World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:4, No:3, 2010

Performance Analysis of List Scheduling in Heterogeneous Computing Systems

Keqin Li

***Abstract*— Given a parallel program to be executed on a heterogeneous computing system, the overall execution time of the program is determined by a schedule. In this paper, we analyze the worst-case performance of the list scheduling algorithm for scheduling tasks of a parallel program in a mixed-machine heterogeneous computing system such that the total execution time of the program is minimized. We prove tight lower and upper bounds for the worst-case performance ratio of the list scheduling algorithm. We also examine the average-case performance of the list scheduling algorithm. Our experimental data reveal that the average-case performance of the list scheduling algorithm is much better than the worst-case performance and is very close to optimal, except for large systems with large heterogeneity. Thus, the list scheduling algorithm is very useful in real applications.**

***Keywords*— Average-case performance, list scheduling algorithm, mixed-machine heterogeneous computing system, worst-case performance.**

I. Introduction

A Key consideration in high performance parallel computing is the match between the computational needs of a parallel application and the advanced capabilities of a parallel machine. Homogeneous computing using one mode of parallelism, while providing good performance for some applications, has severe limitations in satisfying diversified characteristics of program codes efficiently. A parallel program or application may have various tasks with different architectural requirements. For applications with multiple types of parallelism, any single parallel machine will spend most of the time to execute ill-matched codes, resulting in significant degradation of overall system performance.

Heterogeneous computing is a parallel computing technique that solves computationally intensive problems with diverse computing requirements in an environment that incorporates a distributed suite of different autonomous high performance parallel machines interconnected by high speed links and networks. Heterogeneous computing is well-orchestrated, coordinated, and effective use of different types of machines, networks, software, and interfaces to maximize their combined performance and cost-effectiveness. Heterogeneous computing systems have the ability to match computing needs to appropriate computing resources [1], [3], [10], [12].

In a mixed-machine heterogeneous computing system, coarse-grained heterogeneity is supported at task level. A parallel program is divide into a collection of tasks (or modules, code segments), such that within a task, processing requirements are homogeneous. Through offline procedures called code-type profiling and analytical benchmarking [7], [13], the execution time of each task on each machine is measured, and the best machine to execute a task is identified. Then, the problem is scheduling (including mapping), that is, to assign the tasks to the machines and to determine when a task is to be executed. Given a parallel program to be executed on a heterogeneous computing system, the overall execution time of the program is determined by a schedule. A mixed-machine heterogeneous computing system coordinates the execution of tasks of a parallel program on different machines within the system to exploit various architectural features and achieve improved system performance.

Scheduling and mapping of tasks of parallel programs in heterogeneous computing systems have been studied extensively, where parallel programs are modeled by task precedence graphs (directed acyclic graphs) [14] or task interaction graphs (undirected graphs) [6]. For independent tasks [11], many heuristics have been developed and compared for static scheduling (i.e., the complete set of tasks to be scheduled is known in advance) [2] and dynamic scheduling (i.e., tasks arrive at different times and scheduling is performed as tasks arrive) [9].

In this paper, we investigate the problem of scheduling tasks of a parallel program in a mixed-machine heterogeneous computing system such that the total execution time of the program is minimized. Since the problem is NP-hard even in a homogeneous computing system [4], we consider approximation algorithms that produce near-optimal schedules. We analyze the worst-case performance of the *list scheduling* (LS) algorithm. In particular, we prove that the worst-case performance ratio of the list scheduling algorithm is in the range $[\gamma_2, \gamma_1]$, with

$$\gamma_1 = \frac{1}{\alpha}\left(2 - \frac{1}{m}\right),$$

and

$$\gamma_2 = \begin{cases} \frac{1}{\alpha}\left(2 - \left(\frac{2}{\alpha+1}\right)\frac{1}{m}\right) & \text{if } m \leq \frac{2}{1-\alpha}, \\ \frac{1}{\alpha}\left(2 - \frac{1}{m-1}\right) & \text{if } m \geq \frac{2}{1-\alpha}, \end{cases}$$

where m is the number of machines and α is a measure of system heterogeneity with $0 < \alpha \leq 1$, i.e., the smallest ratio of the execution time on the fastest machine to the execution time on the slowest machine for all tasks. Our result includes Graham's classic result in [5] for homogeneous

Manuscript received June 15, 2007.

The author is with the Department of Computer Science, State University of New York, New Paltz, NY 12561, USA. E-mail: lik@newpaltz.edu.

computing systems as a special case. Our result also improves the lower bound in [8] for the worst-case performance ratio of the list scheduling algorithm. We notice that although task scheduling in mixed-machine heterogeneous computing systems have been investigated experimentally, there is lack of such analytical result.

It is clear that precedence constraints, task execution times, together with the machine selection issue, make the scheduling problem in heterogeneous computing systems more complicated than the corresponding scheduling problem in homogeneous computing systems. We find that in the worst case, the list scheduling algorithm has limited ability to take advantage of the power of heterogeneous computing systems with large heterogeneity, due to its simple scheduling and mapping strategies. It is therefore an interesting problem to design more efficient algorithms that are able to assign tasks to the right machines at the right times.

In addition to the worst-case performance analysis, we also examine the average-case performance of the list scheduling algorithm. Our experimental data reveal that the average-case performance of the list scheduling algorithm is much better than the worst-case performance and is very close to optimal, except for large systems with large heterogeneity. Thus, the list scheduling algorithm, though very simple, is useful in real applications.

The rest of the paper is organized as follows. In Section 2, we give a parallel program model for heterogeneous computing, describe the task scheduling problem, and define performance measures. In Section 3, we present the list scheduling algorithm in heterogeneous computing systems. In Section 4, we analyze the worst-case performance ratio of the list scheduling algorithm. In Section 5, we demonstrate experimental data for the average-case performance bound of the list scheduling algorithm. Finally, we conclude the paper in Section 6.

II. Preliminaries

Let $H = (M_1, M_2, ..., M_m)$ be a mixed-machine heterogeneous computing system consisting of a suite of m autonomous machines. The m machines may have different processor architectures (e.g., SIMD, MIMD, VLIW, vector, superscalar, and special processors) and different execution speeds for the same task.

A parallel program P can be specified as a triplet $P = (T, \prec, \tau)$. T is a set of n tasks $T = \{T_1, T_2, ..., T_n\}$, each will be executed on one of $M_1, M_2, ..., M_m$. There is a set of precedence constraints, or, a partial order $\prec$ on T, such that if $T_{i_1} \prec T_{i_2}$, then T_{i_2} cannot start its execution until T_{i_1} is completed. The precedence constraints $\prec$ among the tasks can be represented by a task precedence graph, i.e., a directed acyclic graph (dag) in which, there are n nodes representing the n tasks $T_1, T_2, ..., T_n$, and there is an arc (T_{i_1}, T_{i_2}) if and only if $T_{i_1} \prec T_{i_2}$.

Task execution times are given by $\tau : T \times M \rightarrow (0, +\infty)$, where $\tau(T_i, M_j)$ is the execution time of task T_i on machine M_j. For each task T_i, there exists a permutation $\pi_i = (\pi_i(1), \pi_i(2), ..., \pi_i(m))$, such that

$$\tau(T_i, M_{\pi_i(1)}) \leq \tau(T_i, M_{\pi_i(2)}) \leq \cdots \leq \tau(T_i, M_{\pi_i(m)}).$$

Therefore, machine $M_{\pi_i(k)}$ is the kth fastest machine to execute task T_i. Let α be the largest value such that for each task T_i, the execution times on its fastest machine and the slowest machine are related as $\tau(T_i, M_{\pi_i(1)}) \geq \alpha\tau(T_i, M_{\pi_i(m)})$, where $0 < \alpha \leq 1$. The parameter α is a measure of heterogeneity of H.

A schedule is a specification of when and where to execute the tasks $T_1, T_2, ..., T_n$. Clearly, for a parallel program P and a heterogeneous computing system H, the overall execution time is determined by a schedule. The scheduling problem addressed in this paper is, given a parallel program P and a heterogeneous computing system H, finding a nonpreemptive schedule of tasks in P on H such that the overall execution time of the tasks in P is minimized.

The above scheduling problem is NP-hard even in a homogeneous computing system consisting of identical machines [4]. Therefore, we seek efficient algorithms that are able to generate acceptable schedules. The objective of this paper is to study the performance of the list scheduling algorithm which finds a near-optimal schedule of tasks in a parallel program P on a heterogeneous computing system H.

Let $A(P)$ be the length of a schedule produced by an algorithm A, and $OPT(P)$ be the length of an optimal schedule. Define

$$\gamma_A = \sup_P \left(\frac{A(P)}{OPT(P)} \right)$$

to be the *worst-case performance ratio* of algorithm A. If $A(P) \leq \gamma_1 OPT(P)$ for all P, then γ_1 is called a *worst-case performance bound* of algorithm A, i.e., $\gamma_A \leq \gamma_1$. If for any small $\epsilon > 0$, there exists an instance P such that $A(P) \geq (\gamma_2 - \epsilon)OPT(P)$, then the bound γ_2 is a lower bound for γ_A, i.e., $\gamma_A \geq \gamma_2$.

When a parallel program P contains random precedence constraints and/or random task execution times, both $A(P)$ and $OPT(P)$ are random variables. Define

$$\bar{\gamma}_A(P) = E\left(\frac{A(P)}{OPT(P)} \right)$$

to be the *average-case performance ratio* of algorithm A for random parallel program P, where $E(\cdot)$ stands for the expectation of a random variable. If $A(P)/OPT(P) \leq \beta$, then $\bar{\beta}$, the expectation of β, is called an *average-case performance bound* of algorithm A for random parallel program P, i.e., $\bar{\gamma}_A(P) \leq \bar{\beta}$.

III. The List Scheduling Algorithm

The list scheduling algorithm was originally proposed by Graham for scheduling precedence constrained tasks in homogeneous computing systems with $\alpha = 1$ (that is, a task T_i has the same execution time on all machines $M_1, M_2, ..., M_m$) [5]. A list schedule (i.e., a schedule produced by the list scheduling algorithm) is based on an initial ordering of the tasks $L = (T_{j_1}, T_{j_2}, ..., T_{j_n})$, called a *priority list*. Initially, at time zero, the scheduler instantaneously scans the list L from the beginning, searching for tasks that are ready to be executed, i.e., tasks which have no predecessors under $\prec$ still waiting in L. The first ready task in L is removed from L and assigned

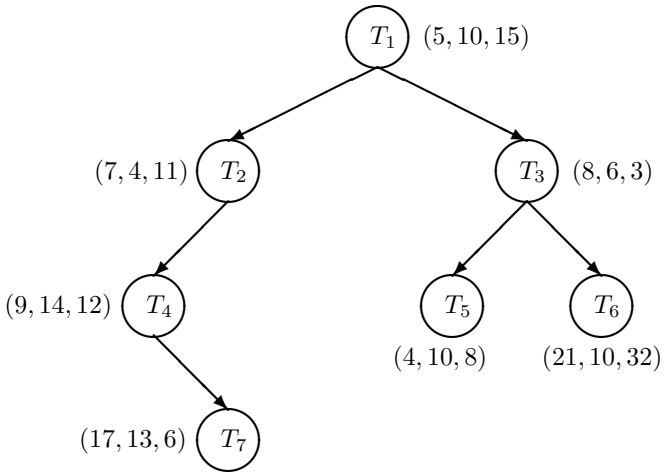

Fig. 1. An example parallel program with $n = 7$ tasks.

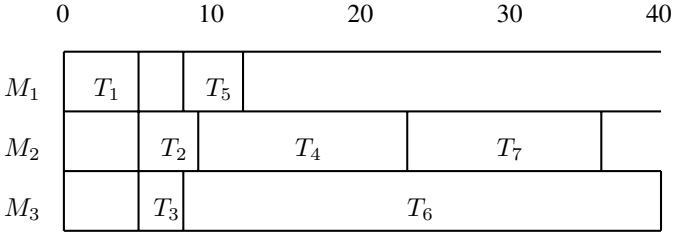

Fig. 2. A list schedule of length 40.

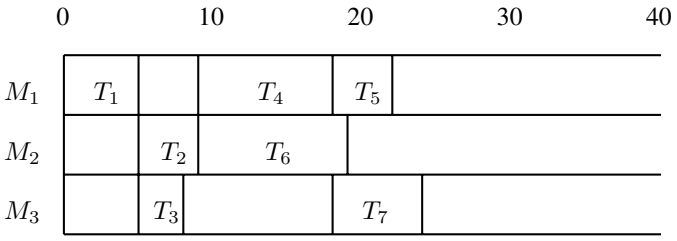

Fig. 3. An optimal schedule of length 24.

to a machine for processing. Such a search is repeated until there is no ready task or there is no more machine available. In general, whenever a machine completes a task, the scheduler immediately scans L, looking for the first ready task to be executed. If such a ready task is found, it is scheduled on that machine; otherwise, the machine becomes idle and waits for the next finished task. As running tasks complete, more precedence constraints are removed and more tasks will be ready.

For a parallel program P, the length (i.e., makespan) of a list schedule certainly depends on the priority list L. However, the performance of a list schedule is quite robust. Graham proved that the list schedule length is no more than twice the optimal schedule length for arbitrary L. In particular, let $LS(P, L)$ denote the length of a list schedule based on a priority list L, and $OPT(P)$ be the length of an optimal schedule. Graham proved the following well known result: for any parallel program P and any priority list L, we have

$$LS(P, L) \leq \left(2 - \frac{1}{m}\right) OPT(P);$$

furthermore, the bound $2-1/m$ is the best possible, i.e., $\gamma_{LS} = 2 - 1/m$ [5].

The excellent performance of the list scheduling algorithm motivates us to apply the strategy to scheduling tasks of parallel programs in mixed-machine heterogeneous computing systems. The extension of the algorithm to heterogeneous computing systems is straightforward. Again, a list schedule is based on a priority list, i.e., an initial ordering of the tasks. The priority list can be arbitrary and does not affect our analytical result. Initially, and whenever a machine completes a task, the scheduler searches for a ready task. If such a task is found, it is assigned to the best idle machine and is executed nonpreemptively on that machine. Ties are broken arbitrarily if there are several best idle machines for the task, e.g., selecting the best machine with the least index. The selection of the best machine is only for practical purpose. Theoretically, it does not matter which idle machine to choose, since the worst-case performance bound to be proven in Section 4 remains the same.

For example, let us consider a parallel program with $n = 7$ tasks shown in Figure 1 on a heterogeneous computing system with $m = 3$ machines. Each task T_i is described by a vector $(\tau(T_i, M_1), \tau(T_i, M_2), ..., \tau(T_i, M_m))$. The $\tau(T_i, M_j)$'s are chosen randomly only for the purpose of illustration. A list schedule based on the priority list $L = (T_1, T_2, ..., T_7)$ is displayed in Figure 2. Initially, at time 0, task T_1 is scheduled on its best machine M_1. When T_1 completes at time 5, tasks T_2 and T_3 are ready for execution and they are assigned to their best machines M_2 and M_3 respectively. When task T_3 completes at time 8, tasks T_5 and T_6 are ready for execution. Task T_5 is scheduled on M_1, since M_1 is the best machine of T_5. Task T_6 is assigned to M_3, since M_3 is the only available machine at this moment. At time 9, task T_2 is finished. Task T_4 is then scheduled on M_2. The completion of task T_5 does not introduce any more ready tasks. Finally, when task T_4 is finished at time 23, task T_7 is scheduled on M_2, since M_2 is the best available machine for T_7. We notice that it is not a good decision to schedule T_6 on M_3, because M_3 is the worst machine of T_6. However, the list scheduling algorithm does not have another choice at time 8. Although four tasks are executed on their best machines, the overall length of the list schedule is still 40, which is longer than that of an optimal schedule as depicted in Figure 3. In the optimal schedule, task T_6 is deferred until task T_2 finishes at time 9 so that T_6 can be executed on its best machine M_2. This means that there is an issue of machine selection. Task T_5 is deferred until T_4 completes because T_4 has a successor T_7, i.e., there is a consideration of the structure of a task precedence graph. It seems that the list scheduling algorithm has very limited capability to deal with such difficulties.

IV. Worst-Case Performance Analysis

In this section, we analyze the worst-case performance of the list scheduling algorithm that produces a near-optimal schedule for a parallel program P on a suite H of heterogeneous machines.

The main result of the paper is the following theorem, where the upper bound for γ_{LS} was originally proved in [8] and is included here for the sake of completeness, and the lower bound for γ_{LS} is improved here.

Theorem 1: The worst-case performance ratio of the list scheduling algorithm is $\gamma_2 \leq \gamma_{LS} \leq \gamma_1$, where

$$\gamma_1 = \frac{1}{\alpha}\left(2 - \frac{1}{m}\right),$$

and

$$\gamma_2 = \begin{cases} \frac{1}{\alpha}\left(2 - \left(\frac{2}{\alpha+1}\right)\frac{1}{m}\right) & \text{if } m \leq \frac{2}{1-\alpha}, \\ \frac{1}{\alpha}\left(2 - \frac{1}{m-1}\right) & \text{if } m \geq \frac{2}{1-\alpha}, \end{cases}$$

in heterogeneous computing systems.

Proof. We notice that when a task completes, ready tasks may be scheduled for execution. We call such a moment as a break point. Let $t_1, t_2, t_3, ..., t_b$ be the break points in a list schedule of a parallel program P based on a priority list L. Since several tasks may complete at the same time, we have $b \leq n$. Such break points divide the time interval $[0, LS(P, L))$ into $(b+1)$ half-open sub-intervals:

$$\begin{aligned} &I_0 = [t_0, t_1), \\ &I_1 = [t_1, t_2), \\ &I_2 = [t_2, t_3), \\ &..., \\ &I_{b-1} = [t_{b-1}, t_b), \\ &I_b = [t_b, t_{b+1}), \end{aligned}$$

where $t_0 = 0$ is the starting time, and $t_{b+1} = LS(P, L)$ is the completion time of P under the list schedule based on L.

We go through the above list of sub-intervals in its reverse order $I_b, I_{b-1}, ..., I_2, I_1, I_0$, and divide the set of sub-intervals $\{I_0, I_1, I_2, ..., I_{b-1}, I_b\}$ into two disjoint subsets $\Im_1$ and $\Im_2$ as follows.

Assume that T_{j_1} is a task that is completed at time $LS(P, L)$ and is scheduled at time t_{k_1}. Then, the sub-intervals I_{k_1}, I_{k_1+1}, ..., I_b are put into $\Im_1$. We say that I_{k_1}, I_{k_1+1}, ..., I_b are *covered* by task T_{j_1}. Now, we ask the question, "Why is task T_{j_1} not scheduled at time t_{k_1-1}?" There are two possible reasons for this, namely,

- (C1) task T_{j_1} is not ready at time t_{k_1-1}, i.e., there is a task T_{j_2} such that $T_{j_2} \prec T_{j_1}$, and T_{j_2} is still in execution or just scheduled at time t_{k_1-1}, or
- (C2) task T_{j_1} is ready at time t_{k_1-1} and is considered for execution; however, it is not scheduled because there is no idle machine to be allocated to T_{j_1}.

Under condition (C1), we assume that T_{j_2} starts at time t_{k_2}, and put I_{k_2}, I_{k_2+1}, ..., I_{k_1-1} into $\Im_1$, and say that these sub-intervals are covered by T_{j_2}. Then, we ask the question, "Why is task T_{j_2} not scheduled at time t_{k_2-1}?" Under condition (C2), I_{k_1-1} is put into $\Im_2$. We notice that all the m machines are busy during I_{k_1-1}; otherwise, task T_{j_1} would be scheduled at time t_{k_1-1}. We then continue to check sub-interval I_{k_1-2} and ask the question, "Why is task T_{j_1} not scheduled at time t_{k_1-2}?"

In general, assume that $I_r, I_{r+1}, ..., I_b$ have been checked and there exists a chain of tasks $T_{j_i} \prec T_{j_{i-1}} \prec \cdots \prec T_{j_1}$ such that each sub-interval in $\Im_1$ is covered by one of the tasks in the chain. Also, task T_{j_i} is ready at time t_r. Our question is "Why is task T_{j_i} not scheduled at time t_{r-1}?" Again, there are two possible reasons, i.e., precedence constraints (C1) and machine constraints (C2). If task T_{j_i} is not ready at time t_{r-1}, i.e., there is a task $T_{j_{i+1}}$ such that $T_{j_{i+1}} \prec T_{j_i}$, and $T_{j_{i+1}}$ is scheduled at time $t_{k_{i+1}}$ and still in execution at time t_{r-1}, we put $I_{k_{i+1}}, I_{k_{i+1}+1}, ..., I_{r-1}$ into $\Im_1$ and say that these sub-intervals are covered by $T_{j_{i+1}}$. If task T_{j_i} is ready at time t_{r-1} but not scheduled for execution, I_{r-1} is put into $\Im_2$. We then continue to check sub-interval I_{r-2}.

Using the above procedure, we obtain two disjoint subsets of sub-intervals $\Im_1$ and $\Im_2$, and a chain of l tasks $T_{j_l} \prec T_{j_{l-1}} \prec \cdots \prec T_{j_1}$ such that each sub-interval in $\Im_1$ is covered by one of the tasks in the chain. We know that all the m machines are allocated during a sub-interval in $\Im_2$; otherwise, more ready tasks would be scheduled at the beginning of the sub-interval. Also, it is clear that during a sub-interval in $\Im_1$, at least one machine is used.

Now, we are ready to establish the upper bound γ_1 for γ_{LS}, that is, for any heterogeneous computing system H, any parallel program P, and any initial priority list L, we have

$$LS(P, L) \leq \frac{1}{\alpha}\left(2 - \frac{1}{m}\right) OPT(P).$$

Let X_1 and X_2 be the total length of all the sub-intervals in $\Im_1$ and $\Im_2$ respectively. Then, we have

$$LS(P, L) = X_1 + X_2.$$

Let M_{p_i} be the machine that executes task T_i in the list schedule. Notice that M_{p_i} is not necessarily the best machine of T_i. In the worst case, M_{p_i} is the slowest machine to execute T_i. (Note: Since we are considering the worst machine, the worst-case performance bound is valid for any machine selection policy.) Nevertheless, in an optimal schedule, the execution time of T_i is still at least $\alpha\tau(T_i, M_{p_i})$. Since $T_{j_l}, T_{j_{l-1}}, ..., T_{j_1}$ must be executed sequentially in any schedule, we obtain

$$OPT(P) \geq \alpha X_1.$$

Let

$$W = \sum_{i=1}^{n} \tau(T_i, M_{p_i})$$

denote the total amount of work performed in the parallel program P under the list schedule. Then, we get

$$OPT(P) \geq \alpha\left(\frac{W}{m}\right).$$

Since at least one machine is used during a sub-interval in $\Im_1$, and all the m machines are allocated during a sub-interval in $\Im_2$, we have

$$W \geq X_1 + mX_2.$$

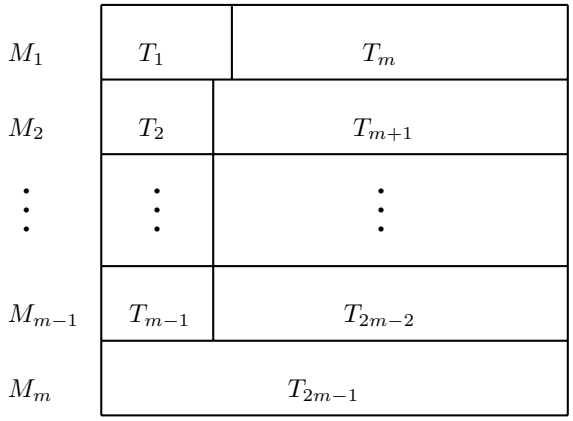

Fig. 4. An optimal schedule of P_1 with length m.

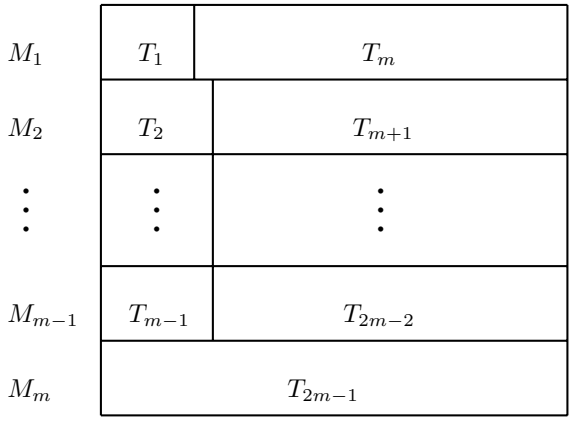

Fig. 6. An optimal schedule of P_2 with length m.

The above discussion can be summarized in the following inequality,

$$\begin{aligned}\frac{LS(P,L)}{OPT(P)} &\leq \frac{X_1+X_2}{\max(\alpha X_1, \alpha(X_1/m)+\alpha X_2)} \\ &= \frac{1}{\alpha}\cdot\frac{X_1+X_2}{\max(X_1, X_1/m+X_2)}.\end{aligned}$$

The inequality for the upper bound in the theorem can be obtained from

$$\frac{X_1+X_2}{\max(X_1, X_1/m+X_2)} \leq 2-\frac{1}{m},$$

which can be derived as follows. If $X_1 \geq X_1/m + X_2$, i.e., $X_2 \leq (1-1/m)X_1$, we get

$$\begin{aligned}\frac{X_1+X_2}{\max(X_1, X_1/m+X_2)} &= \frac{X_1+X_2}{X_1} \\ &= 1+\frac{X_2}{X_1} \\ &\leq 2-\frac{1}{m}.\end{aligned}$$

If $X_1 \leq X_1/m + X_2$, i.e., $X_2 \geq (1-1/m)X_1$, we get

$$\frac{X_1+X_2}{\max(X_1, X_1/m+X_2)} = \frac{X_1+X_2}{X_1/m+X_2}.$$

It is easy to verify that the right hand side of the above equation is a nonincreasing function of X_2 and it reaches its maximum value $2-1/m$ when $X_2 = (1-1/m)X_1$.

As for the first lower bound for γ_{LS}, i.e.,

$$\gamma_{LS} \geq \frac{1}{\alpha}\left(2-\left(\frac{2}{\alpha+1}\right)\frac{1}{m}\right),$$

we construct the following example program P_1. Let T contain $2m-1$ independent tasks T_1, T_2, ..., T_{m-1}, T_m, T_{m+1}, ..., T_{2m-2}, T_{2m-1} with no precedence constraint. The execution times of the $2m-1$ tasks on the m machines are described as follows.

- Task T_1's best machine is M_1 and T_1's execution time on M_1 is $2/(\alpha+1)$.
- For $2 \leq i \leq m-1$, task T_i's best machine is M_i and T_i's execution time on M_i is 1.
- Task T_m's best machine is M_1 and T_m's execution time on M_1 is $m-2/(\alpha+1)$.
- For $m+1 \leq i \leq 2m-2$, task T_i's best machine is M_{i-m+1} and T_i's execution time on M_{i-m+1} is $m-1$.
- Task T_{2m-1}'s best machine is M_m and its execution time on M_m is m.
- For all $1 \leq i \leq 2m-1$, the execution time of T_i on a machine other than its best machine is $1/\alpha$ times that on T_i's best machine.

Figure 4 shows an optimal schedule of P_1, where $OPT(P_1) = m$ and all tasks are executed on their best machines. Now, consider a list schedule with the initial priority list $L = (T_1, T_m, T_{m+1}, ..., T_{2m-2}, T_2, T_3, ..., T_{m-1}, T_{2m-1})$. Assume that the machine with the least index is chosen if there are ties. The list schedule based on L is depicted in Figure 5, where all tasks, except T_1, are executed on their worst machines. It is easy to verify that

$$\frac{2}{\alpha+1}+\frac{m-2}{\alpha} = \frac{1}{\alpha}\left(m-\frac{2}{\alpha+1}\right),$$

that is, tasks $T_1, T_2, ..., T_{m-1}$ are executed on M_1 and their total execution time is identical to the execution time of T_m on M_2. When T_{m-1} is completed, task T_{2m-1} is scheduled on M_1. Thus, the list schedule length is

$$\begin{aligned}LS(P_1,L) &= \frac{2}{\alpha+1}+\frac{m-2}{\alpha}+\frac{m}{\alpha} \\ &= \frac{2}{\alpha+1}+\frac{2m-2}{\alpha}.\end{aligned}$$

Therefore,

$$\begin{aligned}\frac{LS(P_1,L)}{OPT(P_1)} &= \frac{1}{m}\left(\frac{2}{\alpha+1}+\frac{2m-2}{\alpha}\right) \\ &= \frac{1}{\alpha}\left(2-\left(\frac{2}{\alpha+1}\right)\frac{1}{m}\right).\end{aligned}$$

As for the second lower bound for γ_{LS}, i.e.,

$$\gamma_{LS} \geq \frac{1}{\alpha}\left(2-\frac{1}{m-1}\right),$$

we construct the following example program P_2. Let T contain $2m-1$ independent tasks T_1, T_2, ..., T_{m-1}, T_m, T_{m+1}, ..., T_{2m-2}, T_{2m-1} with no precedence constraint. The execution

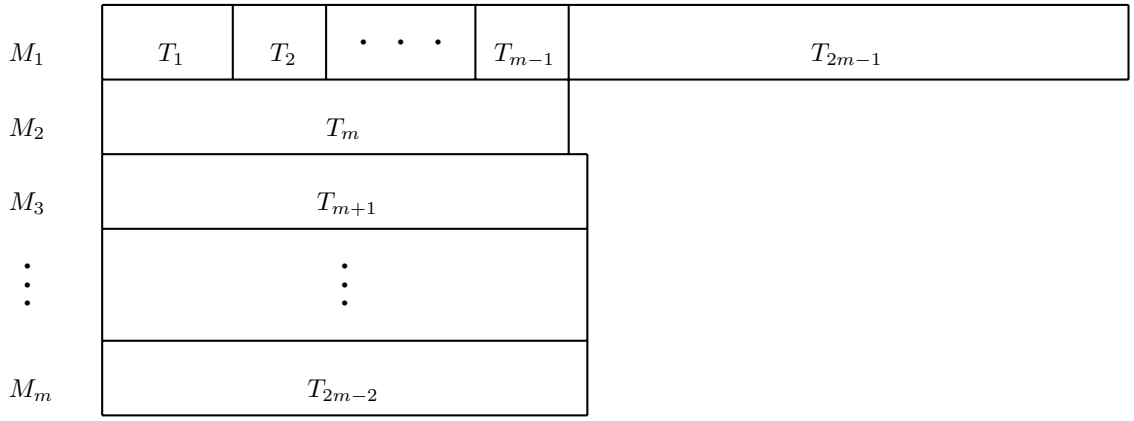

Fig. 5. A list schedule of P_1 with length $2/(\alpha+1)+(2m-2)/\alpha$.

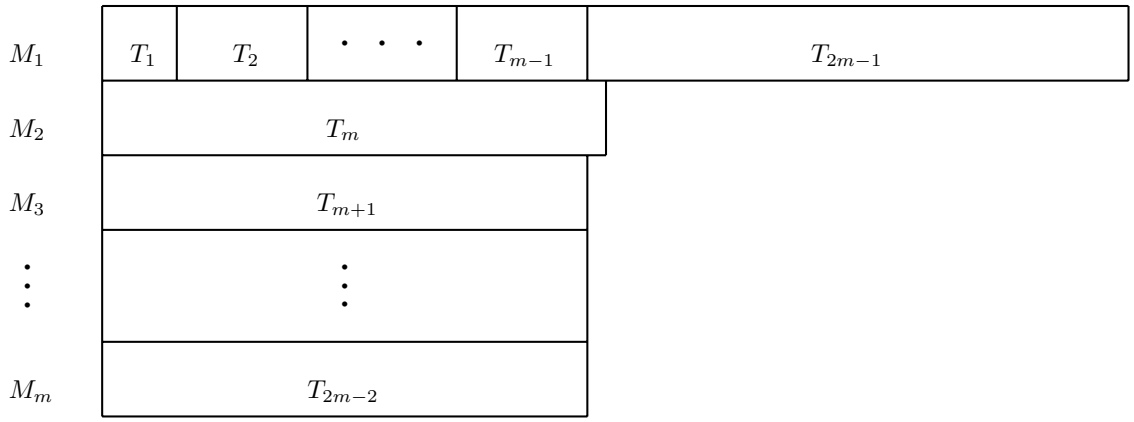

Fig. 7. A list schedule of P_2 with length $(2m-(m-\alpha\delta)/(m-1))/\alpha$.

times of the $2m-1$ tasks on the m machines are described as follows.

- Task T_1's best machine is M_1 and T_1's execution time on M_1 is a very small quantity δ.
- For $2 \le i \le m-1$, task T_i's best machine is M_i and T_i's execution time on M_i is $(m-\alpha\delta)/(m-1)$.
- Task T_m's best machine is M_1 and T_m's execution time on M_1 is $m-\delta$.
- For $m+1 \le i \le 2m-2$, task T_i's best machine is M_{i-m+1} and T_i's execution time on M_{i-m+1} is $m-(m-\alpha\delta)/(m-1)$.
- Task T_{2m-1}'s best machine is M_m and its execution time on M_m is m.
- For all $1 \le i \le 2m-1$, the execution time of T_i on a machine other than its best machine is $1/\alpha$ times that on T_i's best machine.

Figure 6 shows an optimal schedule of P_2, where $OPT(P_2)=m$ and all tasks are executed on their best machines. Now, consider a list schedule with the initial priority list $L=(T_1,T_m,T_{m+1},...,T_{2m-2},T_2,T_3,...,T_{m-1},T_{2m-1})$. The list schedule based on L is depicted in Figure 7, where all tasks, except T_1, are executed on their worst machines. It is easy to verify that

$$\delta+\frac{(m-2)(m-\alpha\delta)}{\alpha(m-1)}=\frac{1}{\alpha}\left(m-\frac{m-\alpha\delta}{m-1}\right),$$

that is, tasks $T_1,T_2,...,T_{m-1}$ are executed on M_1 and their total execution time is identical to the execution time of T_i on M_{i-m+2}, where $m+1 \le i \le 2m-2$. When T_{m-1} is completed, task T_{2m-1} is scheduled on M_1. Thus, the list schedule length is

$$\begin{aligned} LS(P_2,L) &= \delta+\frac{(m-2)(m-\alpha\delta)}{\alpha(m-1)}+\frac{m}{\alpha} \\ &= \frac{1}{\alpha}\left(2m-\frac{m-\alpha\delta}{m-1}\right). \end{aligned}$$

Therefore,

$$\frac{LS(P_2,L)}{OPT(P_2)}=\frac{1}{\alpha}\left(2-\frac{1}{m-1}\cdot\left(1-\frac{\alpha\delta}{m}\right)\right).$$

Since δ can be arbitrarily small, we obtain the second lower bound.

Finally, we notice that

$$\frac{1}{\alpha}\left(2-\left(\frac{2}{\alpha+1}\right)\frac{1}{m}\right) \ge \frac{1}{\alpha}\left(2-\frac{1}{m-1}\right),$$

that is,

$$\left(\frac{2}{\alpha+1}\right)\frac{1}{m} \le \frac{1}{m-1},$$

if and only if $m \le 2/(1-\alpha)$. Hence, we get the combined lower bound γ_2. This completes the proof of the theorem. ∎

TABLE I
NUMERICAL DATA FOR THE PERFORMANCE BOUNDS γ_2 AND γ_1.

	$\alpha = 0.2$		$\alpha = 0.4$		$\alpha = 0.6$		$\alpha = 0.8$		$\alpha = 1.0$	
m	γ_2	γ_1	γ_2	γ_1	γ_2	γ_1	γ_2	γ_1	γ_2	γ_1
2	5.8333	7.5000	3.2143	3.7500	2.2917	2.5000	1.8056	1.8750	1.5000	1.5000
3	7.5000	8.3333	3.8095	4.1667	2.6389	2.7778	2.0370	2.0833	1.6667	1.6667
4	8.3333	8.7500	4.1667	4.3750	2.8125	2.9167	2.1528	2.1875	1.7500	1.7500
5	8.7500	9.0000	4.3750	4.5000	2.9167	3.0000	2.2222	2.2500	1.8000	1.8000
6	9.0000	9.1667	4.5000	4.5833	3.0000	3.0556	2.2685	2.2917	1.8333	1.8333
7	9.1667	9.2857	4.5833	4.6429	3.0556	3.0952	2.3016	2.3214	1.8571	1.8571
8	9.2857	9.3750	4.6429	4.6875	3.0952	3.1250	2.3264	2.3438	1.8750	1.8750
9	9.3750	9.4444	4.6875	4.7222	3.1250	3.1481	2.3457	2.3611	1.8889	1.8889
10	9.4444	9.5000	4.7222	4.7500	3.1481	3.1667	2.3611	2.3750	1.9000	1.9000

Theorem 1 contains Graham's classic result in [5] for homogeneous computing systems as a special case when $\alpha = 1$.

In Table 1, we show the numerical values for the lower bound γ_2 and the upper bound γ_1 for γ_{LS}. We observe that the lower bound and the upper bound are very close, except when both m and α are very small.

It is clear that in addition to precedence constraints and task execution times, there is also a machine selection issue, which makes the scheduling problem more complicated than the one solved in [5]. Theorem 1 implies that in the worst case, the list scheduling algorithm has limited ability to take advantage of the power of heterogeneous computing systems with large heterogeneity, due to its simple scheduling and mapping strategies, that is, scheduling a task as soon as it is ready and not always assigning a task to its best machine. It is therefore an interesting problem to design more efficient algorithms that are able to assign tasks to the right machines at the right times.

V. SIMULATION RESULTS

In this section, we present simulation results on the average-case performance of the list scheduling algorithm in heterogeneous computing systems. Analysis of the average-case performance ratio is beyond the scope of the paper.

For the purpose of average-case performance evaluation, we need models to randomize precedence constraints among the tasks and execution times of the tasks in a parallel program. In our simulations, a parallel program $P = (T, \prec, \tau)$ with random dag structure and random execution times is generated as follows.

- We use the random graph model DAG(n, p) to produce a random dag. In such a random dag with n tasks $T_1, T_2, ..., T_n$, the probability that there exists an arc (T_{i_1}, T_{i_2}) is p, where $0 \leq p \leq 1$, for all pairs (i_1, i_2) with $1 \leq i_1 < i_2 \leq n$. The existence of an arc is independent of the existence of other arcs.
- The $\tau(T_i, M_j)$'s are random variables generated as follows. For each task T_i, we first pick M_k as the worst machine for T_i, where k is uniformly distributed in $\{1, 2, ..., m\}$. We then generate $\tau(T_i, M_k)$ which is uniformly distributed in the interval $(0, 1)$, and for each j, $1 \leq j \leq m$ and $j \neq k$, we generate $\tau(T_i, M_j)$ which is uniformly distributed in the interval $(\alpha\tau(T_i, M_k), \tau(T_i, M_k))$.

Although other models of random dags and random task execution times can be employed, we believe that they will not change our conclusions.

While the list schedule length $LS(P, L)$ can be obtained by a simulation program for arbitrary parallel program P, it is infeasible to calculate $OPT(P)$ in reasonable amount of time. We use the following lower bound for $OPT(P)$,

$$OPT(P) \geq \frac{W^*}{m},$$

where

$$W^* = \sum_{i=1}^{n} \tau(T_i, M_{\pi_i(1)})$$

is the total amount of work performed by the best machines. It is clear that $\bar{\beta}$ is an average-case performance bound, where

$$\beta = \frac{LS(P, L)}{W^*/m}.$$

The purpose of our experiments is to reveal the value of $\bar{\beta}$.

Tables 2–4 show our simulation data for three random parallel programs whose precedence constraints are specified by DAG(n, p), where $n = 200$ and $p = 0, 0.05, 0.1$. For each combination of DAG(n, p), m, and α, we generate 500 samples of a random parallel program. For each sample of a random parallel program, we compute its W^*, simulate the list scheduling algorithm and get the list schedule length, and calculate the value of β. The average value of β is considered as the experimental value of $\bar{\beta}$. The 99% confidence interval of all the data in each table is also given.

From these data, we observe that the average-case performance of the list scheduling algorithm is much better than the worst-case performance and is very close to optimal, except for the case when m is large and α is small. One should also notice that $\bar{\beta}$ only gives an upper bound for $\bar{\gamma}_{LS}$; the estimation of $OPT(P)$ using W^*/m is definitely too optimistic. Therefore, the actual performance of the list scheduling algorithm should be even better.

VI. SUMMARY

We have analyzed the worst-case performance ratio of the list scheduling algorithm for scheduling tasks of a parallel

TABLE II
SIMULATION DATA FOR THE AVERAGE-CASE PERFORMANCE BOUND $\bar{\beta}$ ON DAG(200,0) (99% CONFIDENCE INTERVAL $\pm 0.478\%$).

m	$\alpha = 0.2$	$\alpha = 0.4$	$\alpha = 0.6$	$\alpha = 0.8$	$\alpha = 1.0$
2	1.3383	1.2200	1.1297	1.0596	1.0034
3	1.5858	1.3461	1.1912	1.0848	1.0073
4	1.7741	1.4299	1.2302	1.1015	1.0120
5	1.9284	1.4896	1.2555	1.1135	1.0165
6	2.0584	1.5360	1.2769	1.1235	1.0213
7	2.1636	1.5751	1.2982	1.1333	1.0264
8	2.2558	1.6092	1.3123	1.1414	1.0313
9	2.3417	1.6419	1.3245	1.1485	1.0354
10	2.4142	1.6685	1.3391	1.1561	1.0414

TABLE III
SIMULATION DATA FOR THE AVERAGE-CASE PERFORMANCE BOUND $\bar{\beta}$ ON DAG(200,0.05) (99% CONFIDENCE INTERVAL $\pm 1.297\%$).

m	$\alpha = 0.2$	$\alpha = 0.4$	$\alpha = 0.6$	$\alpha = 0.8$	$\alpha = 1.0$
2	1.3391	1.2181	1.1294	1.0598	1.0038
3	1.5902	1.3452	1.1922	1.0864	1.0088
4	1.7828	1.4364	1.2343	1.1059	1.0162
5	1.9445	1.5027	1.2666	1.1222	1.0253
6	2.0952	1.5637	1.2982	1.1422	1.0360
7	2.2483	1.6256	1.3339	1.1643	1.0553
8	2.4106	1.7023	1.3903	1.1979	1.0885
9	2.6176	1.8115	1.4571	1.2542	1.1322
10	2.8932	1.9547	1.5564	1.3344	1.2031

TABLE IV
SIMULATION DATA FOR THE AVERAGE-CASE PERFORMANCE BOUND $\bar{\beta}$ ON DAG(200,0.1) (99% CONFIDENCE INTERVAL $\pm 1.311\%$).

m	$\alpha = 0.2$	$\alpha = 0.4$	$\alpha = 0.6$	$\alpha = 0.8$	$\alpha = 1.0$
2	1.3380	1.2205	1.1296	1.0600	1.0045
3	1.5944	1.3535	1.1967	1.0903	1.0127
4	1.8303	1.4664	1.2568	1.1252	1.0339
5	2.1347	1.6172	1.3621	1.2050	1.0974
6	2.5431	1.8527	1.5280	1.3384	1.2158
7	3.0061	2.1568	1.7585	1.5138	1.3681
8	3.5409	2.4731	1.9672	1.7147	1.5321
9	4.0979	2.7948	2.2264	1.9248	1.7304
10	4.6251	3.1283	2.4790	2.1359	1.9303

program in a mixed-machine heterogeneous computing system such that the total execution time of the program is minimized. We also presented experimental data to demonstrate the average-case performance which shows that the simple list scheduling algorithm is very useful in real applications.

As a further research direction, it is interesting to close the gap between the lower and upper bounds in Theorem 1. It is also important to design more efficient algorithms with improved worst-case and average-case performance.

REFERENCES

[1] S. Ali, T. D. Braun, H. J. Siegel, and A. A. Maciejewski, "Heterogeneous computing," in *Encyclopedia of Distributed Computing*, J. Urbana and P. Dasgupta, eds., Kluwer Academic, Norwell, MA, 2001.

[2] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, vol. 61, pp. 810-837, 2001.

[3] M. M. Eshaghian, ed., *Heterogeneous Computing*, Artech House, Norwood, MA, 1996.

[4] M. R. Garey and D. S. Johnson, *Computers and Intractability – A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, 1979.

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

[6] M. Kafil and I. Ahmad, "Optimal task assignment in heterogeneous distributed computing systems," *IEEE Concurrency*, vol. 6, no. 3, pp. 42-51, 1998.

[7] A. A. Khokhar, V. K. Prasanna, M. E. Shaaban, and C. L. Wang, "Heterogeneous computing: challenges and opportunities," *Computer*, vol. 26, no. 6, pp. 18-27, 1993.

[8] K. Li and J. E. Dorband, "A task scheduling algorithm for heterogeneous processing," *Proceedings of the 5th High Performance Computing Symposium*, pp. 183-188, 1997.

[9] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, vol. 59, pp. 107-131, 1999.

[10] M. Maheswaran, T. D. Braun, and H. J. Siegel, "Heterogeneous distributed computing," in *Encyclopedia of Electrical and Electronics Engineering*, J. G. Webster, ed., Wiley, New York, pp. 679-690, 1999.

[11] H. J. Siegel and S. Ali, "Techniques for mapping tasks to machines in heterogeneous computing systems," *Journal of Systems Architecture*, vol. 46, pp. 627-639, 2000.

[12] H. J. Siegel, J. K. Antonio, R. C. Metzger, M. Tan, and Y. A. Li, "Heterogeneous computing," in *Parallel and Distributed Computing Handbook*, A. Y. Zomoya ed., McGraw-Hill, New York, pp. 725-761, 1996.

[13] H. J. Siegel, H. G. Dietz, and J. K. Antonio, "Software support for heterogeneous computing," in *The Computer Science and Engineering Handbook*, A. B. Tucker, Jr. ed., pp. 1886-1909, CRC Press, Boca Raton, FL, 1997.

[14] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260-274, 2002.