软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2014,25(7):1403−1415 [doi: 10.13328/j.cnki.jos.004600]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种结合页分配和组调度的内存功耗优化方法*

贾刚勇[1,2], 万 健[1,2], 李 曦[3], 蒋从锋[1,2], 代 栋[4]

[1](杭州电子科技大学 计算机科学与技术系,浙江 杭州 310018)

[2](复杂系统建模与仿真教育部重点实验室(杭州电子科技大学),浙江 杭州 310018)

[3](中国科学技术大学 计算机科学与技术系,安徽 合肥 230027)

[4](Department of Computer Science and Technology, Texas Tech University, Lubbock TX 79409)

通讯作者: 贾刚勇, E-mail: gangyong@mail.ustc.edu.cn

摘 要: 多核系统中,内存子系统消耗大量的能耗并且比例还会继续增大.因此,解决内存的功耗问题成为系统功耗优化的关键.根据线程的内存地址空间和负载均衡策略将系统中的线程划分成不同的线程组,根据线程所属的组,给同一组内的线程分配相同内存rank中的物理页,然后,根据划分的线程组以组为单位进行调度.提出了结合页分配和组调度的内存功耗优化方法(CAS).CAS 周期性地激活当前需要的内存 rank,从而可以将暂时不使用的内存 rank 置为低功耗状态,同时延长低功耗内存 rank 的空闲时间.仿真实验结果显示:与其他同类方法相比,CAS 方法能够平均降低 10%的内存功耗,同时提高 8%的性能.

关键词: 内存;页分配;线程组调度;功耗效率;性能

中图法分类号: TP316

中文引用格式: 贾刚勇,万健,李曦,蒋从锋,代栋.一种结合页分配和组调度的内存功耗优化方法.软件学报,2014,25(7):1403−1415. http://www.jos.org.cn/1000-9825/4600.htm

英文引用格式: Jia GY, Wan J, Li X, Jiang CF, Dai D. Memory power optimization policy of coordinating page allocation and group scheduling. Ruan Jian Xue Bao/Journal of Software, 2014,25(7):1403−1415 (in Chinese). http://www.jos.org.cn/1000-9825/4600.htm

Memory Power Optimization Policy of Coordinating Page Allocation and Group Scheduling

JIA Gang-Yong[1,2], WAN Jian[1,2], LI Xi[3], JIANG Cong-Feng[1,2], DAI Dong[4]

[1](Department of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou 310018, China)

[2](Key Laboratory of Complex Systems Modeling and Simulation of Ministry of Education (Hangzhou Dianzi University), Hangzhou 310018, China)

[3](Department of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China)

[4](Department of Computer Science and Technology, Texas Tech University, Lubbock TX 79409)

Corresponding author: JIA Gang-Yong, E-mail: gangyong@mail.ustc.edu.cn

Abstract: Main memory accounts for a large and increasing fraction of the energy consumption in multi-core systems. Therefore, it is critical to address the power issue in the memory subsystem. This paper presents a solution to improve memory power efficiency through coordinating page allocation and thread group scheduling (CAS). Under the proposed page allocation, all threads are partitioned into different thread groups, where threads in the same thread group occupy the same memory rank. Thread group scheduling is then implemented by adjusting default Linux CFS. The CAS alternates active partial memory periodically to allow others power down and prolongs the idle ranks. Experimental results show that this approach improves energy saving by 10% and reduces performance overhead by 8% comparing with the state of the art polices.

* 基金项目: 国家自然科学基金(61272131, 61003077, 61100193); 江苏省产学研前瞻性联合研究项目(BY2009128)
收稿时间: 2013-12-16; 定稿时间: 2014-03-17

Key words: main memory; page allocation; thread group scheduling; power efficiency; performance

以往,处理器被认为是计算机系统中最耗能的部件.然而,随着越来越多的处理器能耗优化方法的提出和应用,处理器的功耗问题逐步得以缓解.相反地,由于多核系统对内存带宽和容量的需求不断加大,内存的功耗逐渐增加[1−3].并且,目前使用的(DDR*)内存工艺对内存上的功耗管理提出了更加严峻的挑战.目前,内存的功耗已经占据计算机整体功耗的 40%[1],这一比例已接近甚至超过了处理器功耗消耗的比例.如何在保证性能的前提下降低内存的功耗、提高功耗效率,是当今的一个研究热点.

当前,已有大量的研究工作关注内存上的功耗优化问题,研究者们分别通过减少每次访问内存时被激活的内存片数(也称为 rank subsetting)[4,5]或者改变内存的体系结构,从而提高内存的功耗效率[6].这些方法都是通过减少每次内存访问时内存上被激活的物理单元的数量,从而降低内存的动态功耗.Rank subsetting 需要改变内存双列直插式存储模块(DIMM)的物理结构,这种改变需要付出高昂的代价,并且会增加访存延时.改变内存的物理结构,则可能对物理内存的容量和产量产生负面影响[7].

同时,也有一些工作提出功耗感知的操作系统页分配策略[8]或者内存功耗感知的操作系统调度策略[8−12],这些方法都是通过挖掘内存的空闲时间来降低内存的功耗消耗,但是这些方法在不严重影响性能的前提下,不能挖掘足够的空闲时间.

本文提出了将操作系统页分配和线程组调度相结合的内存功耗优化策略.根据共享内存地址空间和负载均衡,将系统中所有的线程划分成线程组;修改操作系统的页分配策略,给相同线程组中的线程分配相同内存 rank 中的物理页(内存 rank 是内存功耗管理的最小单元);优化操作系统的调度方法,实现同一线程组内的线程优先调度.同一线程组中的线程先后执行的过程中只有一个内存 rank 被访问,其他内存 rank 处于空闲状态,从而可以将这些内存 rank 调成低功耗模式.

与已有的研究工作相比,本文的创新之处在于:

1) 通过将页分配和线程组调度相结合,延长了内存空闲状态的时间,从而实现了在没有严重影响性能的前提下,挖掘出更多的空闲时间;
2) 基于共享内存地址空间划分线程组,降低了同一线程组中线程切换的开销;
3) 根据线程组管理内存 rank 的功耗状态,减少内存 rank 状态切换的频率,从而降低了状态切换开销.

实验结果表明:本文提出的 CAS(coordinating page allocation and thread group scheduling)比目前主流的几种方法在内存功耗上平均降低了 10%,同时提高了 8%的性能.

本文第 1 节介绍内存的组织结构并回顾与本文相关的研究工作.第 2 节介绍本文提出的结合页分配和线程组调度的内存功耗优化策略.第 3 节建立模拟实验平台.第 4 节通过大量的模拟实验测试 CAS 策略的功耗和性能.第 5 节总结全文并给出下一步的研究工作.

1 内存组织结构和相关研究

1.1 内存组织结构

本节我们简单地对现有内存的组织结构以及操作系统对内存的管理机制加以介绍.

- 内存的组织结构

目前,主流的内存系统一般包含 1~2 个内存通道(channel),每个内存通道包含 1~2 个双列直插式存储模块(DIMM),每个双列直插式存储模块由 1~2 个内存 rank 组成,每个内存 rank 可以划分成 8 个内存 bank.内存 rank 是内存功耗管理的最小单元.内存 bank 可以并行地访问,因此,不同内存 bank 上的请求可以同时被访问[13],提高了内存的性能.图 1 给出了一个当前内存子系统的组织结构图.

内存可以处于 4 种不同的状态,根据消耗功耗从高到低分别是:active standby,precharge standby,active power-down 和 precharge power-down[9].

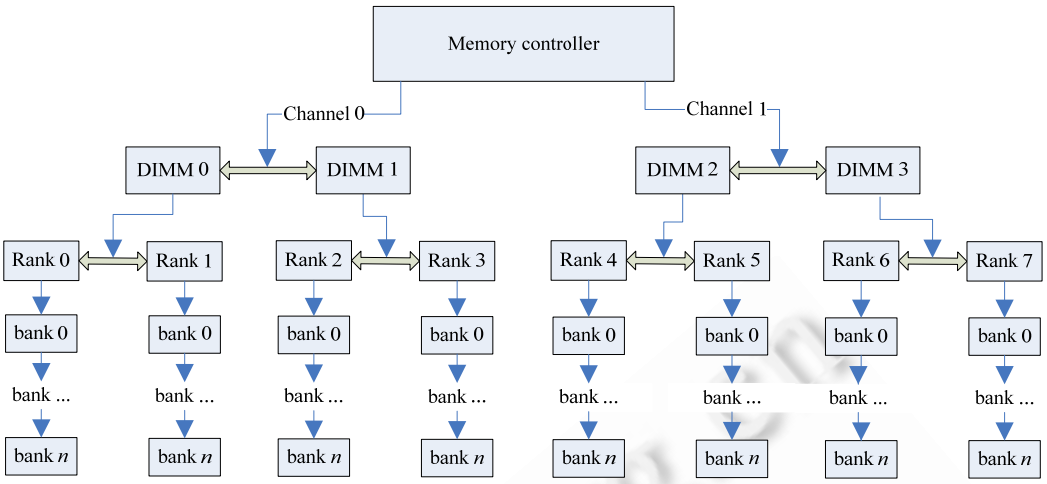


Fig.1 Memory architecture of current subsystem

图 1 当前内存子系统的组织结构

- 操作系统对内存的管理机制

目前,Linux 操作系统使用伙伴算法管理物理内存.Linux 的伙伴算法把所有的空闲页面分为 10 个块组,每组中块的大小是 2 的幂次方个页面.例如,第 0 组中块的大小都为 20(1 个页面),第 1 组中块的大小都为 21(2 个页面),第 9 组中块的大小都为 29(512 个页面).也就是说,每一组中块的大小是相同的,且这些同样大小的块形成一个链表.

我们通过一个简单的例子来说明该算法的工作原理:假设要求分配的块其大小为 128 个页面(由多个页面组成的块叫做页面块).该算法先在块大小为 128 个页面的链表中查找,看是否有这样一个空闲块.如果有,就直接分配;如果没有,该算法会查找下一个更大的块.具体地,就是在块大小为 256 个页面的链表中查找一个空闲块.如果存在这样的空闲块,内核就把这 256 个页面分为两等份,一份分配出去,另一份插入到块大小为 128 个页面的链表中.如果在块大小为 256 个页面的链表中也没有找到空闲页块,就继续找更大的块,即,512 个页面的块.如果存在这样的块,内核就从 512 个页面的块中分出 128 个页面满足请求,然后从 384 个页面中取出 256 个页面插入到块大小为 256 个页面的链表中.然后,把剩余的 128 个页面插入到块大小为 128 个页面的链表中.如果 512 个页面的链表中还没有空闲块,该算法就放弃分配,并发出出错信号.

1.2 相关研究

近年来,国内外学者在内存功耗效率优化方面取得了一定的研究成果,这些研究从不同的角度降低了内存的功耗.

- 操作系统的页管理策略

Lebeck 等人[8]较早研究通过结合操作系统页分配、页迁移策略和内存状态控制实现内存功耗优化的工作,在该工作中定义了两种页管理策略:一种是 the sequential first-touch policy,这种策略是按顺序分配 rank/chip,虽然实验效果比较理想,但该策略未考虑内存碎片以及内存利用率等问题,所以实际系统中无法使用;另一种则是 the frequency policy,这是一种页迁移策略,通过识别经常访问的页,然后将频繁被访问的页聚集,促使内存访问相对集中,可以延长其他内存的空闲时间,但是页迁移的开销太大.Ding 等人[14]也提出了类似于 frequency policy 的页迁移策略,通过减小操作系统中页的大小,充分利用访存局部性,识别出热页后,将其迁移到固定的内存区域,降低其他内存区域的功耗.然而同样因为开销过大,效果有限.

- 操作系统的调度策略

文献[9]提出的操作系统调度策略与本文提出的调度策略比较相似,也是将线程划分后,根据线程组进行调

度,但是他们在划分线程组的过程中没有考虑线程组内线程的特征,即,没有考虑共享内存地址空间,而且没有结合页分配策略,而是使用代价很大的页迁移策略,所以对性能的影响很大,使得功耗效率的提升受到很大的限制.Karlsruhe 大学 Merkel 等人[15]在多核平台根据任务的特征选择内存干扰最小的一组任务并行执行,从而可以提高共享内存的功耗效率.贾刚勇等人[10−12]提出访存亲和性感知的操作系统调度以实现内存功耗的优化.

- 内存通道的划分

根据不同线程的访存行为分配不同的内存通道,可以减少不同线程间使用内存通道的干扰,提高内存功耗效率[16].但是内存通道的划分方法对 cache 的策略有一定的要求,所以在一定程度上限制了这种方法的使用范围.而且因为系统中线程数一般比内存通道数要更多,所以一部分线程必须被分配到同一内存通道中,共享通道的线程间同样会相互干扰,影响效果.

- 内存控制器的调度

根据线程级访存亲和性的不同特征调整内存控制器的调度策略,实现内存功耗效率的优化[17,18].TCM[17]根据线程访存行为特征将线程动态地划分成访存密集型和访存稀疏型两种线程组,给每个线程组分配不同的调度策略,同时解决公平性和吞吐量的问题,提高了内存的功耗效率.但是 TCM 需要修改内存控制器,而且运行时的分组、调度开销比较大.

- Cache 划分

通过软件将共享 cache 划分按需分配给并行执行的线程,能够减少多线程之间相互的干扰,降低 cache 冲突,从而提高 cache 命中率,减少内存访问,提高内存功耗效率[12,19].然而其他共享资源,如内存总线、内存等,都面临着相互竞争、相互干扰的问题,所以需要共同解决.

- 内存状态控制

通过调节内存的状态降低内存的功耗.Delaluz 等人[20]通过挖掘任务对内存访问的特征,根据是否需要使用内存,实时地调节内存的状态:不需要使用内存时,将内存状态调节为低功耗状态,需要使用时,调节为活动状态,从而在保证性能的前提下降低内存功耗.David 等人[21]则将动态电压频率调节手段加入到内存中,使内存也可以实现实时的动态电压频率调节,提高内存的功耗效率,但这需要硬件的更改和开销.

2 内存功耗效率优化方法

本文提出的内存功耗效率优化方法主要是通过结合操作系统的页分配策略和线程组调度来实现.图 1 给出了本文基于的内存组织结构,2 个内存通道,每个通道包含 4 个内存 rank,每个内存 rank 包含 8 个内存 bank.本文分别将所有的线程和内存 rank 划分成 4 个组,其中,将所有的内核线程划分成一个单独的线程组,所以其他所有线程划分成 3 个线程组.同时,将位于不同内存通道上的两个内存 rank 归并成一个内存 rank 组.同一个线程组中的线程优先并发调度,不同线程组的线程周期性地循环调度.任何一个线程组的线程执行时,只会访问两个内存 rank 组,也就是说,只需将这两个内存 rank 组置为工作状态:一个用于内核线程组,一个用于当前执行的用户线程组,其他的内存 rank 组可以置为低功耗状态,从而可以降低功耗.

不同的内存组织结构影响着内存系统的功耗和性能.如果将内存划分成更多的内存 rank 组,那么每个组包含更少的 rank 和 channel 数,也就意味着更多的内存 rank 可以置为低功耗状态,降低更多功耗.但同时也意味着实时可用的带宽更少.

2.1 线程组划分

我们将内存划分成 4 个内存 rank 组,为了将线程组和内存 rank 组一一对应,同样地也将系统中所有的线程基于共享内存地址空间和负载均衡策略划分成 4 个线程组.

首先,将共享内存地址空间的所有线程划分到同一线程组中.操作系统调度时,共享内存地址空间的两个线程之间切换可以避免 TLB 和 cache 的替换,从而可以提高性能.尤其需要指出的是,我们将系统所有的内核线程划分为一个单独的线程组,同时,这个内核线程组占据 2 个内存 rank,并且这两个内存 rank 一直处于工作状态.因为内核线程随机频繁地被当做服务例程调度执行,所以我们将内核线程组所对应的内存 rank 一直处于工作

状态,减少频繁的 rank 状态切换,降低切换带来的代价,提高性能.同时,基于负载均衡策略有利于各组线程的负载实现均衡化,避免线程组间负载差异过大而导致某些线程组没有任务执行而其他线程组负载很重,从而导致性能和功耗效率的下降.

其次,由于根据单一的共享内存地址空间策略划分的线程组一般远远多于 3 个,根据负载均衡策略,我们将所有的这些组进一步划分成 3 个线程组.

于是,系统中所有的线程被划分成了 4 个线程组,其中包含 1 个内核线程组和 3 个非内核线程组.算法 1 给出了本文提出的基于共享内存地址空间和负载均衡的划分策略.$T_1,T_2,\ldots,T_n$ 表示系统中所有的线程,G_1,G_2,G_3,G_4 代表线程划分后最终的 4 个线程组,$TP_1,TP_2,\ldots,TP_m$ 为通过共享内存地址空间策略划分后所有非内核线程临时的线程组.

算法 1. 基于共享内存地址空间和负载均衡策略.

输入:系统中所有线程 $T_1,T_2,\ldots,T_n$;

输出:G_1,G_2,G_3,G_4.

```
FOR (i=1,k=1;i<=n;i++){
  IF (Ti 属于内核线程)
    Ti∈G1;   //将所有内核线程划分到内核线程组 G1
  ELSE {
    FOR (j=i−1;j>0;j−−){
      IF (Ti 与 Tj 共享内存地址空间) {
        FOR (t=1;t<k;t++)
          IF (Tj∈TPt)
            Break;   //查找 Tj 所属的线程组 TPt
          Ti∈TPt;   //将共享内存地址空间的线程划分到同一个线程组中
      }
      ELSE
        Ti∈TPk++;
    }
  }
}  //根据共享内存地址空间将非内核线程划分成 TP1,TP2,…,TPk−1
FOR (i=1,N2=0,N3=0,N4=0;i<k;i++){
  //N2,N3,N4 代表线程组 G2,G3,G4 的线程个数
  IF (N2<=N3 && N2<=N4){
    TPi∈G2;
    N2=N2+TPi 线程个数;
  }
  ELSE IF (N3<N2 && N3<=N4){
    TPi∈G3;
    N3=N3+TPi 线程个数;
  }
  ELSE {
    TPi∈G4;
    N4=N4+TPi 线程个数;
  }
```

} //根据负载均衡策略将非内核线程归并到线程组 G_2,G_3,G_4

图 2 则给出了一个新的线程被创建后加入到一个线程组内的过程:当创建一个线程时,首先查看新创建的线程是否存在与其共享内存地址空间的线程,如果存在,则查找所属的线程组;否则,根据负载均衡选择一个线程组.将新创建的线程加入到所选择的线程组中.

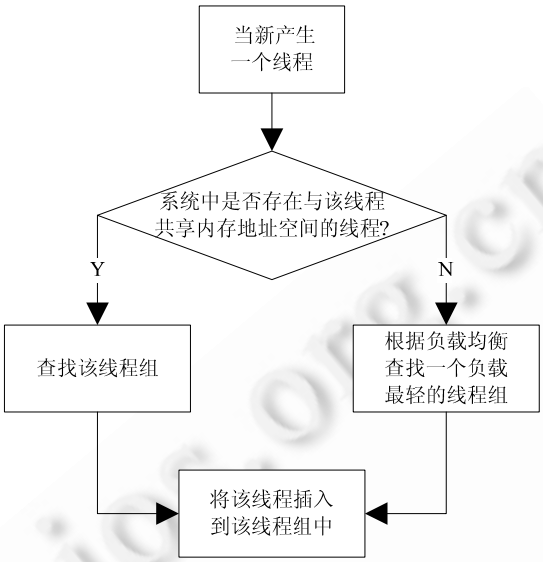


Fig.2 Partition process after a new thread creating

图 2 一个新产生的线程被划分到一个线程组中的过程图

2.2 组感知的操作系统页分配策略

在 Linux 操作系统中,使用经典的伙伴算法来实现页分配,伙伴算法已在第 1.1 节中给出了简单的介绍.根据伙伴算法的原理,一个线程占用的物理内存页可能遍布整个内存的所有 rank 中.这样有利用提高访存的并行性,从而提高访存的性能.但是研究发现:一个线程为了提高并行性,所需的内存 bank 数是有限的,一般达到 16 个内存 bank 数就足够了,更多的内存 bank 数对提高并行性效果并不明显[22].所以在保证性能的前提下,为了最大程度地降低内存功耗,可以将一个线程所占的内存页聚集在几个 rank 内.

因此,我们提出了组感知的操作系统页分配策略,用于最大程度地提高内存功耗效率.基于内存功耗的最小管理单位 rank,我们提出的组感知的页分配策略根据线程所属的线程组来分配物理页,相同线程组内的线程占用同一个内存 rank 组.所以,一个线程所占用的物理页都聚集在两个内存 rank 的所有 bank 内,从而可以在提高功耗效率的同时避免性能的严重下降.

图 3 给出了组感知页分配策略的物理页组织结构,从图中可以发现:我们提出的组织结构与 Linux 操作系统默认组织结构的主要区别在于增加了内存 rank 组的信息,由每个 rank 组来组织各自的物理内存页.内存按照 rank 组被划分成了 4 份,每份都独立地按照 Linux 默认方法组织.每次请求分配一个物理内存块,首先必须确定请求来源于哪个组,再从相应的 rank 组根据默认的伙伴算法分配空闲物理内存块.

算法 2 描述了我们提出的组感知页分配策略.与默认伙伴算法的主要差异在于:我们根据线程所在的组聚集其请求的内存块在其组相应的 rank 组内,而不是扩散于所有的 rank 中.因此,组感知的页分配策略可以延长空闲 rank 的时间,同时保证足够的并行性.

算法 2. 组感知的页分配策略.

输入:线程 T_1;

输出:物理内存块 M.

```
FOR (i=1;i<5;i++)
  IF (T1∈Gi)
    RETURN;   //T1 属于线程组 Gi
在内存 rank 组 i 中查找所需空闲物理块 M;
    将物理块 M 分配给线程 T1;
```

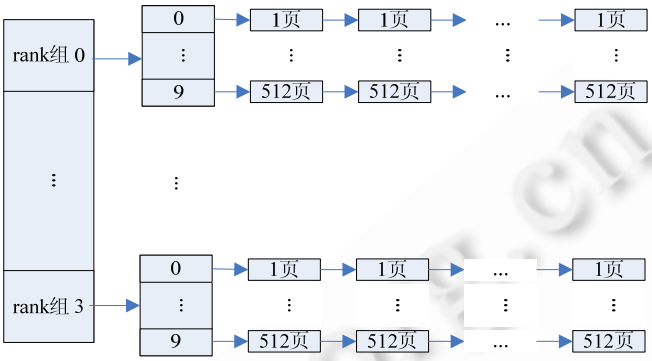


Fig.3 Memory page architecture of the group aware page allocation

图 3 组感知页分配策略的物理页组织结构

2.3 线程调度

我们将系统中所有的线程划分成 4 个线程组,每个线程组组织成红黑树的结构.每个线程组运行于一个处理器核组上,多个核共享一个线程组.每个核采用默认的完全公平性调度策略选择线程组中的线程调度执行.一个线程组和该线程组运行的一个处理器核组以及占据的一个内存组三者形成一个独立的子系统,如图 4 所示.整个系统存在 4 个子系统,子系统间相互独立,互不干扰.在 4 核平台下,每个子系统包含一个核;而在 8 核平台下,每个子系统则包含 2 个核.

我们知道,目前 Linux 操作系统使用完全公平调度(CFS)策略,该策略将系统中线程根据各线程的 *vruntime* 组织成一个红黑树(rb-tree)结构的调度队列.每次调度时,选择红黑树最左下角的线程执行,执行完后根据线程新的 *vruntime* 将线程插回红黑树中.本文提出的线程组调度是基于 CFS 调度策略,结合线程组信息.

当所有的线程划分成线程组后,根据线程组重新排列调度队列,如图 4 所示.属于一个线程组内的所有线程根据各自的 *vruntime* 按照 CFS 调度队列的排列规则组织成一棵红黑树,如图 4 右下角所示.4 个线程组则根据各组的 *vruntime* 组织成一棵红黑树.本文定义各组的 *vruntime* 为组内所有线程 *vruntime* 之和:

$$G_{vruntime}=\sum_{i=1}^{n} vruntime_i \tag{1}$$

其中,$G_{vruntime}$ 和 $vruntime_i$ 分别表示线程组的 *vruntime* 和组内线程 *i* 的 *vruntime*.所以,我们的 CAS 调度队列形成两个层次的红黑树结构.

根据每个线程组的 *vruntime* 分配这个组获得的时间片长度,当一个线程组执行的时间超过了该组获得的时间片长度时,就切换到另一个组执行.而线程所获得的时间片长度则根据线程的 *vruntime* 以及该组所获得的时间片长度来决定,因此,本文提出的 CAS 调度形成了两个层次的调度过程:第 1 层是组调度,组是操作系统分配时间片的单位;第 2 层则是组内的线程调度,该层调度同样也使用完全公平策略.本文提出的 CAS 调度仍然基于现有 Linux 操作系统的 CFS 调度策略,都是根据 *vruntime* 来分配时间片和选择下一个将要运行的线程,所以在响应时间上不会有很大的影响.

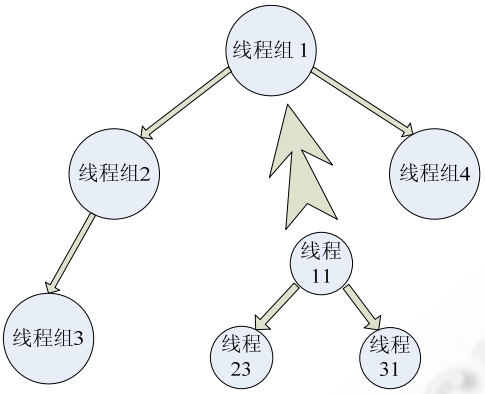


Fig.4 Array scheduling queue according to group

图 4 根据线程组重新排列调度队列

图 5 给出了本文提出的 CAS 调度策略的流程.我们知道,调度的过程主要就是选择下一个将要运行的线程的过程.在 CAS 中,如果当前运行组已经执行的时间还没有超过分配的时间片长度,则选择组内调度队列的最左下角(leftmost)线程作为下一个将要运行的线程;否则,首先选择组间调度队列中最左下角的线程组作为下一个线程组,并在选择的下一个线程组内调度队列中选择最左下角的线程作为下一个将要运行的线程.

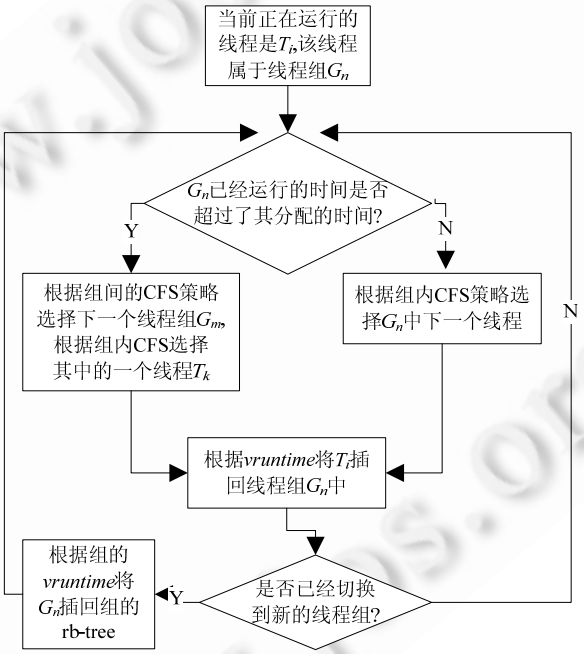


Fig.5 Process of CAS

图 5 CAS 的调度流程

2.4 内存状态控制

本文基于线程组之间的切换来管理内存状态的切换.CAS 根据线程组实现线程组周期性地被调度执行,同时,所需内存 rank 组也是周期性地被访问和处于空闲状态.因为每次将内存 rank 置为低功耗状态需要毫秒级的时间开销(一般是几个毫秒),这段时间不能执行代码,故而伴随着功耗和性能上的开销.因此,为了最大化能耗效

率,必须降低内存 rank 状态的切换频率.但是,关键问题在于,线程组切换的周期需要多长,如果周期太短,内存 rank 切换的频率太高,可能不但没有提高功耗效率反而因切换开销导致功耗效率的下降.

我们知道,Linux 操作系统中最短的调度周期(*sum_runtime*)是 20ms(当系统中线程少于等于 5 个时),因为我们只有 3 个非内核线程组,所以最短的切换周期是 20/3 ms.20/3 ms 已比内存状态的切换代价大了几毫秒,而且系统中一般线程都比较大,所以切换周期也会达到几十毫秒,因而内存 rank 状态的切换代价可以忽略.由此,在根据线程组调度时,基于组间的切换来管理内存状态的切换能够达到内存功耗的下降,同时可以避免频繁的内存状态切换导致的高开销.此外,还可以得知:系统中线程数越多,线程组的切换周期越长,也就可以获得更长的空闲时间,有利于内存功耗的降低.

3 实验与分析

3.1 实验平台与方法

本节将结合页分配和线程组调度的内存功耗优化方法与功耗感知的页分配策略[8]、通过调度实现内存功耗优化策略[9]以及 Linux 自带的调度策略 CFS 进行比较.其中,CFS 没有进行内存功耗的优化.

本文使用 MARSSX86[23]全系统模拟器模拟处理器,在模拟器上运行 Linux 2.6.31 操作系统,并且扩展 DRAMSim 内存模拟器模拟内存子系统.表 1 给出了处理器和内存的配置参数,为了对比内存的功耗,DRAMSim 模拟器实时记录内存通道、rank 和 bank 的状态.内存的功耗根据 Micron 给出的方法[24]来计算.内存的功耗参数选择与文献[7]相同.

Table 1 Processor and memory configuration

表 1 处理器和内存配置

部件	取值
处理器核数	4 核, 2.4GHz
L1 I/D cache(单独)	16KB, 2-way
L2 cache(共享)	64KB
Cache 块大小	64bytes
内存配置	2 GB, 2 channels, 8 ranks, 8 banks per rank

为了更加全面地评价本文的 CAS 方法,我们并发地执行来自 sysbench[25]和 SPEC 2006 标准测试集上不同组合的测试程序.在表 2 中,我们以数字-名字的形式来表示相应的测试程序.这种形式代表的意思是:如果测试程序来自 sysbench 集,则代表该测试程序的线程数;如果来自 SPEC 2006,则代表测试程序的重复个数.

Table 2 Benchmarks

表 2 测试程序集

程序集	测试程序
mix1	12-sysbench cpu, 8-omnetpp
mix2	12-sysbench memory, 8-omnetpp
mix3	6-sysbench cpu, 6-sysbench memory, 8-omnetpp
mix4	24-sysbench cpu, 16-omnetpp
mix5	24-sysbench memory, 16-omnetpp
mix6	12-sysbench cpu, 12-sysbench memory, 16-omnetpp
mix7	48-sysbench cpu, 32-omnetpp
mix8	48-sysbench memory, 32-omnetpp
mix9	24-sysbench cpu, 24-sysbench memory, 32-omnetpp

3.2 实验结果与分析

本节我们首先给出 CAS 策略在延长内存空闲时间上的优势,然后分析 CAS 在提高内存功耗效率上的效果,最后详细分析 CAS 在不同方面对性能的影响.

3.2.1 空闲时间分析

降低内存功耗最重要的工作就是要挖掘足够的内存空闲时间,但是要在不严重影响性能或者不产生很大

代价的前提下挖掘更多的内存空闲时间,对于目前的研究来说较为困难.为了评价本文提出的 CAS 内存功耗优化策略在挖掘内存空闲时间方面的优势,我们提出一个内存 rank 平均空闲时间长度的参数:*ior*,这个参数表示每个处于低功耗状态的内存 rank 被访问(唤醒)之前的平均时长:

$$ior = \frac{\sum_{i=1}^{n} ior_i}{n} \tag{2}$$

其中,ior_i 表示处于低功耗状态的 rank i 被访问之前的平均时长:

$$ior_i = \frac{\sum_{k=1}^{m} Ti_k}{m} \tag{3}$$

其中,Ti_k 表示处于低功耗后的 rank i 被访问之前的时长.*ior* 参数值越大,表示有更多的空闲时间被挖掘.

图 6 展示了归一化的 *ior* 参数值的对比,图中 default,PAPA 和 PPT 分别代表未对内存功耗进行功耗优化的 CFS 策略、功耗感知的页分配策略和通过调度实现内存功耗优化的策略.从图中可以轻易地发现:PAPA 和 PPT 两种策略都能比 default 挖掘出更多的空闲时间,而且本文提出的 CAS 的效果更加明显.还可以发现:随着测试集中测试程序的增加(mix1~mix3 中测试程序 20 个、mix4~mix6 中 40 个、mix7~mix9 中 80 个),*ior* 参数值也逐渐变大,也就意味着更多的空闲时间被挖掘.如果系统中线程数越多,那么每个线程组中的线程数也就越多,也就代表每个线程组在每个调度周期中时间片长度越长,那么处于低功耗状态的内存 rank 可以在更长时间内不被唤醒,从而减少了内存状态切换的频率.

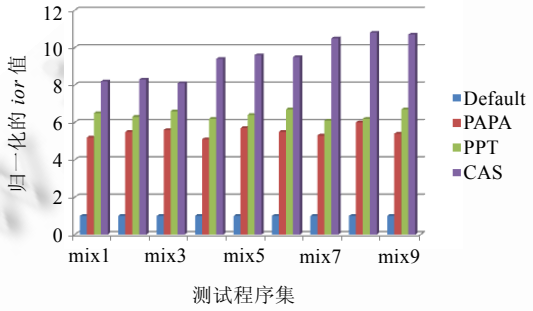


Fig.6 Normalized *ior* parameter value

图 6 归一化的 *ior* 参数值

3.2.2 功耗效率分析

通过挖掘更多的内存空闲时间,结合我们提出的基于组切换的内存状态管理策略,图 7 展示了 CAS 在降低内存功耗上的优势.

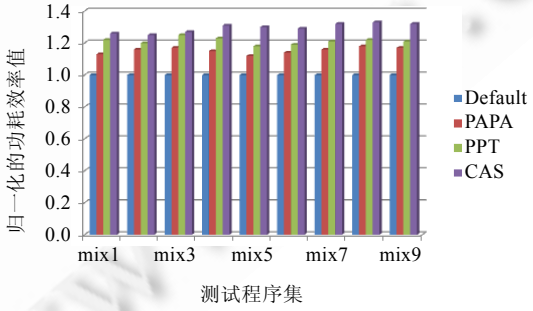


Fig.7 Normalized power efficiency

图 7 归一化的功耗效率

同样地,随着线程数量的增加,内存功耗效率也会随之提高.在 mix1~mix3 情形下,CAS 比 PAPA 平均节省了

9%,比 PPT 节省了 4%的内存功耗;在 mix4~mix6 情形下,CAS 比 PAPA 平均提高了 12%,比 PPT 提高了 7%的内存功耗效率;在 mix7~mix9 情形下,CAS 比 PAPA 平均提高了 16%,比 PPT 提高了 11%的内存功耗.

3.2.3 性能分析

图 8 给出了 4 种策略的归一化完成时间.完成时间越长,说明相应策略下性能下降得越严重.从图 8 可以看出:本文提出的 CAS 比 PAPA 和 PPT 所需完成时间更短,亦即性能比这两种策略更优;而且系统中线程数越多,CAS 的性能越好.在 mix1~mix3 情形下,CAS 比 PAPA 平均提高了 5%,比 PPT 提高了 7%的性能;在 mix4~mix6 情形下,CAS 比 PAPA 平均提高了 7%,比 PPT 提高了 9%的性能;在 mix7~mix9 情形下,CAS 比 PAPA 平均提高了 8%,比 PPT 提高了 12%的性能.

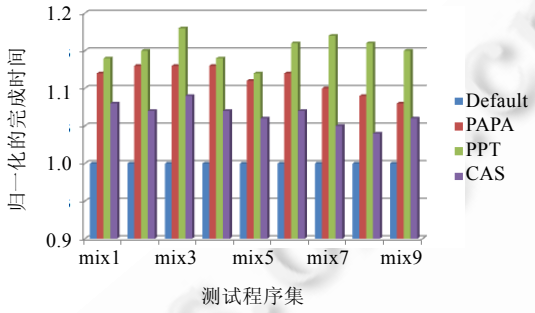


Fig.8 Normalized finish time

图 8 归一化的完成时间

CAS 比 PAPA 和 PPT 具有更好的性能,其主要原因有以下几个方面:

1) CAS 将系统中共享内存地址空间的线程划分到相同的线程组,因共享内存地址空间的两个线程切换执行时可以避免 TLB 以及 cache 的刷新,从而可以提高系统的性能;而且 CAS 策略优先调度线程组内的线程,因此大大提高了共享内存地址空间的线程相互切换的概率,有利于系统性能的提升.表 3 给出了 4 种不同策略下在调度过程中共享内存地址空间线程间相互切换的概率.从表 3 可以发现:CAS 大大提高了共享内存地址空间线程间相互切换的概率,所以有利于提高系统性能.

Table 3 Probability of threads switching between shared memory address space

表 3 共享内存地址空间的线程间相互切换的概率

	Default (%)	PAPA (%)	PPT (%)	CAS (%)
mix1	42	41	54	85
mix2	43	45	58	84
mix3	28	31	46	76
mix4	57	58	62	88
mix5	54	56	61	90
mix6	39	40	52	79
mix7	65	67	73	92
mix8	67	62	75	93
mix9	48	50	64	84

2) CAS 在页分配策略中兼顾了并行性,在减少每个线程占据的内存 rank 数的同时,尽量保证内存 bank 数,有利于保证系统的性能.表 4 给出了 4 种策略下平均每个线程占有的内存 bank 数,从表 4 可以发现:在 CAS 策略下,每个线程平均占有的内存 bank 数尽管比在 default 策略下要小,但却比 PAPA 和 PPT 要大.而且根据文献[13]中(21)的研究,线程所占的内存 bank 数平均能达到 16 个左右,对性能的影响不是很大.所以,本文提出的 CAS 策略有利于保证性能.

3) CAS 根据线程组进行调度,有利于提高 row buffer 命中率的提升,从而提高系统的性能.因为共享内存地址空间的线程间共享一些内存数据,将这些线程并发地执行,可以提高 row buffer 命中的概率.表 5 给出了各种

情形下(20 个线程、40 个线程以及 80 个线程)4 种策略的平均 row buffer 命中率,从表 5 可以明显地发现:在 CAS 策略下,row buffer 命中率高于其他所有的策略,所以 CAS 在性能上具有一定的优势.

Table 4 Average occupied banks for every thread

表 4 每个线程平均占据的内存 bank 数

	Default	PAPA	PPT	CAS
mix	20.8	6.8	6.4	14.6

Table 5 Average row buffer hit ratio in different circumstances

表 5 不同情形下平均 row buffer 命中率

	default (%)	PAPA (%)	PPT (%)	CAS (%)
20 个线程	45.3	43.7	43.1	52.4
40 个线程	46.8	44.2	43.8	54.6
80 个线程	47.6	44.9	42.6	56.3

4 结束语

本文提出了结合操作系统页分配策略和线程组调度策略来提高内存上的功耗效率的优化方法.首先,将系统中所有的线程根据共享内存地址空间和负载均衡策略划分成不同的线程组;其次,本文提出的组感知的操作系统页分配策略将每个线程组内线程所需内存聚集在两个内存 rank,但分布在这两个 rank 的所有内存 bank 中,即,避免扩散到太多的 rank 不利于功耗优化,同时又保证了并行性;最后,基于线程组调度,并基于组进行内存状态切换,在降低内存功耗的同时减少状态切换的频率,降低了状态切换开销.实验结果显示:本文提出的 CAS 策略挖掘了更长的内存空闲时间,因此提高了功耗效率,同时也减少了性能的下降.而且,当系统中线程数越多时,CAS 策略的功耗效率越高.这种特性有利于多核平台下线程数量越来越多的趋势.

致谢 在此,我们向对本文的工作给予支持和建议的同行表示感谢.

References:

[1] Barroso LA, Hölzle U. The datacenter as a computer: An introduction to the design of warehouse-scale machines. In: Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2009. 1−108.

[2] Lefurgy C, Rajamani K, Rawson F, Felter W, Kistler M, Keller TW. Energy management for commercial servers. IEEE Computer, 2003,36(12):39−48. [doi: 10.1109/MC.2003.1250880]

[3] Lim K, Chang J, Mudge T, Ranganathan P, Reinhardt SK, Wenisch TF. Disaggregated memory for expansion and sharing in blade servers. In: Proc. of the Int'l Symp. on Computer Architecture (ISCA). Austin, 2009. 267−278. [doi: 10.1145/1555815.1555789]

[4] Ahn JH, Jouppi NP, Kozyrakis C, Leverich J, Schreiber RS. Future scaling of processor-memory interfaces. In: Proc. of the Super Computing (SC). 2009. 1−12. [doi: 10.1145/1654059.1654102]

[5] Zheng HZ, Lin J, Zhang Z, Gorbatov E, David H, Zhu ZC. Mini-Rank: Adaptive DRAM architecture for improving memory power efficiency. In: Proc. of the Symp. on Microarchitecture (MICRO). 2008. 210−221. [doi: 10.1109/MICRO.2008.4771792]

[6] Udipi AN, Muralimanohar N, Chatterjee N, Balasubramonian R, Davis A, Jouppi NP. Rethinking DRAM design and organization for energy-constrained multi-cores. In: Proc. of the Int'l Symp. on Computer Architecture (ISCA). 2010. 175−186. [doi: 10.1145/1816038.1815983]

[7] Deng QY, Meisner D, Ramos L, Wenisch TF, Bianchini R. MemScale: Active low-power modes for main memory. In: Proc. of the ASPLOS. Newport Beach, 2011. 225−238. [doi: 10.1145/1950365.1950392]

[8] Lebeck AR, Fan XB, Zeng H, Ellis C. Power aware page allocation. In: Proc. of the Architectural Support for Programming Languages and Operating Systems (ASPLOS). 2000. 105−116. [doi: 10.1145/378993.379007]

[9] Lin CH, Yang CL, King KJ. PPT: Joint performance/power/thermal management of dram memory for multi-core systems. In: Proc. of the Int'l Symp. on Low-Power Electronics and Design (ISLPED). 2009. 93−98. [doi: 10.1145/1594233.1594255]

[10] Jia GY, Li X, Wang C, Zhou XH, Zhu ZW. Frequency affinity: Analyzing and maximizing power efficiency in multi-core system. In: Proc. of the IEEE 20th Int'l Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS). 2012. 495−497. [doi: 10.1109/MASCOTS.2012.63]

[11] Jia GY, Li X, Wang C, Zhou XH, Zhu ZW. Memory affinity: Balancing performance, power, thermal and fairness for multi-core systems. In: Proc. of the IEEE Conf. on Cluster Computing. 2012. 605−609. [doi: 10.1109/CLUSTER.2012.33]

[12] Li X, Jia GY, Chen Y, Zhu ZW, Zhou XH. Share memory aware scheduler: Balancing performance and fairness. In: Proc. of the 22th ACM/IEEE Great Lakes Symp. on VLSI (GLSVLSI). 2012. 291−294. [doi: 10.1145/2206781.2206852]

[13] Mutlu O, Moscibroda T. Parallelism-Aware batch scheduling: Enhancing both performance and fairness of shared DRAM system. In: Proc. of the ISCA. 2008. 63−74. [doi: 10.1109/ISCA.2008.7]

[14] Ding C, Zhong YT. Reuse distance analysis. Technical Report, TR741, Department of Computer Science, University of Rochester, 2001. 1−11.

[15] Merkel A, Bellosa F. Memory-Aware scheduling for energy efficiency on multicore processors. In: Proc. of the Hotpower. San Diego: USENIX, 2008. 123−130.

[16] Muralidhara SP, Subramanian L, Mutlu O, Kandemir M, Moscibroda T. Reducing memory interference in multicore systems via application-aware memory channel partitioning. In: Proc. of the MICRO. 2011. 374−385. [doi: 10.1145/2155620.2155664]

[17] Kim Y, Papamicheal M, Mutlu O. Thread cluster memory scheduling: Exploiting differences in memory access behavior. In: Proc. of the MICRO. 2010. 65−76. [doi: 10.1109/MICRO.2010.51]

[18] Kim Y, Han DS, Mutlu O, Harchol-Balter M. ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers. In: Proc. of the HPCA. 2010. 1−12. [doi: 10.1109/HPCA.2010.5416658]

[19] Lin J, Lu QD, Ding XN, Zhang Z, Zhang XD, Sadayappan P. Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems. In: Proc. of the HPCA. Salt Lake City, 2008. 367−378. [doi: 10.1109/HPCA.2008.4658653]

[20] Delaluz V, Vijaykrishnan N, Sivasubramaniam A, Irwin MJ. Memory energy management using software and hardware directed power mode control. Technical Report, CSE-00-004, Department of Computer Science and Engineering, The Pennsylvania State University, 2001. [doi: 10.1109/HPCA.2001.903260]

[21] David H, Fallin C, Gorbatov E, Hanebutte UR, Mutlu O. Memory power management via dynamic voltage/frequency scaling. In: Proc. of the 8th ACM Int'l Conf. on Autonomic Computing (ICAC). 2011. 31−40. [doi: 10.1145/1998582.1998590]

[22] Liu L, Cui ZH, Xing MJ, Bao YG, Chen MY, Wu CY. A software memory partition approach for eliminating bank-level interference in multicore systems. In: Proc. of the PACT. 2012. 367−376. [doi: 10.1145/2370816.2370869]

[23] Patel A, Afram F, Chen SF, Ghose K. MARSSx86: A full system simulator for x86 CPUs. In: Proc. of the DAC. San Diego, 2011. 1050−1055.

[24] Micron Technology Inc. Tn-41-01: Calculating memory system power for DDR3. 2007. http://download.micron.com/pdf/technotes/ddr3/TN41_01DDR3\%20Power.pdf

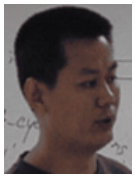
[25] Kopytov A. SysBench: A system performance benchmark. 2004. http://sysbench.sourceforge.net/index.html

贾刚勇(1987－),男,浙江永康人,博士,讲师,主要研究领域为操作系统级功耗管理,系统级性能优化,操作系统调度.
E-mail: gangyong@mail.ustc.edu.cn

蒋从锋(1980－),男,博士,副教授,CCF 会员,主要研究领域为虚拟化,功耗优化.
E-mail: congfengjiang@hdu.edu.cn

万健(1969－),男,博士,教授,博士生导师,主要研究领域为计算虚拟化,网格计算,服务计算,无线传感器网络.
E-mail: wanjian@hdu.edu.cn

代栋(1985－),男,博士,主要研究领域为云计算存储系统,云计算处理系统.
E-mail: daidongly@gmail.com



李曦(1963－),男,博士,副教授,CCF 高级会员,主要研究领域为嵌入式系统设计,操作系统,体系结构.
E-mail: llxx@ustc.edu.cn