维普资讯 http://www.cqvip.com

多处理器单调速率任务分配算法性能评价*)

王 涛 刘大昕

(哈尔滨工程大学计算机科学与技术学院 哈尔滨 150001)

摘 要 多处理器任务分配调度算法是一类经典实时调度算法,然而目前研究在如何根据任务集特征选择任务分配算法方面少见指导性原则,不利于提高多处理器任务分配算法的可调度率及使用尽可能少的处理器达到最优调度结果。基于两种多处理器任务调度策略的比较,本文给出划分策略下的多处理器 RM 调度的可调度条件和任务分配算法及分析。仿真结果表明,各任务分配算法所需处理器数与任务集总利用率成正比。同时,分析总结出各算法适用范围及如何根据任务集利用率选择合适算法的指导原则。最后结果还表明,实际算法性能与理论性能界存在差异。

关键词 多处理器,速率单调,划分,可调度性

The Performance Evaluation of Rate Monotonic Tasks Assignment Algorithms on Multiprocessor

WANG Tao LIU Da-Xin

(Department of Computer Science and Technology, Harbin Engineering University, Harbin 150001)

Abstract Tasks allocation and scheduling algorithms on multiprocessors are a kind of classic real-time scheduling algorithms. However the current study lacks of guiding principles on how to select the proper algorithm based on the task sets' characteristics. It is not conducive to improve the rate of schedulability of the multiprocessor tasks allocation algorithms as well as obtain the optimal results using the minimal the processors. The schedulability conditions and task allocation algorithms as well as analysis of RM scheduling under multiprocessor partition scheme are given based on the comparation of the both multiprocessor tasks scheduling schemes in this paper. Simulation result shows that the number of processors required by all task allocation algorithms is proportional to the total utilization of the task set; the applicable area of the given algorithms and the guiding principles on how to select the proper algorithm based on the task sets' utilization are summed up. The final result also shows that there is difference between the practice performance and the theoretic performance bound of the algorithms.

Keywords Multiprocessor, Rate monotonic, Partition, Schedulability

1 引言

实时系统为满足实时期限,在某些情况下需要繁重的计算请求,从而要求系统具有足够强大的处理能力。同时,硬实时系统要求的高可靠性,也需要冗余的处理能力。这些都意味着需要利用多处理器的优势为实时系统提供强有力的处理能力,为任务系统的可调度性提供保障。迄今为止,国内在实时多处理器任务划分算法性能评价方面的研究较为少见,RM 调度算法[1,2]下尚缺乏对各种任务分配算法的性能和特点系统进行分析评价。中国科学院软件研究所乔颖等人在文[3,4]中提出了一种节约算法和新型任务分配策略,并与其所述近似算法从可调度率上进行了比较。其研究工作考虑到异构处理器环境的集成调度问题,与本文将要探讨的同类多处理器任务分配方法的研究内容属不同范畴。

然而,即使我们希望实时系统能够从多处理器系统中受益,在实时系统中使用多处理器系统仍非常困难。主要的障碍就是多处理器环境下的任务调度算法比单处理器系统要明显的复杂。因为多处理器系统中,调度算法不仅需要对任务集调度排序,还要决定哪些任务需要使用哪些处理器进行调度。大型周期固定优先级任务集在多处理器系统上的可抢占调度问题被认为是难以计算的,是 NP 困难问题[5]。因此,研究集中在开发适合的启发式算法,这些启发式算法与理想的最优算法相比,要求不仅能够高效地实现,还要能够尽量少地使用有限数量的额外处理器。

解决多处理器系统划分方案下的任务调度问题,基本思想是在单处理器上使用 RM 调度算法,并利用多处理器下任务集的可调度条件。开发新型实时任务分配算法成为实时界研究的一个热点问题。本文以下部分安排如下:第 2 节描述、分析、比较两种经典多处理器任务调度策略;第 3 节描述多处理器任务调度模型;第 4 节讨论划分方案下的多处理器 RM 可调度条件并根据可调度条件给出划分方案下的 5 种启发式算法,同时归纳算法的复杂度和性能、分析算法利用率界;第 5 节为上述 5 种算法平均情况下的仿真实验设计及结果,横向比较各算法在不同任务可达利用率参数下的算法性能;最后是本文结论以及未来工作方向。

2 多处理器任务调度策略

多处理器环境的任务调度存在两种策略:全局方案和划分方案[6]。在全局调度方案策略中,实时任务的每一次出现都在不同的处理器上执行,所有处理器上只运行同一种调度

*)本文工作得到黑龙江省自然科学基金资助(项目编号:F2005-02)。王 涛 博士研究生,主要研究方向为实时系统调度等;刘大昕 教授,博士生导师,主要研究方向为数据库与知识库。

维普资讯 http://www.cqvip.com

算法。任务在执行完之前可以被抢占并且可以在不同的处理器间迁移，同时假定多处理器间共享内存的开销非常低。这种方案的主要目标就是为多处理器系统产生一个能够满足它们各自期限任务分配。相反，在划分调度方案中，一个任务的所有出现都在同一处理器上执行，全部任务由任务分配算法预先划分到处理器；每一个处理器可以运行不同或者相同的单处理器任务调度算法。这种划分方法主要用于任务集参数已知的静态优先级调度，对任务集进行脱机分配，它充分利用了“分治算法”的设计思想，将难解的问题分解为 n 个子问题，分而治之，逐一突破。

划分方案与全局方案相比，具有几个优势。首先，由于多处理器调度的开销仅仅在于给处理器分配任务，并且这个操作在任务第一次执行前仅仅执行一次，所以划分调度的复杂性更低；第二，一旦给处理器分配任务的操作完成，剩下的工作就可以由单处理器调度算法完成。图 1 是两种策略的示意图，图(a)为划分方案，任务由分配算法分配给处理器，每个处理运行单处理调度算法，被划分的子任务集与处理器绑定，不再变化；图(b)是全局方案，调度算法对任务调度排序，任务在执行过程中可在不同处理器间迁移。

划分方案的性能由两个因素决定：给处理器分配任务的任务分配算法和每个处理器上决定任务执行顺序的任务调度算法。对一个给定的调度算法，一个最优的任务分配算法可以用最少数量的处理器为每一个处理找到一个可行的调度。

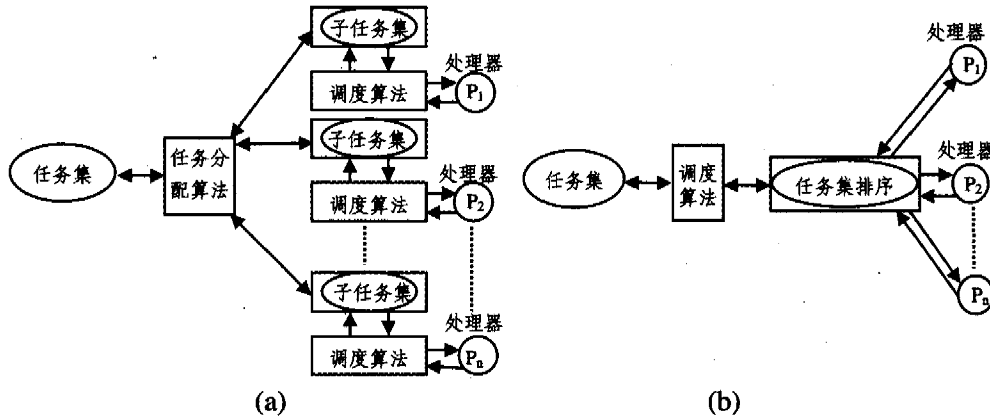

图 1　划分方案和全局方案

但是，为固定优先级调度算法（包括静态优先级 RM 算法和动态优先级 EDF 算法）找到一个最优任务分配问题，却是一个 NP 困难问题[5]。在一个多处理器系统上分配实时任务的任何一个实际调度算法，其计算复杂性和算法的性能总是相互矛盾的。早期的研究主要集中在性能保证和最坏情况界方面对算法的组合分析。度量一个任务分配算法 A 的性能渐进性界，使用公式，$\mathcal{R}_A=\lim\limits_{N_{opt}\to\infty} N/N_{opt}$，其中 N 为启发式分配算法所需处理器数量，N_{opt} 为最优分配算法所需处理器数量。但 $\mathcal{R}_A$ 只是用于比较不同任务分配算法的性能，代表的是算法与最优算法相比较的性能界。对任务在算法下的可调度性测试并不实用，因为最优算法所需的处理器数在多项式时间内无法获得。

3　多处理器任务调度模型

考虑一组周期任务在同等多处理器上的可抢占调度问题。多处理器和任务集描述如下：

设 $\tau=\{\tau_1,\tau_2,\cdots,\tau_n\}$ 为含有 n 个周期任务的任务集合，$\tau_i=(C_i,T_i),(i=1,\cdots,n)$，其中 τ_i 表示相互独立的单个任务（假设任务间没有任何依赖关系和资源约束），其执行时间和周期分别为 C_i 和 $T_i(T_i>0)$，相对期限为 D_i。本文仅考虑任务周期和相对期限相等并且任务执行时间小于等于周期的情况，即 $T_i=D_i$ 且 $0\leqslant C_i\leqslant T_i$。任务 τ_i 的利用率为 $u_i=C_i/T_i$，任务集的总利用率为 $U=\sum\limits_{i=1}^{n}C_i/T_i$。设 α 为任务最大可达利用率，$\alpha=\max u_i,(i=1,\cdots,n)$，显然 $\alpha\in(0,1]$。仿真实验中，将根据不同 α 的取值比较算法的性能。

$P=\{P_1, P_2,\cdots, P_m\}$ 为含有 m 个具有相同处理能力的同等处理器集；η_m 为第 m 个处理器 P_m 的利用率，即分配给该处理器的任务的总利用率 U_m。假设处理器间无任何共享资源，同时假定每个处理上运行 RM 调度算法，即如果 $T_i<T_j$，则 τ_i 比 τ_j 具有更高的优先级，优先在该处理器上执行。另外，假定任务调度是以可抢占的方式进行并且中断后可恢复执行。

4　多处理器任务分配算法

多处理器任务划分方案存在两种情况：一种是处理器数量固定，一种是无限制处理器数量情况。使用固定数量处理器，目的是找到一种可行的任务分配算法，并在固定数量处理上测试任务集的可调度性。无限处理器数量的情况下的任务分配算法，目的是找到最小的处理器数量，可行地分配所有任务。这个问题类似于经典组合优化理论中的装箱问题。

设有 n 种物品，它们的尺寸分别为 $s_1,\cdots,s_n$；另有任意多个箱子(Bins)，假定每个箱子的容量为一固定常数 C。

优化问题：求使用最小箱数将物品装入的装箱方法。

判定问题：任意给定 k 个箱子，是否存在一种装箱方法，用小于等于 k 个箱子将这些物品全部装入。

相对于装箱问题，任务调度系统中处理器对应于箱子，任务对应于物品，物品重量对应于任务利用率因子。但是这种简单的固定箱子容量的装箱问题与多处理器任务调度问题有一个关键的不同点，即简单装箱问题中箱子大小是不变的，而对应于箱子大小尺寸的处理器利用率是一个由预先定义的函数确定的变量。预先定义的函数即为任务集可调度条件，它是关于任务数的函数。任务分配算法中，可调度条件决定任务是否可被分配给处理器进行可行的调度。

周期任务的处理器分配问题不仅要依靠分配算法自身，同时需要依靠算法使用的任务集可调度条件。多处理器系统

维普资讯 http://www.cqvip.com

的可调度条件正是基于单处理器系统的基础建立的。本节讨论多处理器划分调度下的任务集可调度性判定条件、算法和算法的性能界。

4.1 多处理器 RM 可调度性判定条件

Liu 和 Layland 的可调度条件是基于最坏情况(Worst-Case,WC)对任务集总利用率的界定(以下简称 LLWC),在多处理器环境下依然可以作为 RM 算法在单个处理上处理所分配的任务可调度性的判定条件,因此在此有必要重申 LLWC 条件[1]。

定理 1(LLWC 条件) 给定周期任务集 τ,如果总利用率满足 $U\leqslant n(2^{1/n}-1)$,则任务集在 RM 算法下是可调度的。当 $n\to\infty$时,处理器利用率 $f(n)=n(2^{1/n}-1)\to\ln2$。

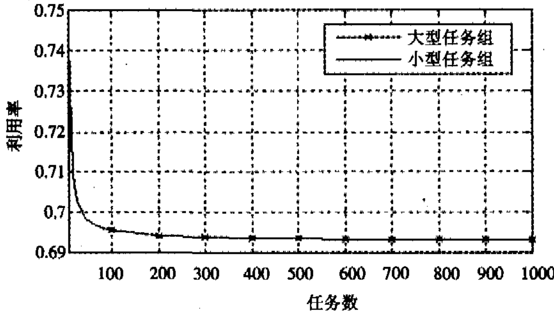

图 2 利用率随 n 变化关系

为分析 LLWC 条件,我们选择了两组各包含不同数量任务的任务集:第一组中每个任务集含任务数量在[100,1000]之间均匀分布,称为大型任务组;第二组中每个任务集包含任务数量在[5,95]之间均匀分布,称为小型任务组。试验表明,小型任务组利用率随任务数递减的速度较快,大型任务组利用率随任务数的增加,利用率趋于平缓,向 $\ln2\approx0.69$ 逼近。图 2 表明函数 $f(n)=n(2^{1/n}-1)$为连续严格单调递减函数,当 $n\to\infty$时,处理器利用率 $f(n)$将最终稳定在 0.69 附近。

LLWC 条件是充分但非必要条件,可能存在任务集能够满足任务的各自期限,但不满足该条件的情况。Burchad 等人针对具有简单周期(具有周期任务的系统具有简单周期是指,对于系统中的任意一对任务 T_i 和 T_k,当相应周期 $p_i<p_K$ 时,p_k 是 p_i 的整数倍。)和周期接近的任务集提出更高的处理器利用率上界。这里需要指出,下面将要引出的定理并不能很好地应用于任务利用率较高的任务集合。由于该条件是面向任务周期的,因此我们简单地称其为 PO (Period Oriented 面向周期)条件[7]。

定理 2(PO 条件) 对于给定任务集,$\tau=\{\tau_1,\tau_2,\cdots,\tau_n\}$,定义

$$\gamma=\max_{1\leqslant i\leqslant n}D_i-\min_{1\leqslant i\leqslant n}D_i$$

其中 $D_i=\log_2T_i-\lfloor\log_2T_i\rfloor(i=1,\cdots,n)$

若 $\gamma\leqslant1-1/n$,并且总利用率满足 $U\leqslant(n-1)(2^{\gamma/(n-1)}-1)+2^{1-\gamma}-1)$,则任务集在该处理器下用 RM 算法是可调度的;若 $\gamma>1-1/n$,并且总利用率满足 $U\leqslant n(2^{1/n}-1)$,则任务集在该处理器下用 RM 算法是可调度的。以上两个条件都是紧密的。PO 条件中的 γ,直接的含义是度量系统中任务偏离简单周期的距离。为便于算法实现,推论 1[7] 给出简化的 PO 条件将作为简单周期和近简单周期任务集在分配给处理器时的可调度性判定标准。

推论 1 对于给定任务集 $\tau=\{\tau_1,\tau_2,\cdots,\tau_n\}$和定理 2 定义的 γ,若总利用率 $U\leqslant\max\{\ln2,1-\gamma\ln2\}$,则任务集在该处理器下用 RM 算法是可调度的。

脱机任务分配算法需要调度开始前就知道确定的任务集某属性(如任务数量和周期)。对于那些用 LLWC 条件不能判定可调度性的任务集,可以用下面的条件进行可调度性划分的判定。该条件按照周期递增将任务集排序,因此称为周期递增(IP,Increasing Period)条件[8],它比较任务集前 $n-1$ 个任务的总利用率和第 n 个任务的利用率,判断任务集是否可调度。

定理 3(IP 条件) 给定周期任务集 $\tau=\{\tau_1,\tau_2,\cdots,\tau_n\}$的周期满足 $T_1\leqslant T_2\leqslant\cdots\leqslant T_n$。设 $U_{n-1}=\sum_{i=1}^{n-1}C_i/T_i\leqslant(n-1)(2^{1/(n1)}-1)$,若满足 $C_n/T_n\leqslant2(1+U_{n-1}/n-1)^{-(n-1)}-1$,则任务在 RM 算法下可调度。当 $n\to\infty$,任务 τ_n 的最小利用率达到 $2e^{-U_{n-1}}-1$。

IP 条件仅与任务集中任务数和总利用率有关,所以可以作为任务集可调度性的测试条件,其中函数 $f(U_{n-1},n)=2(1+U_{n-1}/n-1)^{-(n-1)}-1$ 是 U_{n-1} 和 n 的严格递减函数。IP 条件将用于下文的 RMNF、RMFF 以及 RMBF 算法的可调度条件。

为更精确地刻画 RM 算法的行为特征,Lehoczky 等人对随机产生的周期任务集从任务细分利用率概率分布的角度做了随机分析,给出一种基于累积处理器时间需求的 RM 可调度充分必要条件[9]。该条件表明,对随机周期任务集,随着任务数量的增加,任务执行时间变得不重要,任务利用率会根据任务周期趋于一个常数。

定理 4(NS 条件) 设给定周期任务集 $\tau=\{\tau_1,\tau_2,\cdots,\tau_n\}$的周期满足 $T_1\leqslant T_2\leqslant\cdots\leqslant T_n$。用 $W_i(t)=\sum_{j=1}^{i}C_j\cdot\lceil t/T_j\rceil$表示[0,$t$]时间间隔任务集前 i 个任务对处理器的累积需求,定义 $L_i(t)=W_i(t)/t$,$L_i=\min_{\{0<t\leqslant T_i\}}L_i(t)$。则 $L=\max_{\{1\leqslant i\leqslant n\}}L_i$。

①第 i 个任务 τ_i 能被可行调度,当且仅当 $L_i\leqslant1$。

②任务集 $\tau=\{\tau_1,\tau_2,\cdots,\tau_n\}$用 *RM* 算法可调度,当且仅当 $L\leqslant1$。

4.2 RM 任务划分算法

目前几乎所有的单处理器任务分配方案都是基于文[1]提出的 RM 可调度充分条件,因而现存 RM 任务划分方案与启发式装箱方法存在不同。基于装箱问题的 NF 和 FF 启发式划分算法,文[8]提出两种多处理器周期任务分配算法 RMNF 和 RMFF。在装箱问题中,FF 和 NF 算法都是联机算法,不同于装箱算法,在多处理器任务分配问题中,需要根据任务的利用率判定该任务是否可以分配给一个处理器,因而要预先获知任务集的一些属性,如任务数量、周期、执行时间等,因此 RMNF 和 RMFF 算法是脱机算法。在提出 2 种算法的同时,文[8]分析了算法的性能比 $\mathcal{R}_A=\lim_{N_{opt}\to\infty}N/N_{opt}$。在最坏情况下,RMNF 和 RMFF 算法的性能上界分别为 2.67 和 2.23,下界为 2.4 和 2.0。两种算法都使用 IP 条件作为可调度判定条件并具有相同的复杂性 $O(n\log n)$。下面分别给出两种基于装箱划分算法的任务分配算法的伪代码描述。

很明显,两算法的性能上下界之间存在缝隙,是不紧密的。这里就产生一个问题,这个上界和下界是不是就是真实的界?在实际实现算法需要多处理器执行一个周期任务集的

维普资讯 http://www.cqvip.com

情况下，这个问题可能加重。一方面，需要充足数量的处理器保障最坏情况下任务集能满足硬实时期限；另一方面，又要使用尽可能少的处理器，避免资源浪费。为此，文[10]给出了 RMNF 和 RMFF 的紧密性能界，分别为 $\mathcal{R}_{RMNF}=2.67$，$\mathcal{R}_{RMFF}=2.33$，同时在证明的过程中，纠正了文[8]关于 RMFF 上界证明中的错误。

上述算法并没有注意到任务集以下特征，即当某个处理器上的任务只是简单的周期任务时，任务的可调度利用率会比普通任务高。为此文[11]给出了如下定理。

算法. RMNF

Input: Task set $\tau=\{\tau_1,\tau_2,\cdots,\tau_n\}$ and a set of processors $\{P_1,\cdots,P_m\}$
Output: j; number of processors required.
1. Tasks are sorted in non-decreasing order of their periods
2. $i:=1$; $j:=1$; /* $i=i^{th}$ task, $j=j^{th}$ processor */
3. while ($i\leqslant n$) do
4. if ($u_i\leqslant$ Condition IP) then
5. $U_j:=U_j+u_i$;
6. else
7. $U_{j+1}:=U_{j+1}+u_i$;
8. $j:=j+1$;
9. $i:=i+1$;
10. return (j);
11. end

算法. RMFF

Input: Task set $\tau=\{\tau_1,\tau_2,\cdots,\tau_n\}$ and a set of processors $\{P_1,\cdots,P_m\}$
Output: j; number of processors required.
1. Sort the tasks by increasing periods
2. $i:=1$; $j:=1$; /* $i=i^{th}$ task, $j=j^{th}$ processor */
3. while ($i\leqslant n$) do
4. $q:=1$;
5. while ($u_i>2(1+U_q/k_q)^{-kq}-1$) do
6. $q:=q+1$; /* q denotes the processor index */
7. $U_q:=U_q+u_i$; $k_q:=k_q+1$;
8. if ($q>j$) then
9. $j:=q$;
10. $i:=i+1$;
11. return (j);
12. end

定理 5 任务系统 τ 中具有独立可抢占的周期任务，并且任务的相对期限等于各自周期。如任务集可划分为 h 个包含简单周期任务的不相邻子集 $S_1,S_2,\cdots,S_h$，并且 h 个子集中任务的总利用率 $U(S_i)$，$(i=1,2,\cdots,h)$ 满足以下不等式：

$$(1+U(S_1))(1+U(S_2))\cdots(1+U(S_h))\leqslant 2$$

则该任务系统 τ 按照 RM 算法是可调度的。

从定理 5 可知，任务集按速率单调算法在某处理器上调度，若将任务划分为少量仅包含简单周期任务的若干子集，可以显著提高可调度利用率。因此，一种多处理器分配算法首先将任务集按照条件划分为多个子集，只要子集总利用率不大于分配给它的处理器利用率上限，那么没给自己都是可调度的。如果有多个子集分配给同一个处理器，那么它们的可调度利用率就是关于子集数的函数，与任务数无关。基于上述思想，Burchad 等人提出了定理 2 的 PO 条件和 RMST 及 RMGT 算法[7]，并且证明其性能界分别为 $\mathcal{R}_{RMST}\leqslant 1/(1-\alpha)$（$\alpha$ 为任务最大可达利用率）和 $\mathcal{R}_{RMGT}=7/4$，计算时间复杂性都是 $O(n\log n)$。下面给出两种算法的伪代码描述。

RMGT 算法不依赖于一个任务的利用率，它首先将任务集按照利用率划分为两个子集。利用率小于等于 1/3 的任务使用 RMST 算法处理；利用率大于 1/3 的大型任务使用最先适合方式，分配给最多拥有一个按 RMST 算法分配的任务的处理器。

算法. RMGT

Input: Task set $\tau=\{\tau_1,\tau_2,\cdots,\tau_n\}$ and a set of processors $\{P_1,\cdots,P_m\}$
Output: j; number of processors required.
1. Obtain $D_i:=\log_2 T_i-\lfloor\log_2 T_i\rfloor\log_2 T_i$ for all tasks and sort the task set such that $0\leqslant D_i\leqslant\cdots\leqslant D_n<1$
2. $i:=1$; $j:=1$; /* $i=i^{th}$ task, $j=j^{th}$ processor */
3. $D:=D_i$; $\gamma_j:=0$; $U_j:=u_i$; $i:=i+1$;
4. while ($i\leqslant n$) do
5. $\gamma_j:=D_i-D$; /* γ_j value is updated */
6. if ($U_j+u_i\leqslant\max\{\ln 2, 1-\gamma_j\ln 2\}$ then
7. $U_j:=U_j+u_i$; /* Task τ_i to processor P_j */
8. else
9. $j:=j+1$; $D:=D_i$; $\gamma_j:=0$;
10. $U_j:=U_j+u_i$; /* Task τ_i to processor P_{j+1} */
11. $i:=i+1$;
12. return(j);
13. end

算法. RMST

Input: Task set $\tau=\{\tau_1,\tau_2,\cdots,\tau_n\}$ and a set of processors $\{P_1,\cdots,P_m\}$
Output: j; number of processors required.
1. Partition the task set in two groups:
$G_1:=\{\tau_i\mid u_i\leqslant 1/3\}$; $G_2:=\{\tau_i\mid u_i>1/3\}$;
2. Allocate tasks from the first group using RMST.
3. Allocate tasks from the second group as follows:
4. $i:=1$; $U_m:=u_i$;
5. $i:=i+1$;
6. Use heuristic First-Fit to find a processor k that executes only one task (e.g, task τ_w) and satisfies the following conditions:
$\lfloor T_w/T_i\rfloor(T_i-C_i)\geqslant C_w$ or $T_w\geqslant\lceil T_w/T_i\rceil C_i+C_w$ if $T_i<T_w$
$\lfloor T_i/T_w\rfloor(T_w-C_w)\geqslant C_i$ or $T_i\geqslant\lceil T_i/T_w\rceil C_w+C_i$ if $T_i\geqslant T_w$
7. if such processor exists then
8. $U_k:=U_k+u_i$; /* Allocate task τ_i to processor P_k and set it as fully utilized
9. else
10. $U_l:=u_i$; /* Allocate task τ_i to an idle processor with index l. */
11. goto 5
12. Terminate when all tasks from group G_2 are allocated.
13. return(j) /* j number of processors required in sets G_1 and G_2 */
14. end

5 仿真试验

实时调度理论中的算法分析，由于可靠性对系统至关重要，系统调度必须满足任务期限才有意义，因此理论上有必要讨论算法在最坏假设下的性能界。但最坏情况分析并不能代表算法的真实运行情况，会受到多种软硬件环境的干扰和影响，因此在实验中，往往考虑平均情况下的算法性能表现才更有现实意义。实验考察了各多处理器任务分配算法在固定数量处理器下划分相同任务集所需处理器数量。

5.1 处理器集和任务集描述

考虑到实际应用中多数情况是在固定处理器数量系统上的任务划分，因此仿真试验设计使用 30 个处理器，并假定对任务集而言每个处理器上运行相同速率单调度算法，处理器处理能力相同。同时，为保证任务集样本可划分，最大任务集

维普资讯 http://www.cqvip.com

限制含 250 个任务，所有任务集中大多数任务的利用率均匀分布在 30/250=0.12 以内。然后，为满足不同最大可达利用率分组测试的需要，对每组任务集设置若干机动调整任务，以保证任务集在一定的小范围内均匀达到每组最大可达利用率的要求。但在分组中，不同算法严格按照对同一任务集的测试，以保证实验结果能够公平反映各划分算法的性能。

任务集根据最大可达利用率分为 3 组：$\alpha=0.3$，$\alpha=0.6$，$\alpha=0.9$。每组中在[25,250]中每隔 25 取 1 个值，产生 10 个大小不同的任务集。每个任务集中任务周期取值范围为 $1\leqslant T_i\leqslant 1000$，在[1,1000]产生 250 个均匀分布的值。任务执行时间根据 $C_i=T_i*u_i$ 计算，u_i 为任务利用率。为保证多数任务集任务利用率小于 0.12，先以 0.12/250 比值为间隔，在 0.12 以内产生 250 个利用率 u_i。根据处理器需求调整前面几个利用率值，均匀分布至最大可达利用率并刚好能够划分 250 个任务，计算出执行时间并不再变化，此后，每次减少 25 个任务，在每组中产生 10 个任务集。3 组不同最大可达利用率的任务集共产生出 30 个大小不同的任务集。实验使用 XML 文件格式作为任务集载体，所有算法运行环境均为 Windows2000 advanced server，PIV1.8，512M RAM。

5.2 性能指标

由于最优划分所需处理器数量无法计算，所以使用任务集总利用率 U 作为划分所需处理器数量的下界，即划分任务集最少需要 U 个处理器。一个性能指标定义为实际使用处理器数量与被划分任务集总利用率的比值：

$$\rho=\frac{M}{U}$$

其中 M 为试验所需实际处理器数，U 为任务集总利用率。

5.3 实验结果及分析

表 1　不同 α 值的仿真结果

任务数		25	50	75	100	125	150	175	200	225	250
$\alpha=0.3$	RMBF	3	4	5	6	8	10	13	16	20	24
	RMGT	3	3	4	5	7	8	11	13	16	19
	RMNF	3	4	5	6	8	10	13	16	20	24
	RMFF	3	4	5	6	8	10	13	16	20	24
	RMST	3	3	4	5	7	8	11	13	16	19
$\alpha=0.6$	RMBF	5	6	7	8	10	12	15	18	22	26
	RMGT	6	6	7	8	10	11	13	16	18	21
	RMNF	5	6	7	8	10	13	16	19	23	27
	RMFF	5	6	7	8	10	12	15	18	22	26
	RMST	6	6	6	7	8	10	13	15	18	21
$\alpha=0.9$	RMBF	8	9	10	11	13	16	19	22	26	30
	RMGT	9	9	10	11	13	14	16	19	21	24
	RMNF	9	9	10	12	13	16	19	22	26	30
	RMFF	8	9	10	11	13	16	19	22	26	30
	RMST	9	10	11	13	14	17	17	19	21	24

算法运行结果见表 1。实验数据表明，随着各算法所需数理器数随任务数增加而增加，RMGT 和 RMST 算法表现出优异的性能，同样大小任务集下所需处理器最少，两算法性能表现非常接近，尤其在 $\alpha=0.3$ 组的任务划分表现出同样的性能，符合两算法的理论依据，因为 RMGT 算法在 $\alpha\leqslant 0.3$ 时使用 RMST 划分任务集。在 $\alpha=0.6$ 时，RMGT 性能有所下降，因为算法需要使用最先适合的方式处理利用率大于 0.3 的任务而需要额外的处理器。$\alpha=0.9$ 时，RMST 算法性能明显下降，证明小任务速率单调算法适合于简单周期较低利用率任务划分，不适合高利用率任务集。另外 3 种 RMBF、RMNF、RMFF 算法始终表现了相近的性能，符合使用同样可调度条件的理论依据。

图 3 直观描述了 3 组任务集与所处理器关系的变化曲线。图 3(a) 中，RMBF、RMNF、RMFF3 种算法性能表现相同，曲线重合；RMGT 和 RMST 算法重合，表现出相同性能。随着利用率提高，各算法出现差异，表明在对于相同任务集由于算法各有针对，所以会出现差别。图 3(b) 清楚表明 RMNF 算法使用了最多数量的处理器，从而算法性能最低。图 3(c) 出现部分重合的曲线，表明算法对高利用率任务集合性能会有下降趋势，尤其 RMST 算法在任务集小于 125 以前出现了曲线高于其他另外 3 种算法的现象，原因是 RMST 算法不适合高利用率较小的任务集划分。

表 2 是各算法的最终性能指标。数据表明各算法使用处理器数与任务集总利用率成正比。理论上 ρ 值应该随最大利用率增加而增加，但受个体任务在任务集中出现的位置不同和额外的上下文切换开销等因素影响，实验结果并非与理论一致，不过总体 ρ 值还是没有出现很大的波动，在正常范围之内，在一定程度上也反映出各算法性能较稳定，与理论上的 $\mathcal{R}_A=\lim\limits_{N_{opt}\to\infty} N/N_{opt}$ 比较，由于任务集中多数任务采用较小利用率，各算法 ρ 值分别低于最大利用率界 $\mathcal{R}_A$。

结论　通过本文的研究工作，对多处理器任务调度问题认识到以下几点：(1) 多处理器任务分配算法渐近性能界 $\mathcal{R}_A=\lim\limits_{N_{opt}\to\infty} N/N_{opt}$ 用来度量算法性能好坏，目的是使用尽可能少的处理器可行的分配任务集，因此设计一种任务分配算法，要尽可能降低 $\mathcal{R}_A$ 的值，尽可能计算紧密的 $\mathcal{R}_A$；(2) 多处理器可调度判定条件，用于分配算法本身在进行任务划分时的判定依据，是算法的核心。准确合理的可调度条件公式，可以提高算法对任务集划分的效率，提供可调度率；(3) 分配算法的利用率界分析是分配算法分析的主要核心内容，用于对不同特征任务集选择不同分配算法进行任务划分（通过对任务集总利用率与算法利用率界的比较，判断使用该算法对任务集是否可以产生可行分配）。

未来工作将集中在以下几个方面：第一，考虑资源约束条件下的多处理器任务调度问题；第二，考虑开始时间抖动问题，提出降低抖动发生的方法；第三，多处理器联机任务分配相关算法研究。

参考文献

1 Liu C L, Layland J W. Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of ACM, 1973, 20(1): 174～189

2 王永吉，陈秋萍. 单调速率及其扩展算法的可调度性判定. 软件学报，2004，15：799～814

3 乔颖，王宏安，戴国忠. 一种新的实时多处理器系统的动态调度算法. 软件学报 2002，13(1)：51～58

4 乔颖，邹冰，方亭，王宏安，戴国忠. 一种实时异构系统的集成动态调度算法. 软件学报 2002，13(12)：2251～2258

5 Leung J Y T, Whitehead J. On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks. Performance Evaluation, 1982(2): 237～250

6 Oh Y, Son S H. Allocating Fixed-Priority Periodic Tasks on Multiprocessor Systems. Real-Time Systems, 1995, 9(3): 207～239

7 Burchard A, Liebeherr J, Oh Y, et al. New Strategies for Assigning Real-Time Tasks to Multiprocessor Systems. IEEE Transactions on Computers, 1995, 44(12)

8 Dhall S K, Liu CL. On a real-time scheduling problem. Operations Research, 1978, 26(1): 127～140

9 Lehoczky J P, Sha L, Ding Y. The Rate-Monotonic Scheduling Algorithm: Exact Characterization and Average Behavior. In: IEEE Real-Time Systems Symposium, 1989. 166～171

10 Oh Y, Son S H. Tight Performance Bounds of Heuristics for a Real-Time Scheduling Problem: [Technical Report]. CS-93-24. Uni-

维普资讯 http://www.cqvip.com

versity Of Virginia. Dept. of Computer Science, May 1993
11 Kuo T W, Mok A K. Load adjustment in adptive real time systems. Proceedings of IEEE Real Time Systems Symposium, Dec. 1991

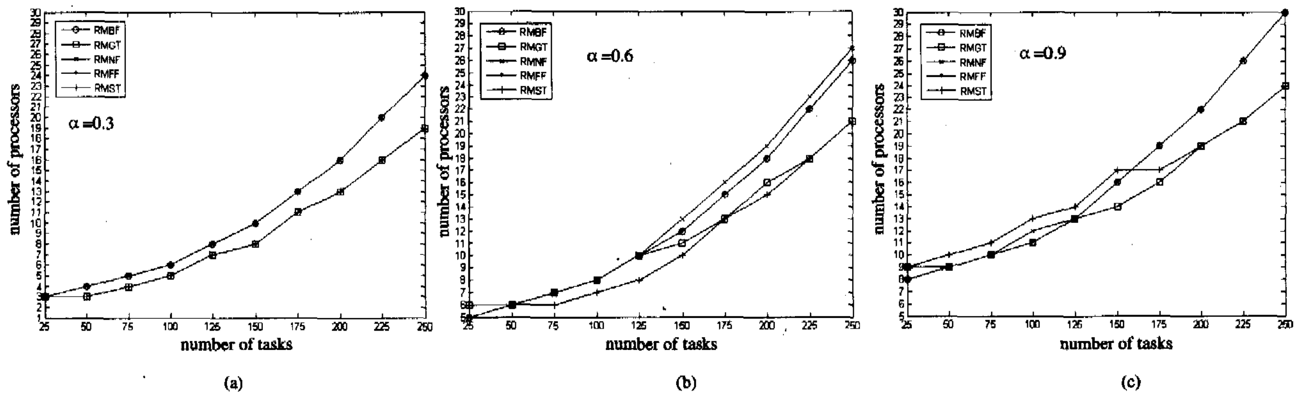

图 3 任务数与处理器数关系曲线

表 2 仿真结果

α	ρ_{RMBF}	ρ_{RMGT}	ρ_{RMNF}	ρ_{RMFF}	ρ_{RMST}
α=0.3	[1.41,1.66]	[1.13,1.47]	[1.41,1.66]	[1.41,1.66]	[1.13,1.47]
α=0.6	[1.40,1.54]	[1.16,1.70]	[1.42,1.54]	[1.40,1.54]	[1.15,1.54]
α=0.9	[1.20,1.41]	[1.13,1.35]	[1.28,1.41]	[1.20,1.41]	[1.13,1.48]

(上接第 263 页)

个处理器必须等待，等待期间通过短循环的不断测试，最终获得锁。对这种方法的改进是使用后退循环，不再不停地访问锁状态，以减少总线竞争，但同样有系统能量的消耗。嵌入式 SMP 系统的内核中设计一对低功耗的锁等待睡眠指令，一旦等待锁，该线程立即进入睡眠，锁释放时立刻唤醒等待该锁的线程。这种旋转锁的设立，减少了总线竞争并明显地降低了系统的消耗。

4.2 Cache 的一致性

一般 SMP OS 能提供 Cache 一致性的映像区，把数据放在 Cache 中能维持数据的稳定。现有维持一致性的方法通常是通过增加信号，扩展系统总线来控制和检测其他处理器的 Cache。在嵌入式系统中，系统总线的时钟通常慢于 CPU，上述方法除了造成 CPU 与 Cache 之间的瓶颈外，还大大增加了总线的流量和功耗。多内核嵌入式处理器系统设计了一个位于多处理器之间的智能控制单元 SCU(Snoop Control Unit)，它主要用于多处理器的数据一致性控制，由它提供快速的数据路径，使数据在各个 CPU 的 Cache 之间高速移动。SCU 保留了每个缓存行(cache line)物理地址标签的副本，并智能地监督处理器对缓存行的操作。如果一个处理器修改了一个缓存行，另一个处理器对该缓存行先读后写，SCU 就假定这个地址今后也会进行同样的操作，如果同样的操作再次发生，SCU 会自动把缓存行的状态置为无效，而不是消耗能量先把它置为共享状态，这就可以在不引发外部内存操作的情况下，让处理器把缓存行直接传输到其他的处理器中，从而降低对有着共同缓存行的处理器进行 Cache 操控的机率。

4.3 多处理器的通信

SMP 多处理器系统用旋转锁来同步处理器之间的通信和对共享资源的访问，避免了通过访问存储器实现通信。异步运行的系统之间必须经常进行同步，使用的一种方法是通过中断系统来激活另一个处理器。这里的中断是一个特殊的中断系统，它通过 I/O 外设而不是 CPU 来触发中断。假设一个多线程应用程序，在某一个处理器上运行的线程会改变处理器的状态，并且这种改变对该应用程序运行在其他处理器上的线程不是硬件一致的。为了维护一致性，操作系统必须把这些对内存映射的修改同步到其他处理器中，首先把新的内存映射配置到自身的处理器中，随后用低竞争的私有外设总线发中断控制信号，中断控制器立刻触发其他的处理器，使它们根据中断 ID 信息，进行各自的内存映射更新。利用这种中断机制，除了可以对应用程序进行动态负载均衡外，也可以对中断处理进行动态负载均衡，即目前负载最轻的处理器最适合处理中断，并把中断发往该处理器。

结束语 嵌入式处理器往往针对应用固有的并行性特征设计微体系结构，这是大幅度提高性能、降低功耗和设计复杂性的有效方法。这几年，像 ARM、MIPS、POWER PC 等主流嵌入式处理器都为此作了很大的努力，并以很快的速度推出新的处理器体系架构，为嵌入式系统满足更强的功能、更复杂的运算、更低的功耗，创造了应用的基础条件。作者在近期的一些嵌入式项目开发中，切身体会到嵌入式处理器结构的发展，特别是并行技术的创新和应用，对更高需求的满足、研发周期的缩短和开发成本的降低所带来的深刻影响[6]，并以此让更多从事嵌入式开发的同仁分享嵌入式处理器发展的成果。

参考文献

1 Seal D. ARM Architecture Reference Manual [M]. 2nd ed. Pearson Education Limited, 2001
2 Sloss A, et al. ARM System Developer's Guide [M]. Mor-gan Kauffman. 2004
3 ARM Data Engines for Complex Multimedia Solutions . http://www.arm.com/.
4 Liu J W S. 姬孟洛，李军，王馨，等译. 实时系统[M]. 北京：高等教育出版社，2003
5 Andrew N. Sloss. 沈建华译. ARM 嵌入式系统开发-软件设计与优化[M]. 北京：北京航空航天大学出版社，2005
6 魏洪兴，周亦敏. 嵌入式系统设计与实例开发实验教材 I [M]. 北京：清华大学出版社，2005