计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2015.20140706
52(11):2599-2607, 2015

一种减少竞争的内存划分方法

贾刚勇[1] 李 曦[2] 万 健[1] 王 超[2] 代 栋[3]
[1](杭州电子科技大学计算机学院 杭州 310018)
[2](中国科学技术大学计算机科学与技术学院 合肥 230027)
[3](德州理工大学计算机系 美国德克萨斯州卢伯克市 79409)
(dong.dai@ttu.edu)

A Memory Partition Policy for Mitigating Contention

Jia Gangyong[1], Li Xi[2], Wan Jian[1], Wang Chao[2], and Dai Dong[3]
[1](*School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou* 310018)
[2](*School of Computer Science and Technology, University of Science and Technology of China, Hefei* 230027)
[3](*Department of Computer Science and Technology, Texas Tech University, Lubbock, TX, USA* 79409)

Abstract In modern multi-core system, memory is shared among more and more concurrently running threads. Therefore, memory contention and interference are more and more serious, which induces performance degradation differently, unfairness resource sharing and priority inversion even starvation. In this paper, we firstly analyze the problems induced by contention and interference in detail, and propose a pseudoshare framework which brings shared memory to be exclusive likely in multi-core system. The pseudoshare framework contains three main components: 1) Partition threads, cores and memory into thread, core and memory groups respectively, and one thread group runs on one core group occupying unique one memory group, and one core group is only scheduled for its corresponding thread group accessing its unique memory group. Through this way, memory access among core groups is isolated; 2) Analyze the behavior characteristics of threads, especially the threads' performance influence after memory is shared for each thread; 3) According to the performance influence of parallel running threads, allocate memory bandwidth for each thread. We implement pseudoshare in both 4-core and 8-core platforms. Experimental results show pseudoshare optimizes both the problems of performance degradation differently and unfairness resource sharing in both 4-core and 8-core, 9.8% and 22.5% on average for interference and fairness respectively. Moreover, pseudoshare solves starvation and reduces 5.3% power on average.

Key words memory contention; memory interference; performance degradation differently; unfairness resource sharing; power efficiency

摘 要 多核系统中内存被越来越多地核共享,因此,日益加深的内存竞争和内存干扰导致了越来越严重的核间不同程度的性能下降、不公平的资源共享和优先级翻转甚至饿死等问题.首先分析内存共享引发的问题,接着提出伪共享(pseudoshare)方法减少内存竞争。伪共享的框架包含3个部分:1)将所有的线程、处理器核以及内存分别划分成线程组、处理器核组以及内存组,1个线程组运行在1个核组上同时使用1个内存组,从而形成一个个相互独立的子系统,子系统间互不干扰;2)分析线程的内存行为

收稿日期:2014-07-29;**修回日期**:2015-02-28
基金项目:国家自然科学基金项目(61272131,61003077,61100193,61402140);浙江省自然科学基金项目(LQ14F020011)
通信作者:贾刚勇(gangyong@hdu.edu.cn)

特征,获取线程所需的内存带宽;3)给每个线程分配内存带宽.伪共享的方法通过内存的划分减少核间对内存的干扰和竞争,同时通过内存带宽划分提高了公平性.实验结果显示,伪共享的方法降低了 9.8%的内存干扰,提高了 22.5%的公平性,并且降低了 5.3%的能耗.

关键词 内存竞争;内存干扰;差异化的性能下降;不公平的资源共享;能耗

中图法分类号 TP302

多核系统不仅仅在桌面和服务器领域相对流行,而且在嵌入式领域也已经相当普遍.但是现有的多核系统为了高效地利用硬件资源,需要多个核共享一些硬件结构,其中内存就是 1 种重要的共享资源.资源的多核共享存在一些缺陷,即 1 个核上运行的线程不再是独立的,而是需要跟其他核上运行的线程相互协调使用共享资源.

核间的协调使用共享资源会产生很多问题,包括:1)并行执行的线程间性能不可预测地下降,每个线程性能的下降跟并行执行的其他线程密切相关;2)不公平的共享硬件资源,这种不公平性可能打乱服务质量(QoS)的需求;3)优先级反转,高优先级的线程因为获取太少的内存导致频繁地放弃处理器,从而更少地被调度执行,更严重的情况下可能导致饿死.图 1 展示了并行执行的各线程性能下降的差异性,给出 4 个线程并行执行时相对单独执行时各线程性能下降的比例.显然,各线程性能下降的差异是巨大的.

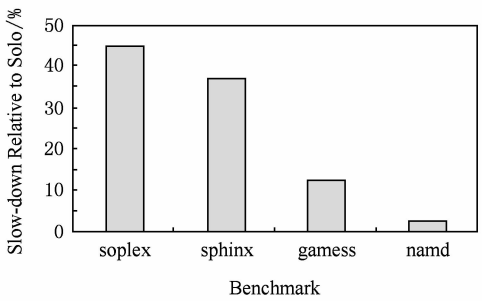


Fig. 1 Performance degradation of each thread relative to running solo.

图 1 每个线程相对单独执行时性能下降的比例

当前也有一些工作提出减少内存竞争和线程间内存干扰提高系统性能和公平性的方法.比如,线程调度策略[1-4],这些策略利用线程的不同特征组合调度从而减少内存竞争和相互干扰、内存调度[5],以及内存通道/Rank/Bank 划分[4,6]等.

尽管这些方法能够有效地减少竞争和干扰,但是基本存在以下 4 个问题:1)引入了不可扩展的额外硬件代价;2)目标比较单一,只能针对系统性能、线程性能的可预测性以及公平性三者中的一个进行优化;3)效果起伏不定,过分依赖并行执行的线程,有时效果很好,有时甚至出现负面的影响;4)打乱线程的优先级,经常出现优先级反转的现象.

本文提出了 1 种有效减少内存竞争和线程间相互干扰从而同时提高系统性能、线程可预测性以及公平性的方法,并且不需要引入额外的硬件,而且不会出现负面影响的情形.我们提出了 1 个伪共享(pseudoshare)的框架,这个框架将系统中的线程、处理器核以及内存分别划分成线程组、处理器核组和内存组,每个线程组跟 1 个处理器核组绑定,并且通过内存 Bank 感知的页分配策略,使每个核组只能使用 1 个内存组.1 个核组以及相应的 1 个线程组和 1 个内存组这三者形成 1 个独立的子系统(subsystem),子系统间相互独立、互不干扰.其次,根据并行执行的线程行为特征,给每个线程分配内存带宽.所有的子系统和内存带宽的划分形成伪共享的框架.

与已有的研究工作相比,本文的创新之处在于:

1) 划分线程、处理器核以及内存分别形成线程组、核组和内存组,从而进一步形成独立的子系统,子系统间相互独立,互不干扰;

2) 根据并行执行的线程行为特征,给每个线程分配内存带宽,实现线程间性能的公平下降;

3) 基于内存 Bank 感知的页分配策略,将每个线程所需的内存聚集在一个内存组内.

为了更好地评价本文提出的伪共享方法的有效性,我们将伪共享的方法同时实现于 4 核和 8 核平台.实验结果表明,本文提出的伪共享的方法平均减少了 9.8%的内存干扰,同时提高了 22.5%的公平性,并且降低了 5.3%的能耗.

1 内存组织结构和相关研究

1.1 内存组织结构

本节我们简单地介绍一下现有内存的组织结构以及操作系统对内存的管理机制.

1) 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 个的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,它可以并行访问,因此不同内存 Bank 上的请求可以同时被访问[3],提高内存的性能. 图 2 给出了一个当前内存子系统的组织结构图:

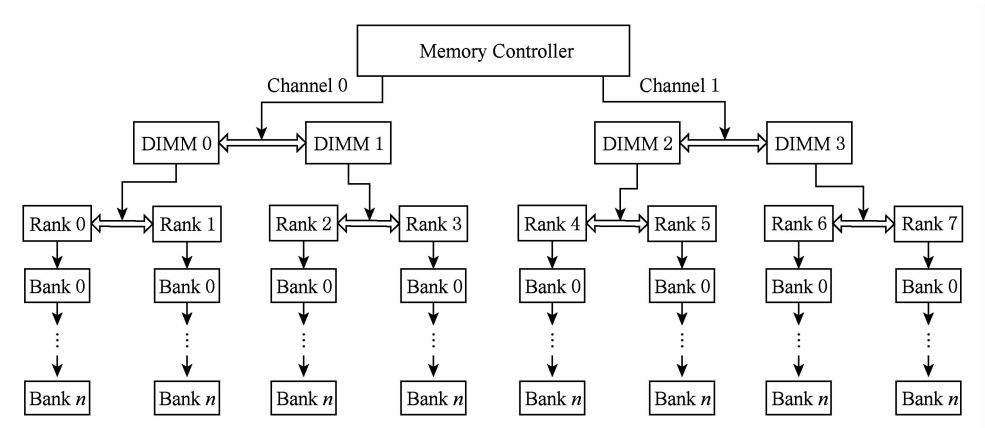


Fig. 2 Organization of a modern memory subsystem.

图 2 当前内存子系统的组织结构

2) 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,且同样大小的块形成一个链表.

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来降低内存的功耗.

1) 操作系统的页管理策略. Lebeck 等人[7]是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 2 种页管理策略:①顺序分配策略(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,但是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;②频率分配策略(frequency policy). 这是 1 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[8]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.

2) 操作系统的调度策略. 台湾国立大学 Yang 教授小组[9]提出的操作系统调度策略跟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,功耗效率的提升受到很大的限制. 本文[10-13]提出访存亲和性感知的操作系统调度实现内存功耗的优化.

3) 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[4];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、影响效果.

4) 内存控制器的调度. 根据线程级访存亲和性不同特征,调整内存控制器的调度策略,实现内存功耗效率的优化[1-2]. 基于线程组的内存调度策略(thread cluster memory scheduling, TCM)[1]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 2 个线程组,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.

5) 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的

相互干扰,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[14-15].然而,其他共享资源,如内存总线、内存等,都面临相互竞争、相互干扰,所以需要共同解决.

2 伪共享的框架

在 Linux 操作系统中使用伙伴算法进行物理页的分配,每次分配空闲列表的第一个页块,因此一个线程占据的内存页遍布整个内存.伙伴算法利用了内存的并行性从而提高性能.但是通过实验我们可以发现每个线程所需的内存 Bank 数是有限的[16].

2.1 线程所需内存 Bank 数

为了证明每个线程所需的内存 Bank 数是有限的,我们对大量的测试程序在分配 4～64 个内存 Bank 时的性能进行对比.图 3 给出了每个线程的性能与分配的内存 Bank 数量之间的关系.跟我们的设想相同,每个线程所需的 Bank 数量是有限的,大部分的程序只需 16 个 Bank 就已经足够了,分配更多的 Bank 对线程性能的提升基本没有帮助.

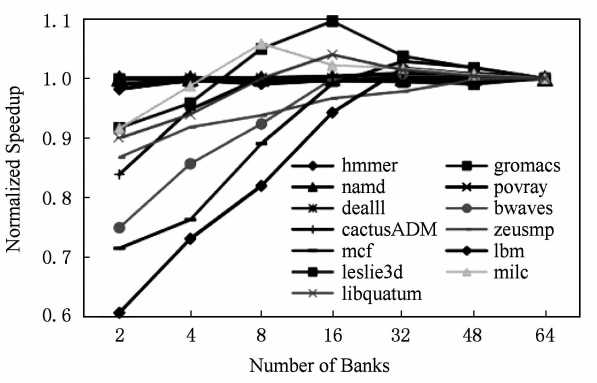


Fig. 3 Performance improvement along with Bank amount added.

图 3 伴随内存 Bank 数量的增加线程性能的变化趋势

因为内存间存在的依赖性以及高 Cache 的命中率,1 个线程的访存并发性不会很高.尽管如此,伙伴算法还是过分地利用 Bank 级的并行性,从而超过了线程本身的并行性需求.然而,超过线程的并行性需求不但没有性能的提升反而加剧了线程间内存的干扰.

为了在保证满足线程访存并行性的前提下尽量降低线程间内存的相互干扰,我们提出了内存 Bank 感知的内存划分.

2.2 线程和内存的划分

我们将内存划分成 4 个内存组,每个内存组包含独立的并且对于 1 个线程所需并行性已然足够的 16 个内存 Bank,同时将系统中所有的线程和处理器核也分别划分成 4 个线程组和处理器核组.1 个线程组运行于 1 个处理器核上,并且占用 1 个独立的内存组.

针对处理器核的划分,我们根据平均以及共享 Cache 的原则,尽量将共享 Cache 的核划分到相同的处理器核组中.

针对线程的划分,需要根据 3 个原则,分别是共享内存地址空间、负载均衡以及优先级.

1) 将共享内存地址空间的所有线程划分到同一个线程组中.共享内存地址空间的线程间相互切换可以避免 TLB 和 Cache 的刷新,从而可以提高系统的性能.

2) 根据优先级和负载均衡策略将共享内存地址空间的线程组划分成 4 个线程组.相同优先级的线程尽量均衡地划分到 4 个线程组中.

图 4 给出了 1 个新的线程被划分到 1 个线程组中的过程.当 1 个新的线程产生后,如果这个新线程与已有的线程共享内存地址空间,则将这个新线程加入到共享内存地址空间的线程的那个线程组中;如果不存在,则根据优先级和负载均衡原则将该新线程加入到 1 个线程组中.

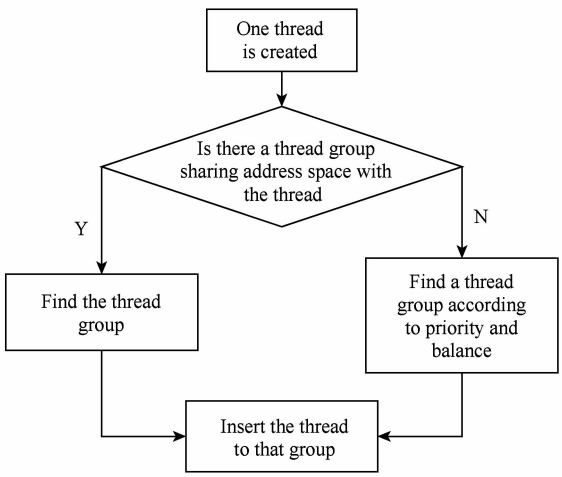


Fig. 4 Process of a new thread partitioned into one group.

图 4 一个新产生的线程被划分到一个线程组中的过程图

2.3 内存 Bank 感知的操作系统页分配策略

系统中的所有线程已经划分成 4 个线程组,每个线程组使用默认的完全公平调度策略(completely fair scheduling, CFS)基于线程的虚拟执行时间(Vruntime)组织成红黑树.因此,系统中有 4 个红黑树.

但是在给每个线程分配物理内存时我们不使用Linux操作系统默认的伙伴算法,而是使用我们提出的内存Bank感知的页分配策略,该策略能够在满足线程并行性的前提下尽量减少线程间内存的相互干扰,给每个线程分配足够、但是不超过需求的内存Bank.根据线程所在的线程组,分配相应内存组内的页给线程,相同线程组内的线程占据同一内存组,所以,1个线程所占据的内存只分布在1个内存组中而不是整个内存,从而可以保证性能同时减少线程间的干扰.

图5给出了内存Bank感知页分配策略的物理页组织结构.这个结构跟默认的伙伴算法的区别主要在于我们增加了内存Bank的信息.每个内存Bank组维护1个空闲页的列表,这样整个内存被划分成了4个部分.每次线程向内存请求1个空闲页块时,首先必须确定内存组号,再根据内存组号查找空闲页块.算法1描述了内存Bank感知的页分配策略.

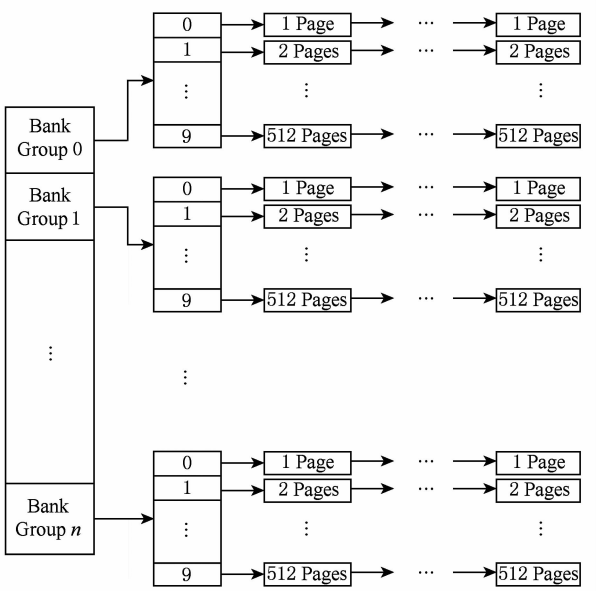


Fig. 5 Physical page organization of Bank aware page allocation.

图5 内存Bank感知页分配策略的物理页组织结构

算法1. 内存Bank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 */
  End If
End For
```

在内存Bank组 i 中查找所需空闲物理块 M;

将物理块 M 分配给线程 T_1.

2.4 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1个处理器核组上,多个核共享1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.1个线程组和该线程组运行的1个处理器核组以及占据的1个内存组这三者形成1个独立的子系统,如图6所示,图6中 n 为总核数除以4的值.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1个核,而在8核平台下每个子系统则包含2个核.

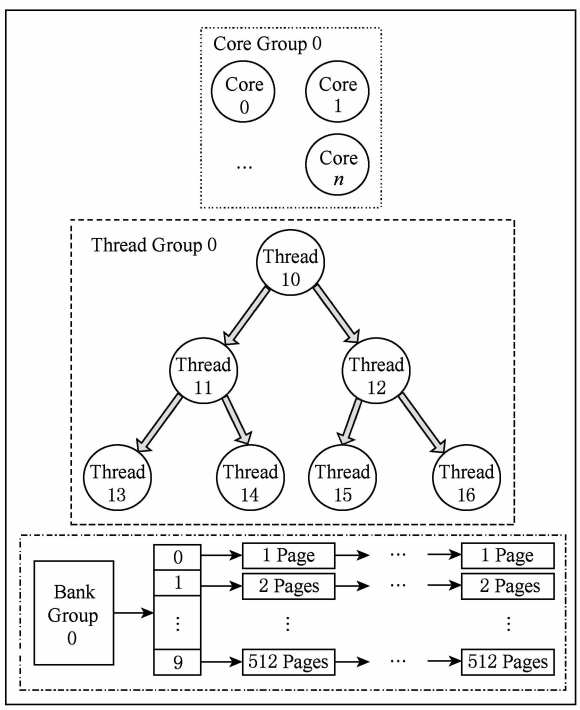


Fig. 6 One core, one thread group and one memory group form a subsystem in 4-core circumstance.

图6 由1个核组、1个线程组、1个内存组形成的一个独立的子系统结构

2.5 内存带宽划分

在整个系统形成相互独立的4个子系统后,因子系统间互不干扰,内存的竞争主要集中在内存带宽上.如果按照并行执行的线程行为特征公平地给线程分配内存带宽,那么线程间性能的不均等下降以及公平性问题都会得到解决.

假设 N 个线程并行执行,$T_1, T_2, \cdots, T_N$.通过处理器的性能监视器(performance monitor unit, PMU),可以统计线程执行的指令数和访存的指令

数,分别用 I_i 和 M_i 表示;同时用 B 表示系统总共的内存带宽.我们需要公平地给每个线程分配内存带宽,分别用 $B_1, B_2, \cdots, B_N$ 表示每个线程所需的带宽.每个线程的性能下降可以用 $(I_i+B_i)/(I_i+M_i)$ 表示,其中为了竞争内存带宽,$B_i<M_i$.所以,对于随机的 i 和 j,$1\leqslant i,j\leqslant N$,为了保证公平性必须满足:

$$(I_i+B_i)/(I_i+M_i)=(I_j+B_j)/(I_j+M_j), \quad (1)$$

$$(B_1+B_2+\cdots+B_N)=B. \quad (2)$$

通过式(1)(2),可以公平地给并行执行的每个线程分配所需的内存带宽.

3 实验与分析

本文使用 MARSSX86[17] 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,8 核平台下也使用类似的参数.

Table 1 Processor and Memory Configuration

表 1 处理器和内存配置

Function Unit	Configuration
Processor	4 Cores, 2.4 GHz
L1 I/D Cache (Solo)	16 KB, 2-Way
L2 Cache(Share)/KB	64
Cache Block/B	64
Memory	2 GB, 2 Channels, 8 Ranks, 8 Banks per Rank

为了更加全面地评价本文的伪共享方法,我们并发执行来自 SysBench[18],SPEC2000,SPEC2006 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 SysBench 的测试程序,表示该测试程序的线程数;针对 SPEC2006 测试程序则每个线程重复 3 个.为了更好地分析实验结果,将测试程序集分为不同的类型:内存敏感性和内存不敏感性.表 2 中测试程序集从 Mix1~Mix9 这 9 个组的内存敏感度越来越低,同时采用 *Weighted Speedup* 和 *Maximum Slowdown* 这 2 个参数来评价系统的性能和公平性:

$$Weighted\ Speedup=\sum_{i=1}^{N}\frac{IPC_i^{\text{shared}}}{IPC_i^{\text{alone}}}, \quad (3)$$

$$Maximum\ Slowdown=\max_i\frac{IPC_i^{\text{alone}}}{IPC_i^{\text{shared}}}. \quad (4)$$

Table 2 Benchmark Sets

表 2 测试程序集

Workloads	Benchmarks from SysBench, SPEC2000 and SPEC2006
Mix1	18-CPU, povray, tonto, calculix, perlbench, namd, wrf
Mix2	18-CPU, perlbench, namd, wrf, dealII, gcc, sjeng
Mix3	18-CPU, dealII, gcc, sjeng, gobmk, gromacs, h264ref
Mix4	9-CPU, 9-memory, gobmk, gromacs, h264ref, bzip2, hmmer, astar
Mix5	9-CPU, 9-memory, h264ref, bizp2, hmmer, astar, cactus, omnetpp
Mix6	9-CPU, 9-memory, hmmer, astar, cactus, omnetpp, xalanc, sphinx3
Mix7	18-memory, cactus, omnetpp, xalancbmk, sphinx3, gems, lbm
Mix8	18-memory, xalancbmk, sphinx3, gems, lbm, soplex, leslie3d
Mix9	18-memory, gems, lbm, soplex, leslie3d, libquantum, mcf

4 实验结果与分析

本节我们首先对比伪共享方法在系统性能上的效果,其次给出公平性方面的提升,最后分析功耗上的优势以及对系统带宽的敏感性.

4.1 系统性能的分析

我们将伪共享的方法与其他 3 种方法进行对比,包括 CFS,TCM 和组划分策略(bank partition management, BPM).其中,CFS 是现在 Linux 操作系统默认的方法,在这里作为对比的标准;TCM 是目前平衡系统性能和公平性最好的调度方法之一;BPM 则是目前通过内存 Bank 划分解决内存竞争和干扰最好的方法之一.

图 7 给出了相对于 CFS 在 4 核和 8 核平台下各方法的性能提升.从图 7 可以轻易地发现,伪共享的方法在 4 核和 8 核平台下都比其他方法要好;但是,伪共享的方法在 4 核下性能优势更加明显,因为 8 核平台下 2 个处理器核共享 1 个独立的子系统,子系统内部的 2 个核竞争内存资源导致性能有所下降,而且随着测试程序集对内存敏感度的降低,系统的性能提升越来越高.

伪共享方法的性能在 4 核和 8 核下相对 CFS 分别提升了 10.5%和 11.8%.系统性能的提升主要源于划分的独立子系统,子系统间互不干扰,减少了内存的竞争和线程间的干扰,以及增加了共享内存地址空间的线程间的相互切换从而降低切换代价.

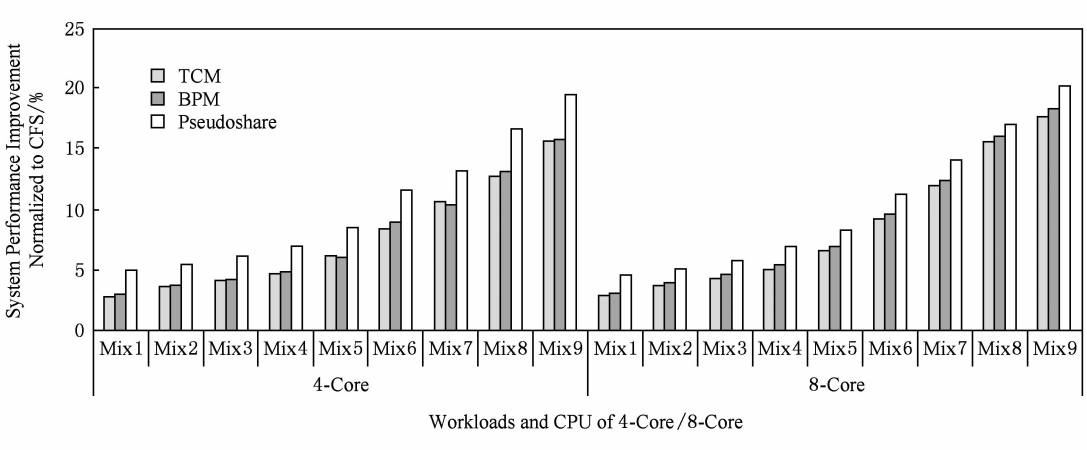


Fig. 7 System performance improvement in both 4-core and 8-core circumstances.

图 7 分别在 4 核和 8 核下性能的提升

相对于 TCM,在 4 核和 8 核下伪共享的方法分别提升了 3.3%和 2.1%的性能;相对于 BPM,在 4 核和 8 核下伪共享的方法分别提升了 2.6%和 1.7%的性能.

4.1.1 Row Buffer 失效率的下降

伪共享方法通过结合独立子系统和线程调度 2 个部分来实现系统性能的提升,我们详细地分析每个部分的效果.首先分析独立子系统在减少 Row Buffer 失效率上的效果.

图 8 给出了 4 核平台下相对 CFS 的其他 3 种方法在 Row Buffer 失效率上的效果.从图 8 可以明显地看出伪共享方法比 TCM 和 BPM 更加有效地降低 Row Buffer 的失效率.Row Buffer 失效率的下降是系统性能提升的主要原因.

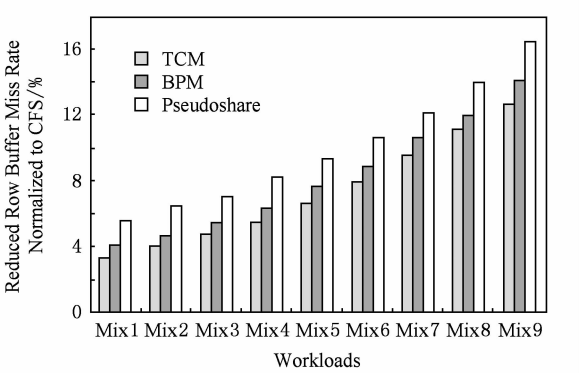


Fig. 8 Reduced Row Buffer miss rate across 9 workloads.

图 8 9 个测试程序集 Row Buffer 失效率的下降

在 4 核平台下,相对 CFS,TCM 和 BPM,伪共享方法分别下降了 10.2%,2.8%和 1.9%的 Row Buffer 失效率;在 8 核平台下,伪共享方法同样具有较好的效果,分别下降了 9.4%,2.3%和 1.6%的 Row Buffer 失效率.

通过划分线程组、处理器核组和内存组,并且 1 个线程组运行于 1 个处理器核组同时使用 1 个内存组,从而形成 4 个独立的子系统,能够很好地减少内存竞争和线程间相互干扰.这部分的效果是系统性能提升的主要原因.

4.1.2 切换代价的下降

本节我们分析通过线程调度降低切换代价.表 3 给出了 4 核平台下各种方法共享内存地址空间的线程间切换的平均概率,这个概率越高表示切换的代价越小,也即系统性能会相对更好.从表 3 可以得出,伪共享方法能够明显地提高共享内存地址空间线程切换的概率,而且随着多线程应用越来越普遍,提升的概率会越来越高.这也是性能提升的一个原因.

Table 3 Ratio of Switching Among Sharing Memory Address Threads

表 3 共享内存地址空间的线程间相互切换的概率

Method	Switching Ratio /%
CFS	13
TCM	12
BPM	17
Pseudoshare	34

4.2 公平性分析

图 9 给出了 3 种方法的公平性相对 CFS 的提升.显然,伪共享方法比其他方法在公平性上都要更优.这主要是因为伪共享方法根据并行执行的线程行为特征公平地分配内存带宽,理论上保证了线程间公平性.

图 10 给出了伪共享方法相对没有进行内存带宽划分条件下的公平性的提升.从这个结果可以进一步证明本文提出的内存带宽划分策略的有效性.

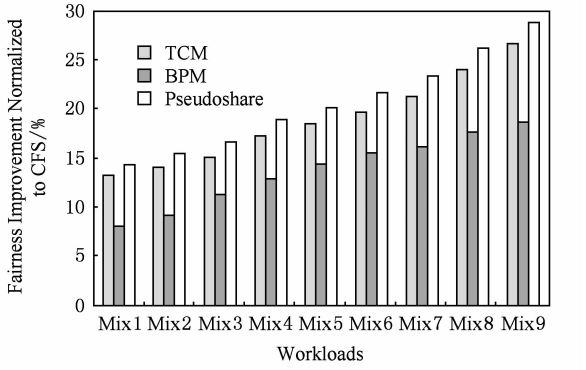


Fig. 9 Fairness improvement normalized to CFS in 4-core.

图 9 在 4 核下相对 CFS 公平性的提升

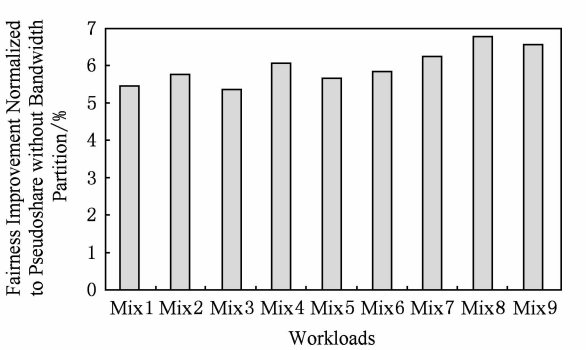


Fig. 10 Fairness improvement relative to pseudoshare without bandwidth partition.

图 10 相对于没有带宽划分的伪共享的公平性提升

4.3 功耗分析

在内存的功耗消耗中,Row Buffer 失效时的内存访问是最耗能的操作,因为 Row Buffer 失效后的内存访问需要将阵列中整行数据移动到 Row Buffer 中. 伪共享方法因降低了(Row Buffer)的失效率以及线程间切换的代价从而降低了内存的功耗. 通过模拟结果,伪共享方法在 Open-page 策略下能够降低 5.3%的内存功耗.

4.4 对系统内存带宽大小的敏感度分析

随着系统中处理器核数越来越多,因为处理器上引脚数有限,所以整个系统的内存带宽受限,导致平均每个核可用的内存带宽越来越少. 内存带宽已经成为片上多核进一步发展的一个主要限制. 每个核可用的内存带宽越来越少导致了越来越严重的竞争和干扰,同时也越来越迫切地需要解决竞争和干扰问题. 为了评价伪共享方法在各核可用带宽越来越小的条件下的效果,我们模拟每个核平均可用的内存带宽从 1.2 GBps 下降到 0.6 GBps 时各种情形下的效果.

图 11 给出了伪共享方法对内存带宽的敏感度. 从图 11 可以发现,随着带宽下降,线程间公平性越来越好,说明伪共享方法的效果也越来越好.

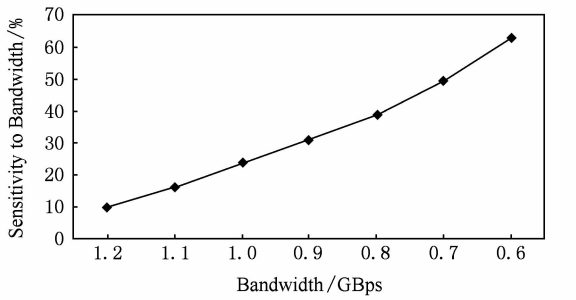


Fig. 11 Sensitivity to the bandwidth using pseudoshare.

图 11 伪共享方法对内存带宽的敏感度

5 总 结

本文提出了 1 种伪共享的方法,该方法通过划分线程组、处理器核组以及内存组,并且 1 个线程组运行于 1 个处理器核组同时使用 1 个内存组,从而形成一个个独立的子系统,子系统间相互独立、互不干扰,从而减少内存竞争和线程间的干扰;针对并行执行的线程,根据线程的行为特征公平地给每个线程分配内存带宽,从而提高线程间的公平性. 实验结果显示,伪共享方法在 4 核和 8 核平台下分别降低了 10.2%和 9.4%的 Row Buffer 失效率,同时降低了 161%和 136%的切换代价,从而提升了 10.5%和 11.8%的性能. 除此之外,伪共享方法在 2 种平台下分别提升了 20.7%和 24.2%的公平性,而且降低了 5.3%的内存功耗.

参 考 文 献

[1] Kim Y, Papamicheal M, Mutlu O. Thread cluster memory scheduling: Exploiting differences in memory access behavior [C] //Proc of the 43rd IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2010: 65-76

[2] Kim Y, Han D, Mutlu O, et al. ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers [C] //Proc of the 16th IEEE Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2010: 87-98

[3] Mutlu O, Moscibroda T. Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems [C] //Proc of the 35th Int Symp on Computer Architecture. New York: ACM, 2008: 63-74

[4] Muralidhara S, Subramanian L, Mutlu O, et al. Reducing memory interference in multicore systems via application-aware memory channel partitioning [C] //Proc of the 44th IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2011: 374-385

[5] Ausavarungnirun R, Chang K, Subramanian L, et al. Staged memory scheduling: Achieving high performance and scalability in heterogeneous systems [C] //Proc of the 39th Int Symp on Computer Architecture. New York: ACM, 2012: 416-427

[6] Jeong M, Yoon D, Sunwoo D, et al. Balancing DRAM locality and parallelism in shared memory CMP systems [C] //Proc of the 18th IEEE Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2012: 53-64

[7] Lebeck A, Fan Xiaobo, Zeng Heng, et al. Power aware page allocation [C] //Proc of the Int Conf on Architectural Support for Programming Language and Operating Systems (ASPLOS2000). New York: ACM, 2000: 105-116

[8] Ding Chen, Zhong Yutao. Reuse distance analysis, UR-CS-TR-741 [R]. Rochester, NY: University of Rochester, 2001

[9] Lin C H, Yang C L, King K J. PPT: Joint performance/power/thermal management of dram memory for multi-core systems [C] //Proc of the Int Symp on Low Power Electronics and Design (ISLPED2009). Piscataway, NJ: IEEE, 2009: 93-98

[10] Wang Lei, Liu Daofu, Chen Yunji, et al. Survey on partitioning and scheduling policies of shared resources in chip-multiprocessor [J]. Journal of Computer Research and Development, 2013, 50(10): 2212-2227 (in Chinese)
(王磊, 刘道福, 陈云霁, 等. 片上多核处理器共享资源分配与调度策略研究综述[J]. 计算机研究与发展, 2013, 50(10): 2212-2227)

[11] Jia Gangyong, Li Xi, Wang Chao, et al. Frequency affinity: Analyzing and maximizing power efficiency in multi-core system [C] //Proc of the 20th Symp on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS). Piscataway, NJ: IEEE, 2012: 495-497

[12] Jia Gangyong, Li Xi, Wang Chao, et al. Memory affinity: Balancing performance, power, thermal and fairness for multi-core systems [C] //Proc of the 2012 Cluster Computing. Piscataway, NJ: IEEE, 2012: 605-609

[13] Li Xi, Jia Gangyong, Chen Yun, et al. Share memory aware scheduler: Balancing performance and fairness [C] //Proc of the 22nd Great Lakes Symp on VLSI (GLSVLSI). New York: ACM, 2012: 291-294

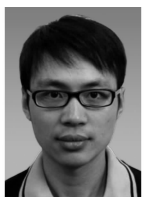
[14] Cho S, Jin L. Managing distributed, shared L2 caches through OS-level page allocation [C] //Proc of the 39th IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2006: 455-465

[15] Lin Jiang, Lu Qingda, Ding Xiaoning, et al. Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems [C] //Proc of the 14th IEEE Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2008: 367-378

[16] Liu Lei, Cui Zehan, Xing Mingjie, et al. A software memory partition approach for eliminating bank-level interference in multicore systems [C] //Proc of the Int Conf on Parallel Architectures and Compilation Techniques (PACT2012). New York: ACM, 2012: 367-375

[17] Avadh P, Afram F, Chen Shunfei, et al. MARSSx86: A full system simulator for x86 CPUs [C] //Proc of the Design Automation Conference (DAC2011). New York: ACM, 2011: 1050-1055

[18] Kopytov A. SysBench: A system performance benchmark [EB/OL]. (2004-05-16) [2014-06-13]. http://sysbench.sourceforge.net/index.html

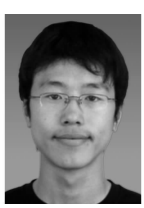


Jia Gangyong, born in 1987. PhD. His main research interests include system level power management, system level performance optimization and operating system scheduling.

Li Xi, born in 1963. PhD, associate professor. Member of China Computer Federation. His main research interests include embedded system, operating system and computer architecture (llxx@ustc.edu.cn).

Wan Jian, born in 1969. PhD, professor, PhD supervisor. Member of China Computer Federation. His main research interests include computing virtualization, grid computing, service computing and wireless sensor networks (wanjian@hdu.edu.cn).

Wang Chao, born in 1985. PhD, associate professor. Member of China Computer Federation. His main research interests include FPGA, big data, reconfigurable computing, MPSoC (cswang@ustc.edu.cn).



Dai Dong, born in 1985. Postdoctor. His main research interests include cloud computing focusing on the backend storage and processing system (dongdaily@gmail.com).