一个新的相关任务调度算法*)

张建军[1,2] 李庆华[1]

(华中科技大学计算机科学与技术学院 武汉 430074)[1] (海军工程大学理学院 武汉 430033)[2]

摘 要 现已有许多调度算法在某些特定条件下能产生最优调度。Darbha 和 Agrawal 提出的 TDS 算法能产生最优调度,其最优条件比较苛刻,实用性不强。Park 和 Choe 提出一种扩展调度算法(Extended TDS),虽然其最优条件比 TDS 算法的约束条件宽松些,但在任务数较多时难以满足,并且形式过于复杂。因此,本文提出一种能产生最优调度的新算法,该算法既考虑合并其它父任务以减少通讯时间,同时尽可能少地合并其它任务,从而尽量减小任务的启动时间。该算法不仅最优条件简单、宽松,而且具有与 TDS 算法相同的时间复杂度 $O(v^2)$。

关键词 任务复制,最优条件,最优调度

A New Scheduling Algorithm for Dependent Tasks

ZHANG Jian-Jun[1,2] LI Qing-Hua[1]

(Department of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074)[1]

(College of Science, Naval University of Engineering, Wuhan 430033)[2]

Abstract There are some algorithms that are able to find an optimal schedule under certain conditions. The task duplication based scheduling (TDS) algorithm proposed by Darbha and Agrawal generates an optimal schedule, but its optimality condition is so restricted that it cannot be applied. Thus, Park and Choe proposed an extended TDS algorithm whose optimality condition is less restricted than that of TDS algorithm, but the condition is very complex and is difficult to satisfy when the number of tasks is large. In this paper, we propose a new algorithm that can generate an optimal schedule, which attempts to merge tasks of parent clusters as little as possible in order to reduce the start time of task.. The propose algorithm not only has a simple and loose optimality condition, but has the same time complexity as TDS algorithm that is $O(v^2)$.

Keywords Task duplication, Optimality condition, Optimal schedule

1 引言

任务调度就是根据一定的调度规则和调度策略,把组成并行程序的一组任务或构成工作负载的一组作业,按照一定执行时序分配到并行分布系统的多个计算节点上,以最小化并行应用程序的完成时间,以期取得较好的系统性能。任务调度的目标是把组成并行程序的一组相关任务分配到多个处理机,以使得程序的完成时间最短。通常,我们用有向无环图(DAG)来描述相关任务。任务图由一个四元组 $G=(V, E, w, c)$来表示:其中,$V=\{n_1, n_2, \cdots, n_m\}$是任务集,$v=|V|$表示任务的个数;$E$ 是边的集合,$e=|E|$表示边的个数。w 是计算开销集合:$w(n_i)$ 是任务 $n_i(n_i \in V)$的计算开销;c 是通信开销集合:对于每条边,如从任务 n_i 到任务 n_j 的边 $e(n_i, n_j) \in E$,则它的通信开销为 $c(n_i, n_j)$。对于任务 n_i,$pred(n_i)=\{n_k | e(n_k, n_i) \in E\}$表示其父任务节点集合,$succ(n_i)=\{n_k | e(n_i, n_k) \in E\}$表示其子任务节点集合。如果$|pred(n_i)| \geqslant 2$,则任务 n_i 称为 *join* 任务;如果$|pred(n_i)|=0$,则任务 n_i 称为初始(entry)任务;如果$|succ(n_i)|=0$,则任务 n_i 称为终止(exit)任务。

当前,基于任务复制的调度已成为研究热点[1~4, 8]。任务复制是指在不同处理机上执行相同的任务,一个任务可以被同时分配到多个处理机上,从而可以减少任务间的通讯时间,提前任务的开始时间和完成时间。任务复制不仅能保留程序最初的并行性,而且能减少任务间的通信开销。因此,任务复制是消除任务间通信开销的非常有效的方法。一般来说,基于任务复制的调度绝对优于不是基于任务复制的调度[4]。然而,无论任务是否复制,多处理机调度问题始终是一个 NP 完全问题。现已有许多基于任务复制的调度算法在任务满足某些条件时能产生最优调度[1~4,8]。Darbha 和 Agrawal 提出了一个基于任务复制(TDS)的调度算法[1]。对于每个 join 任务,该算法在满足以下两个条件之一的情况下能产生最优调度:

$$\begin{cases} w(n_i) \geqslant c(n_j, n_a), & \text{if } est(n_i) \geqslant est(n_j) \\ w(n_i) \geqslant (c(n_j, n_a) + est(n_j) - est(n_i)), & \text{if } est(n_i) < est(n_j) \end{cases}$$

其中,n_i 和 n_j 分别表示具有最大和次最大$\{ect(n_j)+c(n_j, n_a) | n_j \in pred(n_a)\}$值的 n_a 的父任务。该算法的主要缺陷是 join 任务只与具有最大值的父任务合并,而不考虑其它的父任务,从而失去了产生更短调度的机会,尤其是当通信开销较大的情况下。因此,TDS 算法只适合通信开销较小的情况。针对 TDS 算法存在的问题,Park 和 Choe 随后提出了一种扩展调度算法(extended TDS)[2]。由于该算法考虑合并其它父任务,因而能产生比 TDS 更短的调度。对于每次合并,

*)国家自然科学基金项目(6027307)。**张健军** 副教授,博士研究生,主要研究方向为并行与分布式计算、网格计算;**李庆华** 教授,博导,主要研究方向为并行与分布式计算、高性能算法。

如果能满足下列条件，该算法能产生最优调度：

$$\min_{n_e \in M'} [ect(n_e) + \max_{n_h \in succ(n_e) \cap M'} (c(n_i, n_h) + w(n_h) + \sum_{n_j \in S(n_h) \cap M'} w(n_j))] \geq ct(M')$$

其中，$M = \bigcup_{i=1}^{r} C(n_i)$，$M' = merge(M, C(n_i))$，而 $S(n_i)$ 表示任务 n_i 的所有子孙任务。虽然该算法的最优条件比 TDS 算法的条件要宽松些，但形式过于复杂，并且在任务数较多时难以满足。

因此，针对上述算法存在的问题，本文提出一种新算法，该算法既考虑合并其它父任务以减少通讯时间，同时尽可能少地合并父任务，而不盲目合并父任务 Cluster 中的所有祖先任务，因而能产生比 TDS 和 extended TDS 算法更短的调度。而且，该算法最优条件简单、宽松，具有与 TDS 算法相同的时间复杂度 $O(v^2)$。

2 新调度算法

本算法的目的是尽量减少程序的完成时间。对于任务 n_i，$est(n_i)$ 和 $ect(n_i)$ 分别表示其最早开始时间和最早完成时间。这里考虑的是非抢占式调度，因此 $ect(n_i) = est(n_i) + w(n_i)$。我们把分配到同一处理机上的任务集合称为一个族(Cluster)，对于族 C 中每个任务 n_i，$st(n_i, C)$ 和 $ct(n_i, C)$ 分别表示 n_i 在族 C 中的开始时间和完成时间。因此，对于任何族 C，$est(n_i)$ 总是小于或等于 $st(n_i, C)$，即 $est(n_i) \leqslant st(n_i, C)$。$ct(C)$ 表示族 C 中最后一个任务 n_{end} 的完成时间，即 $ct(C) = ct(n_{end}, C)$。程序的最早完成时间实际上等于终止任务 n_{exit} 的最早完成时间，即 makespan $= ect(n_{exit})$。为了计算 n_{exit} 的最早完成时间，必须先计算其父任务的最早完成时间 ect 和最早开始时间 est。

```
Algorithm Schedule(V,E,w,c)
Begin
  est(n1)=0;C(n1)={n1};
  ReadyTaskQueue=succ(n1);
  while (|ReadyTaskQueue|>0) do
    na←delete a task from ReadyTaskQueue;
    Compute_est(na,C(na),est(na));
    ReadyTaskQueue←{nk|For∀ni∈pred(nk),∃C(ni)});
  endwhile
  i=n;j=0;
  while i>1 do
    if task ni has not been scheduled
    then {j=j+1; Assign C(ni) to processor Pj;}
    i=i-1;
  endwhile
End
Procedure Compute_est(na,C(na),est(na))
Begin
  // DAT(C(n1),na)≥…≥DAT(C(ns),na)≥DAT(C(ns+1),na)
  =0
  k=1;M1=C(n1);
  do
    k=k+1;Mk=Mk-1∪{nk};
  while (k≤s)and(ct(Mk-1)≤DAT(C(nk-1),na)) and (ct(Mk)>
  DAT(C(nk),na))
  C(na)=Mk-1∪{na};
  est(na)=max(ct(Mk-1),DAT(C(nk),na));
End
```

图 1 算法描述

算法 *Schedule*() 依次从开始任务到终止任务分别计算每个 *ready* 任务的 *est* 和构造其 Cluster。如果一个任务的所有父任务准备就绪，那么这个任务称为 *ready* 任务。首先，初始任务 n_1 没有父任务，理所应当成为 *ready* 任务；一旦某任务的所有父任务计算完毕，该任务即可加入 *ReadyTaskQueue*，其中 *ReadyTaskQueue* 是 *ready* 任务的集合队列。*Compute_est*() 是算法的主要部分，其主要工作是计算 *ready* 任务的 *est* 和构造其 Cluster。一旦所有任务的 *est* 计算完毕，其对应的 Cluster 集合也就构造出来，并将各个 Cluster 分配到不同的处理机。

2.1 计算 *est*

算法的主要部分在于如何得到任务的 *est*，图 1 列出了如何计算 *est* 的详细过程。下面我们通过图示来说明 *est* 的计算方法。对于任务 n_a，$est(n_a)$ 取决于其父任务的个数。如果 n_a 只有一个父任务，$C(n_a)$ 应该等于 $C(n_i) \cup \{n_a\}$，如图 2(*b*) 所示。如果 n_a 不与 $C(n_i)$ 合并，n_a 的开始时间就会延迟至 $ect(n_i) + c(n_i, n_a)$，我们把这个值定义为 $DAT(C(n_i), n_a)$，如图 2(*a*) 所示。如果 n_a 与 $C(n_i)$ 合并，由于 n_a 与 n_i 在同一 Cluster 中，通讯时间 $c(n_i, n_a)$ 可忽略不计；因此，n_a 的开始时间可提前到 $ect(n_i)$。所以，我们可以得到：$est(n_a) = ect(n_i)$ 且 $C(n_a) = C(n_i) \cup \{n_a\}$。

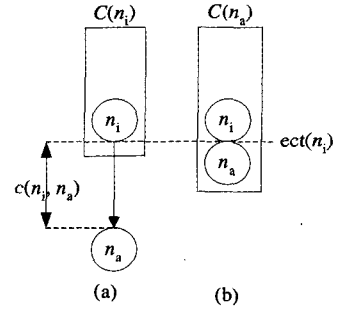
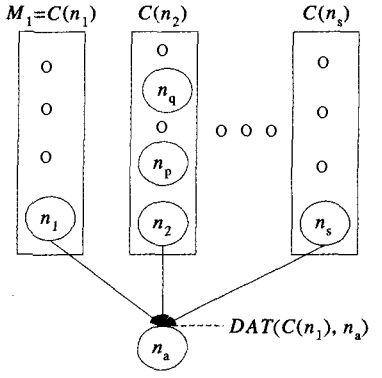
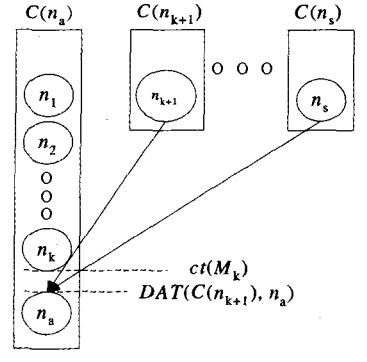


图 2 一个父任务



(a) 归并前



(b) 归并后

图 3 多个父任务

如果任务 n_a 有 s ($s>1$) 个父任务 n_1，n_2，…，n_s，其中 $DAT(C(n_1), n_a) \geqslant DAT(C(n_2), n_a) \geqslant \cdots \geqslant DAT(C(n_s), n_a)$。

首先，如果把 $C(n_1)$ 与 n_a 合并，n_a 的开始时间可以提前到 $\max(ct(C(n_1)), DAT(C(n_2), n_a))$。否则，$n_a$ 的开始时间会延迟到 $DAT(C(n_1), n_a)$。随后，寻找最大索引值 $k(2 \leqslant k \leqslant s)$，$k$ 是使得 n_a 的开始时间最小的索引值，它满足下列条件：$ct(M_k) \leqslant DAT(C(n_k), n_a)$ 而且 $ct(M_{k+1}) > DAT(C(n_{k+1}), n_a))$，其中，$M_k = C(n_1) \bigcup_{i=2}^{k} \{n_i\}$。因此，$k$ 是缩短 n_a 的开始时间的最大值。一旦 k 确定，任务的最早开始时间就确定。因此，$est(n_a) = \max(ct(M_k), DAT(C(n_{k+1}), n_a))$，而 $C(n_a) = M_k \bigcup \{n_a\}$。

基于上述操作，能够得到 $est(n_a) \leqslant \max(ect(n_1), DAT(C(n_2), n_a))$。因此，该算法能产生比 TDS 算法更好的调度。当通信开销较小的情况下，即使 k 不存在，也能产生与 TDS 相同的调度长度。由于父任务 Cluster 中的某些任务可能会延迟后续任务的开始时间，因此算法并不试图合并父任务 Cluster 中的所有任务，而只是试图合并父任务，则 $ct(M_k) \leqslant ct(\bigcup_{i=1}^{k} C(n_i))$。因此，$est(n_a) = \max(ct(M_k), DAT(C(n_{k+1}), n_a)) \leqslant \max(ct(\bigcup_{i=1}^{k} C(n_i)), DAT(C(n_{k+1}), n_a))$。所以，该算法能产生比 extended TDS 算法更短的调度。

2.2 最优证明

对于任何 join 任务，下面分析 *Compute_est*() 的最优性。通过合并父任务（按父 Cluster 的完成先后时间排序）来减少通讯时间，但这样合并父任务是否能最优地减少 join 任务的 est，定理 1 对此进行证明。

定理 1 考虑一个 join 任务 n_a，具有 s 个父任务 $n_1, n_2, \cdots, n_s$，其中 $DAT(C(n_1), n_a) \geqslant DAT(C(n_2), n_a) \geqslant \cdots \geqslant DAT(C(n_s), n_a)$。

如果下列条件满足：

$$DAT(C(n_p), n_{i+1}) \leqslant st(n_{i+1}, M_{i+1}) \quad \text{if} \quad ct(M_{i+1}) \leqslant \max_{n_p \in pred(n_{i+1}) \wedge n_p \notin M_i} DAT(C(n_{i+1}), n_a))$$

则 *Compute_est*() 能产生 n_a 的最小开始时间 $est(n_a) = \max(ct(M_k), DAT(C(n_{k+1}), n_a))$，其中 $M_k = C(n_1) \bigcup_{i=2}^{k} \{n_i\}$，而且 k 是能减少 n_a 的开始时间的最大值。

证明：

对于 join 任务 n_a，父任务 $C(n_1)$ 首先应该考虑合并。通过合并使得 $c(n_1, n_a)$ 变为 0。因此，n_a 的开始时间可以减小，但仍然受限于 $DAT(C(n_2), n_a)$。对于父任务 n_2 的每个父任务 n_p，如果 n_p 不在 M_1 中，即 $n_p \in pred(n_2) \wedge n_p \notin M_1$，而且 n_p 到达 n_2 的时间小于或等于 n_2 在 M_2 中的开始时间的话，即 $DAT(C(n_p), n_2) \leqslant st(n_2, M_2)$，$n_2$ 应该合并，即 $M_2 = M_1 \bigcup \{n_2\}$。因此，由于 $c(n_2, n_a)$ 的变 0 使得 n_a 的开始时间可以减少。不断重复这样一个过程，直到考虑父任务 n_{k+1} 时，因为合并 n_{k+1} 会增加 n_a 的开始时间，即 $ct(M_k \bigcup \{n_{k+1}\}) > DAT(C(n_{k+1}), n_a)$，也就是说 k 是能减少 n_a 的开始时间的最大索引值。因此，$est(n_a) = \max(ct(M_k), DAT(C(n_{k+1}, n_a))$ 就是最小的、最优的。

2.3 时间复杂度分析

算法 *schedule*() 类似于广度优先搜索，复杂度为 $O(e+v)$，其中 e 和 v 分别表示图中边和任务的个数。对于每个任务 n_i，必须计算其 $est(n_i)$，因此要计算 v 次 est。每个任务的 est 是由 *Compute_est*() 来完成。如果图中任务的最大入度为 d，即任务最多有 d 个父任务，由于每次合并仅仅考虑父任务，因此对于每个 join 任务合并的最大复杂度为 $O(d)$，而 join 任务的 d 个父任务的排序时间为 $O(d \log d)$。因此，过程 *Compute_est*() 的复杂度就为 $O(v(d \log d + d))$，整个算法的复杂度为 $O(e+v+v(d \log d+d))$。一般来说，$d \ll v$，因此 $d \log d < v$，而对于稠密图 $O(e)$ 相当于 $O(v^2)$。因此，算法最坏情况下的复杂度为 $O(v^2)$。表 1 给出了该算法与 TDS 和 extended TDS 算法的时间复杂度。

表 1 该算法与其它两种算法的时间复杂度的比较

Algorithms	Proposed algorithm	TDS algorithm	Extended TDS algorithm
Time Complexity	$O(v^2)$	$O(v^2)$	$O(dv^2)$

3 实例分析

对于图 4(a)，图 4(b)～4(d)显示了算法的实现过程。首

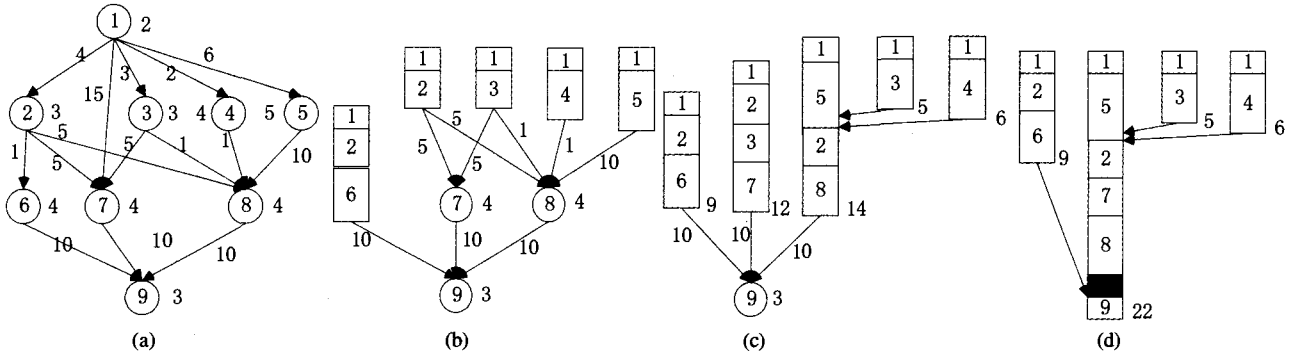


图 4 算法的一个图解例子

先，$est(n_1)$ 设为 0。由于只有一个父任务，因此 $est(n_i)$ 和 $C(n_i)(i=2..6)$ 很容易计算和构造。由于任务 n_7 有 2 个父任务 n_2 和 n_3，其 Cluster 为 $C(n_2)$ 和 $C(n_3)$。如果 n_2 和 n_3 都并入 $C(n_1)$，那么 $ct(C(n_1) \bigcup \{n_2\} \bigcup \{n_3\}) = 8 < DAT(C(n_3), n_7) = 10$，而且 n_3 的父任务 n_1 在 $C(n_2)$ 中，无须考虑。因此，$est(n_7)$ 是最优的且 $C(n_7) = \{1,2,3,7\}$。任务 n_8 的情况和 n_7 类似。下面重点考虑任务 n_9，其父任务 Cluster 排序为 $C(n_8), C(n_7)$ 和 $C(n_6)$。首先考虑把任务 n_9 并入 $C(n_8)$，则 $st(n_9) = 22$；如再与 n_7 合并，$st(n_9)$ 减少为 19，因为 $est(n_9) = \max(ct(C(n_8) \bigcup \{n_7\}), DAT(C(n_6), n_9)) = \max(18, 19) = 19$。而且对于 n_7 的父任务 n_3，虽然不在 $C(n_8)$ 中，但 $DAT(C(n_3), n_7) = 10 = st(n_7, C(n_8) \bigcup \{n_7\}) = 10$ 最优满足条件。所以，任务 n_3 不必合并，则 $est(n_9)$ 是最优的。图 5(d)显示的是最后的调度结果。我们可以看到任务 n_7，n_8 和 n_9 均不满足 TDS 算法的最优条件，因此由 TDS 算法所产生的开始时间均大于我们所计算的开始时间。对于任务 n_9，也不满足 ex-

tended TDS算法的最优条件。因此，由该算法所产生的任务 n_9 的启动时间也不可能是最优的。因此，我们的算法能产生比 TDS 和 extended TDS 更短的调度。表 2 列出了上述 3 种算法产生的每个任务的 est 和 cluster。

表 2　上述三种算法所产生的调度结果

	The proposed algorithm			Extended TDS algorithm			TDS algorithm	
Task	Cluster	est	ect	Cluster	est	ect	est	ect
1	{1}	0	2	{1}	0	2	0	2
2	{1,2}	2	5	{1,2}	2	5	2	5
3	{1,3}	2	5	{1,3}	2	5	2	5
4	{1,4}	2	6	{1,4}	2	6	2	6
5	{1,5}	2	7	{1,5}	2	7	2	7
6	{1,2,6}	5	9	{1,2,6}	5	9	5	9
7	{1,2,3,7}	8	12	{1,2,3,7}	8	12	13	17
8	{1,5,2,8}	10	14	{1,5,2,8}	10	14	11	15
9	{1,5,2,7,8,9}	19	22	{1,5,2,3,7,8,9}	21	24	25	28

4　性能分析与比较

调度长度是调度算法的主要性能指标。由于任务图具有不同的特征，因此有必要对调度长度进行标准化，从而对调度算法的性能进行客观比较。

$SLR=\frac{makespan}{\sum_{n_i\in CP_{MIN}} w(n_i)}$，$SLR$ 称为调度长度比率，其分母是图中关键路径上任务的计算时间的和，分子是实际调度长度。显然，$SLR\geqslant 1$，而且 SLR 越小，表明该算法的调度长度越短，性能越好。

下面，我们采用随机生成的任务图对上述 3 种算法进行性能比较。我们通过变换以下 3 个重要参数来产生任务图：

①任务图的结点个数—V

②入度—Indegree：任务的父结点的个数。

③通信与计算时间比率—CCR：是指任务图中结点间通信量的平均值与结点计算量的平均值的比值。CCR 反映了任务通信时间与计算时间的一个大致的比例关系。

对于上述 3 种参数，分别取以下值：

- V={20,50,100,200,500}
- Indegree={1,2,5,10}
- CCR={0.1,0.5,1,5,10}

因此，这 3 种参数的组合共产生 100 种具有不同参数的任务图；而同一种任务图，随机生成 20 个图。因此，总的任务图个数为 2000。设置不同参数，产生不同参数组合的随机任务图，可以最大限度地保证客观地测试各种算法的性能。

首先，考虑任务图大小对各算法性能的影响，如表 3(a)所示。我们不难看出，随着任务图大小的不断增加，各算法的性能都有不同程度的下降；但是，对于不同大小的任务图，我们的算法始终优于其它算法。

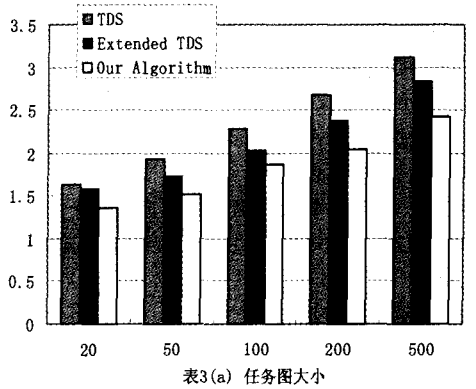


表3(a)　任务图大小

其次，考虑任务的入度(indegree)和通信与计算时间比率(CCR)对各算法的影响，如表 3(b)、3(c) 所示。很明显，所有算法都对 indegree 和 CCR 非常敏感。这是因为较大的 indegree 和 CCR 对任务的启动时间会有较大的影响，从而会导致差异较大的调度结果。尤其是 TDS 算法，其波动最大，仅仅在 indegree 和 CCR 的取值越小时性能较好，而取值越大时性能会大幅下降。我们的算法对于不同的 indegree 和 CCR 都有较好的性能，虽然有所下降，但幅度较小；而且，indegree 和 CCR 越大，我们的算法越优于其它算法。

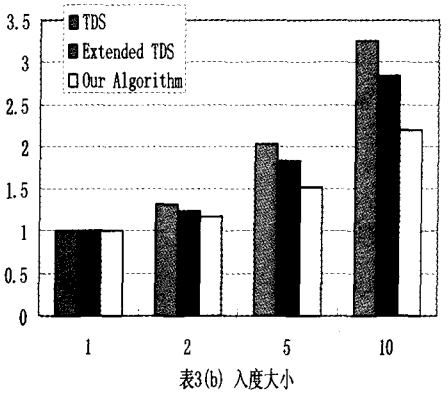


表3(b)　入度大小

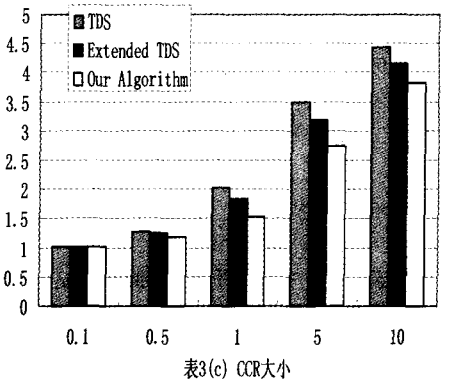


表3(c)　CCR大小

结论　本文针对 TDS 和 extended TDS 算法存在的问题，提出了一个新算法。该算法既考虑合并其它父任务以减少通讯时间，同时尽可能少地合并父任务，从而尽量减小任务的启动时间，因而能产生比 TDS 和 extended TDS 算法更短的调度。而且，该算法最优条件简单、宽松，具有与 TDS 算法相同的时间复杂度 $O(v^2)$。大量实验数据表明，该算法的性能明显优于其它算法。

(下转第 278 页)

行程序的文件头结构有 e_entry 成员，这就是程序的入口点。然后用汇编无条件跳转语句跳到系统的动态链接器的 entry point。

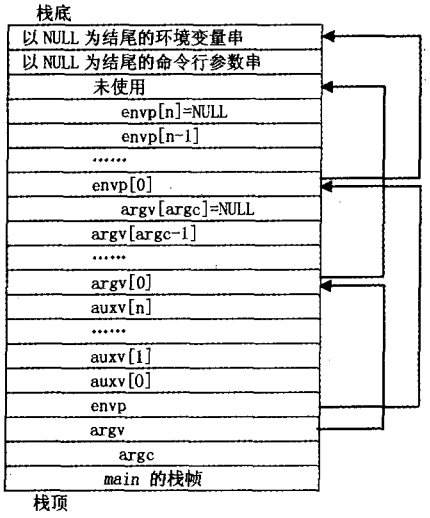


图 3　栈的结构

6　实验和分析

测试环境：

1. Red Hat Enterprise Linux AS release 4 (Nahant Update 2)

2. glibc. 2. 3. 4

3. 大页面大小为 256M，普通页面大小为 16k

实验方法：

每次申请一块表 1 所示大小的内存，分别用大页面和普通页面。对其每一个单元顺序地进行赋值操作，并记录时间。然后用所申请内存的大小除以所用时间得出效率，参见表 1。

表 1　测试用例及结果

申请的内存大小（单位：B）	使用大页面效率（单位：MB/S）	不使用大页面效率（单位：MB/S）	标志
256	1220. 20	1220. 70	A
1024	9882. 81	9765. 62	B
8192	9765. 62	10020. 83	C
16354	13020. 83	13531. 25	D
262144	11792. 45	10373. 44	E
1048576	10515. 25	8319. 47	F
16777216	2170. 21	2000. 09	G
268435456	2154. 65	1850. 0	H
1073741824	2199. 53	1900. 54	I

结论：

其中到标志 D 时，所申请的内存大小为 16k，此时由于普通页面不需要换页面，因此表现出来的是大页面和普通页面的效率差不多，但是所申请的内存大小更大的时候，由于大页面不需要更换页面或者更换的次数比较少，因此表现出来的效率要高。

但是大页面不是在任何场合都能够高效运转，因为如果运行小程序，大页面的内存浪费比较严重，而且如果程序中调用多次 fork()，则由于要反复申请大页面，开销很大。

结束语　本文提出了一个实现 ELF 可执行文件的 Data Segment 的大页面化的方法。这是一个很新的课题，对技术能力要求比较高。下一阶段的工作是对 ELF 可执行文件的执行空间中的栈和堆使用大页面。

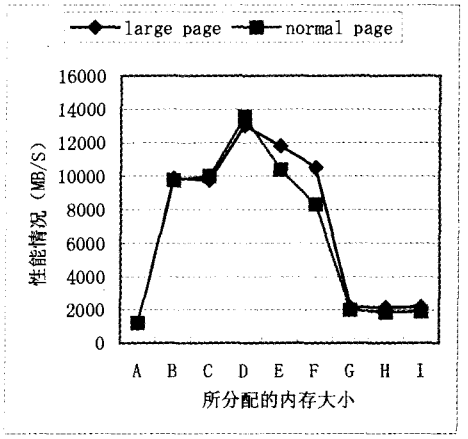


图 5　比较结果

参 考 文 献

1　Bryant R E, O'Hallaron D 著. 龚奕利，雷迎春译. 深入理解计算机系统. 中国电力出版社，2004. 524

2　倪继利著. Linux 内和分析及编程. 电子工业出版社，2005，9：86，552～555

3　http://www. phrack. org/phrack/58/p58-0x05 grugq & scut，2001

4　http://lists. grok. org. uk/pipermail/full-disclosure/attachments/20040101/fea4fb1f/ul_exec. txt

5　http://downloads. securityfocus. com/library/subversiveld. pdf grugq，2001

6　http://www. madchat. org/coding/Cheating_elf. pdf

7　http://www. skyfree. org/linux/references/ELF_Format. pdf

8　http://www. iecc. com/linkers

（上接第 272 页）

参 考 文 献

1　Darbhá S, Agrawal D P. Optimal Scheduling Algorithm for Distributed-Memory Machines. IEEE Trans, Parallel and Distributed Systems，1998，9(1)：87～94

2　Park C I, Choe T Y. An Optimal Scheduling Algorithm Based on Task Duplication. IEEE Trans Computers，2002，51(4)：444～448

3　Kwok Y K, Ahmad I. Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors. IEEE Trans Parallel and Distributed Systems，1996，7(5)：506～521

4　Ahmad I, Kwok Y K. On Exploiting Task Duplication in Parallel Program Scheduling. IEEE Trans Parallel and Distributed Systems，1998，9(19)：872～891

5　Yang T, Gerasoulis A. DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors. IEEE Trans Parallel and Distributed Systems，1994，5(9)：951～967

6　Gerasohlis A, Yang T. A Comparison of Clustering Heuristics for Scheduling DAGs on Multiprocessors. J Parallel and Distributed Computing，1992，16(4)：276～291

7　Palis M A, Liou Jing-Chiou, Wei D S L. Task Clustering and Scheduling for Distributed Memory Parallel Architecture. IEEE Trans Parallel and Distributed Systems，1996，7(1)：46～55

8　Colin J Y, Chretienne P. Cpm Scheduling with Small Computation Delays and Task Duplication. Operation Research，1991，39(4)：680～684