多处理器并行 EDPF 优化实时调度算法

胡天一,徐中伟
HU Tian-yi,XU Zhong-wei

同济大学 电子与信息学院,上海 200331
School of Electronics and Information,Tongji University,Shanghai 200331,China
E-mail:justice623@126.com

HU Tian-yi,XU Zhong-wei.Parallelized EDPF real-time optimization scheduling algorithm for multiprocessors.Computer Engineering and Applications,2007,43(19):35-38.

Abstract: Real-time multiprocessors scheduling is always an important subject in real-time system research.In this paper,according to the characters of these systems as deadline requirement and effectiveness,we develop an optimization scheduling algorithm based on parallelized EDPF.The improved algorithm is suitable for the parallelized tasks.It takes processing time into account besides the deadline and resource requirement of the task sets in order to reduce scheduling backtracks and increase effectiveness.After a great deal of simulation,we analyze the effect of success ratio by several parameters.At last,the new algorithm is proved to be better than the Myopic algorithm.

Key words: real-time multiprocessors;deadline;parallelized;scheduling backtrack;success ratio

摘 要:实时多处理器系统的任务调度问题始终都是一个重要课题。针对该系统须保证任务截止期和有效性的特点,提出了一种并行 EDPF(Earliest Deadline and Processing Time First)优化调度算法。该算法适用于可并行任务,并在考虑到了任务集的截止期和资源因素基础上,加入了运行时间因素,达到了减少调度返回次数以及提高有效性的目的。最后通过大量的仿真,分析了一些必要参数对调度成功率的影响,并通过比较证明了该算法明显优于 Myopic 算法。

关键词:实时多处理器;任务截止期;并行;调度返回;调度成功率
文章编号:1002-8331(2007)19-0035-04 **文献标识码**:A **中图分类号**:TP27

1 引言

随着科学的发展和对系统性能要求的提高,一般的控制系统已经无法满足要求,因此多处理器实时系统凭其高性能和高可靠性,越来越多地应用于各种领域,如航空电子控制系统、核反应堆控制系统等等。但是随着系统中处理器数目的增加,实时系统的任务调度问题更为突出。调度的实质是资源的分配,而实时调度强调的是任务的时间约束[3]。实时系统的基本问题,就是要保证系统中的任务满足其时间要求,从而保证系统实时性。

Mok 和 Dertouzos 在 1983 年指出 MSP(Multi-processor Scheduling Problem)是一个 NP-hard 问题[5],不存在一个理想的调度算法。因此需要研究新的可行的算法来达到最优化调度。目前,已有多种算法被提出,这些算法主要分为静态调度算法和动态算法调度。静态调度算法主要有启发式算法、近似算法、状态空间算法、遗传算法等,其优点是简单、稳定,但是缺乏灵活性。而动态调度算法更适用于较为复杂和恶劣的环境,与静态调度相比,区别之处在于:前者在任务运行时,自动更新优先级已反映变化了的条件。其中具有代表性的有:

(1)EDF(Earliest Deadline First)[4],ESF(Earliest Start Time First),MPF(Minimum Processor Time First),MLF(Minimum Laxity First),这些算法按任务截止期、开始运行时间、最小计算时间、最小延迟度决定调度优先级,克服了静态调度的局限性,但它们都存在没有考虑资源和时间复杂度大等问题;

(2)Myopic 算法,结合了 EDF 和 ESF 局部算法,并解决了资源控制问题,是目前使用较多的一种经典算法。然而当任务允许并行处理,且运行时间很短空闲时间很长时,Myopic 算法的有效性就会大大降低。本文在 Myopic 算法的基础上,提出了一种新的实时多处理器的动态调度算法——并行 EDPF 算法,有效地解决了这一问题。

2 基本理论

2.1 任务模型

假设一个由 N 个处理器组成的实时系统,$N>1$。

(1)每个任务 $T(i)$是非周期性的,$a(i)$、$w(i)$、$d(i)$分别代表到达时间、准备时间、截止时间;$c(i,j)$是任务 $T(i)$在 j 个处理器上并行运行的最不利计算时间($1\leqslant j\leqslant N$)。

(2)每个任务可能需要一些类似数据结构、变量、通信缓冲区的管理资源,每个任务有 2 种接入资源的方式:

①专用接入,在这种情况下,其他任务无法使用该资源;

②共享接入,在这种情况下,可以由不同的任务共享使用

基金项目:国家自然科学基金(the National Natural Science Foundation of China under Grant No.60674004)。
作者简介:胡天一(1982-),男,硕士研究生,主要研究方向:并行处理、任务调度;徐中伟(1964-),教授,博士生导师。

同一个资源。在专用接入方式下,如果两个任务同时需要同一个资源,就会产生资源冲突。

(3)当一个任务并行执行时,所有其并行子任务(又称分割任务),为了达到同步管理的目的必须同时开始运行。*Max_split* 表示任务的最大并行级别。任务 $T(i)$同时在 $j(j\geq 2)$个处理器上运行时的最不利计算时间 $c(i,j)=\lfloor c(i,j-1)*(j-1)/j\rfloor+1$。

(4)任务为非抢占式的(一旦一个任务或者分割任务开始执行了,就必须到任务截止)。

(5)准线性上升假设:每个任务 $T(i)$在 j 和 k 个处理器上运行时的最不利计算时间$(j<k)$满足 $j*c(i,j)<k*c(i,k)$,准线性是由于在单个任务分割时通信和同步的损耗引起。

2.2 术语

定义 1 当一个任务的截止期和资源需求都满足时,判定该任务为可行;当一个任务集所有任务都可行时,判定该任务集的调度为可行。

定义 2 对一个任务集的子集进行的调度称为局部调度。当该子集扩展其任务补集内任一任务的调度仍为可行调度时,则称该局部调度强可行。当对任务集进行局部调度时,采用固定大小的子集进行依次调度。这样子集称为调度窗口,K 表示调度窗口大小。

定义 3 $ET_r(k,s)(ET_r(k,e))$是资源 $Resource(k)$共享(专用)方式下的最早有效时间。

定义 4 $P=\{P_1,P_2,P_3,\cdots\}$,$R=\{R_1,R_2,R_3,\cdots\}$,分别表示处理器和资源集。$ET_t(i)$是任务 $T(i)$的理想最早可运行时间。于是,

$$ET_t(i)=\mathrm{MAX}(w(i),\mathrm{MIN}_{j\in P}(Atime(j)),\mathrm{MAX}_{k\in R(i)}(ET_r(k,u)))$$

其中,$w(i)$,$R(i)\subseteq R$ 分别是任务 $T(i)$的到达时间和所需资源集,$Atime(j)$表示处理器 P 最早可运行任务的时间;$\mathrm{MIN}_{j\in P}(Atime(j))$为系统的处理器最早有效时间;$\mathrm{MAX}_{k\in R(i)}(ET_r(k,u))$表示任务 $T(i)$所需资源的最早可用时间,且当资源为共享访问方式时,$u=s$,当资源为专用访问方式时,$u=e$。

定义 5 $gaprate(i)$表示任务 $T(i)$的延迟度,$gaprate(i)=(d(i)-c(i,1))/c(i,1)$,反映了 $T(i)$任务截止期的急迫程度,当对任务集进行调度时,其作为处理器选择和是否进行并行处理的重要因素。

定义 6 任务目标函数定义为:

$$H(T(i))=d(i)+W_1*ET_t(i)+W_2*gaprate(i)$$

W_1、W_2为权值,分别表征了最早有效时间和延迟度对目标函数影响的程度,当 W_2为 0 时,即为 Myopic 算法,W_1、W_2都为 0 时,算法退化为 EDF 算法。

定义 7 调度成功率定义为:

$$success\_rate=\frac{\text{调度成功任务数}}{\text{任务总数}}$$

3 EDPF 算法

3.1 算法描述

在选择处理器方法时,Myopic 算法采用的是 EDF 方式,这样选择的缺点是没有考虑到任务运行时间的因素,当任务运行时间很短而可延迟时间很长时,会导致调度方案的不可行。EDPF 算法有效地减少了调度失败的可能,在考虑任务最早有效时间的基础上加入运行时间因素,使得调度方案可行性增加。且在任务无法满足时间截止期的情况下,采用并行处理技术,大大提高了系统的有效性。

这里采用与 Myopic 算法一样的系统[1],该系统中有 M 个资源以及 1 个资源列表 *ResourceList*[],其中每个资源对应一个入口,且每个资源对应一个访问任务数和一个模式参数决定资源是共享(专用)模式。可以对 EDPF 算法进行如下描述:

(1)将任务队列中的任务按截止期的非递减顺序排列。开始时,局部调度为空。

(2)通过对调度窗口中的 K 个(或少于 K 个)任务进行可行性检测来决定当前的局部调度是否为强可行的。若是,则 *feasible*=true;否则,*feasible*=false;

K_C为已经完成的可行性检测任务数。

$T(i)$为当前任务序列第(K_C+1)个任务。

Num_split 是 $T(i)$的并行级别。

P_Degree 是已经完成可行性检测的 K 个任务的并行级总和。

(2.1)首先,令 *Num_split=Max_split*;$K_C=0$;*P_Degree*=0;*feasible*=true。

(2.2)while(feasible==true)

①if(*K*−*P_Degree*<*Num_split*)令 *Num_split=K−P_Degree*;

②计算任务 $T(i)$的最早有效时间 $ET_t(i)$;

③寻找满足 $ET_t(i)+c(i,j)\leq d(i)$,且 $1\leq j\leq Num\_split$ 两个条件的最小处理器号 j;

④如果这个 j 存在,则令 $K_C=K_C+1$;*P_Degree=P_Degree*+1;

⑤否则 if(*Num_split*<*Max_split*)退出;

(3)if(feasible==true)

(3.1)计算调度窗口中的任务的目标函数 H 的数值,$H(T)=d(i)+W_1*ET_t(i)+W_2*gaprate(i)$;

(3.2)选择目标函数值最小的任务 T 扩充当前调度。否则,

(3.3)调度返回至上一级;

(3.4)在此层的调度窗口中,选择目标函数值次优的任务 $T(i)$扩充当前调度。选择处理器来运行;

(4)将调度窗口向后移动一个任务;

(5)重复步骤(2)~(4)的操作,直到以下条件中的任意一个得到满足:

①得到了一个完全的可行调度;

②已经达到了最大的返回次数或 $H(T)$函数的最大估算值;

③已经无法再返回。

3.2 算法举例和性能分析

例 设一个任务队列中有 9 个任务,这些任务分别为 $T(1)$、$T(2)$、$T(3)$、$T(4)$、$T(5)$、$T(6)$、$T(7)$、$T(8)$、$T(9)$,系统中有 3 个处理器,分别为 $P(1)$、$P(2)$、$P(3)$,有 2 个资源,分别为 $R(1)$、$R(2)$,且每个资源只有一个实例。$T(1)$、$T(8)$均以共享的方式访问资源 $R(1)$,而任务 $T(9)$则以专用的方式访问资源 $R(2)$。各任务的参数见表 1。

表 1 实时任务集举例

任务 $T(i)$	准备时间 $r(i)$	运算时间 $c(i,1)$	运算时间 $c(i,2)$	截止期 $d(i)$	资源需求
1	0	15	8	16	共享
2	0	10	6	11	
3	0	13	7	14	
4	8	3	2	27	
5	10	5	3	28	
6	13	18	10	29	
7	15	9	5	30	
8	19	17	9	38	共享
9	22	15	8	40	专用

首先设可检测窗口大小 K，权值 W_1、W_2，最大返回次数 *backtracks_num*,最大分割数 *Max_split* 分别为 3,1,1,1,3,则图 1(a)为 Myopic 算法进行调度的结果,图 1(b)为 EDPF 算法进行调度的结果。图中的一个方框表示调度中的一个结点。每个方框中的 3 个数字分别表示系统中 3 个处理器的最早可运行任务的时间。箭头旁的标号 $T_i(j)$表示任务 $T(i)$在处理器 $P(j)$上运行。

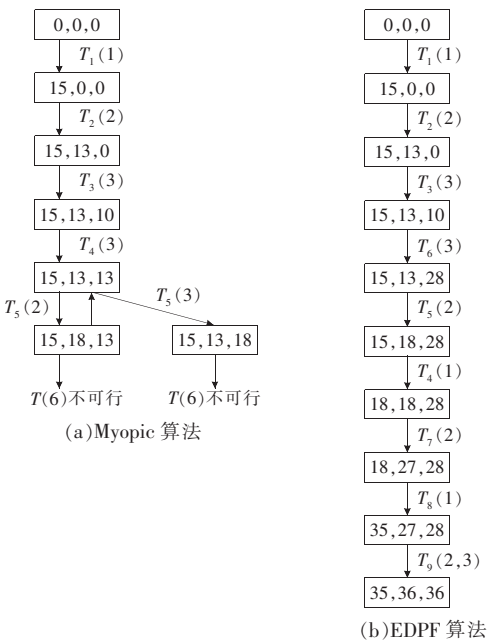

图 1　Myopic 算法和 EDPF 算法的搜索树

该例子中,如果使用 Myopic 算法,则在对由任务 $T(6)$、$T(7)$、$T(8)$所组成的调度窗口进行可行性检查时,由于任务 $T(6)$是不可调度的,将导致一次调度返回。而当重新对 $T(6)$、$T(7)$、$T(8)$所组成的调度窗口进行可行性检测时,又一次由于 $T(6)$的不可调度引发返回,然而此时已达到了最大返回次数,即导致了对 $T(6)$调度的失败,于是使用此算法未能找到该任务集的可行调度方案。如果使用 EDPF 算法对该例中的任务进行调度,则对任务 $T(4)$、$T(5)$、$T(6)$进行调度时,$T(4)$在处理器 $P(1)$、$P(2)$、$P(3)$上运行都可以满足其截止期,此时计算这 3 个任务在 $P(3)$上运行时的 H 值:

$$H(T_6)=29+10+(29-10)/18\approx 40$$

$$H(T_5)=28+10+(28-10)/5\approx 42$$

$$H(T_4)=27+10+(27-10)/3\approx 43$$

因此,EDPF 算法为 $P(3)$选择了 $T(6)$,下一步通过比较 $H(T_5)$和 $H(T_4)$又为 $P(2)$选择了 $T(5)$。当对任务 $T(9)$进行调度时,单处理器已经无法满足任务的截止期,因此需要通过并行处理来完成任务的调度,于是 $T(9)$便由 $P(2)$、$P(3)$同时处理,这样就避免了 Myopic 算法的返回问题,并且找到了一个可行调度。此外,改进算法的复杂度与 Myopic 算法相同,当存在 n 个需要调度的任务时,时间复杂度为 $O(Kn)$。

4　仿真试验

为了研究并行 EDPF 算法,本文作一些模拟。这里最关心的是一个任务集的所有任务在截止期前是否可以完成。因此,本文主要是对算法的调度成功率(success ratio)进行研究。一般来说,实时系统的调度算法的一个最重要的衡量标准是调度成功率,在前面已经作了定义。需要进行分析的参数有延迟度、调度窗口、权值、资源使用情况、最大调度返回次数对调度成功率的影响,并依次与 Myopic 算法的调度成功率进行比较。

这里,采用文[1]中的任务集生成方法,生成了 80 个可调度的任务集,每一个任务集含有 40~60 个任务。具体参数如表 2 所示。

表 2　仿真参数表

参数	说明
Checkwin_k	调度窗口大小
W_1	任务最早开始时间权值
W_2	任务延迟度权值
Max_c	任务最大执行时间
Min_c	任务最小执行时间
Pro_Num	系统中处理器个数
Res_Num	系统中资源类型数
Back_Num	最大允许返回数
gaprate	任务截止期延迟度
Use_P	任务调用一个资源的比例
Share_P	任务调用一个共享资源的比例
Max_split	任务的最大可并行级别

在仿真中,固定地将调度长度设置为 800。同时,将系统中的处理器个数设为 3,将系统中的资源个数设为 2,并假设每个资源只有一个实例。此外,将任务的最大运行时间和最小运行时间分别设定为 60 和 30。在这种参数取值下的模拟结果分别如图 2~图 6 所示。

4.1　调度窗口的大小对调度成功率的影响

在仿真中将 *Use_P*,*Share_P*,W_2,W_1,*Back_Num*,*gaprate* 和 *Max_split* 分别设置为 0.2,0.5,4,3,8,0.2 和 3，其仿真结果如图 2 所示。*Checkwin_k* 依次取 3,4,5,6,7,8,9,可以看出可检测窗口大小增大时,Myopic 算法和 EDPF 算法的调度成功率也随之增大。从图中可以看出，在调度窗口的大小相同时，EDPF 算法的调度成功率要高于 Myopic 算法。由于调度窗口的加大使得算法拥有更强的预见性启发式搜索时,调度成功率大大增加。

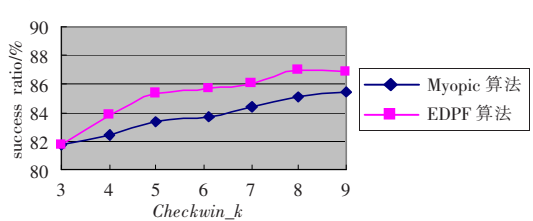

图 2　调度窗口的影响

4.2　最大返回次数对调度成功率的影响

仿真中，将 *Use_P*,*Share_P*,W_2,W_1,*gaprate* 和 *Max_split* 分别设置为 0.2,0.5,4,3,8,0.2 和 3。图 3 显示了最大返回次数对算法调度成功率的影响。*Back_Num* 取 0~30,每回增加 5 次,当允许的最大返回次数变大时,两个算法的 *success ratio* 都将随之增加。在最大返回次数的变化过程中,EDPF 算法的调度成功率始终优于 Myopic 算法。

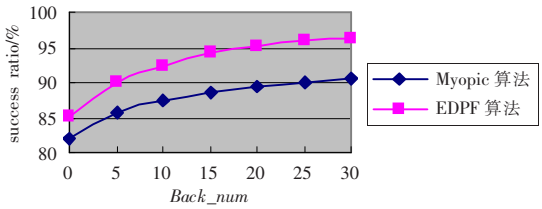

图 3 最大返回次数的影响

4.3 权值 W_1 对调度成功率的影响

在仿真中,*Checkwin_k*,*Use_P*,*Share_P*,W_2,*Back_Num*,*gaprate* 和 *Max_split* 分别设置为 10,0.2,0.5,0,30,0.2 和 3。图 4 显示了权值 W_1 对调度成功率的影响。从仿真结果可以看出,在权值的变化过程中,EDPF 算法的调度成功率都比 Myopic 算法高。W_1 从 0~6 逐点进行测试,发现,两种算法都有略微提升,曲线变化平稳,同时,当 W_1=0 时,两种算法退化成最早截止期 EDF 算法,该算法没有考虑处理器和资源问题,所以调度成功率相对较低。

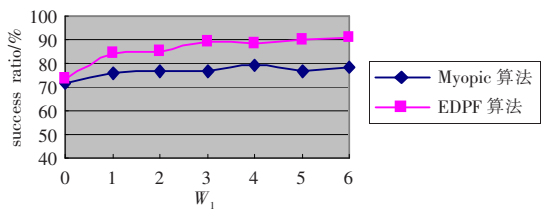

图 4 权值 W_1 的影响

4.4 任务的资源利用率对调度成功率的影响

图 5 显示了当任务使用资源利用率发生变化时,算法调度成功率的变化情况。*Max_split*=3,*Share_P*=0.5,*Checkwin_k*=8,W_1=3,W_2=4,*Back_Num*=30,*gaprate*=0.3。从仿真结果来看,随着 *Use_P* 的增大,即资源利用率的增大,两种算法的调度成功率都较快下降,从 95%以上下降到 50%左右。原因在于,当资源利用率增大时,任务使用资源数变多,使得发生资源冲突的可能性也随之增大。同时,通过比较可以发现,新的算法较 Myopic 算法有较大的提高。

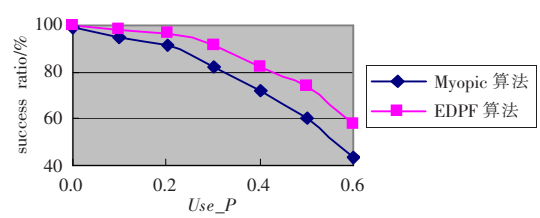

图 5 资源利用率的影响

4.5 任务的延迟度对调度成功率的影响

在仿真中,将 *Max_split*,*Use_P*,*Share_P*,*Checkwin_k*,W_1,W_2,*Back_Num* 分别设定为 3,0.2,0.5,10,3,4 和 30。图 6 体现了任务的延迟度对调度成功率的影响。仿真结果显示,当任务的延迟度增加时,两个算法的调度成功率都将增加,而且 EDPF 算法的调度成功率始终高于 Myopic 算法。

4.6 权值 W_2 对调度成功率的影响

在仿真中,将 *Max_split*,*Use_P*,*Share_P*,*Checkwin_k*,W_1,*Back_Num* 分别设定为 1,0.2,0.5,10,10 和 30。图 7 体现了权值 W_2 对调度成功率的影响。W_2=0 时,两者一致,都在 73%左右;随着 W_2 的增大,调度成功率逐渐增加,但是当 W_2>4 时,成功率开始下降,这是由于此时延迟度成了主要因素,弱化了任务截止期对调度成功率的影响。

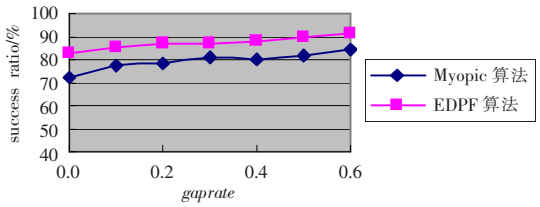

图 6 延迟度的影响

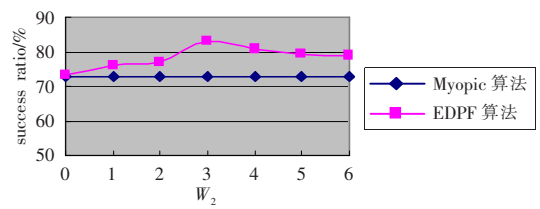

图 7 权值 W_2 的影响

5 结论

目前,MSP 问题仍然是一个重要的研究领域,迄今为止,已存在了许多算法,包括静态调度算法和动态调度算法。其中,静态调度算法当环境变得更加复杂和恶劣时,这些算法就不再适用,此时动态调度算法则有了更大的适用范围。但是这些算法大都采用的是 EDF 方式,这样的选择缺点是没有考虑到任务运行时间的因素,当任务运行时间很短而延迟时间很长时,可能会导致调度方案的不可行。

本文在这些算法的基础上提出了基于并行 EDPF 的算法,增加了一种针对实时多处理器系统的多任务调度解决方案。若系统中的任务具有并行性,则 EDPF 算法可以有效地减少调度失败的可能,并在考虑任务最早有效时间的基础上加入运行时间因素,使得调度方案可行性的增加,大大提高了系统的有效性。通过仿真实验,可以看到 EDPF 算法的调度成功率要优于 Myopic 算法的调度成功率。(收稿日期:2007 年 1 月)

参考文献:

[1] Ramamritham K J,Stankovic A,Shiah P-F.Efficient scheduling algorithms for real-time multiprocessor systems[J].IEEE Transactions on Parallel and Distributed Systems,1990,1(2):184-194.

[2] Douglass B P.Real-time design patterns[M].北京:北京航空航天大学出版社,2004.

[3] Galli D L.Distributed Operating Systems Concept and Practice[M].北京:机械工业出版社,2005.

[4] Chetto H,Chetto M.Some result of the earliest deadline scheduling algorithm[J].IEEE Transactions on Parallel and Distributed Systems,1989.

[5] Mok A K.Fundamental design problems of distributed systems for the hard real-time environment.Cambridge,MA:Department of Electronic Engineering and Computer Sciences,MIT,1983.