嵌入式系统的细粒度多处理器实时抢占式调度算法

李 扬[1] 王春明[2]

[1](江苏商贸职业学院电子与信息学院 江苏 南通 226011)

[2](南通大学计算机科学与技术学院 江苏 南通 226019)

摘 要 现有的嵌入式实时系统调度算法一般以任务级为调度单位,对此提出一种细粒度的线程级多处理器实时调度算法。采用 DAG 图描述实时系统的任务,并采用任务分解法将其分解为线程形式;为任务级调度采用基于干扰的可调度性分析,为线程级调度采用基于工作负载的可调度性分析;将线程的偏移、截止期与优先级作为三个调度目标,设计混合线程级调度算法。仿真实验结果表明,算法对于多线程任务的实时系统具有较好的性能。

关键词 多核处理器 并行任务 云计算 多线程任务 调度算法 嵌入式系统

中图分类号 TP391.9 **文献标识码** A DOI:10.3969/j.issn.1000-386x.2019.04.032

REAL-TIME PREEMPTIVE SCHEDULING ALGORITHM FOR FINE-GRAINED MULTIPROCESSOR IN EMBEDDED SYSTEM

Li Yang[1] Wang Chunming[2]

[1](*School of Electronics and Information, Jiangsu Vocational College of Business, Nantong 226011, Jiangsu, China*)

[2](*School of Computer Science and Technology, Nantong University, Nantong 226019, Jiangsu, China*)

Abstract The existing scheduling algorithms of embedded real-time system treat task level as scheduling unit generally. We proposed real-time scheduling algorithm for fine-grained thread-level multiprocessor. DAG graph was used to describe the tasks in real-time system, and the task decomposition method was adopted to decompose the tasks into threads. The interference-based schedulability analysis was introduced for task-level scheduling, and the workload-based schedulability analysis was introduced for thread-level scheduling. We treated offset, deadline and priority as three scheduling targets and designed hybrid thread-level scheduling algorithm. Simulation experimental results show that the proposed algorithm has better performance in real- time system with multi-threaded tasks.

Keywords Multi-core CPU Parallel tasks Cloud computing Multi-threaded tasks Scheduling algorithm Embedded system

0 引 言

如今 AMD 与 Intel 等处理器厂商已经发布 8 个核心以上的 CPU,未来 CPU 的核心数量更可能达到百级,由此增加了多处理器、多任务调度问题的难度[1]。传统的研究大多集中于单处理器,而针对单处理器的诸多研究成功无法扩展至多处理器系统[2]。为了充分利用多核心处理器的计算能力,任务调度算法需要支持任务之间的并行化[3-4],此外,对于多线程的任务,更可能发生同一任务的不同线程在多个处理器核心上并发执行的情况,因此,细粒度的任务调度算法成为一个亟待解决的问题[5]。

实时调度研究中有两个基本问题:(1) 调度算法

收稿日期:2018-10-17。江苏省“333 工程”科研项目(BRA2018220);南通市港闸区科技项目(GZKJ2018JHK008);江苏高校‘青蓝工程’资助项目(201706-202006);南通市科技计划项目(GY12015037);中华供销合作总社职业教育专项研究课题(NGX1425)。
李扬,副教授,主研领域:嵌入式系统,物联网。**王春明**,副教授。

需为任务或线程分配合适的优先级,且满足系统的截止期约束;(2) 为调度算法提供可调度性分析,证明算法满足系统的截止期约束[6-7]。以往的研究大多从单线程多任务、多处理器的实时调度方面解决上述两个问题。近期,出现多个考虑多线程多任务的多处理器调度研究,其中可调度性分析的研究为主要目标,但针对此场景的调度算法优化研究相对较少[8-10]。

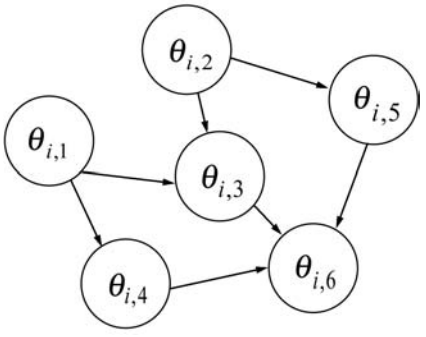
单线程任务中,作业是基本的调度单位,该场景的实时调度算法根据优先级变化的时间点主要分为三类[11]:(1) 任务级固定优先级算法;(2) 作业级固定优先级算法;(3) 动态优先级算法。如果将线程作为调度单位,则为调度问题增加了一个维度,从而提升了问题的复杂度。

文献[12]提出了最优优先级分配算法(OPA),该研究证明了 OPA 是任务级固定优先级分配问题的最优算法,本文期望将 OPA 算法扩展至线程级调度场景,但由此带来两个问题:线程级依赖关系与线程级调度算法的可调度性分析。本文首先采用任务分解方法[13]为各线程分配偏移时间与截止期;然后,基于干扰的线程级分析证明本文扩展调度算法与 OPA 算法兼容,从而证明本文线程级调度算法也是最优调度算法;最终,设计了完整的线程级调度算法。

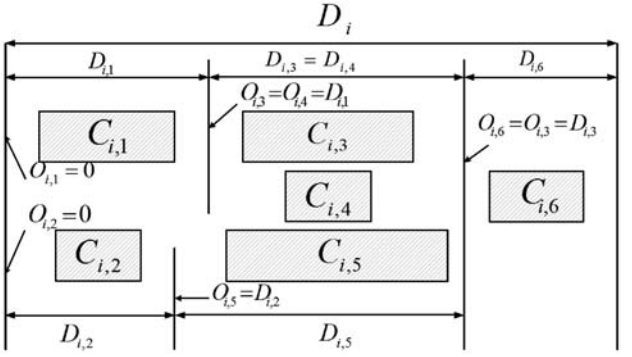
1　系统模型

1.1　有向无环图(DAG)任务

本文采用 DAG(有向无环图)表示并行任务[14],假设任务集合为 τ,一个任务 $\tau_i \in \tau$ 表示为一个有向无环图,如图 1(a)所示。τ_i的顶点 $v_{i,p}$代表一个线程 $\theta_{i,p}$,一条有向边($v_{i,p}$到 $v_{i,q}$)代表线程之间的依赖关系。一个 DAG 任务包含若干个作业,作业之间最小间隔为 T_i,各作业的截止期应当在 D_i以内,设 J_i^h表示 τ_i的第 h 个作业,任务分解的结果如图 1(b)所示。



(a) DAG 任务



(b) 任务分解

图 1　任务示意图

1.2　任务分解

可将一个 DAG 任务分解为独立、有序的子任务集合,线程之间具有依赖关系与执行出口,为 DAG 任务的各线程分配对应的偏移与截止期。

将任务 τ 分解产生的线程集合设为 τ^{decom},τ^{decom}中线程的数量设为 n。对于任务 τ_i,定义任务的主线程为 $\theta_{i,1}$,τ_i 中其他各线程 $\theta_{i,p}$ 表示为($T_{i,p}$, $C_{i,p}$, $D_{i,p}$, $O_{i,p}$),其中 $T_{i,p}$是最小间隔(等于 T_i),$C_{i,p}$与 $D_{i,p}$分别表示最坏情况的运行时间及其截止期,$O_{i,p}$为对应的偏移(从 $O_{i,1}=0$ 开始)。

假设主线程 $\theta_{i,1}$在 $r_{i,1}^h$点运行,则作业 j_i^h具有绝对的截止期 $d_{i,1}^h = r_{i,1}^h + D_{i,1}$。那么,依赖于 $\theta_{i,1}$的下一个线程 $\theta_{i,p}$将在 $r_{i,p}^h = r_{i,1}^h + O_{i,p}$点运行,其截止期为 $d_{i,p}^h = r_{i,p}^h + D_{i,p}$。将 $\theta_{i,p}$的执行窗口定义为时间段($r_{i,p}^h$,$d_{i,p}^h$]。

1.3　CPU 平台与调度算法

假设多核 CPU 由 m 个相同的处理器组成,调度策略为全局任务-线程级固定优先级调度,每个线程 $\theta_{i,p}$可以动态地在处理器之间迁移,且为所有线程分配静态的优先级 $P_{i,p}$。将优先级高于 $P_{i,p}$的线程集合设为 $hp(\theta_{i,p})$。

2　可调度性分析

DAG 任务分解为线程级之后,各线程即设置了偏移与截止期,无需考虑线程间的依赖关系。基于干扰的可调度性分析兼容 OPA,所以本文采用该方案[15]。

2.1　基于干扰的可调度性分析

线程级干扰可如下定义:

干扰 $I_{k,q}(a,b)$:如果 $\theta_{k,q}$为 ready 状态,但[a,b)中更高优先级的线程正在运行,此时的时隙总和定义为干扰 $I_{k,q}(a,b)$。

干扰 $I_{(k,q)\to(k,q)}(a,b)$:[a,b)中 $\theta_{i,p}$正在运行,且

$\theta_{k,q}$为 ready 状态，此时的时隙总和定义为干扰 $I_{(k,q)\to(k,q)}(a,b)$。

$I_{k,q}(a,b)$与 $I_{(k,q)\to(k,q)}(a,b)$之间的关系是可调度性分析的重要基础，$I_{k,q}(a,b)$与 $I_{(k,q)\to(k,q)}(a,b)$之间的关系可推导为[16]：

$$I_{k,q}(a,b)=\frac{\sum_{(i,p)\neq(k,q)} I_{(i,p)\to(k,q)}(a,b)}{m} \tag{1}$$

假设 $J^*_{k,q}$表示 $\theta_{k,q}$中总干扰最大的作业，该作业的总干扰可表示为：

$$I^*_{k,q}\triangleq\max_h(I_{k,q}(r^h_{k,q},d^h_{k,q}))=I_{k,q}(r^*_{k,q},d^*_{k,q}) \tag{2}$$

文献[15-16]定义了以下可调度性判定条件：

引理 1 当且仅当所有线程 $\theta_{k,q}$满足以下条件，则 m 个相同处理器组成的多处理器平台中集合 τ^{decom}是可调度的：

$$\sum_{\theta_{i,p}\in\tau^{\text{decom}}\setminus\{\theta_{k,q}\}}\min(I^*_{(i,p)\to(k,q)},D_{k,q}-C_{k,q}+1)<m:(D_{k,q}-C_{k,q}+1) \tag{3}$$

文献[17]对引理 1 扩展，获得了以下线程级判定条件：

引理 2 如果 τ^{decom}是可调度的，则 τ 也是可调度的。

2.2 基于工作负载的可调度性分析

首先需要确保各线程的截止期早于任务的截止期(引理 1)，然后需要计算其他线程对目标线程 $\theta_{k,q}$的干扰(式(3))。

本文推导进程对线程干扰的上界，即任务 τ_i对 $\theta_{k,q}$的干扰，表示为 $\sum_{\forall\theta_{i,p}\in\tau_i}\min(I^*_{(i,p)\to(k,q)},D_{k,q}-C_{k,q}+1)$。首先根据 τ_i释放作业的偏移点，计算 τ_i在 $\theta_{k,q}$运行窗口中的执行长度；然后，选择 τ_i最大运行长度对应的偏移。

考虑两个场景计算最大工作负载：$i\neq k$ 与 $i=k$，原因在于 $i=k$ 表示两个干扰线程属于同一个任务，说明 τ_i中作业释放的偏移已经自动给定。

2.2.1 $i\neq k$ 情况的负载最大化

为了简化描述，对于时间段$(x,y]$，如果作业的启动时间在 x 之前，截止期在$(x,y]$内，则简称为前置作业；如果作业的启动时间与截止期均在$(x,y]$内，则简称为期内作业；如果作业的启动时间在$(x,y]$内，截止期在 y 之后，则简称为后置作业。

假设 τ_i的作业为周期地启动，定义 $\Delta_i(x,y)$为$(x,y]$ 中前置任务启动时间点与 x 的差值，如图 2 所示。

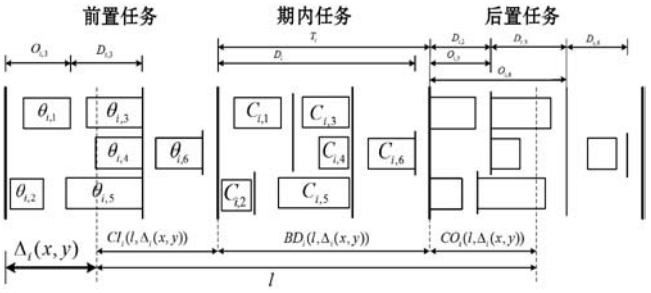


图 2 τ_i 中所有线程的最大工作负载

对于一个给定的 $\Delta_i(x,y)$，如果$(x,y]$时长为 l，可将$(x,y]$时间段分为前置时段 $CI_i(l,\Delta_i(x,y))$，期内时段 $BD_i(l,\Delta_i(x,y))$以及后置时段 $CO_i(l,\Delta_i(x,y))$，三种时段可表示为：

$$CI_i(l,\Delta_i(x,y))=\min(T_i-\Delta_i(x,y),l) \tag{4}$$

$$BD_i(l,\Delta_i(x,y))=\left\lfloor\frac{l-CI_i(l,\Delta_i(x,y))}{T_i}\right\rfloor\cdot T_i \tag{5}$$

$$CO_i(l,\Delta_i(x,y))=l-CI_i(l,\Delta_i(x,y))-BD_i(l,\Delta_i(x,y)) \tag{6}$$

对于给定的 $\Delta_i(x,y)$，$(x,y]$中每个线程的负载贡献度(如图 2 所示)可计算为：

$$W_{i,p}(l,\Delta_i(x,y))=W^{CI}_{i,p}+W^{BD}_{i,p}+W^{CO}_{i,p} \tag{7}$$

式中：

$$W^{CI}_{i,p}=[\min(O_{i,p}+D_{i,p},\Delta_i(x,y)+l)-\max(\Delta_i(x,y),O_{i,p})]^{C_{i,p}}_0$$

$$W^{BD}_{i,p}=\left\lfloor\frac{l-CI_i(l,\Delta_i(x,y))}{T_{i,p}}\right\rfloor\cdot C_{i,p}$$

$$W^{CO}_{i,p}=[CO_i(l,\Delta_i(x,y))-O_{i,p}]^{C_{i,p}}_0$$

$[X]^b_a$表示 $\min(\max(X,a),b)$。此时论证目标变为：对于 $W^{CI}_{i,p}$，首先寻找 $\theta_{i,p}$中前置作业的运行窗口与$(x,y]$重叠的部分，将其表示为$(a,b]$。不失一般性，将 $r^{CI}_{i,0}$设为 0，则前置作业在 $O_{i,p}$启动。如果 $\Delta_i(x,y)<O_{i,p}$，则 a 等于 $O_{i,p}$；否则，a 等于 $\Delta_i(x,y)$，如图 2 所示。此时 $\theta_{i,p}$中前置作业的截止期为 $O_{i,p}+D_{i,p}$。如果 $\Delta_i(x,y)+l>O_{i,p}+D_{i,p}$，则 b 等于 $O_{i,p}+D_{i,p}$；否则 b 等于 $\Delta_i(x,y)+l$，说明进前置作业与$(x,y]$产生重叠。由此可总结为：a 等于 $\max(\Delta_i(x,y),O_{i,p})$，$b$ 等于 $\min(O_{i,p}+D_{i,p},\Delta_i(x,y)+l)$。在$(a,b]$中，前期作业的运行时间在 0 与 $C_{i,p}$ 之间。因此获得了式(7)的 $W^{CI}_{i,p}$。

$\theta_{i,p}$期内作业的数量可计算为$\left\lfloor\frac{l-CI_i(l,\Delta_i(x,y))}{T_{i,p}}\right\rfloor$，所以 $W^{BD}_{i,p}$等于作业数量乘以运行时间 $C_{i,p}$。$W^{CO}_{i,p}$的推导过程与 $W^{CI}_{i,p}$相似。

对于偶发任务：此时可简单地检查$(x,y]$中 $\theta_{i,p}$的运行量，则 $\Delta_i(x,y)$上界为 $W_{i,p}(l,\Delta_i(x,y))$。

考虑任务 τ_i 中 $\Delta_i(x,y)$ 的所有可能值,优先级高于 $\theta_{k,q}$ 的所有线程工作负载之和即为 τ_i 对线程 $\theta_{k,q}$ 的最大干扰的上界,表示为:

$$W_i(D_{k,q}) = \max_{0\leqslant\Delta_i(x,y)<T_i}\sum_{\forall\theta_{i,p}\in hp(\theta_{k,q})} \min(W_{i,p}(D_{k,q},\Delta_i(x,y)),D_{k,q}-C_{k,q}+1) \tag{8}$$

在时段$(x,y]$中,对于给定的 $\Delta_i(x,y)$,证明 $W_{i,p}^{CI}$、$W_{i,p}^{BD}$ 与 $W_{i,p}^{CO}$ 分别是 $\theta_{i,p}$ 中前置作业、期内作业与后置作业运行量(执行时长)的上界。

2.2.2　$i=k$ 情况的负载最大化

该情况下 τ_k 的作业被同一个任务干扰,因此可自动地决定 τ_k 中作业启动的偏移时间。为了计算 $i=k$ 情况的最大工作负载,本文仅需要考虑与线程 $\theta_{k,q}$ 重叠的部分执行窗口,使用式(7)可计算这些线程的负载贡献量。因此,τ_k 中优先级高于线程 $\theta_{k,q}$ 的其他所有线程最大负载计算为:

$$W_k(D_{k,q}) = \sum_{\forall\theta_{k,p}\in hp(\theta_{k,q})} \min(W_{k,p}(D_{k,q},O_{k,q}),D_{k,q}-C_{k,q}+1) \tag{9}$$

基于式(8)和式(9)计算的干扰上界,为本文线程级调度算法设计了以下可调度性判定:

理论 1　对于 m 个相同的处理器平台的线程级调度算法,如果每个线程 $\theta_{k,q}$ 满足以下的不等式,则集合 τ^{decom} 为可调度。

$$\sum_{\forall\tau_i\neq\tau_k} W_i(D_{k,q}) + W_k(D_{k,q}) < m\cdot(D_{k,q}-C_{k,q}+1) \tag{10}$$

证明　上文所述 $W_k(D_{k,q})$ 是 $\theta_{k,q}$ 运行窗口中优先级高于 $p_{k,q}$ 的最大运行量。当且仅当 A 优先级高于 B 时,A 才对 B 产生干扰,式(3)左式的上界为式(10)的左式。根据引理 1,可证明该理论。

3　线程级最优优先级分配算法

问题描述为:给定一个分解线程的集合 τ^{decom},为每个线程 $\theta_{i,q}\in\tau^{\text{decom}}$ 分配优先级 $P_{i,p}$,因此根据基于工作负载的可调度性判定(理论 1)认为分解集合为可调度的。

OPA 算法通过优先级分配为各任务分配一个优先级,如果所有任务 τ_i 满足以下两个条件,则该可调度性判定为 OPA 兼容:

条件 1:任务 τ_i 的可调度性对其他优先级的任务顺序不敏感。

条件 2:如果交换 τ_k 与 τ_i 之间的优先级使得 τ_k 的优先级提高,交换前后 τ_k 的可调度性不变。

算法 1 为本文改进的 OPA 调度算法,该算法从优先级最低的线程开始迭代地为所有线程分配优先级,其中 τ^{decom} 为操作系统中待分配优先级的线程集。第 k 次迭代中,τ^{decom} 分为两个不相交的子集:$A(k)$ 与 $R(k)$,其中:

1) $A(k)$ 表示第 k 步之前已分配优先级的线程子集;

2) $R(k)$ 表示剩余线程的子集。

理论 1 的可调度性判定是 OPA 兼容的,说明该判定满足线程级可调度性的条件 1 与条件 2,因此算法 1 的优先级分配过程正确,证明方法如下:

理论 2　理论 1 的可调度性判定兼容算法 1。

证明　根据本文的可调度性判定证明线程级调度算法满足条件 1 与条件 2:

(1) 满足条件 1 证明:式(10)的左式计算了每个任务对线程 $\theta_{k,q}$ 干扰的上界,任务的干扰上界计算为优先级高于 $\theta_{k,q}$ 的其他线程工作负载之和。而该计算过程不依赖相应的优先级顺序。因此,满足条件 1。

(2) 满足条件 2 证明:交换 $\theta_{k,q}$ 与 $\theta_{i,p}$ 优先级,$\theta_{k,q}$ 的优先级提高。因为 $\theta_{k,q}$ 的优先级提高,所以 $hp(\theta_{k,q})$ 交换后变得更小。因此,式(8)、式(9)的 $W_i(D_{k,q})$ 与 $W_k(D_{k,q})$ 交换之后变得更小,满足条件 2。

计算复杂度分析:分解任务集合 τ^{decom} 的线程数量设为 n,根据可调度性判定,如果存在所有线程可调度的情况,则算法可成功找到该优先级分配策略,所以对于 n 个线程的场景,最多需$\frac{n(n+1)}{2}$次可调度性判定。

算法 1　线程级 OPA 调度算法

输入:τ^{decom}
输出:调度状态
1: $k\leftarrow 0, A(1)\leftarrow\varnothing, R(1)\leftarrow\tau^{\text{decom}}$
2: **DO** {
3:　　$k\leftarrow k+1$
4:　　**IF**(*thread_level_sch*($A(k),R(k)$) == FAILURE);
5:　　　　**return** FALSE;　　//不可调度
6: } **WHILE**($R(k)$!= NULL)
7: **return** TRUE;　　//可调度

算法 2　thread_level_sch 函数

输入:$A(k),R(k)$
输出:分配结果
1: **FOREACH** thread $\theta_{p,q}\in R(k)$ {
2:　　**IF**($\theta_{p,q}$可调度,根据理论 1,$R(k)$ 中其他线程的优先级均高于 k) **THEN** {
3:　　　　为 $\theta_{p,q}$ 分配优先级 k;

```
4:          R(k+1)←R(k)\{θ_{p,q}};
5:          A(k+1)←A(k)\{θ_{p,q}};
6:          return SUCCESS;
7:      }
8: }
9: return FAILURE;
```

4 动态截止期的线程级调度算法

4.1 算法设计

上文考虑了线程静态偏移与截止期的情况，本算法的决策目标为各线程的偏移、截止期与优先级三个目标，所以根据理论 1 的可调度性分析，并行任务系统是可调度的。本文动态截止期的线程级优先级分配策略的主要思想为：首先通过算法 1 分配优先级，如果算法 1 分配失败，则调节一些线程的偏移与截止期，使得调节后某个线程可成功地分配优先级，如果找到该调节方法，则继续使用算法 1 分配优先级；否则，认为调度失败，如算法 3 所示。算法 3 中输入为操作系统中待分配优先级的线程集，输出为调度的状态。

将线程结束时间与截止期之间的最小距离定义为线程 $\theta_{k,q}$ 的富余时间，表示为 $S_{k,q}$。根据理论 1 的可调度性判定，$S_{k,q}$ 可近似为：

$$S_{k,q} = D_{k,q} - C_{k,q} - \left\lfloor \frac{\sum_{\forall \tau_i \neq \tau_k} W_i(D_{k,q}) + W_k(D_{k,q})}{m} \right\rfloor \quad (11)$$

定义线程的归一化富余时间 $\overline{S}_{k,q} = S_{k,q}/D_{k,q}$。可将线程的截止期调节变换为线程的富余时间调节，即富余时间多的线程将部分富余时间赠予时间紧缺的线程，将赠予方简称为赠予线程，被赠予方成为受赠线程。最终，截止期调节问题可分为三个子问题：赠予线程、受赠线程的决策策略，以及赠予时间量的分配。

线程间赠予富余时间的目标是为受赠线程分配一个优先级，使得受赠线程变为可调度，然而由此可能对其他可调度的线程产生边际效应。考虑该问题，本文设计了以下三个子问题解决策略：

（1）受赠线程的决策策略：应当保证已分配优先级线程具有足够的富余时间，所以决策策略为最小化总赠予时间的量。据此将所需赠予时间最少的线程作为受赠线程，时期变为可调度状态，见算法 4 的第 3 行。

（2）赠予线程的决策策略：确定受赠线程后，建立一个赠予线程候选集合（算法 4 第 4 行），属同一个任务的候选线程已经分配了优先级，应当能够赠予富余时间，且不影响所有线程的可调度性。为了避免破坏已分配优先级线程的可调度状态，本文选择候选集合中归一化富余时间量最大的线程作为赠予线程（算法 4 第 6 行）。

（3）赠予时间量的分配：

引理 3 如果线程 $\theta_{k,q}$ 的截止期 $D_{k,q}$ 减小，则对 $\theta_{k,q}$ 最坏情况的干扰随之单调减小。

证明 式（8）、式（9）中，分别计算了 τ_i 与 τ_k 的最大执行量。式（7）中给定 t，$W_{i,p}(D_{i,p},t)$ 与 $W_{k,q}(D_{k,q},t)$ 随 $D_{k,q}$ 下降而单调下降。式（8）中，因为需要调节 $\Delta_i(x,y)$ 使得 $W_i(D_{k,q})$ 最大化，所以 $W_k(D_{k,q})$ 不会随 $D_{k,q}$ 下降而提高，因此，可看出 $W_i(D_{k,q})$ 与 $W_k(D_{k,q})$ 随 $D_{k,q}$ 下降呈单调下降。由此证明引理 3。

引理 3 说明如果赠予线程减小其截止期（将富余时间赠予受赠线程），赠予线程可能获得其他的富余时间。因此，每个赠予步骤中将赠予时间设置较小，从而增加找到其他富余时间的概率（算法第 7 行），赠予完成后，每个赠予线程保持更新其富余时间量，以期找到其他的富余时间（算法 11 行）。

算法 3 动态截止期的线程级调度算法

```
输入:τ^decom
输出:调度状态
1: k←0,A(1)←∅,R(1)←τ^decom
2: DO {
3: IF(thread_level_sch(A(k),R(k)) == FAILURE) {
4:      IF(deadline_change(A(k),R(k)) == FAILURE) {
5:          return FALSE;
6:      }
7: } WHILE(R(k) ! = NULL);
8: return TRUE
```

算法 4 deadline_change 函数

```
输入:A(k), R(k)
        //A(k)已调度的任务集,R(k)尚未调度的任务集
输出:分配结果
1: F←R(k);
2: DO {
3:      根据理论 1 选择受赠线程,表示为 θ*_{s,e} ∈ F;
4:      建立候选赠予线程的集合 DC_s(θ*_{s,e});
5:      保存 τ_s 中所有线程的当前偏移与截止期;
6:      WHILE(DC_s(θ*_{s,e}) ! = NULL) {
7:          从 DC_s(θ*_{s,e}) 重搜索归一化富余时间最多的线程,表示为 θ+_{s,i} ∈ DC_s(θ*_{s,e});
8:          调节任务中所有线程的偏移与截止期,从而实现 θ+_{s,i} 向 θ*_{s,e} 赠予富余时间量 Ω;
9:          IF(根据理论 1,线程 θ*_{s,e} 是可调度的) THEN {
```

10:　　**return** SUCCESS;
11:　　更新 $\theta_{s,i}^{+}$的富余时间;
12:　}
13:　保存 τ_s中所有线程的偏移与截止期;
14:}**WHILE**(F! = NULL)
15:**return** FAILURE

4.2　计算复杂度分析

本文线程级调度算法迭代地为各线程分配优先级。如果算法 2 分配失败,则启动算法 4,算法 4 中建立基于候选线程的集合,并为 n 个线程进行可调度性判定,复杂度为 $O(n^2)$。算法中循环体(6 ~ 12 行)最多重复 $S_{s,e}^{*}/\Omega$ 次,其中 $S_{s,e}^{*} < T_s$。因为最外层循环(2 ~ 14 行)最多重复 n 次,所以算法 4 的可调度性判定次数为 $\max\{O(n^3), O(nT_{\max})\}$,其中 $T_{\max}$代表所有任务中的最大 T_i。因为本算法中截止期的调节最多发生 n 次,所以可调度判定的次数为 $\max\{O(n^4), O(nT_{\max})\}$。

5　仿真实验与结果分析

5.1　仿真环境

为了简化表达,定义以下术语:C_i表示 τ_i($\sum_q C_{i,q}$)内所有线程最差情况的运行时间总和,U_{sys}表示系统利用率,L_i是 τ_i最坏情况的执行时间,LU_{sys}定义为任务 $\tau_i \in \tau$ 中最大的 L_i/D_i值。

采用文献[18]的方法生成 DAG 任务,对于任务 τ_i,其参数如下设置:节点数量 n_i均匀选择于区间[1, $N_{\max}$]内,其中 $N_{\max}$为单个任务的线程数量上限。每对节点之间边随机地生成,概率为 p。根据 $C_{i,p}$值所属的三个范围[1,5]、(6,10]、(11,40],分别为任务 τ_i与每个线程分配一个类型(轻型、中等、重型),同时确定 T_i($=D_i$)参数,因此 C_i/T_i值分别在[0.1, 0.3],(0.3, 0.6]或(0.6, 1.0]随机地选择。

为了全面地测试本算法对并行 DAG 任务的性能,共设计了三种测试统计量:(1) 系统利用率;(2) 并行度;(3) 节点总数量。第一组实验评估了系统对 U_{sys}的调度性能变化情况,结果如图 3 所示。第二组实验评估了算法对不同任务内并行度的调度性能变化情况,结果如图 4 所示。第三组实验评估了算法对 τ 中节点总数量的调度性能变化情况,结果如图 5、图 6 所示。将本文算法与基本 OPA 算法[12]、B_task_EDF[19]以及 C_task_EDF[20]三种算法进行横向比较,评估本算法的性能。

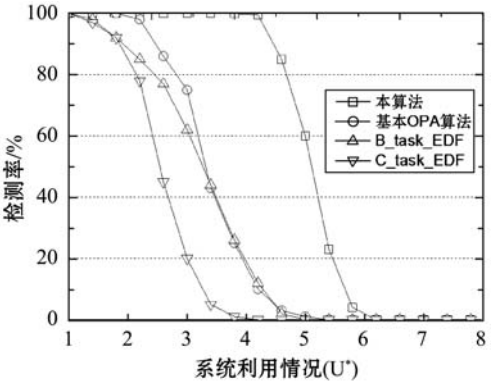


图 3　不同 U_{sys}的调度性能变化情况

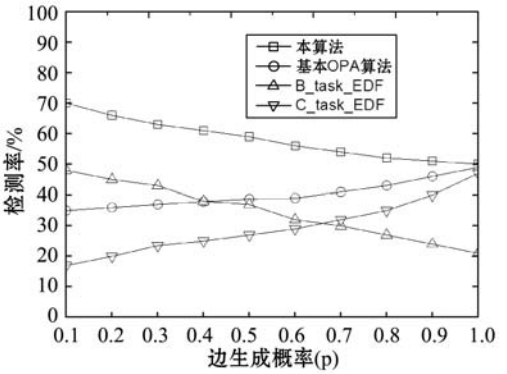


图 4　不同任务内并行度的调度性能变化情况

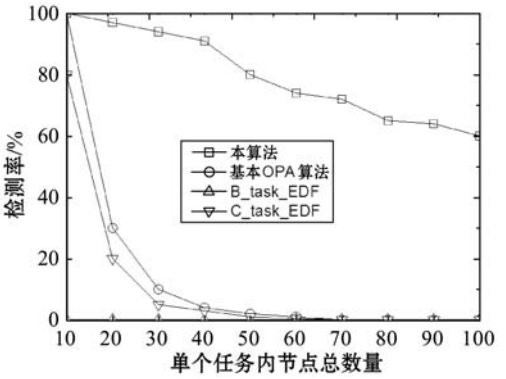


图 5　对 τ 中节点总数量的调度性能变化情况

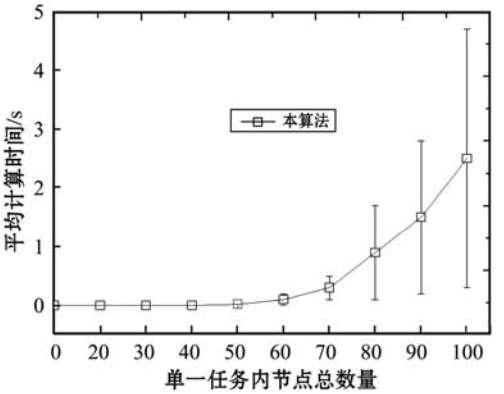


图 6　本调度算法的计算时间与任务内节点总数量的关系

5.1.1　第一组实验的任务集生成方法

设置 $p = 0.5$、$N_{\max} = 10$、$m = 8$ 的情况,生成 1 000 个任务集。生成准确的 U_{sys}难度较大,所以本文生成任务所采用的 U^* 上下界分别为 $U_{\max}^{*} = U^* + 0.005$ 与

$U^*_{min} = U^* - 0.005$。生成任务集的步骤为：

步骤 1：随机地将任务逐个加入任务集中；

步骤 2：如果某个任务加入之后导致 $U^* > U^*_{max}$，则忽略该集合并返回步骤 1；

步骤 3：如果一个任务加入后导致 $U^*_{min} \leqslant U^* \leqslant U^*_{max}$，则采用该任务集进行实验。

将 U^* 从 1 增加至 8，步长设为 0.4，因此实验的任务集数量为 18 000。

5.1.2 第二组实验的任务集生成方法

设置 $m = 8$、$N_{max} = 10$、p 为变量因子，生成 1 000 个任务集，步骤如下：

步骤 1：使用上述参数生成 m 个任务的任务集；

步骤 2：如果任务集的 U_{sys} 大于 m，则忽略该任务集并返回步骤 1；

步骤 3：采用此时的任务集进行实验，然后向该任务集添加一个任务，并返回步骤 2 直至生成 1 000 个任务集。

对于节点间的边，如果 $p = 0$，则没有边（不存在父线程），此时任务内的并行度为最大；如果 $p = 1$，每个节点均与其他节点连接，说明一个任务内同时仅一个线程运行；随着 p 值增加，每个 DAG 任务 τ_i 内边的数量增加，由此导致关键的执行路径 L_i 变长，然后 LU_{sys} 增加。

为了针对不同的任务内并行度进行仿真，在 10 个不同的 p 值下，进行仿真实验，p 值从 0.1 增加至 1，步长为 0.1，共有 10 000 个仿真。

5.1.3 第三组实验的任务集生成方法

第三组实验采用第一组实验的参数设置，将任务逐个加入任务集直至 τ 中节点数量达到给定的节点总数量，任务的节点数量范围为[1，给定的总节点数量 $-\tau$ 中节点数量]。将节点总数量从 10 增加至 100，$p = 0.5$，$U^* = m/2$。每个数据点包含 1 000 个仿真，共有 10 000 个仿真。

5.2 实验结果与分析

从图 3 中可看出，U^* 从 1 到 5.8，本算法的检测率远超过其他三个算法。将本算法与 OPA 基本算法相比，可看出本文细粒度的调度算法（线程为调度单位）明显提高了调度算法的性能，同时本算法优于两个 EDF 的增强算法，可看出本算法中采用线程级调度、动态线程截止期等设计具有显著的效果。

图 4 所示为检测率与 p 值的关系，p 值增加引起每个 DAG 任务的边数量增加，从而使得节点间依赖的关系度升高。从图中可看出，线程级优先级分配算法在 p 值较小时具有较好的性能，而当 p 值为 1 时，本算法与基本 OPA 算法、C_task_EDF 性能接近，此时一个 DAG 任务中所有节点均被连接，任务均为单线程任务。B_task_EDF 的检测率随着 p 值的增加而降低，主要原因是该方法的调度性判定 LU_{sys} 是否小于阈值 $m/(2m-1)$，随着 p 值的增加，LU_{sys} 值升高，导致可调度性下降。

为了测试本算法对不同数量线程的性能，将线程数量设为因变量，统计系统检测率的变化情况。图 5 中可看出，节点数量较多时，本算法优于其他三个算法。节点数量越多，本算法调节线程截止期的机会越多，因此可调度性越高。

图 6 是本算法计算时间与任务内节点数量的关系，可看出节点数量越高，算法的计算时间越高。虽然对于 100 个线程的任务，本算法的计算时间约为 2.2 s，但是本算法将实时系统的检测率提高到约 60%。

6 结 语

现有实时系统调度算法将任务作为调度单位，严重限制了多线程任务的调度性能，并且已有的算法大多对一些经典的调度算法进行可调度性分析，以期通过优化可调度性判定条件提高算法的可调度性。然而受限于原调度算法的性能限制，通过收紧可调度性判定条件所获得的性能提升往往较为有限。本文提出了一种细粒度的线程级多处理器实时调度算法，并给出了详细的可调度性分析，仿真实验结果表明，本算法对于多线程任务的实时系统具有较好的性能，对高线程数量的任务具有明显地优势。

参 考 文 献

[1] 姜浩，杜琦，郭敏，等. 面向 ARMv8 64 位多核处理器的 QGEMM 设计与实现[J]. 计算机学报，2017，40(9)：2018－2029.

[2] 焦文喆，翟正军，王国庆. 时间触发 AFDX 调度设计及实时性分析[J]. 计算机工程，2016，42(7)：42－48.

[3] 徐俊，汤庸，刘道余. 基于混合差分粒子群算法的 MapReduce 任务调度算法研究[J]. 小型微型计算机系统，2016，37(7)：1479－1481.

[4] 杨辉华，张晓凤，谢谱模，等. 基于布谷鸟搜索的多处理器任务调度算法[J]. 计算机科学，2015，42(1)：86－89.

[5] Melani A，Bertogna M，Bonifaci V，et al. Response-Time Analysis of Conditional DAG Tasks in Multiprocessor Systems [J]. Molecular Immunology，2015，25(4)：329－335.

[6] 梁浩，晏立，沈项军. 全局固定优先级实时调度算法分析[J]. 计算机工程，2017，43(12)：65－68.

[7] Guan N，Han M，Gu C，et al. Bounding Carry-in Interference to Improve Fixed-Priority Global Multiprocessor Schedu-

ling Analysis[C]//IEEE International Conference on Embedded and Real-Time Computing Systems and Applications. IEEE, 2015:11-20.

[8] Qi Wang, Parmer, G. FJOS: Practical, predictable, and efficient system support for fork/join parallelism[C]//2014 IEEE Real-Time and Embedded Technology and Applications Symposium(RTAS). IEEE Computer Society,2014:25-36.

[9] Li J, Luo Z, Ferry D, et al. Global EDF scheduling for parallel real-time tasks[J]. Real-Time Systems, 2015, 51(4): 395-439.

[10] Axer P, Quinton S, Neukirchner M, et al. Response-Time Analysis of Parallel Fork-Join Workloads with Real-Time Constraints[C]//Euromicro Conference on Real-Time Systems. IEEE Computer Society, 2013:215-224.

[11] Davis R I, Burns A. A survey of hard real-time scheduling for multiprocessor systems[J]. Acm Computing Surveys, 2011, 43(4).

[12] Davis R I, Burns A. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems[J]. Real-Time Systems,2011,47(1):1-40.

[13] Ferry D, Li J, Mahadevan M, et al. A real-time scheduling service for parallel tasks[C]//Real-Time and Embedded Technology and Applications Symposium(RTAS), 2013 IEEE 19th. IEEE, 2013: 261-272.

[14] 林英, 孟正, 康雁,等. 多核下一种线程调度算法的研究与实现[J]. 计算机技术与发展, 2013(10):19-22.

[15] Davis R I, Burns A. Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems[C]//Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE. IEEE, 2009: 398-409.

[16] Bertogna M, Cirinei M, Lipari G. Improved schedulability analysis of EDF on multiprocessor platforms[C]//Real-Time Systems, 2005.(ECRTS 2005). Proceedings. 17th Euromicro Conference on. IEEE, 2005: 209-218.

[17] Saifullah A, Li J, Agrawal K, et al. Multi-core real-time scheduling for generalized parallel task models[J]. Real-Time Systems, 2013, 49(4): 404-435.

[18] Cordeiro D, Mounié G, Perarnau S, et al. Random graph generation for scheduling simulations[C]//Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques. 2010.

[19] Baruah S. Improved Multiprocessor Global Schedulability Analysis of Sporadic DAG Task Systems[C]//Proceedings of the 2014 26th Euromicro Conference on Real-Time Systems. IEEE, 2014:97-105.

[20] Chwa H S, Lee J, Phan K M, et al. Global edf schedulability analysis for synchronous parallel tasks on multicore platforms[C]//Proceedings of the 2013 25th Euromicro Conference on Real-Time Systems. IEEE, 2013: 25-34.

(上接第 197 页)

[3] Zhang X F, Ni D, Gou Z J. Sparse representation and PCA method for image fusion in remote sensing[C]//Proceedings of 2016 the 2nd international conference on control, automation and robotics, 2016: 324-330.

[4] 赵学军,刘静. 基于 Shearlet 和稀疏表示的遥感图像融合[J]. 科学技术与工程, 2017, 17(4): 255-259.

[5] 赵延芳,王涛,李鹏. 基于 ZY3 遥感图像的反立体校正方法研究[J]. 信息技术, 2017, 15(3): 60-64.

[6] Wang W H, Sun S L, Jiang M X. Traffic lights detection and recognition based on multi-feature fusion[J]. Multimedia Tools and Applications, 2017, 76(13): 14829-14846.

[7] Panchal S, Thakker R A. Improved image pansharpening technique using nonsubsampled contourlet transform with sparse representation[J]. Journal of the Indian Society of Remote Sensing, 2017, 45(3): 385-394.

[8] Peng G, Wang Z Y, Liu S Q. Image fusion by combining multiwavelet with nonsubsampled direction filter bank[J]. Soft Computing, 2017, 21(8):1977-1989.

[9] Liu C P, Long Y H, Mao J X. Energy-efficient multi-focus image fusion based on neighbor distance and morphology[J]. Optik-International Journal for Light and Electron Optics, 2016, 127(23): 11354-11363.

[10] Udhaya Suriya T S, Rangarajan P. Brain tumour detection using discrete wavelet transform based medical image fusion[J]. Biomedical Research, 2017, 28(2): 684-688.

[11] Bao W X, Wang W, Zhu Y X. Pleiades satellite remote sensing image fusion algorithm based on shearlet transform[J]. Journal of the Indian Society of Remote Sensing, 2018, 46(1): 19-29.

[12] 李旭,高雅楠, Yue S. 一种尺度感知型遥感图像融合新方法[J]. 宇航学报,2017,38(12):1348-1353.

[13] Zhang J. A new saliency-driven fusion method based on complex wavelet transform for remote sensing images[J]. IEEE Geoscience and Remote Sensing Letters, 2017, 14(12):2433-2437.

[14] 杨艳春,李娇,王阳萍. 图像融合质量评价方法研究综述[J]. 计算机科学与探索,2018, 12(7): 1021-1035.

[15] Sanli F B, Abdikan S, Esetlili M T, et al. Evaluation of image fusion methods using PALSAR, RADARSAT-1 and SPOT images for land use/land cover classification[J]. Journal of the Indian Society of Remote Sensing, 2017, 45(4): 591-601.