 100 times, and the average of the 100 values is considered as $\mathbf{E}(\mathrm{LTF}(L)/\mathrm{OPT}(L))$ or $\mathbf{E}(\mathrm{LTF}^*(L)/\mathrm{OPT}(L))$, which is displayed in the table. The maximum 99% confidence interval of all the data in the table is $\pm 0.1010213\%$.

We have the following observations.

- The average-case performance of algorithm LTF in Table 4 is much better than the worst-case performance in Table 3.
- The performance of algorithm LTF* is noticeably better than that of algorithm LTF. The performance of LTF* is very close to optimal and practically very useful.

5. Online scheduling on multiple multicore processors

5.1. The RTF scheduling algorithm

Our algorithm for non-clairvoyant online scheduling of independent parallel tasks on multiple multicore processors is extended from the SIMPLE algorithm, which was originally proposed for non-clairvoyant online scheduling of independent parallel tasks on a single parallel computing system [17]. It can be easily extended to multiple parallel computing systems such as multiple multicore processors.

Our algorithm works as follows. The list L of n parallel tasks is divided into $m+d$ sublists $L_1, L_2, \ldots, L_m, L_{m+1}, L_{m+2}, \ldots, L_{m+d}$, where $L_k = (T_{b_{k-1}+1}, T_{b_{k-1}+2}, \ldots, T_{b_k})$, for all $1 \leq k \leq m+d$, with $b_0 = 0$, $b_{m+d} = n$. The indices $b_1, b_2, \ldots, b_m, b_{m+1}, b_{m+2}, \ldots, b_{m+d-1}$ and the integer d are defined below. Notice that $L = L_1 L_2 \cdots L_{m+d}$ is a simple concatenation of the $m+d$ sublists.

Let $T_{i_1}, T_{i_2}, \ldots, T_{i_d}, \ldots$ be the sequence of tasks completed in a schedule produced by our algorithm. Assume that task T_{i_k} is completed at time c_k, where $1 \leq k \leq d$.

Initially, at time 0, tasks in L_j are scheduled for execution on the jth multicore processor, for all $1 \leq j \leq m$. The index b_j is as large as possible, i.e., the total size of tasks in L_j does not exceed M_j,

$$s_{b_{j-1}+1} + s_{b_{j-1}+2} + \cdots + s_{b_j} \leq M_j,$$

but the total size of tasks in L_j plus that of T_{b_j+1} exceeds M_j,

$$s_{b_{j-1}+1} + s_{b_{j-1}+2} + \cdots + s_{b_j} + s_{b_j+1} > M_j,$$

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

Later, at time c_k when task T_{i_k} is completed on the jth multicore processor, where $1 \leq k \leq d$, tasks in L_{m+k} are scheduled for execution on the jth multicore processor. Again, the index b_{m+k} is as large as possible, for all $1 \leq k \leq d-1$. Let $C_j(t)$ be the number of active cores allocated to some task in execution on the jth multicore processor at time t. Then, we have

$$C_j(c_k) + s_{b_{m+k-1}+1} + s_{b_{m+k-1}+2} + \cdots + s_{b_{m+k}} \leq M_j,$$

but

$$C_j(c_k) + s_{b_{m+k-1}+1} + s_{b_{m+k-1}+2} + \cdots + s_{b_{m+k}} + s_{b_{m+k}+1} > M_j,$$

for all $1 \leq k \leq d-1$.

The integer d is defined in such a way that at time c_d when task T_{i_d} is completed on the jth multicore processor, all tasks in L_{m+d} can be scheduled for execution on the jth multicore processor, and the jth multicore processor still has more available cores to accommodate more tasks.

The above algorithm is essentially the same as LTF, except that tasks in L are in their original order, which is a random order of tasks. Hence, we call the above algorithm *random task first* (RTF). Since tasks are scheduled in the given order without any preprocessing, RTF is an online scheduling algorithm.

5.2. Average-case analysis

In this section, we analyze the asymptotic average-case performance of algorithm RTF for non-clairvoyant online scheduling of independent parallel tasks on multiple multicore processors with $M_1, M_2, \ldots, M_m$ cores.

We make the following assumptions for analyzing the average-case performance of algorithm RTF.

- The task sizes $s_1, s_2, \ldots, s_n$ are i.i.d. random variables with a common probability distribution in the range $[1..M^*]$.
- The task execution times $t_1, t_2, \ldots, t_n$ are i.i.d. random variables with a common probability distribution.
- The probability distribution of task sizes is independent of the probability distribution of task execution times.

We also need the following core allocation model. Assume that there is a multicore processor with M cores, and there are n tasks with sizes $s_1, s_2, \ldots, s_n$. Tasks are scheduled as many as possible on the multicore processor for simultaneous execution, i.e., we find i such that $s_1+s_2+\cdots+s_i \leq M$, but $s_1+s_2+\cdots+s_i+s_{i+1} > M$; or if there are no enough tasks, all tasks are scheduled for execution. If $s_1, s_2, \ldots, s_n$ are i.i.d. random variables, the total number of active cores allocated is also a random variable. We use $C(n, M)$ to denote the mean of this random variable, i.e., the expected number of active cores. It is clear that $C(n, M)$ increases with n. However, $C(n, M) = C_M$ for all $n \geq M$, since a multicore processor with M cores can accommodate at most M tasks.

Let p_s be the probability that the size of a task T_i is $s_i = s$, where $s = 1, 2, 3, \ldots$. The following recurrence relation characterizes

$C(n, M)$ [17]:

$$C(n,M) = \begin{cases} \sum_{s=1}^{M} sp_s, & \text{if } n = 1; \\ \sum_{s=1}^{M} p_s(s + C(n-1, M-s)), & \text{if } n > 1. \end{cases}$$

The above recurrence relation can be used to calculate $C(n, M)$ for any n, M, and any probability distribution of tasks $(p_1, p_2, p_3, \ldots)$.

As an interesting and important special case, if task sizes have a uniform distribution in the range $[1..D]$, i.e., $p_1 = p_2 = \cdots = p_D = 1/D$, where $D \leq M$, then we have

$$C_M = M - \left(3 - \left(1 + \frac{1}{D}\right)^{D+1}\right) D - 1.$$

The reader is referred to [17] for the derivation.

Our main result of this section is the following theorem.

Theorem 3. *To schedule any list L of independent parallel tasks on m multicore processors with $M_1, M_2, \ldots, M_n$ cores respectively by using the RTF algorithm, we have*

$$\mathbf{E}(RTF(L)) \leq \beta \mathbf{E}(OPT(L)),$$

as $n \to \infty$, where the asymptotic average-case performance bound is

$$\beta = \frac{M}{C_{M_1} + C_{M_2} + \cdots + C_{M_n}}.$$

If task sizes are uniformly distributed in the range $[1..D]$, where $D \leq M^*$, we have

$$\beta = \frac{M}{M - m((3 - (1 + 1/D)^{D+1})D + 1)}.$$

Proof. Let us divide the time interval $[0, \text{RTF}(L))$ into $d + 1$ subintervals $[c_0, c_1)$, $[c_1, c_2)$, $[c_2, c_3)$, ..., $[c_{d-1}, c_d)$, and $[c_d, \text{RTF}(L))$. Let $\tau_k = c_k - c_{k-1}$, where $1 \leq k \leq d$, and $\tau_{d+1} = \text{RTF}(L) - c_d$ be the lengths of the $d + 1$ subintervals. Notice that at time c_d, all tasks in L_{m+d} are scheduled for execution, and there is no more task to be scheduled. Hence, $\tau_{d+1} \leq t^*$. Therefore, we have

$$\text{RTF}(L) \leq \sum_{k=1}^{d} \tau_k + t^*,$$

and

$$\mathbf{E}(\text{RTF}(L)) \leq \sum_{k=1}^{d} \mathbf{E}(\tau_k) + \mathbf{E}(t^*).$$

Notice that the total amount of work to be performed for all the n tasks is

$$W = \sum_{i=1}^{n} s_i t_i,$$

and we have

$$\text{OPT}(L) \geq \frac{W}{M},$$

and

$$\mathbf{E}(\text{OPT}(L)) \geq \frac{\mathbf{E}(W)}{M}.$$

Let $C_j(k)$ denote the number of active cores on the jth multicore processor during the time interval $[c_{k-1}, c_k)$, where $1 \leq j \leq m$ and $1 \leq k \leq d$. Then, we have

$$W \geq \sum_{k=1}^{d} \tau_k (C_1(k) + C_2(k) + \cdots + C_m(k)),$$

where we ignore the amount of work performed during $[c_d, \text{RTF}(L))$. Therefore, we get

$$\mathbf{E}(W) \geq \sum_{k=1}^{d} \mathbf{E}(\tau_k)(\mathbf{E}(C_1(k)) + \mathbf{E}(C_2(k)) + \cdots + \mathbf{E}(C_m(k))).$$

We observe that initially, each multicore processor is packed to the maximum extent, until there is no enough core to accommodate the next task. Later, when task T_{i_k} is completed at time c_k on the jth multicore processor, where $1 \leq k \leq d - 1$, the jth multicore processor is again packed to the maximum extent, until there is no enough core to accommodate the next task. Therefore, we have $\mathbf{E}(C_j(k)) = C_{M_j}$, for all $1 \leq j \leq m$ and $1 \leq k \leq d$, which implies that

$$\mathbf{E}(W) \geq (C_{M_1} + C_{M_2} + \cdots + C_{M_m}) \sum_{k=1}^{d} \mathbf{E}(\tau_k).$$

and

$$\mathbf{E}(\text{OPT}(W)) \geq \frac{C_{M_1} + C_{M_2} + \cdots + C_{M_m}}{M} \sum_{k=1}^{d} \mathbf{E}(\tau_k).$$

Summarizing the above discussion, we get

$$\mathbf{E}(\text{RTF}(L)) \leq \left(\frac{M}{C_{M_1} + C_{M_2} + \cdots + C_{M_n}}\right) \mathbf{E}(\text{OPT}(W)) + \mathbf{E}(t^*),$$

and

$$\frac{\mathbf{E}(\text{RTF}(L))}{\mathbf{E}(\text{OPT}(W))} \leq \frac{M}{C_{M_1} + C_{M_2} + \cdots + C_{M_n}},$$

as $n \to \infty$ and $\mathbf{E}(t^*)/\mathbf{E}(\text{OPT}(W)) \to 0$.

If task sizes are uniformly distributed in the range $[1..D]$, where $D \leq M^*$, we have

$$\begin{aligned} \sum_{j=1}^{m} C_{M_j} &= \sum_{j=1}^{m} (M_j - (3 - (1 + 1/D)^{D+1})D - 1) \\ &= M - m((3 - (1 + 1/D)^{D+1})D + 1). \end{aligned}$$

The theorem is proven. □

5.3. Simulation results

In this section, we demonstrate simulation results on the average-case performance of algorithms RTF for non-clairvoyant online scheduling of independent parallel tasks on multiple multicore processors.

In Table 5, we show our experimental data on the average-case performance of algorithms RTF and RTF*. The number of multicore processors is $m = 5$. The numbers of cores are set as $(M_1, M_2, M_3, M_4, M_5) = (128, 128, 192, 256, 256)$. For each combination of k, where $1 \leq k \leq 10$, and each of the three probability distributions, and the RTF and RTF* algorithms, we generate a list L of n =50,000 random tasks. Task sizes are i.i.d. random variable from a given probability distribution, where we set $D = \lfloor M^*/k \rfloor$. Task execution times are i.i.d. random variables uniformly distributed in the range (0, 10). For each list of random tasks, we apply algorithm RTF or RTF*, and get the schedule length $\text{RTF}(L)$ or $\text{RTF}^*(L)$. The experiment is repeated for 100 times, and the average of the 100 values is considered as $\mathbf{E}(\text{RTF}(L))$ or $\mathbf{E}(\text{RTF}^*(L))$. As for $\mathbf{E}(\text{OPT}(L))$, we simply use the lower bound

$$\mathbf{E}(\text{OPT}(L)) \geq \mathbf{E}(W)/M = \frac{N}{M}\mathbf{E}(s_i)\mathbf{E}(t_i) = 2.5(D + 1)N/M.$$

The ratio of $\mathbf{E}(\text{RTF}(L))/\mathbf{E}(\text{OPT}(L))$ or $\mathbf{E}(\text{RTF}^*(L))/\mathbf{E}(\text{OPT}(L))$ is displayed in the table. The maximum 99% confidence interval of all the data in the table is $\pm 0.1378706\%$.

As comparison, we also show the analytical asymptotic average-case performance bound in Theorem 3. It is observed that the analytical result is very close to the experimental result.

Table 5
Simulation results of RTF on multiple multicore processors.

k	Uniform		Binomial		Geometric	
	Analysis	Simulation	Analysis	Simulation	Analysis	Simulation
1	1.2606742	1.2751010	1.2200202	1.2532891	1.2131789	1.2460074
2	1.1220632	1.1317983	1.0980104	1.0982232	1.0972197	1.1114637
3	1.0766544	1.0833610	1.0586856	1.0588793	1.0609908	1.0700421
4	1.0568853	1.0620539	1.0435530	1.0442393	1.0452747	1.0515330
5	1.0434784	1.0478795	1.0335985	1.0359015	1.0345496	1.0407647
6	1.0359712	1.0394772	1.0280461	1.0302214	1.0285208	1.0342347
7	1.0304114	1.0347201	1.0239242	1.0265633	1.0240463	1.0304326
8	1.0267380	1.0309127	1.0211917	1.0239852	1.0210859	1.0268341
9	1.0230906	1.0277685	1.0184712	1.0218115	1.0181435	1.0241272
10	1.0194690	1.0238680	1.0157610	1.0194173	1.0152198	1.0230443

6. Summary

We have studied the problem of non-clairvoyant scheduling of independent parallel tasks on single and multiple multicore processors. Our results are summarized as follows.

- For a single multicore processor, we have derived an asymptotic worst-case performance bound for the non-clairvoyant offline scheduling algorithm LTF. The result improves our previous result on a single parallel computing system.
- For multiple multicore processors, we have derived an asymptotic worst-case performance bound for the LTF algorithm. This result is the first attempt in scheduling parallel tasks on multiple parallel computing systems.
- For multiple multicore processors, we have also derived an asymptotic average-case performance bound for the non-clairvoyant online scheduling algorithm RTF. The result extends our earlier result on a single parallel computing system.

There are still much work that remains to be done. We mention several topics which are worth of further and deeper investigation.

- First, on a single multicore processor, the worst-case performance of algorithm LTF* is unknown and is definitely an interesting topic.
- Second, on multiple multicore processors, the worst-case performance of algorithm LTF needs more careful analysis.
- Third, the worst-case performance of algorithm LTF without the assumption of $s^* \leq M^*$ is also unknown and worth of investigation.
- Finally, the worst-case performance of algorithm LTF* on multiple multicore processors is challenging and interesting.

It is also worth mentioning that the problem investigated in this paper can be further extended by breaking the boundaries among the multicore processors. This means that core allocation can be performed across different multicore processors. However, as mentioned earlier, such an investigation requires additional consideration of inter-processor communication times.

Acknowledgments

The author is grateful to the anonymous reviewers for their suggestions to improve the manuscript.

References

[1] A.K. Amoura, E. Bampis, C. Kenyon, Y. Manoussakis, Scheduling independent multiprocessor tasks, Algorithmica 32 (2002) 161–247.
[2] J.H. Anderson, J.M. Calandrino, Parallel real-time task scheduling on multicore platforms, in: 27th IEEE International Real-Time Systems Symposium, 2006, pp. 89–100.
[3] C. Augonnet, S. Thibault, R. Namyst, P.-A. Wacrenier, STARPU: A unified platform for task scheduling on heterogeneous multicore architectures, in: 15th International Euro-Par Conference, in: LNCS, vol. 5704, 2009, pp. 863–874.
[4] B.S. Baker, D.J. Brown, H.P. Katseff, A 5/4 algorithm for two-dimensional packing, J. Algorithms 2 (1981) 348–368.
[5] B.S. Baker, J.S. Schwarz, Shelf algorithms for two-dimensional packing problems, SIAM J. Comput. 12 (1983) 508–525.
[6] J. Blazewicz, J.K. Lenstra, A.H.G. Rinnooy Kan, Scheduling subject to resource constraints: Classification and complexity, Discrete Appl. Math. 5 (1983) 11–24.
[7] R. Bleuse, S. Kedad-Sidhoum, F. Monna, G. Mounie, D. Trystram, Scheduling independent tasks on multi-cores with GPU accelerators, Concurr. Comput.: Pract. Exper. 27 (6) (2015) 1625–1638.
[8] A. Feldmann, J. Sgall, S.-H. Teng, Dynamic scheduling on parallel machines, Theoret. Comput. Sci. 130 (1994) 49–72.
[9] M.R. Garey, R.L. Graham, Bound for multiprocessor scheduling with resource constraints, SIAM J. Comput. 4 (1975) 187–200.
[10] I. Golan, Performance bounds for orthogonal oriented two-dimensional packing algorithms, SIAM J. Comput. 10 (1981) 571–582.
[11] R.L. Graham, Bounds on multiprocessing timing anomalies, SIAM J. Appl. Math. 2 (1969) 416–429.
[12] K. Jansen, Scheduling malleable parallel tasks: An asymptotic fully polynomial time approximation scheme, Algorithmica 39 (2004) 59–81.
[13] D.S. Johnson, et al., Worst-case performance bounds for simple one-dimensional packing algorithms, SIAM J. Comput. 3 (1974) 299–325.
[14] K. Lakshmanan, S. Kato, R. Rajkumar, Scheduling parallel real-time tasks on multi-core processors, in: 31st IEEE International Real-Time Systems Symposium, 2010, pp. 259–268.
[15] K. Li, Analysis of an approximation algorithm for scheduling independent parallel tasks, Discrete Math. Theor. Comput. Sci. 3 (1999) 155–166.
[16] K. Li, Job scheduling and processor allocation for grid computing on metacomputers, J. Parallel Distrib. Comput. 65 (11) (2005) 1406–1418.
[17] K. Li, An average-case analysis of online non-clairvoyant scheduling of independent parallel tasks, J. Parallel Distrib. Comput. 66 (5) (2006) 617–625.
[18] M. Pricopi, T. Mitra, Task scheduling on adaptive multi-core, IEEE Trans. Comput. 63 (10) (2014) 2590–2603.
[19] D.B. Shmoys, J. Wein, D.P. Williamson, Scheduling parallel machines on-line, in: Proc. 32nd IEEE Symposium on Foundations of Computer Science, 1991, pp. 131–140.
[20] B. Wang, Task parallel scheduling over multi-core system, in: First International Conference on Cloud Computing, in: LNCS, vol. 5931, 2009, pp. 423–434.
[21] Y. Wang, K. Li, H. Chen, L. He, K. Li, Energy-aware data allocation and task scheduling on heterogeneous multiprocessor systems with time constraints, IEEE Trans. Emerg. Top. Comput. 2 (2) (2014) 134–148.
[22] G. Xie, R. Li, K. Li, Heterogeneity-driven end-to-end synchronized scheduling for precedence constrained tasks and messages on networked embedded systems, J. Parallel Distrib. Comput. 83 (2015) 1–12.
[23] Y. Xu, K. Li, L. He, L. Zhang, K. Li, A hybrid chemical reaction optimization scheme for task scheduling on heterogeneous computing systems, IEEE Trans. Parallel Distrib. Syst. 26 (12) (2015) 3208–3222.
[24] L. Zhang, K. Li, C. Li, K. Li, Bi-objective workflow scheduling of the energy consumption and reliability in heterogeneous computing systems, Inform. Sci. 379 (2017) 241–256.

Dr. Keqin Li is a SUNY Distinguished Professor of computer science in the State University of New York. He is also a Distinguished Professor of Chinese National Recruitment Program of Global Experts (1000 Plan) at Hunan University, China. He was an Intellectual Ventures endowed visiting chair professor at the National Laboratory for Information Science and Technology, Tsinghua University, Beijing, China, during 2011–2014. His current research interests include parallel computing and high-performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU–GPU hybrid and cooperative computing, multicore computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of things and cyber–physical systems. He has published over 560 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently serving or has served on the editorial boards of IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Computers, IEEE Transactions on Cloud Computing, IEEE Transactions on Services Computing, and IEEE Transactions on Sustainable Computing. He is an IEEE Fellow.