# 多核系统共享内存资源分配和管理研究

高 珂[1),2)] 陈荔城[1),2)] 范东睿[1)] 刘志勇[1),3)]

[1)](中国科学院计算技术研究所计算机体系结构国家重点实验室 北京 100190)

[2)](中国科学院大学 北京 100049)

[3)](中国科学院计算技术研究所移动计算与新型终端北京市重点实验室 北京 100190)

**摘 要** 对于共享内存资源的多核系统来说,分配和管理有限的内存资源是一个非常重要且具有挑战性的问题.随着处理器核数的快速增长,不同线程间的访存请求对系统中共享内存的竞争也愈发激烈,由此导致的对系统性能和系统公平性的影响也更加显著.为了缓解这一问题,除了增加可用共享资源外,公平高效地管理和利用共享内存资源至关重要.在各类共享资源中,对系统性能影响最大的是共享 Cache 和 DRAM.文中将这两级共享内存资源的分配和管理研究归结为三个重要方面,包括共享缓存分区、访存请求调度以及地址映射优化,并从优化系统吞吐率和公平性方面分析总结了一系列共享缓存分区策略,从缓解多线程对 DRAM 的竞争和相互干扰方面分析概括了一系列访存调度算法和地址映射策略.最后对共享内存资源未来的研究和发展做了总结和展望.

**关键词** 多核系统;共享缓存;缓存分区;动态随机存储器;访存调度;地址映射

**中图法分类号** TP302 **DOI 号** 10.3724/SP.J.1016.2015.01020

# Shared Memory Resources Allocation and Management Research on Multicore Systems

GAO Ke[1),2)] CHEN Li-Cheng[1),2)] FAN Dong-Rui[1)] LIU Zhi-Yong[1),3)]

[1)](*State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100190)

[2)](*University of Chinese Academy of Sciences, Beijing* 100049)

[3)](*Beijing Key Laboratory of Mobile Computing and New Terminals, Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100049)

**Abstract** It is an important as well as a challenging problem to allocate and manage limited memory resources on multicore systems. With the rapid growth of the number of processor cores, memory access requests from different threads to shared memory system become increasingly competitive, and the resulted impact on system performance becomes increasingly significant. Fair and efficient management and usage of shared memory resources are essential in addition to increasing available shared resources. Amongst various types of shared resources, shared Cache and DRAM have the greatest impact on system performance. This paper attributes shared memory resources management to three important issues, including Cache partitioning, scheduling and address mapping. We introduce and analyze a series of shared Cache partitioning strategies to optimize system throughput and fairness, a full range of DRAM memory access scheduling algorithms as well as mapping strategies to alleviate multiple threads competition and mutual interference. Future research issues on the allocation and management of shared memory resources are also discussed in this paper.

收稿日期:2013-08-28;最终修改稿收到日期:2014-10-31.本课题得到国家“九七三”重点基础研究发展规划项目基金(2011CB302501)、国家自然科学基金(61020106002,61221062,61332009,61173007,61100013)以及 NSFC 与香港 RGC 合作项目(61161160566)资助.

**高 珂**,男,1983 年生,博士研究生,中国计算机学会(CCF)会员,主要研究方向为多核处理器体系结构、内存体系结构.E-mail:gaoke@ict.ac.cn.**陈荔城**,男,1986 年生,博士研究生,中国计算机学会(CCF)会员,主要研究方向为内存体系结构、虚拟机内存管理.**范东睿**,男,1979 年生,博士,研究员,主要研究领域为众核处理器体系结构、高通量处理器体系结构.**刘志勇**(通信作者),男,1946 年生,博士,研究员,博士生导师,主要研究领域为高性能算法与体系结构、并行处理.E-mail:zyliu@ict.ac.cn.

**Keywords**　multicore systems；shared cache；cache partition；DRAM；memory access scheduling；address mapping

## 1　引　言

处理器和内存系统是冯诺依曼体系结构计算系统的两大核心部件. 处理器的任何操作必须首先从内存系统中取出指令和数据，然后处理器再将操作完成的数据保存回内存中. 这种以内存为中心的体系结构使得内存墙[1-2]（Memory Wall）问题成为限制计算系统整体性能的关键因素. 在过去三十年中，由摩尔定律所推动的处理器体系结构改进，使得处理器的性能得到高速的发展，而内存系统的性能改善却远远落后于处理器. 多线程执行和多核处理器的出现，使得这一问题的影响更加严重. 来自多个处理器核的访存请求对共享内存资源（Cache 和 DRAM）产生激烈的竞争，对内存系统产生越来越大的压力. 内存系统的性能成为系统规模扩展和性能提升的关键因素，内存资源的优化配置和高效管理已经成为提升多核系统性能的关键技术.

现代多核处理器通常包含私有缓存（Private Cache）和末级共享缓存 LLC（Last Level Cache）. 图 1 所展示的是一个典型的多核系统. 这种多级层次结构是缓解处理器与 DRAM 内存之间性能差异最常用的方法. 它利用程序局部性原理，把不同速度、不同容量和不同造价的 SRAM 缓存插在处理器和 DRAM 之间. 通过减少对 DRAM 的访问次数来大幅缩短访存延迟. 私有缓存的特点是更靠近处理器，为每个核独占，容量较小，但访问速度较快，不同缓存之间可能存在数据的冗余备份，需要进行一致性维护. 末级共享缓存的特点是容量较大但访问速度较慢，可以缓存多个进程或线程的独有或共享数据，多核共享可以减少数据的冗余，并降低多核间的通信延迟，但这将导致多核对共享资源的竞争访问和相互干扰，产生较高的缓存失效率（Cache Miss Rate），影响整个系统的吞吐率和公平性[3-4]. 由于 Cache 比 DRAM 更靠近计算单元，故更容易影响计算性能. 一种直观的方式是将 Cache 资源划分为若干独立的部分，并分别赋予不同的线程，从而缓解线程间对共享 Cache 的竞争和相互干扰. 采用分区的方法从 20 世纪 90 年代以来，一直被人们反复研究，通常结合各种替换策略和实现机制来适应不同的计算环境，以达成不同的性能目标.

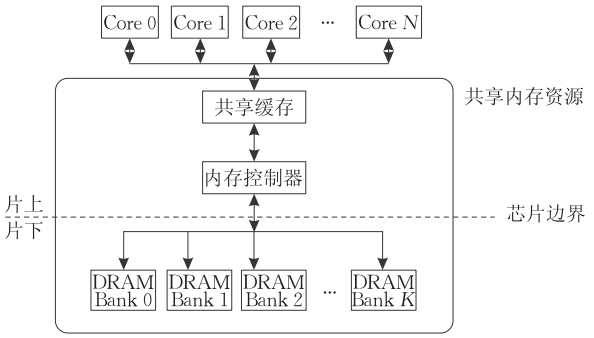



图 1　多核系统共享内存资源示意图

从主存储器的角度讲，影响系统性能的主要因素包括内存控制器的地址映射（Address Mapping）和访存调度策略（Scheduling）. 当发生末级缓存失效时，处理器不得不延长访存路径，访问 DRAM 存储器. 由于片下内存的访问延迟比缓存延迟大的多，通常存在数量级上的差距，导致处理器所需要的数据无法及时返回，处理器进入阻塞①状态并等待请求返回，严重浪费计算资源，并对系统性能造成巨大影响. 学术界和工业界对这类问题已有大量的相关研究，因为上述任何一个方面的改进，都可以降低访存的延迟进而提高系统的性能. 在多核平台上，为了并发地处理不同核的访存请求，现代的 DRAM 系统通常包含多个 Bank，以并发地独立处理各自的访存请求. 然而如果来自不同进程/线程的访存请求访问同一个 Bank，则会产生 Bank 上的冲突. Bank 冲突将引发两个对性能的“负”影响：（1）潜在的可被并行处理的访存请求，将不得不被串行地执行；（2）Bank 上的行缓冲（Row Buffer）将产生“颠簸”. 这两个“负”影响都会加大访存延迟，使得“内存墙”问题在多核计算环境中更为突出. 地址映射是否合理会严重影响程序的 DRAM 访问行为，例如连续的访存请求被映射到相同 Bank 的不同行中，就会造成 Bank 冲突. 地址映射的优化目标就是最小化 Bank 冲突，并同时增加数据局部性. 访存调度策略是另一种重要的优化方式，它可以根据不同的计算环境因地制宜地采用不同的调度算法，并较易于与现有系统集成. 从 21 世纪初开始，这一方法被人们反复地使用. 该方法可被概括为通过调整内存控制器调度队列中访存请求的顺序，实现缓解线程间的访存干扰，同时兼顾公平性（Fairness）和吞吐率

① 对于乱序执行处理器来说，重排序缓存 ROB 满时才发生阻塞.

(Throughput)的目标.

我们认为,针对多核系统中线程间的访存干扰问题,共享内存资源的分配和管理研究可以归结为两个系统层次上的三个重要方面,包括共享缓存分区、访存请求调度以及地址映射优化.本文将在这三个方面介绍、概括以及展望未来的发展方向.本文第2节从Cache层次介绍并分析片上共享缓存分区的相关研究,主要包括共享缓存分区框架以及针对多核环境下缓存分区的重要目标和近期的研究特点;第3节我们从DRAM存储层次对内存控制器优化技术进行概括总结,包括访存调度算法和地址映射两个重要方面,将调度算法按照不同的实现层次进行分类归纳和总结,并对单核时代到多核时代地址映射优化的软硬件相关研究进行分类和概括;第4节总结全文,并分析讨论内存共享资源未来的研究和发展趋势.

## 2 共享缓存分区

在框架(framework)上,我们将共享缓存分区归结为三个主要的构成部分:缓存性能信息探测和收集、缓存分区算法以及具体的缓存分区实现机制.随着多核/众核时代的来临,线程级大规模并行的趋势越来越明显,可并行的多应用程序行为特征更为复杂和多样,在给定不同目标和体系结构特征的限制下,共享缓存分区面临新的问题和挑战.本节主要分为两部分:第1部分集中分析和总结与缓存分区框架相关的若干问题,其中包括硬件缓存性能计数器的设计和采用,实际应用中的几种衍生和折中缓存分区算法,以及这些分区算法在软硬件环境下的具体实现机制;第2部分集中讨论多核/众核时代到来之后,线程间干扰所产生的新问题和新趋势,分析并总结了面向吞吐率和公平性的缓存分区研究,以及针对线程数量进一步增长所带来的扩展性问题的细粒度分区研究.

### 2.1 共享缓存分区框架

共享缓存分区研究在单核时代就已经出现,主要涉及的是指令和数据的缓存分配问题.Stone等人[5]研究了多道程序在单核处理器上的最优化分区问题,他认为缺失率是缓存大小的函数,图2所示为缺失率函数随分区状态转换图.其中$M(x)$代表进程的缓存缺失率,$I$和$D$分别代表指令流和数据流,$C$为共享缓存的容量大小,$S(x)$代表每个进程在缓存容量$x$下的概率.Stone通过推导证明指令流和数据流存在最优化分区的情形:当不同源访存缺失率函数的导数相等时,即是最优化分区的情形.然而在竞争的情况下,对共享缓存的无约束管理,无法实现最优化.

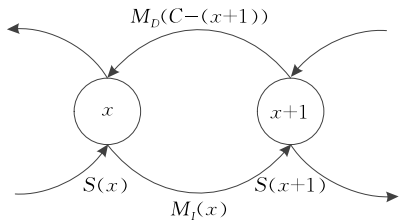



图2 缓存分区状态转换图

Suh等人[6-7]认为根据Stone的简单贪婪算法可以找到最优的缓存分区,但指出这种方法只适合缓存缺失率函数是单调的情形.在实际应用中,缓存缺失率函数并不是严格单调的,这导致如果采用分割算法,将带来很大的实现开销.Suh等人[6]提出了自己的缓存分区整体框架,并重点研究了缓存的分配算法.Suh等人给出了多核共享缓存分区的形式化定义:假设当前运行的进程数为$N$,若按照缓存块的粒度完成分区,进程间共享缓存块的个数是$C$,问题是将共享缓存分割成$N$个互不相交的子集,目标是最小化整体的缓存缺失率.每次分区的时间间隔记为$T$(该时间包含重新分区所导致的开销).设$C_i$是分配给第$i$个进程的缓存块的个数,分区需要产生一个包含$N$个元素的序列$\{C_1,C_2,\cdots,C_N\}$.定义$m_i(c)$是第$i$个进程在时间$T$内所发生的缺失率,最优化分区就是使得表达式$\sum_{i=1}^{N} m_i(c_i)$最小化,其限制条件为$\sum_{i=1}^{N} C_i = C$. Suh等人对缓存分区问题形式化的描述对基于性能探测和收集的缓存分区框架研究具有指导意义.

#### 2.1.1 性能信息探测和收集

现代商用处理器一般包含硬件缓存性能计数器.通过性能计数器可以在程序运行过程中动态和周期性地收集特定信息,比如访存次数、缺失次数等.这种精确地获取程序访存行为特征信息的探测装置,可有效辅助缓存分区选择机制做出相应的分配和调度策略.Suh等人[7]提出通过边界收益(Marginal Gains)的方法来寻找适合各个程序的分区大小.对组相联缓存增加路计数器(Way Counter),当一个最近期使用MRU(Most Recently Used)缓存路中的缓存行发生命中时,Counter(0)增加1.Counter(1)统计次最近期MRU的缓存行命中次数,以此类推,Counter($N$)统计近期最少使用LRU(Least Recently Used)的缓存命中次数.除了这些

信息外，每个缓存块还需要增加进程的 ID 信息，另外每个处理器包含一个缓存块计数器，用以保存处理器所包含的缓存块数. 这种每路缓存对应一个计数器的结果是将一路缓存作为缓存分区单元，在系统的缓存路数足够多且线程数较少时，这种做法可以取得较为理想的效果. 通过对性能信息进行监控，既可以得到线程随可用缓存空间减小而导致的缓存失效率的增加趋势，也能反过来获取当线程的可用缓存空间增加时，线程的缓存失效率降低的趋势. 这种随着缓存空间的增减带来的性能改变即为边际效益. 在具体的分区实现机制中，收集哪些信息与特定的缓存分区目标有关，比如对缓解多核多线程程序的访存竞争和污染，提高系统整体性能而言，需要收集总的缓存缺失率信息[3]；对更加公平地对待并行运行的多道程序，保证各个进程的服务质量 QoS (Quality of Service)而言，需要收集各个进程在单独运行和共享运行下的缓存失效比信息[4].

2.1.2　缓存分区分配算法

依据探测和收集获取的性能信息，可以设计相应的算法来确定每个处理器核可用的缓存空间配额. 缓存分区分配算法的有效性对缓存分区机制性能具有重要的影响，对缓解多核之间的访存竞争和污染具有重要作用. Hassidim[8] 从理论上证明了为多核处理器中的每个核寻找最优的共享缓存分区配额是一个 NP 难的问题. 当处理器核的数量达到一定规模时，对每一种可能性的分区分别评估是不切实际的，例如对于一个 32 路的末级共享缓存进行分区，四核的情况下将会有 6545 种分区可能，八核时将会达到 15 380 937 种可能. 因此，在具体实现缓存分区机制的时候，都会有所侧重和权衡. 比如在缓存分区研究早期提出并被广泛研究的常用算法有贪婪算法[9]、基于分段的凸函数算法[5]以及前瞻预测算法[3]等等.

2.1.3　缓存分区实现机制

缓存分区除了需要通过算法来确定分区在大小和数量的最优划分外，还需要一个底层的实现机制. 根据实施的路径和时机的不同，可将缓存分区实现机制分为两类：

(1)严格限定缓存位置分区. 概括地说是严格限制访存地址在缓存中的位置. 路分区(Way Partitioning)或者列分区(Column Caching)方式是将缓存按路进行分割，并限制每个分区映射到系统可用路的子集中. 路分区比较简单，但也存在较明显的问题：以路粒度进行分区，可用分区数受限于路的数目，可扩展性较差；另外也会导致每个分区的关联度降低，影响分区后的性能. 为了避免关联度的缺失，可以将缓存按组(Set)分区，比如可重配 Cache[10] 和 Molecular Cache[11]. 还可以通过页着色技术，利用操作系统的虚存管理来限制进程可用的物理地址到固定的缓存组中.

(2)基于替换策略的分区. 另外一种分区实现机制是通过修改缓存的分配和替换策略来实现. 这种机制没有对缓存分区位置进行严格限制.

如表 1 所示，我们统计了从 2000 年开始至今共享缓存分区所采用的不同实现机制. 其中路分区[6,10,12]最简单，但存在分区粒度较大，分区数目受限于可用路数，可扩展性较差等问题. 为了避免关联度的降低，可以采用组分区[10-11]的方式，然而这种方法需要可配置的译码器或者需要对缓存阵列做较大的修改. 通过操作系统的页着色[13]来限制一个进程的物理地址映射到固定的缓存组中，无需硬件修改，但是同样存在分区粒度较大的问题，并且软件的方法在页面重着色过程中需要进行内存拷贝，开销较大，因此要求重着色分区操作不能太频繁. Xie 等人[14]提出的 PIPP 伪分区机制，是一种基于修改缓存替换策略的分区机制，这种机制避免限制内存物理地址在缓存中的位置，通过分别赋予每个分区在 LRU 替换链中不同的插入位置来实现分区，并在发生 Cache 命中访问时缓慢地提升该缓存行的位置，而不像原 LRU 直接将其插入到 LRU 链头. PIPP 是一种结合 UCP[3]方案的分区机制，然而同样存在扩展性的问题. 随着处理器核数的快速增长，共享缓存分区需要向更细粒度划分的方向发展.

**表 1　共享缓存分区机制分类**

| 发表年份 | 研究者 | 分区机制 | 硬件代价 |
|---|---|---|---|
| 2000 | Chiou 等人[12] | 路分区 | 低 |
| 2000 | Ranganathan 等人[10] | 组分区/路分区 | 高 |
| 2002 | Suh 等人[6] | 路分区 | 低 |
| 2004 | Iyer 等人[15] | 基于替换策略 | 低 |
| 2006 | Varadarajan 等人[11] | 组分区 | 高 |
| 2008 | Lin 等人[13] | 页着色 | 无(软件) |
| 2009 | Xie 等人[14] | 基于替换策略 | 低 |
| 2011 | Sanchez 等人[16] | 细粒度 | 低 |
| 2013 | Cook 等人[17] | 路分区 | 低 |

## 2.2　多核处理器缓存分区

多核环境下共享缓存会同时接收来自于多个线程的访存请求，而单个线程的访存行为特征在时间局部性和空间局部性上具有很大不同，随着并行执行的线程数量的增长，程序行为特征更加复杂多样，这加重了线程间的相互干扰，降低了共享缓存的使用效率. 因此多核环境下的缓存分区必须要考虑如

何控制线程自身对缓存的需求，以及对其他线程的干扰程度. 最大公平性(Fairness)和最大化吞吐率(Throughput)是目前多核环境下所要达到的两大目标. 另外由于线程数量的增长所导致分区粒度的扩展性问题也已成为新的研究方向.

2.2.1 公平性分区

Kim 等人[4]研究了以公平性为目标的缓存分区机制，他们认为操作系统的线程调度依赖于硬件提供公平的缓存共享的能力，并提出了 5 种公平性的衡量标准. Kim 等人首先对系统的公平性给出了一个理想化的定义：

如果分区满足$\frac{T_{shr1}}{T_{ded1}}=\frac{T_{shr2}}{T_{ded2}}=\cdots=\frac{T_{shrn}}{T_{dedn}}$，那么认为系统是绝对公平的，其中 $T_{shri}$ 表示多个线程并行运行时线程 $i$ 的运行时间；$T_{dedi}$ 表示线程 $i$ 独占处理器资源时的运行时间. 在实际系统中，$T_{shri}$ 信息易于获取，但是 $T_{dedi}$ 信息很难取得. 因此 Kim 等人提出另外 5 种可用指标来衡量系统公平性. 所用到的信息包括各线程独占缓存时产生的失效数 *Miss_dedi* 和缓存失效率 *Missr_dedi*，以及多个线程并行时的缓存失效数 *Miss_shri* 和缓存失效率 *Missr_shri*.

通过对其不同的计算，可以产生不同的衡量标准，包括平衡各线程缓存失效数增加的比例；平衡各线程的缓存失效数；平衡缓存失效率增加的比例；平衡各线程并行运行增加的缓存失效率. 通过带有这些衡量标准的实验表明以改善系统公平性为目标的缓存分区机制通常可以同时提高系统吞吐率，但是以最大化吞吐率为目标的缓存分区则无法保障公平性. 分区如果没有公平性考虑，则会出现诸如线程饥饿、优先级倒置以及某些线程非常依赖于其他线程的运行等问题.

2.2.2 最大化吞吐率

Qureshi 等人在文献[3]中提出了一种基于收益率的缓存分区 UCP(Utility-based Cache Partition)机制. UCP 的优化目标是最大化系统吞吐率. 与 Suh 对边际效益的定义相同，UCP 中把随着缓存空间增加而减少的缓存失效数称之为收益(Utility). 针对以前监控器存在的缺点，UCP 提出一种新的收益监控器 UMON(Utility Monitor)对其进行改进，每个 UMON 增加一组辅助标签目录 ATD(Auxiliary Tag Directory). ATD 除了不包含具体的数据项外，与共享缓存保持相同的结构，如图 3 所示. 各线程间的 ATD 保持相互独立，从而避免了并行运行线程对所获取的缓存收益信息的影响，能够更为准确地反映一个线程独立的访存行为特征.

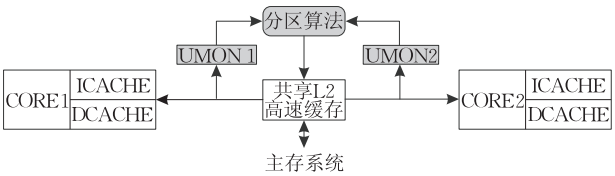



图 3 UCP 分区框架(阴影代表增加的结构)

在 UCP 中根据不同程序从额外的缓存空间获取的收益情况不同，Qureshi 等人[3]将程序划分为 3 类：当分得的缓存空间增多时，其收益不会发生显著变化的称为低收益类(Low Utility)；随着所分配的缓存空间增加其收益持续上升的程序称为高收益类(High Utility)；当分配的缓存空间增加到某个临界点后其收益不再继续提升的称为饱和收益类(Saturating Utility). 低收益类程序在并行运行时，彼此间对于自身可用的缓存空间不敏感，因而缓存分区并非必需；当饱和收益类程序共同运行时，只要知道各程序的缓存需求，即可分别为其分配合理的缓存空间；当饱和收益类程序与低收益类程序共同运行时，优先满足饱和收益类的缓存需求；而高收益类的程序在与其他类程序共同运行时，由于高收益类程序总是对于可用缓存空间大小很敏感，因此需要特别对待.

所做分区方式的可能数目随并行程序数量的增加呈指数增长，因此通过验证每种可能的分区来选择一个最好的分区策略变得不切实际. Qureshi 等人[3]提出前瞻算法作为一种可扩展的替代算法. 通过对程序的访存特征进行研究并分类，解释不同的程序从缓存分区的获益状况，可以帮助制定有效的缓存分区策略. 然而 UCP 中所提出的分类方式更多的是对于一个线程收益曲线的直观判断，并没有提出一个形式化的算法来对程序进行准确归类，因而很难在硬件上实现. 另外，UCP 通过在每个核的 UMON 中增加 ATD，以获取更为准确的访存信息，但是这也带来了较大的硬件开销.

除了硬件方法辅助分区外，软件的方法也可以达到共享缓存分区的效果. Lin 等人[13]提出一种采用页着色技术来提高吞吐率的分区方式. 这是一种纯软件的缓存划分方法，它通过操作系统的虚实地址映射来实现 Cache 划分. 目前大部分计算机系统的最后一级 Cache 是物理地址索引的，所以如果操作系统分页能使这些程序仅访问 Cache 的某一部

分，即可实现分区的效果. 如图 4 所示，物理地址中物理页索引的部分与 L2 缓存组索引重合的部分是操作系统分页算法可以控制的，示例中显示重合的部分有 4 位. 这 4 位代表 16 种不同取值，即可以把 Cache 划分成 16 个不同的区域，通常被称作把 Cache 划分成 16 种不同的颜色. 操作系统的页面存储分配算法可以限制程序所使用的物理页面地址的 4 位取一种或者多种颜色，通过为不同程序选择不同的颜色，就能保证这些程序映射到 Cache 的不同部分. 软件页着色的方法受限于颜色数固定的地址映射方式，另外存在分区粒度较大和分区不能够太频繁的限制.

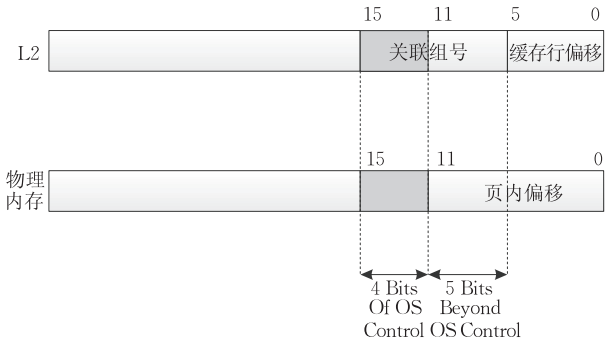



图 4　操作系统控制的 L2 Cache 划分

2.2.3　细粒度分区

目前有关共享缓存分区的趋势在朝向更细粒度的方向发展，这也是多核时代的必然要求. Sanchez 等人在文献[16]中针对之前的 UCP 策略分区粒度大、可扩展性差、效率低等问题，提出一种新的缓存分区策略 Vantage. Vantage 以缓存行为单位进行缓存分区，解决了分区粒度大的问题. Vantage 以较小的代价把共享缓存分为数十个分区，解决了随着线程数增多缓存分区可扩展性差的问题；同时在 Vantage中，不会破坏缓存的关联度，避免由于缓存关联度降低导致的缓存失效率上升；并且线程间仍然执行严格分区，避免了线程间的相互干扰. 与之前的缓存分区方案的另外一个不同点在于，Vantage 中并不会把全部共享缓存空间都分配给线程使用，而是保留一小部分(例如 10%的缓存空间). 当线程对于缓存的实际需求超出缓存分区策略分配给该线程的缓存空间时，可以共用保留的这部分共享缓存，而不是占用其他线程的缓存空间.

另外，随着处理器核数的迅速增长，在大规模的处理器核上进行末级缓存划分越来越具有挑战性. Song 等人[18]针对众核处理器[19]结构，提出了一种隐式动态的共享 Cache 空间的划分机制，用于隔离不同线程在共享 Cache 中的数据，减少多个线程访问缓存的冲突问题.

## 3　内存控制器优化

DRAM 内存系统主要由内存控制器和内存芯片组成. 内存控制器通过一个片选信号控制一个内存 Rank，每个 Rank 由多个 DRAM 颗粒组成. 为了提高访存的并发度，DRAM 颗粒由多个 Bank 体并联而成，如主流的 DDR3 颗粒包含 8 个 Bank 体. Bank 体主要由存储阵列和行缓存(Row Buffer)组成，行缓存用于存放正在被读写访问的内存行. 如图 5 所示.

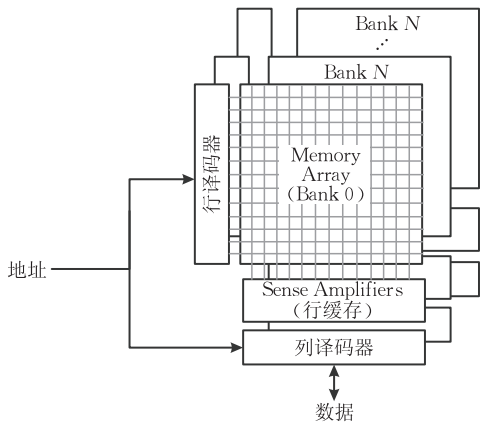



图 5　DRAM 组织结构

由于 DRAM 访存协议的复杂性、大量的时序参数、多变的芯片组织形式以及所面对的不同应用程序特征，在一个给定设计目标的限制下，如何挖掘内存控制器的设计空间，甚至可以与指令集的微结构设计相提并论[20]. 设计目标可能是最小化面积开销、最低功耗要求、最大化系统性能、最大化系统公平性，或者在这几种相互冲突的设计目标中取得权衡(Tradeoff). 其中访存请求调度算法和地址映射机制，对于内存控制器的设计和实现来说至关重要，因此本文将集中在这两个方面进行研究阐述. 我们以表 2 给出自 2006 年以来关于内存控制器优化的相关工作，并根据调度算法在不同的系统层次上将其分成三大类：后端调度、前端调度以及进程调度；对于地址映射机制，根据不同程序的行为特征，程序数据在物理地址中的位置是否可变，可将其归结为静态和动态的方法. 本文将在后面各小节中分别作进一步阐述. 另外，随着内存芯片的容量、工作频率和带宽的进一步增加，在某些大型系统中 DRAM 内存消耗的功耗，甚至超过处理器核本身的功耗. 内

存控制器的另一个优化目标是降低 DRAM 功耗. 最近几年关于 DRAM 存储系统的低功耗研究主要体现在两个方面:(1)采用片上访存调度或者存储管理的办法,借助 DRAM 存储系统的低功耗模式[21],或者通过调度的方式进行动态电压和频率调整节能[22-23];(2)通过改变 DIMM 结构来达到降低功耗的目的[24].

**表 2 内存控制器优化技术比较**

| 发表年份 | 研究者 | 优化目标 | 优化方式 |
|---|---|---|---|
| 2006 | Nesbit 等人[25] | 性能 | FQ 算法 |
| 2007 | Mutlu 等人[26] | 公平性 | STFM 算法 |
| 2008 | Ipek 等人[27] | 性能 | 自学习 |
| 2008 | Mutlu 等人[28] | 公平性和性能 | PAR-BS |
| 2010 | Kim 等人[29] | 扩展性 | ATLAS 算法 |
| 2010 | Kim 等人[30] | 公平性和性能 | TCM 算法 |
| 2010 | Ebrahimi 等人[31] | 公平性 | FST 算法 |
| 2011 | Ebrahimi 等人[32] | 多线程性能 | 并行调度 |
| 2011 | Muralidhara 等人[33] | 性能 | 通道分区 |
| 2011 | Kaseridis 等人[34] | 性能 | 映射和调度 |
| 2012 | Ausavarungnirun[35] | 异构系统性能 | SMS 调度 |
| 2012 | Jeong 等人[36] | 性能 | Bank 分区 |
| 2013 | Ghose 等人[37] | 性能 | Load 预测 |

### 3.1 访存请求调度算法

访存请求调度单元是内存控制器的核心部件,也是提高多核处理器内存系统性能的关键模块. 长期以来受内存芯片结构特征和应用程序访存特征的限制,内存系统一直面临着理论峰值带宽高,而实际有效带宽利用率低的问题. 访存请求调度是在不同的目标指导下,对访存请求进行重新排序的过程. 调度对象是内存控制器队列中的所有请求.

当来自多核的访存请求进入内存控制器后,请求将被缓存在单个或者多个队列中,并被转换成一系列状态和数据. 请求队列可能有多种组织方式,一种方式是每个 Bank 分配一个独立的队列,如图 6 所示. 另一种方式是对所有 Rank 和 Bank 只维护一个全局的请求队列,在这种队列结构的组织下,需要采用较复杂的硬件逻辑来执行请求的重新排序. DRAM 内存控制器可以依据许多因素对队列中的访存请求进行调度,包括请求的优先级、给定请求的可获得的资源数、请求的 Bank 地址、请求的到达时间或者访问历史等信息.

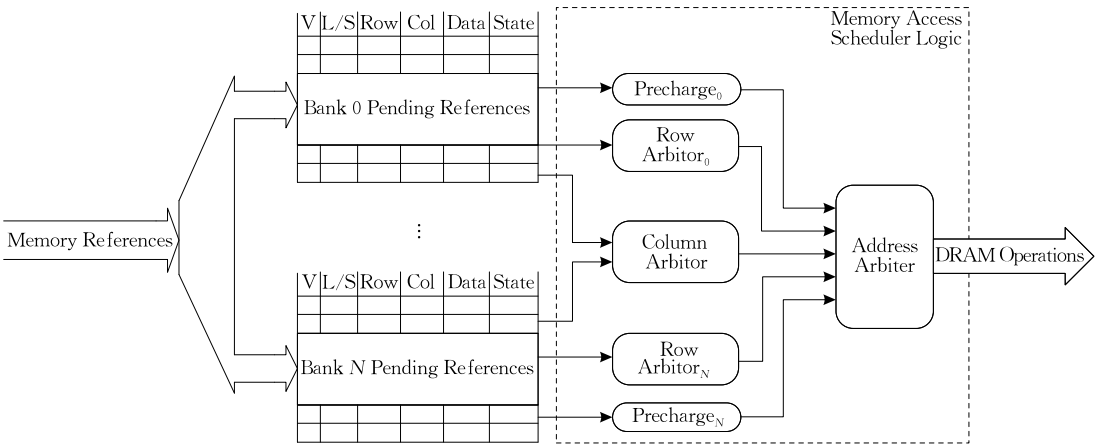


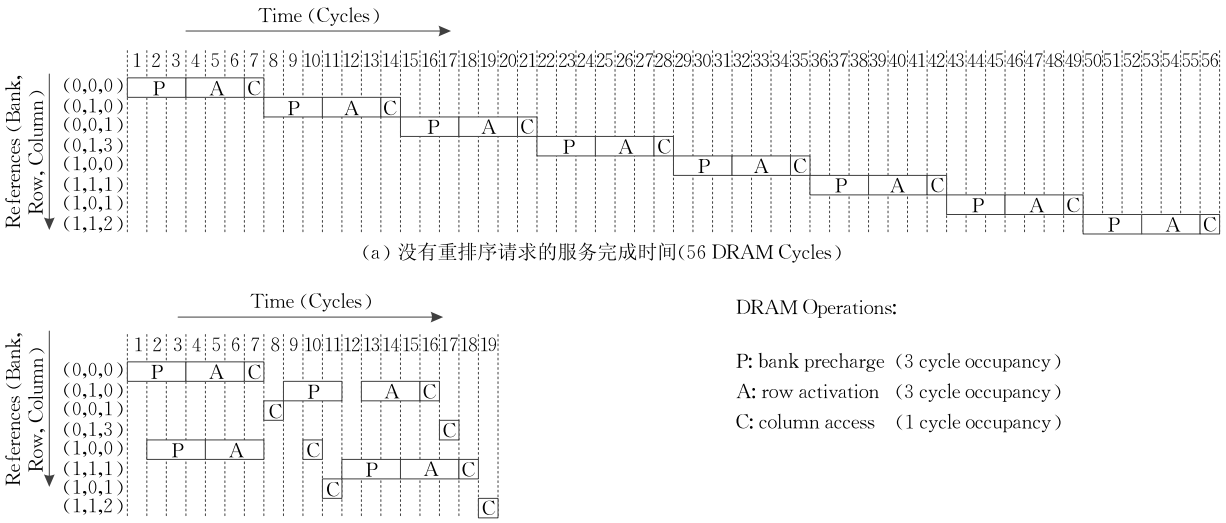

图 6 访存请求调度器结构图

#### 3.1.1 后端调度算法

早期的访存调度算法大多面向单核处理器,主要优化目标是减少访存延迟,提高带宽的利用效率,通常所采用的方法是挖掘访存请求的 Bank 级并行度和 Row Buffer 局部性. 它需要在一定时间周期内寻找局部最优的访存序列. 由于这一时期的访存调度策略主要面向 DRAM 的 3D 结构特点进行优化,我们将此类研究工作称之为后端访存调度算法研究.

对于处在内存控制器访存队列中的访存请求,访存调度器会根据 DRAM 访问协议(如 DDR3 协议)以及资源限制情况选择一部分访存请求进行响应. 最简单的调度顺序是先来先服务 FCFS(First Come First Serve),访存请求严格按照到达时间的先后顺序被调度执行,并且多个访存操作之间不能流水,后一个访存请求必须等前一个访存请求的所有操作结束之后才能开始,因此这种策略会使 DRAM 的 Bank 级并行性得不到利用.

Rixner 等人[38]提出了行缓存命中优先的调度策略 FR-FCFS(First Ready-First Come First Serve),如图 7 所示. 这种调度策略可在满足资源和时序的限制下,同时发掘 Bank 级并行性和 Row Buffer 局部性. 访存请求可以被乱序调度,且发送到不同 Bank 的访存请求可以流水执行,另外当访存请求所需访问的数据就在当前行缓存时,这些访存请求将被赋予更高的优先级并被优先调度执行,以提高行缓存命中率来降低访存的延迟. 但这种调度没有考虑应用的需求,是一种局部的调度优化策略.




(a) 没有重排序请求的服务完成时间(56 DRAM Cycles)

(b) 有重排序请求的服务完成时间(19 DRAM Cycles)

图 7　是否有重排序请求的两种服务完成时间

Hur 等人[39]认为 DRAM 调度算法不应仅采用贪婪式的局部调度，还应该依赖于应用的需求，并根据某些特定反馈信息来选择合适的调度算法. 他们提出一种根据访存请求的历史信息进行决策的自适应访存调度算法，调度器不再维护一个统一的访存请求队列，而是将读写请求分开并分别为其维护一个独立的访存队列. 仲裁器根据访存的历史信息以及两个队列中读写请求的比例来决定下一个被调度的访存请求是读还是写. 这种考虑读写平衡的调度算法为访存请求调度器提供了一种新的设计参考，该方法需要与 FR-FCFS 调度方法结合起来才能获得比较好的结果.

Lee 等人[40]提出了预取(Pretech)感知的访存调度算法，他们认为如果一个时间段内无用的预取太多，那么由预取触发的访存请求应该比正常访存请求的优先级低，这样使有效的访存请求能更快被响应. 而在预取效果好时，预取请求应该与普通读写访存请求具有相同的调度优先级，这样能达到隐藏延迟的效果. 该算法需要在处理器内增加一些结构来记录预取的有效性. 另外如果一个预取请求长时间没有被响应，而且又被后来的正常读写请求所替代，那么这个预取请求将是无用的，需要把这个预取请求从访存队列中删除.

内存控制器中的这些调度算法可以显著地改善 DRAM 的行缓存局部性，提高 Bank 的并行度，从而降低处理器的阻塞时间，并增强预取的效率. 但这一时期针对的主要是单核处理器，所采用的优化决策都是基于当前看到的或历史的访存序列，当处理器发展到多核时代，片上集成的功能部件越来越多时，访存调度研究开始更多的关注多个访存源之间的调度.

### 3.1.2　前端调度算法

多核环境下 DRAM 会同时接收来自于多个线程或者多个进程的访存请求，这些访存请求会进入内存控制器竞争内存资源，一个线程的访存请求不仅可以推迟其他线程访存请求的响应时间，也会破坏其他线程访存请求的行缓存局部性. 因此 DRAM 的内存控制器调度必须考虑如何控制多核环境下访存请求的响应顺序. 最大化吞吐率(Throughput)和最大公平性(Fairness)，是目前前端调度算法所要达到的两大主要目标. 吞吐率指的是单位时间内完成的访存请求数目. 公平性指的是线程共享运行相比于线程单独运行所造成的减速比(Slowdown)差异问题，公平性问题也是服务质量(Quality of Service)问题.

Zhu 等人[41]认为在同时多线程处理器 SMT (Simultaneous Multi-Threading)中，随着线程数目的增多，内存系统的访存压力越来越大，导致 Bank 竞争加剧，通过改善 Row Buffer 局部性来提高性能变得越来越困难，而通过线程感知的调度策略则可以有效改善性能. 最少访存请求优先原则认为如果访存队列中来自于某个进程的访存请求数量少于其他同时执行的进程，那么该进程的访存请求应该获得较高的优先级. 这是因为一旦响应了该进程的少数几个访存请求，该进程就可以继续往下执行，同时对其他进程延迟的影响也比较小.

针对存储级并行性对多核系统性能影响的发现，Mutlu 等人[28]提出了并行性感知的访存请

求批量调度方式 PAR-BS(Parallelism-Aware Batch Scheduling).这种调度方式包括批量调度和并行性感知两个方面.批量调度指的是对访存队列中来自不同进程的访存请求按到达时间的先后顺序进行分组,前一组的所有访存请求的优先级都高于后一组的所有请求,但每一个分组内部的所有访存请求可以按照普通的访存请求调度策略进行调度.而并行性感知指的是调度时考虑每一个进程的 Bank 级并行性,尽可能使同一个进程发送到不同 Bank 的访存请求被并行执行.批量调度不但为并行性感知调度方式提供了基础,而且批量调度的分组大小可以控制性能的公平性,因为可以采用轮询的方式响应各个进程的分组,而同一个分组内部的调度策略可以尽可能地考虑提高性能,这样能够兼顾系统性能与公平性.图 8 展示了 PAR-BS 的调度过程.

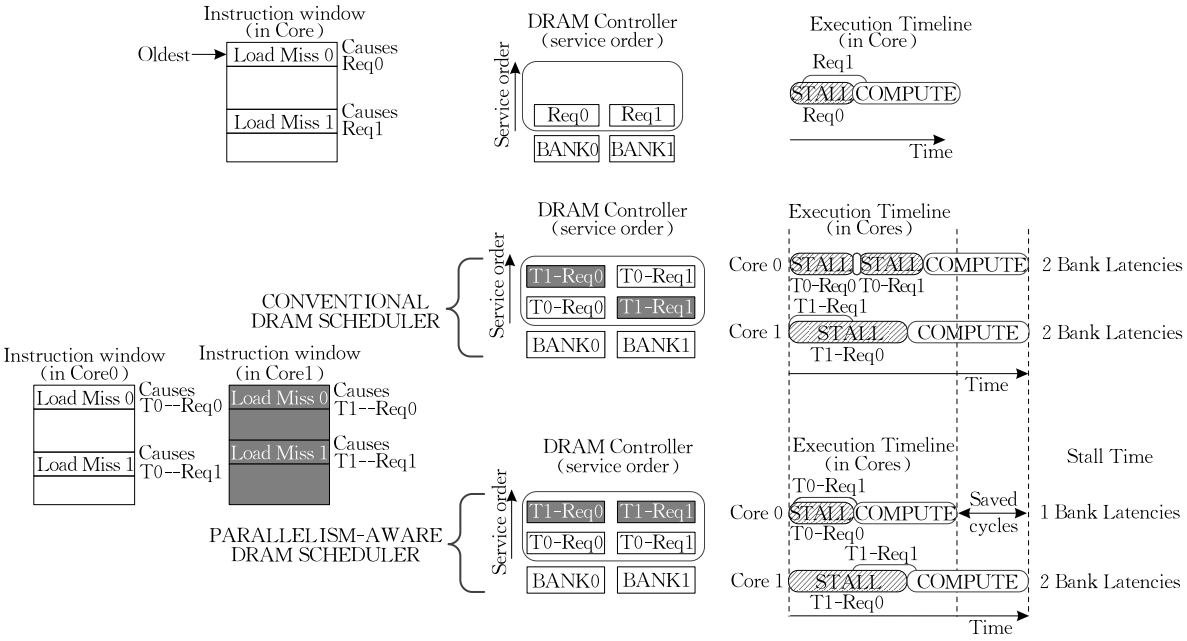



图 8 增加并行感知的 DRAM 访存请求调度模型

在批量调度方式中,分组的大小是一个重要的参数.如果分组太大,那么分组内采用的 FR-FCFS 调度策略会导致行缓存局部性好的程序比局部性较差的程序得到更多的访存响应,从而破坏系统公平性.而如果分组太小又容易破坏程序的行缓存局部性.因此分组的大小需要权衡系统性能与公平性这两个方面.Mutlu 等人[28]通过实验选择了一个经验性值作为分组大小.

通常情况下,为了提高内存系统的吞吐率,一般优先调度延迟敏感的线程.因为延迟敏感的线程以计算为主,访存量小,优先调度对系统的公平性影响不大.而带宽敏感型线程对内存系统公平性产生较大影响.Kim 等人首次把区分不同线程的访存行为作为优化的主要手段,在运行时将并行的线程划分为访存密集和访存非密集两类,并始终赋予后者较高的优先级,被称为线程簇调度算法[30] TCM (Thread Cluster Memory scheduling).与 PAR-BS 相比,TCM 更为有效地提高了系统的吞吐率,因为访存非密集型的程序对系统的吞吐率影响更为显著.另外 Kim 等人认为多核处理器平台访存请求的到达规律符合 Pareto 随机分布,所以采用排队论的最小获得服务线程优先的思想,提出了 ATLAS 算法[29].该调度算法在每个调度周期内按照每个线程已经获得的带宽大小,对线程进行排序,并赋予获得最小带宽线程的访存请求最高的优先级.

Ipek 等人[27]指出依靠一些专家根据影响性能和公平性的一些要素制定调度策略的方法无法预计调度决策的长期效果,而且这种方法无法从已经完成的调度决策中汲取经验.因此提出通过机器学习的方法来设计一个可以自动学习,并且可以灵活适应执行环境变化的访存调度请求策略.采用强化学习的机器学习方法,Ipek 描绘了强化学习的过程,学习主体(调度决策机制)会与周围的环境执行多次交互,每次交互学习主体都会感知环境的状态,并且决定自身的行动.学习主体的行动将改变环境的状态,环境的新状态将作为学习主体的下一次输入,同时环境会作相应的改变.学习主体就在重复的交互过程中学习如何才能达到状态到行动的最佳映射,

使得学习主体从环境中获得的特定回馈最大.

3.1.3　进程调度算法

从某种意义上说，进程调度应该被看作系统软件的一部分而不是由内存控制器所控制的，但由于进程的行为对于存储系统的使用效率具有重要影响，所以在内存控制器的访存调度策略中考虑进程的行为特征进行访存请求的调度优化也会影响存储系统的效率. 多个进程的并行执行虽然可以达到更高的带宽利用率，却容易导致超线性的性能降低. Zhang[42]提出了多核环境下基于程序访存阶段的粗粒度进程调度策略，通过进程调度，程序执行过程中的访存总量虽然没有改变，但是程序对访存带宽的突发式访存得到了“平坦化”，从而减少算法部件的等待时间，提高系统吞吐率. Zhang 等人构造了识别程序所处访存阶段的 PIN 工具，用来识别程序访存阶段的边界，他们还提出了考虑程序全局访存行为的独立进程调度算法，实现了基于程序访存阶段的粗粒度进程调度. 带宽感知的进程调度，是一种相对有效和容易实现的用于缓解带宽竞争的方式.

另外，Xu 等人[43]通过带宽敏感线程调度手段较为有效地缓解了对带宽的不恰当竞争而引发的若干问题，进而提高了整体吞吐率. 具体做法是，通过性能监控单元 PMU(Performance Moniter Unit)监视并发执行的每个进程的访存带宽，并利用进程调度的手段使整个系统的带宽利用率趋于“均衡”(即当前工作集的平均带宽需求). 这一做法源于这样一个观察：带宽的利用率在 10 ms 的时间片内(Linux 进程调度的时间片是 100ms)仍然有较大的波动，如果调度算法在某一个时间段内使带宽的利用达到峰值(Peak Bandwidth)，那么将会引发超线性的性能下降. 因此，该调度算法将调度的时间片减小，并使得带宽利用率始终维持在某个“均衡”的状态，这一做法在真实的系统上可以取得平均近 5%的性能提升.

多核处理器对共享内存资源的竞争是导致系统性能下降的主要原因，并且使得性能变得愈发不可预测，并在这类系统上产生更为严重的服务质量问题，Xu 等人[44]针对 QoS 问题，提出一种进程调度策略来改善系统公平性. 其策略是对于并行运行的多个程序，保证相同权重的应用具有相同的性能减速比. 基本方法是在系统运行过程中，监控所有程序的性能变化. 当发现一个程序相对于其他进程来说，遭遇到更多的减速并积累了更多需要完成的工作时，便分配较多的 CPU 时间片给它，从而改善系统的公平性. Xu 等人的方法可以赋给不同的进程以不同的权重，并提供一个有效和健壮的调节器允许 OS 自由地调整公平性和吞吐率.

### 3.2　地址映射优化

影响内存系统延迟和带宽维持的因素有很多，除了访存请求调度之外，另一个可以直接影响内存系统性能的是地址映射. 地址映射指的是一个给定的物理地址，如何将其分片划分来确定 Channel、Rank、Bank、Row 以及 Column 在内存地址上的位置. 地址映射的优化目标是在提供一定 Bank 间并发度的情况下降低 Bank 内的行缓存冲突. 如果地址映射与程序的运行时访存行为不相匹配，例如连续的访存请求被映射到相同 Bank 的不同行中，就会造成 Bank 冲突. 我们将用于消除 Bank 冲突的方法分成两类：静态的方法和动态的方法.

静态的方法主要集中在早期的地址映射研究当中. 存储体斜排(Skewing)技术在 20 世纪 70 年代已经被提出用以解决多个存储体中的访存冲突问题[45-46]. 针对斜排技术，Gao 等人[47-48]在质数内存系统下引入孙子定理来解决 Bank 冲突问题. Frailong 等人[49]提出按位异或函数(XOR)进行地址映射缓解存储体访问冲突. XOR 映射方式的好处是易于在硬件中实现且开销很低. 多种 XOR 地址映射方式被提出以解决各种常用的地址访问模式中的存储体访问冲突问题[49-54]. 对于其他的存储层次，Liu 等人[55]提出采用 XOR 地址映射的方法减少 Cache 访问中的行访问冲突问题，Zhang 等人[56]提出采用 XOR 地址映射方法减少存储体访问中的行缓存(Row Buffer)冲突问题. Hsu 和 Smith[57]在具有 Cached DRAM 的向量超级计算机系统中提出并研究了内存的交叉存储(Interleaving)技术，该技术可以实现提高数据局部性同时避免 Bank 冲突的目标. 早期的地址映射方法通常是针对数据结构中的访问冲突而提出的，而处理器核数及可并行执行的线程个数的不断增多，对地址映射提出了新的挑战，静态地址映射方法已经无法满足要求. 动态的地址映射方法，可以同时避免线程内以及线程间的 Bank 冲突. 例如，Impulse[58]内存系统通过引入一层新的影子(Shadow)地址空间，并在内存控制器中实现从影子地址到物理地址的动态重映射来改善内存性能，但是该方法需要修改应用程序和操作系统以改变数据最终在 DRAM 中的存储位置. Mi 等人[59]和 Liu 等人[60]通过页着色技术对这一问题进行了进一步的研究.

### 3.2.1 静态方法

页交叉存储(Page Interleaving)是一种典型的静态地址映射方式.图 9 展示的是我们通过 HMTT 卡[61]测得的 Intel 服务器真实的地址映射.从图中可以看出,这种地址映射方式将内存地址空间按页面大小分配到不同 Bank 的相同行中.目的是使访问同一个页面内连续地址的访存请求,被映射到同一个 Bank 的相同行中增加行缓存局部性,而使相隔距离超过一个 DRAM 页面大小的访存请求尽量映射到不同的 Bank 上,以增加 Bank 级并行度.

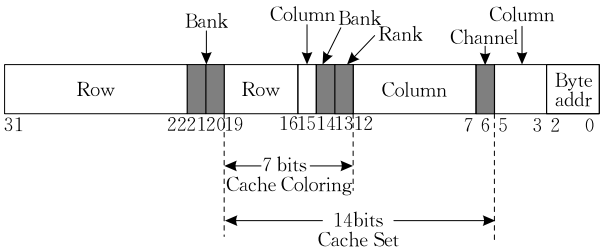



图 9 Intel Xeon 5645 服务器真实地址映射(双通道,32 个 Bank,每个 Bank 容量 128 MB)

页交替对地址连续的访存请求会产生连续的行缓存命中,不过这种地址映射方式有一个缺点是没有考虑 Cache 对访存请求序列的影响.Zhang 等人[56]在分析了大量的行缓存失效的情况之后发现,行缓存冲突的产生有 3 个原因:L2 Cache 冲突失效、L2 写回以及特定的程序访存模式.Cache 冲突失效以及 Cache 写回在这种地址映射中,必然导致 DRAM 的行缓存冲突.下面以 Cache 写回为例说明这种映射方式的缺点:假设 Cache 写回请求 $W$ 是由读请求 $R$ 触发的,那么 $R$ 和 $W$ 肯定被映射到相同的 Cache 块.所以它们的 Cache Set 索引一定相同,而 $R$ 和 $W$ 访问的是不同的物理地址(否则不会产生写回),那么它们的 Cache Tag 一定不同.通常末级共享缓存的大小除以相联度之后的值要大于 DRAM 的页大小,所以 Bank 将会是 Cache Set 中的一部分,而 Cache Tag 往往是 Row 索引的一部分,这就意味着 $R$ 和 $W$ 访问的是同一个 Bank 内部的不同 Row,所以 $R$ 和 $W$ 这两个连续的访存请求必然导致行缓存冲突.

针对这种情况,Zhang 等人[56]提出一种新的地址映射方式:XOR 地址映射方式.XOR 是一种常用的用于散列地址的异或方法,Liu 等人[55]曾提出基于 XOR 的方法在缓存-主存之间进行地址映射.如图 10 所示,通过物理地址中 Bank 索引对应的位与 Cache Tag 的低位进行 XOR 运算,结果作为新的 Bank 索引,使得原来映射到相同 Bank 的不同 Row 的地址,被随机分布到不同的 Bank 上,这样可以显著减少由于 Cache 的冲突失效和 Cache 写回引起的行缓存冲突.引入 XOR 地址映射使 DRAM 的地址映射与 Cache 地址映射不对称,从而消除某种程度的 Row Buffer 冲突.Zhang 等人还证明了 XOR 地址映射能够保持物理地址到 DRAM 的一一映射的关系.

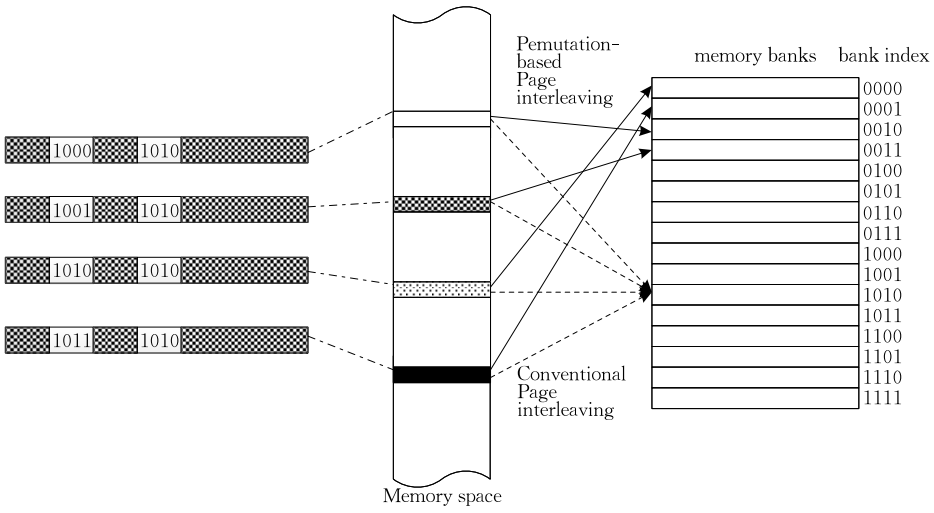



图 10 XOR 地址映射方法

### 3.2.2 动态方法

由于多道程序的访存序列之间通常不存在 DRAM 行缓存局部性,因此不同程序的访存请求往往访问 DRAM 中同一 Bank 的不同行,导致 Row Buffer 的局部性被破坏,造成行缓存冲突.不加约束的竞争会使程序的执行时间难以预测,同时造成系统资源浪费,降低系统的吞吐率.Mi 等人[59]认为通过内存控制器调度的方式很难解决这个问题,因为

调度的作用范围受到指令窗口和访存请求队列长度的限制，不能彻底消除程序间的竞争造成的行缓存冲突. Mi 提出了多道程序间 Bank 划分的思想，他通过页着色的方式(依据物理地址位中的 Bank 位)将不同进程分配到具有不同颜色的 Bank 上，实现不同进程对 Bank 访问的隔离，以避免在 Bank 上相互干扰和冲突.

Liu 等人[60]进一步在真实多核机器上，采用页着色技术，通过纯软件的方式实现了 Bank 级划分机制 BPM(Bank-level Partition Mechanism). 基于 BPM，他们做了广泛的实验，综合所有随机产生的工作集测试数据，实验结果表明能得到平均 4.7% 的性能提升(最高达 8.6%)，同时降低 4.5%(最高达 15.8%)的不公平性，这主要是通过减少平均 15%的 Bank 干扰(Row Buffer Miss 率的降低)带来的. 此外由于行缓存命中访问的增加，导致内存芯片使用效率的提升，这带来 5.2%的内存系统能耗的节省. 从以上地址映射的发展来看，动态的地址映射技术是多核系统发展的必然趋势.

## 4　总结与展望

共享缓存和 DRAM 内存系统是目前制约多核系统性能的关键因素，随着处理器核数的快速增长，来自不同核不同线程的访存请求对有限的共享内存资源产生越来越激烈的竞争，因此挖掘和提高共享内存资源利用效率具有重要的理论价值和实际意义. 本文主要在共享缓存分区、内存请求调度以及地址映射三方面对共享内存资源分配和管理研究进行了综述，并给出了未来的发展趋势.

在第 2 节中，本文从系统吞吐率和系统公平性角度介绍了一系列对共享缓存分区的相关研究、算法和实现机制. 然而近十年的研究发现，共享缓存分区并不总是有效，因为 Cache 本身会自动和动态调整每个程序的资源分配，而共享缓存分区却限制程序对其他分区中缓存资源的利用. 本文认为虽然分区在一定程度上解决了共享缓存干扰和污染的问题，但是随着多核/众核时代的到来，一些算法的复杂度和硬件开销也将大幅增长，导致其失去实用性和可扩展性. 另外，处理器核数和线程数规模进一步增大，共享缓存容量也呈现出增长的趋势，然而这并未导致进程/线程对缓存需求的进一步增加，同时运行的多道程序以及多线程程序，在云计算和大数据时代展现出越来越明显的访存行为特征上的差异，比如 Web 服务、搜索引擎、视频服务、生物计算等应用. 本文认为基于工作集特征，对缓存容量需求以及带宽需求进行分类，以及轻量级的快速硬件分区机制将是未来的一个发展趋势.

在第 3 节中，本文从访存请求调度算法和地址映射角度介绍了一系列对 DRAM 内存系统的相关研究. 访存请求调度在本质上是改变内存控制器对请求的服务顺序，以更适应现代 DRAM 内存的存储结构并在不同系统优化目标之间做出权衡，比如服务质量、实时性要求或最大化吞吐率等. 访存请求调度越来越趋向从全局了解不同线程的需求. 应用级或者结构级的信息将参与调度，比如将线程按照访存密集、非密集分别调度；或者将结构级信息，比如 load/store 指令参与控制器调度；或者通过 MSHR 进行流速控制等方式. 不过随着处理器核数的增长，调度算法的作用范围越来越受到指令窗口和访存请求队列长度的限制.

地址映射相对于调度算法可以在很大程度上缓解线程之间对 DRAM 资源的争抢，但静态的映射方法不适应多核时代所带来越来越多的线程干扰情形，最大化 Bank 级并行和 Row Buffer 命中难以同时满足. 在线程数量较少的情况下，可以通过 Bank 级并行增加性能，但在大量挖掘线程级并行的多线程处理器中，这种最大化单线程性能的地址映射方式，会造成线程的相互干扰，减少 Row Buffer 命中，降低内存带宽和性能. 通过限制单线程可用的 Bank 数量，虽然可能降低单线程性能，但在大量线程级并行的应用中，却可有效降低线程间对 Bank 的相互干扰，会提升系统整体吞吐率，进而提高系统整体性能. 本文认为以往面向充分挖掘 Bank 级并行的映射方式，将会被动态以及面向线程级 Bank 划分的映射方式所替代.

总之，随着处理器核数的增加，多核系统会产生大量的访存请求，对系统中共享内存资源的访问竞争加剧. 在摩尔定律的推动下，存储容量依然在快速增长，Row Buffer 也有增大的趋势. 但是大量线程相互干扰所导致的 Row Buffer 局部性变差已成为影响内存性能的一个重要问题. 增加系统中内存通道或 Bank 数量是提高内存性能的有效方法之一，但这种方式会带来较高的成本以及受到处理器引脚(Pin)数量的限制. 本文认为独立异构的 DRAM 微结构技术、高速串行总线、消息内存以及板上缓存，已成为 DRAM 内存系统的发展趋势和重要的研究方向.

**致 谢** 感谢评审专家付出的辛勤劳动和对本文提出的中肯意见!

## 参 考 文 献


[1] Wulf W A, Mckee S A. Hitting the memory wall: Implications of the obvious. ACM SIGARCH Computer Architecture News, 1995, 23(1): 20-24

[2] Asanovic K, Bodik R, Catanzaro B C, et al. The landscape of parallel computing research: A view from berkeley. EECS Department, University of California, Berkeley: Technical Report: UCB/EECS-2006-183, 2006

[3] Qureshi M K, Patt Y N. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches//Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture. Orlando, USA, 2006: 423-432

[4] Kim S, Chandra D, Solihin Y. Fair cache sharing and partitioning in a chip multiprocessor architecture//Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques. Juan-Les-Pins, France, 2004: 111-122

[5] Stone H S, Turek J, Wolf J L. Optimal partitioning of cache memory. IEEE Transactions on Computers, 1992, 41(9): 1054-1068

[6] Suh G E, Devadas S, Rudolph L. A new memory monitoring scheme for memory-aware scheduling and partitioning//Proceedings of the 8th International Symposium on High-Performance Computer Architecture. Cambridge, USA, 2002: 117-128

[7] Suh G E, Rudolph L, Devadas S. Dynamic partitioning of shared cache memory. The Journal of Supercomputing, 2004, 28(1): 7-26

[8] Hassidim A. Cache replacement policies for multicore processors//Proceedings of the 1st Symposium on Innovations in Computer Science. Beijing, China, 2010: 501-509

[9] Fox B. Discrete optimization via marginal analysis. Management Science, 1966, 13(3): 210-216

[10] Ranganathan P, Adve S, Jouppi N P. Reconfigurable caches and their application to media processing//Proceedings of the 27th International Symposium on Computer Architecture. Vancouver, Canada, 2000: 214-224

[11] Varadarajan K, Nandy S K, Sharda V, et al. Molecular caches: A caching structure for dynamic creation of application-specific heterogeneous cache regions//Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture. Washington, USA, 2006: 433-442

[12] Chiou D, Jain P, Devadas S, Rudolph L. Dynamic cache partitioning via columnization//Proceedings of the 37th Annual Design Automation Conference. Los Angeles, USA, 2000: 416-419

[13] Lin J, Lu Q, Ding X, et al. Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems//Proceedings of the 14th International Conference on High Performance Computer Architecture. Salt Lake City, USA, 2008: 367-378

[14] Xie Y, Loh G H. PIPP: Promotion/insertion pseudo-partitioning of multi-core shared caches//Proceedings of the 36th Annual International Symposium on Computer Architecture. Austin, USA, 2009: 174-183

[15] Iyer R. CQoS: A framework for enabling QoS in shared caches of CMP platforms//Proceedings of the 18th Annual International Conference on Supercomputing. Malo, France, 2004: 257-266

[16] Sanchez D, Kozyrakis C. Vantage: Scalable and efficient fine-grain cache partitioning//Proceedings of the 38th Annual International Symposium on Computer Architecture. San Jose, USA, 2011: 57-68

[17] Cook H, Moreto M, Bird S, et al. A hardware evaluation of cache partitioning to improve utilization and energy-efficiency while preserving responsiveness//Proceedings of the 40th Annual International Symposium on Computer Architecture. Tel Aviv, Israel, 2013: 308-319

[18] Song Feng-Long, Liu Zhi-Yong, Fan Dong-Rui, Zhang Jun-Chao, Yu Lei. An implicitly dynamic shared cache isolation in many-core architecture. Chinese Journal of Computers, 2009, 32(10): 1898-1904(in Chinese)
(宋风龙, 刘志勇, 范东睿, 张军超, 余磊. 一种片上众核结构共享 cache 动态隐式隔离机制研究. 计算机学报, 2009, 32(10): 1898-1904)

[19] Fan D, Zhang H, Wang D, et al. Godson-T: An efficient many-core processor exploring thread-level parallelism. IEEE Micro, 2012, 32(2): 38-47

[20] Jacob B, Ng S, Wang D. Memory Systems: Cache, Dram, Disk. Burlington, USA: Morgan Kaufmann, 2008

[21] Yoon D H, Jeong M K, Erez M. Adaptive granularity memory systems: A tradeoff between storage efficiency and throughput. ACM SIGARCH Computer Architecture News, 2011, 39(3): 295-306

[22] Deng Q, Meisner D, Ramos L, et al. Memscale: Active low-power modes for main memory. ACM SIGPLAN Notices, 2011, 46(3): 225-238

[23] Deng Q, Meisner D, Bhattacharjee A, et al. Coscale: Coordinating CPU and memory system DVFS in server systems//Proceedings of the 45th International Symposium on Microarchitecture (MICRO). Washington, USA, 2012: 143-154

[24] Yoon D H, Chang J, Muralimanohar N, Ranganathan P. BOOM: Enabling mobile memory based low-power server DIMMs//Proceedings of the 39th Annual International Symposium on Computer Architecture (ISCA). Washington, USA, 2012: 25-36

[25] Nesbit K J, Aggarwal N, Laudon J, Smith J E. Fair queuing memory systems//Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture. Washington, USA, 2006: 208-222

[26] Mutlu O, Moscibroda T. Stall-time fair memory access scheduling for chip multiprocessors//Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture. Washington, USA, 2007: 146-160

[27] Ipek E, Mutlu O, Martínez J F, Caruana R. Self-optimizing memory controllers: A reinforcement learning approach//Proceedings of the 35th International Symposium on Computer Architecture. Washington, USA, 2008: 39-50

[28] Mutlu O, Moscibroda T. Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems//Proceedings of the 35th Annual International Symposium on Computer Architecture. Beijing, China, 2008: 63-74

[29] Kim Y, Han D, Mutlu O, Harchol-Balter M. ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers//Proceedings of the 16th International Symposium on High Performance Computer Architecture (HPCA). Bangalore, India, 2010: 1-12

[30] Kim Y, Papamichael M, Mutlu O, Harchol-Balter M. Thread cluster memory scheduling: Exploiting differences in memory access behavior//Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). Washington, USA, 2010: 65-76

[31] Ebrahimi E, Lee C J, Mutlu O, Patt Y N. Fairness via source throttling: A configurable and high-performance fairness substrate for multi-core memory systems//Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems. Pittsburgh, USA, 2010: 335-346

[32] Ebrahimi E, Miftakhutdinov R, Fallin C, et al. Parallel application memory scheduling//Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture. New York, USA, 2011: 362-373

[33] Muralidhara S P, Subramanian L, Mutlu O, et al. Reducing memory interference in multicore systems via application-aware memory channel partitioning//Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture. New York, USA, 2011: 374-385

[34] Kaseridis D, Stuecheli J, John L K. Minimalist open-page: A DRAM page-mode scheduling policy for the many-core era//Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture. New York, USA, 2011: 24-35

[35] Ausavarungnirun R, Chang K K-W, Subramanian L, et al. Staged memory scheduling: Achieving high performance and scalability in heterogeneous systems//Proceedings of the 39th International Symposium on Computer Architecture. Portland, Oregon, 2012: 416-427

[36] Jeong M K, Yoon D H, Sunwoo D, et al. Balancing DRAM locality and parallelism in shared memory CMP systems//Proceedings of the 18th International Symposium on Hgh Performance Computer Architecture. Washington, USA, 2012: 1-12

[37] Ghose S, Lee H, Martínez J F. Improving memory scheduling via processor-side load criticality information//Proceedings of the 40th Annual International Symposium on Computer Architecture. Tel-Aviv, Israel, 2013: 84-95

[38] Rixner S, Dally W J, Kapasi U J, et al. Memory access scheduling//Proceedings of the 27th Annual International Symposium on Computer Architecture. Vancouver, Canada. 2000: 128-138

[39] Hur I, Lin C. Adaptive history-based memory schedulers//Proceedings of the 37th International Symposium on Microarchitecture. Portland, Oregon, 2004: 343-354

[40] Lee C J, Mutlu O, Narasiman V, Patt Y N. Prefetch-aware DRAM controllers//Proceedings of the 41st IEEE/ACM International Symposium on Microarchitecture. Washington, USA, 2008: 200-209

[41] Zhu Zhichun, Zhang Zhao. A performance comparison of DRAM memory system optimizations for SMT processors//Proceedings of the 11th International Symposium on High-Performance Computer Architecture. Washington, USA, 2005: 213-224

[42] Zhang Chao. Analysis of Program Behavior in the Parallel Program Services[Ph. D. dissertation]. Institute of Computing Technology, Chinese Academy of Sciences, Beijing, 2008(in Chinese)
(张超. 服务于程序并行的程序行为分析研究[博士学位论文]. 中国科学院计算技术研究所, 北京, 2008)

[43] Xu D, Wu C, Yew P-C. On mitigating memory bandwidth contention through bandwidth-aware scheduling//Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques. Vienna, Austria, 2010: 237-248

[44] Xu D, Wu C, Yew P-C, et al. Providing fairness on shared-memory multiprocessors via process scheduling//Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems. London, UK, 2012: 295-306

[45] Budnik P, Kuck D J. The organization and use of parallel memories. IEEE Transactions on Computers, 1971, 100(12): 1566-1569

[46] Batcher K E. The multidimensional access memory in Staran. IEEE Transactions on Computers, 1977, 100(2): 174-177

[47] Gao Q. The Chinese remainder theorem and the prime memory system//Proceedings of the 20th Annual International Symposium on Computer Architecture. San Diego, USA, 1993: 337-340

[48] Gao Qing-Shi, Liu Zhi-Yong. A prime memory system scheme based on the Chinese remainder theorem. Journal of Computer Research and Development, 1995, 32(5): 1-7(in Chinese)
(高庆狮, 刘志勇. 一个基于孙子定理的素数存储系统方案. 计算机研究与发展, 1995, 32(5): 1-7)

[49] Frailong J M, Jalby W, Lenfant J. XOR-schemes: A flexible data organization in parallel memories//Proceedings of the International Conference on Parallel Processing. University Park, USA, 1985: 276-283

[50] Raghavendra C, Boppana R V. On methods for fast and efficient parallel memory access//Proceedings of the International Conference on Parallel Processing. Urbana-Champaign, USA, 1990: 76-83

[51] Liu Z, Li X. XOR storage schemes for frequently used data patterns. Journal of Parallel and Distributed Computing, 1995, 25(2): 162-173

[52] Liu Z, Li X, You J-H. On storage schemes for parallel array access//Proceedings of the 6th International Conference on Supercomputing. Washington, USA, 1992: 282-291

[53] Liu Z, You J-H. An implementation of a nonlinear skewing scheme. Information Processing Letters, 1992, 42(4): 209-215

[54] Liu Z, You J-H, Li X. The odd-even expansion storage scheme and its implementation issues//Proceedings of the 6th International Parallel Processing Symposium. Washington, USA, 1992: 550-557

[55] Liu Zhi-Yong, Li En-You, Qiao Xiang-Zhen. Cache memory system address mapping technology and device. CN1217505A, 1999-05-26(in Chinese)
(刘志勇, 李恩有, 乔香珍. 高速缓冲存储器系统中的地址映射变换技术与装置. CN1217505A, 1999-05-26)

[56] Zhang Z, Zhu Z, Zhang X. A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality//Proceedings of the 33rd Annual ACM/IEEE International Symposium on Microarchitecture. Monterey, USA, 2000: 32-41

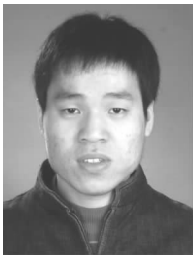
[57] Hsu W-C, Smith J E. Performance of cached DRAM organizations in vector supercomputers//Proceedings of the 25th Annual International Symposium on Computer Architecture. San Diego, USA, 1993: 327-336

[58] Carter J, Hsieh W, Stoller L, et al. Impulse: Building a smarter memory controller//Proceedings of the 5th International Symposium on High-Performance Computer Architecture. Washington, USA, 1999: 70-79

[59] Mi W, Feng X, Xue J, Jia Y. Software-hardware cooperative DRAM bank partitioning for chip multiprocessors//Ding Chen, Shao Zhiyuan, Zheng Ran, eds. Network and parallel computing. Berlin Heidelberg: Springer-Verlag, 2010: 329-343

[60] Liu L, Cui Z, Xing M, et al. A software memory partition approach for eliminating bank-level interference in multicore systems//Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques. Minneapolis, USA, 2012: 367-376

[61] Bao Y, Chen M, Ruan Y, et al. HMTT: A platform independent full-system memory trace monitoring system//Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems. Annapolis, USA, 2008: 229-240



**GAO Ke**, born in 1983, Ph. D. candidate. His research interests include multicore processor architecture and memory architecture.

**CHEN Li-Cheng**, born in 1986, Ph. D. candidate. His research interests include memory architecture and virtual machine memory management.

**FAN Dong-Rui**, born in 1979, Ph. D., professor. His research interests include many-core architecture and high-throughput processor architecture.

**LIU Zhi-Yong**, born in 1946, Ph. D., professor, Ph. D. supervisor. His research interests include high performance algorithm and architecture, parallel processing.

**Background**

With the development of multicore systems, memory access requests from different threads to shared memory system become increasingly competitive. Cache partitioning is an effective approach for mitigating the contention. Meanwhile, Modern multi-banked DRAM memory contains a row buffer that serves as a cache to memory accesses. The memory accesses that exhibit row buffer reuse will be served significantly faster. Memory controller optimizations such as memory access schedulers and optimized address mapping schemes attempt to improve and utilize row buffer locality to boost the overall system performance.

However, these existing memory systems optimizations suffer from some limitations. In this paper, we review above three important issues of shared memory resources management and analyze these optimizations technology. Cache partitioning is very sensitive to workloads so that it is not always beneficial. Memory access scheduler can only see a small number of memory transactions issued to the memory controller. Thus, its improvement on row buffer locality can be limited. Address mapping schemes cannot exploit the data access patterns in a program. As a result, they cannot enforce an ordered data mapping in memory. In the end, we overview state of the art of multicore memory system development and also propose the future research avenues.

This work is in part supported by the National Basic Research Program(973 Program) of China(No.2011CB302501), the National Natural Science Foundation of China (Nos. 61020106002, 61221062, 61332009, 61173007, and 61100013).