第42卷 第2期 西 安 交 通 大 学 学 报 Vol. 42 №2
2008年2月 JOURNAL OF XI′AN JIAOTONG UNIVERSITY Feb. 2008

基于反馈的片上多处理器系统层次负载平衡算法

王鹏，董渭清，王甜
（西安交通大学计算机科学与技术系，710049，西安）

摘要：针对现有负载平衡算法未能有效利用片上多处理器系统线程级并行性，没有考虑线程间数据共享与通信以及产生颠簸等问题，提出了一种基于反馈的层次负载平衡算法. 采用层次式调度，将属于同一进程的多个线程静态调度到特定规模的内核子集上，并在此基础上根据系统实时负载情况在特定内核子集内动态迁移线程，以降低同一进程的多个线程之间的通信代价. 在线程迁移过程中，引入反馈机制，即以系统颠簸情况为反馈信息，实时调整迁移数目，最终使系统较快地达到平衡. 实验表明，基于反馈的层次负载平衡算法能使系统的平衡程度达到较高的水平，引入的反馈机制可使系统平衡速度提高28%，并在系统平衡前使“颠簸”数目减少54%.

关键词：片上多处理器；多线程；负载平衡；反馈
中图分类号：TP31 **文献标志码**：A **文章编号**：0253-987X(2008)02-0179-05

Hierarchical Load Balance Algorithm Based on Feedback in Chip Multiprocessors System

WANG Peng, DONG Weiqing, WANG Tian
(Department of Computer Science and Technology, Xi′an Jiaotong University, Xi′an 710049, China)

Abstract: Focusing on the problem that existing load balance algorithms failed to make full use of chip multiprocessors (CMP) system's thread level parallelism (TLP) and did not take data sharing, communication, and excessive transformation among threads into account, a hierarchical load balance algorithm based on feedback is proposed. Hierarchical schedule algorithm is adopted in order to reduce the cost of communication. Thus, threads of the same process are assigned into the same set of cores and then migrated into the range of its assigned core set dynamically. Feedback is used during the migration of threads according to the number of excessive transformation that could be computed through information feedback mechanism. The experiment shows that the algorithm is capable of achieving high balancing degree. Moreover, the balancing speed of hierarchical load balance methods with feedback mechanism is increased by 28%, and the excessive transformation is reduced by 54%.

Keywords: chip multiprocessors; multi-thread; load balancing; feedback

片上多处理器(CMP)系统[1-3]是一种紧耦合的系统，可大大提高线程级并行性(TLP)[4]. 与进程相比，线程只需要极少量的系统资源，其迁移的代价也比较小. 因此，相对于任务级并行的多处理机系统等松耦合系统，CMP系统更适用于频繁、灵活的动态调度.

静态负载平衡算法，如适用于超立方体网络的LDLPT (largest dimension largest processing time)算法[5-6]、LDF (largest dimension first)算法[7]等，不适用于细粒度调度，不能充分利用CMP线程级并行性的特点，也无法达到较高的平衡程度，特别是当系统负载发生变化时，不能根据具体情况调整系统负载，致使系统失衡. 动态负载平衡算法，如位交换法(dimension exchange method, DEM)[8]

收稿日期：2007-07-16. 作者简介：王鹏(1983—)，男，硕士生；董渭清(联系人)，男，教授.

和梯度法(gradient based method,GBM)[9]等,虽能根据系统负载情况进行实时调整,但一般采用固定比例的迁移量,需要经多次迭代才能使系统达到平衡,同时还伴随着大量的"颠簸"现象.

本文针对以上问题,提出了适用于CMP的基于反馈的层次负载平衡算法,并对4-超立方体网络的CMP系统进行了验证.

1 层次负载平衡算法的核心思想

调度通常以进程为单位,但是这种粗粒度的调度不够灵活,原因在于进程拥有大量的系统资源,而资源的每一次迁移都伴随着重分配以及大量数据的传输,这给互连网络带来了巨大的压力,并且在迁移后系统不能立即恢复执行状态.

高线程级并行性的CMP系统,是以线程为调度单位进行调度的,因为线程基本上不占用系统资源,只是在系统运行中占用了少量、必不可少的34资源.

但是,同属一个进程的多个线程具有大量的共享数据,如果简单地以线程为动态调度单位将数据调度到系统的任意一个内核上,必然给线程带来高昂的通信代价,也限制了整个系统的加速比.考虑到同属一个进程的多个线程之间存在着密切的关系,应尽量地在把共享数据存储到内核高速缓存之后,将关系密切的线程分配到邻近的内核上,以使线程能快速地访问共享数据.这样,同一进程中的多个线程将被调度到物理距离或逻辑距离较近的几个内核上,从而使得系统的通信代价减小,系统的加速比提高.

基于以上分析知,在以超立方体网络连接的CMP中,负载平衡应采用层次调度法.首先以进程为调度单位,在进程层采用LDLPT算法进行调度,并将同一进程下的多个线程调度到以某个子立方体网络连接的内核子集上,使系统初步平衡,之后在线程层以线程为单位进行调度,通过循环迭代达到线程层负载平衡.每次循环应在各内核与其相邻内核组成的内核子集的内部达成局部负载平衡,由于各内核子集交叉覆盖了整个内核集,所以经过多次迭代,线程将由系统中的重载区域扩散至轻载区域,最终使系统达到全局负载平衡.

2 线程层负载平衡

在实际应用中,有很多任务(如过程控制系统、飞行控制系统等)的正确性不仅依赖于计算的逻辑结果,而且与计算时间有关[10].对于实时性要求比较高的任务,当调度的进程确定分配到某一内核子集的时候,该进程应立即执行,而LDLPT算法则不然,它需要等待正在执行的任务结束后才开始执行新的任务.假设系统采用LDLPT算法已经完成了进程层调度,并将属于同一个进程的所有线程分配到由某一个子立方体网络连接的限定的内核子集上,各进程同时开始执行,那么再根据系统实时负载情况在限定的内核子集上以线程为单位进行调度,致使整个系统达到负载平衡.

2.1 基本定义

为了衡量系统平衡程度,首先引入系统平衡函数的概念.

记基于m维超立方体网络的CMP中第i个内核C_i的负载线程数为a_i,每个内核平均负载线程数为

$$\bar{a}=\sum_{i=1}^{2^m}a_i/2^m$$

则内核C_i的全局绝对失衡数为

$$a'_i=|a_i-\bar{a}|$$

定义1 用平衡函数

$$I_m=\sum_{i=1}^{2^m}a'_i/\sum_{i=1}^{2^m}a_i \tag{1}$$

来衡量系统平衡程度,I_m小表明系统平衡,相反则表明系统失衡,显然$0\leqslant I_m\leqslant 1$.

系统的负载平衡问题可以通过调整每个内核的负载线程数a_i,即降低I_m来解决.

在m维的超立方体网络上,每个内核均与m个内核直接相连,并时刻保持其与m个邻内核的负载信息,且定期交换线程个数以及处理内核资源的属性.如果某一内核资源用量超过了限定阈值,这时需将部分线程迁出,而当某一内核的负载超过自身以及四邻区域负载的平均值,将视为过载.此时,在就绪的队列中选取k条线程迁往m个邻内核中负载最低者.

记某一内核与m个邻内核组成一个内核子集为S,该内核就绪线程数为b_0,其m个邻核的就绪线程数依次为$b_1,b_2,\cdots,b_m$.

定义2 记内核在S的负载失衡数为

$$\Delta b_i=b_i-\bar{b}=b_i-\frac{\sum_{i=0}^{m}b_i}{m+1} \tag{2}$$

式中:$\bar{b}$为内核子集S的平均就绪线程数.

如果$\Delta b_i>0$,内核C_i在S中属于重载,应从其

负载中选取 k_i 条线程迁往 m 个邻内核中的负载最轻者，则

$$k_i = L\Delta b_i \tag{3}$$

式中：L 为迁移系数，$0 \leqslant L \leqslant 1$.

选用“二分之一法”，令 $L \equiv 1/2$，即每次从重载内核 C_i 可选择

$$k_i = \frac{b_i - (\sum_{i=0}^{m} b_i)/(m+1)}{2} \tag{4}$$

条线程迁往 m 个邻内核中的负载最轻者.

2.2　线程层负载平衡算法

若进程 P_i 所属的 n 个线程 $T_i = \{T_{i1}, T_{i2}, \cdots, T_{in}\}$ 分配到 d_i 维子立方体，则线程迁移只能限制在该子立方体所连接的内核子集之中，此时应为每个线程标注内核集标记符，限制其迁移范围.

内核 C_i 的线程层负载平衡执行算法如图 1 所示；该算法以式(1)、式(4)以及内核集标识符为基础. H_I 为系统平衡阈值，可直接反映系统负载的平衡程度，若 I_m 小于该阈值，则认为系统处于负载平衡状态. 当 H_I 较大，线程只需少量迁移，此时 I_m 值较大(认为系统平衡)；当 H_I 较小，线程需进行多次迁移，此时 I_m 值较小. H_B 为内核失衡阈值，可反映局部平衡程度，若该值设置得较高，那么只有当某一内核相对其 m 个邻内核超载较为严重时才迁出线程，此时系统的平衡程度较低，反之亦然.

当内核 C_i 的失衡数 $\Delta b_i > H_B$ 时，可认为该内核重载，应适当迁出线程.

经过多次循环迭代，系统负载从重载区域向轻载区域扩散，最终达到负载平衡. 在此过程中，迭代次数能直接反应算法的性能.

3　基于反馈的加速方法

一般，层次负载平衡算法的性能要优于单一负载平衡算法，但采用静态迁移量法有一定缺陷，即如果系统出现“颠簸”现象，若仍将 $k_i \equiv \Delta b_i/2$ 条线程迁出，结果会加剧“颠簸”. 如果系统严重失衡，在无“颠簸”现象的情况下，每次需要迁移大量的线程，若仍将 $k_i \equiv \Delta b_i/2$ 条线程迁出，会限制系统平衡的速度. 因此，根据系统实时情况，动态调整迁移量 k_i 是非常必要的. 若系统没有发生“颠簸”现象，可适当增大每次迁移的负载量；若系统发生“颠簸”现象，应适当减小每次迁移的负载量.

3.1　反馈信息的采集

线程若在某一内核与其相邻内核之间来回迁移，则说明发生了无助于系统平衡的迁移，且一定出现了“颠簸”现象. 为了检测“颠簸”现象，应记录内核与其邻内核之间的线程迁移情况，如果前后 2 次 2 个内核之间的线程迁移方向相反，则说明出现了“颠簸”现象.

```
初始状态：经进程层调度后的负载分配.
结束状态：系统全局负载平衡.
方法：
1) while(1) {
2)     根据式(1)计算I_m;
3)   if(I_m<H_I)
4)     break;
5)   根据式(2)计算 Δb_i;
6)   if(Δb_i>H_B) {
7)     根据式(4)计算k_i;
8)     令C_n为C_i的m个邻内核中负载最轻者;
9)  A:利用线程内核集标识符在C_i负载线程中查询所有可以
      迁往C_n的线程集T, 记线程条数为K, 并标识迁往的C_n;
10)  if(K≥k_i) {
11)    从T 中选择k_i条线程记入迁移线程缓冲集;
12)  }
13)  else if(0<K<k_i){
14)    从T 中选择K条线程记入迁移线程缓冲集，并标识迁
       往的C_n;
15)      k_i=k_i-K;
16)    令C_n为C_i的m个邻负载次轻内核;
17)    goto A;
18)  }
19) }
20) 等待其他内核完成以上工作;
21) 将迁移线程缓冲集中的线程迁往目的内核;
22) }
```

图 1　内核 C_i 线程层负载平衡执行算法

为此，在基于 m 维超立方体网络的 CMP 中，为每个内核增设了一个 $2m$ 位移位寄存器 R，前 m 位用来记录内核与 m 个邻内核之间的上一次的迁移信息，后 m 位记录内核与 m 个邻内核之间的本次的迁移信息. 如果线程从相邻内核迁入本内核，则对应位记为“1”，否则记为“0”，开机时初始化为高阻“Z”. 在 4 维超立方体网络的 CMP 中，某一内核 C_i 与 4 个邻内核之间的迁移信息如表 1 所示.

表 1　某一内核与 4 个邻内核之间的迁移信息

邻核	1#	2#	3#	4#
上次迁移信息	迁出	迁入	迁入	迁出
本次迁移信息	迁出	迁入	迁出	迁入

将 $R_1 \sim R_4$ 位分别与 $R_5 \sim R_8$ 位进行异或运算，可生成 4 位结果 X. 如果前后 2 次迁移方向相反，即 $R(j)$ 与 $R(j+4)$ 不同，其异或结果为“1”则表示某一

内核与相邻内核之间发生颠簸现象.将 X 的各位进行或运算,将结果存入 16 位寄存器 Y 中的对应位,如此以来,Y 中第 i 位可表明 C_i 是否发生“颠簸”现象.

依据表 1,反馈信息的采集系统及其采集过程如图 2 所示.

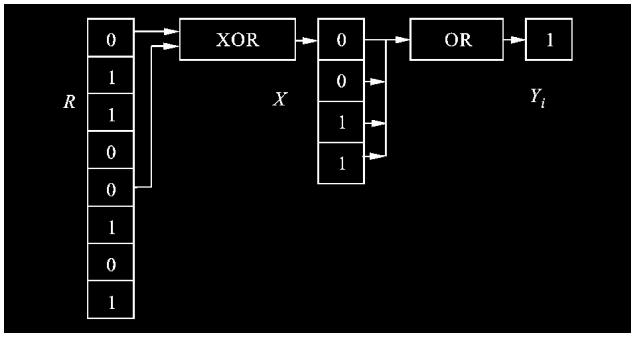

图 2 反馈信息采集系统

如果 $Y_i=1$,则表明在上一次的迁移中,该内核发生了“颠簸”现象,否则系统正常.

3.2 迁移系数调整

为了能动态地调整迁移负载量 k_i,应为每个内核引入迁移系数 L_i,并且令

$$k_i = L_i\left(b_i - \frac{\sum_{i=0}^{m} b_i}{m+1}\right) \tag{5}$$

式中:L_i 为第 i 个内核的迁移系数($0 \leqslant L_i \leqslant 1$).当 $L_i \equiv \frac{1}{2}$,该方法则退化为“二分之一”法.

在初始状态下,令 $L_i=\frac{1}{2}$,在执行过程中算法则可根据反馈信息来调整 L_i,从而调整了 k_i.如果 $Y_i=1$,则 L_i 减小 0.1,否则 L_i 增大 0.1.基于此,迁移的调整方法如下.

(1)根据本次迁移的情况,对 $R(m+1)$ 至 $R(2m)$ 赋值.

(2)组合逻辑电路快速生成 Y_i,并调整 L_i.

(3)R 左移 m 位,并为下一次迁移提供本次的迁移信息.

4 验证与分析

通过实验在 CMP 系统上验证了所提算法及静态迁移量法的负载平衡情况,并对结果进行了分析.实验中,CMP 各内核采用 4 维超立方体网络的连接方式,进程集是 32 个生命周期相同的进程,其详细信息如表 2 所示.

经进程层调度之后,平衡函数 $I_m=0.41$,此时

表 2 实验进程集

进程	P_1	$P_2 \sim P_3$	$P_4 \sim P_5$
线程数	260	65	35
进程	P_6	$P_7 \sim P_{15}$	$P_1 \sim P_{32}$
线程数	17	2	1

系统并未达到较高的平衡程度,在此基础上应该进行线程层调度.图 3 描绘了采用静态迁移量法和基于反馈的动态调整迁移量法的层次负载平衡的执行情况.

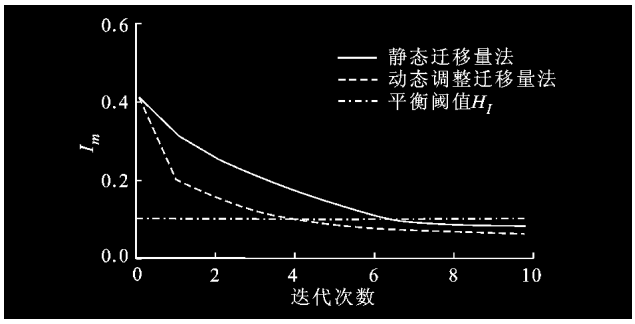

图 3 系统平衡函数随迭代次数的变化

设 $H_I=0.1$,即 $I_m=0.1$ 时,认为系统处于基本负载平衡状态.

利用静态迁移量法需要进行 7 次迭代可使 I_m 降到 0.09,以达到 H_I 的要求.如果系统在未达到平衡时,利用了动态调整迁移量法,它会根据反馈信息确定未发生“颠簸”现象的内核,这些内核若仍需迁出线程,则增大其相对迁移比例,由此大大加快了迁移速度.实验结果也显示出,采用动态调整迁移量法只需进行 5 次迭代便使 I_m 降至 0.08,以达到 H_I 的要求,显然比前者加速了 28.6%.

采用动态调整迁移量、静态迁移量这 2 种方法的系统颠簸变化如图 4 所示.从图中看出,若有足够的反馈信息,动态调整迁移量法的性能比较好.相反,当系统刚开始运行而不能提供足够的反馈信息时,动态迁移量法的优越性不太明显.假设 $I_m=0.1$ 时系统基本平衡,则静态迁移量法在系统平衡前的 7 次迭代中共发生颠簸 22 次,而动态调整迁移量法在系统未达到平衡时,根据反馈信息可以确定发生“颠簸”现象的内核.这些内核如果仍需迁出线程,则减小其相对迁移比例,以防止颠簸现象再次发生.实验表明,动态调整迁移量法在使系统平衡程度满足 H_I 时,前 5 次迭代中共产生 10 次颠簸,比静态迁移量法减少了 54.5%.

若实验中采用静态负载平衡算法,其 I_m 为 0.41,这个结果只能说明系统的平衡程度较低,而采用层次负载平衡算法,I_m 可降至 0.1,这说明系统的平衡程度大大提高.

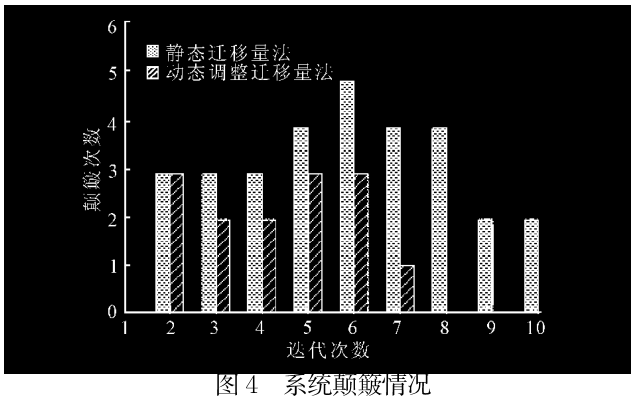

图 4　系统颠簸情况

5　结束语

本文针对现有负载平衡算法不能充分适应 CMP 线程级并行性的问题,提出了基于反馈动态调整迁移量的层次调度算法.实验表明,采用所提算法的 CMP,在执行多线程任务时可使系统快速达到负载平衡状态.相对于以往固定迁移比例的方法,动态调整迁移量法可显著提高系统平衡速度,同时有效控制了系统"颠簸"现象的发生.本文的平衡函数 I_m 随着平衡次数的增加而下降,虽在开始阶段很快下降,但是伴随着 I_m 的减小,其下降速度明显减缓.

参考文献:

[1] PARKHURST J, DARRINGER J, GRUNDMANN B. From single core to multi-core: preparing for a new exponential [C] // Proceedings of the ACM Great Lakes Symposium on VLSI. New York: Association for Computing Machinery, 2006:210-216.

[2] TENDLER J, DODSON S, FIELDS S, et al. Sinharoy POWER4 system microarchitecture [J]. IBM Journal of Research and Development, 2002, 46(1):5-26.

[3] HOFSTEE P. Power efficient processor architecture and the cell processor [C]// Proceedings of 11th International Symposium on High-Performance Computer Architecture. Los Alamitos, USA: IEEE Computer Society, 2005: 258-262.

[4] SPRACKLEN L, ABRAHAM S G. Chip multithreading: opportunities and challenges [C]// Proceedings of the 11th International of Symposium on High-Performance Computer Architecture. Los Alamitos, USA: IEEE Computer Society, 2005:248-52.

[5] CHEN Guan′ing, LAI Ten-Hwang. Scheduling independent jobs on partitionable hypercube [J]. Parallel and Distributed Computing, 1991,12(1):74-78.

[6] 肖建华,陈建二,陈松乔.超立方体网络中任务调度的一个新近似算法 [J].小型微型计算机系统, 2001, 22(8):913-916.
XIAO Jianhua, CHEN Jian′er, CHEN Songqiao. New algorithm for jobs scheduling in hypercube network [J]. Mini-Micro Systems,2001, 22(8):913-916.

[7] ZHU Yahui, MOHAN A. Job scheduling on a hypercube [J]. IEEE Trans on Parallel and Distributed System, 1993,4(1):62-69.

[8] LIN F C H, KELLER R M. The gradient model load distribution methods [J]. IEEE Trans on Software Engineering, 1987,13 (1):32-38.

[9] CYBENKO G. Dynamic load balancing for distributed memory multiprocessors [J]. Journal of Parallel and Distributed Computing, 1989,7(2):279-301.

[10] MANIMARAN G, MURTHY C S R. An efficient dynamic scheduling algorithm for multiprocessor real-time systems [J]. IEEE Trans on Parallel and Distributed Systems, 1998,9(3):312-319

(编辑　苗凌)