计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2015.20148178
52(11):2577-2588, 2015

一种缓解多线程访存干扰的 VRB 内存机制

高 珂[1,3] 范东睿[1] 刘志勇[1,2]
[1](计算机体系结构国家重点实验室(中国科学院计算技术研究所) 北京 100190)
[2](北京市移动计算和新型终端重点实验室(中国科学院计算技术研究所) 北京 100190)
[3](中国科学院大学 北京 100049)
(gaoke@ict.ac.cn)

Decoupling Contention with VRB Mechanism for Multi-Threaded Applications

Gao Ke[1,3], Fan Dongrui[1], and Liu Zhiyong[1,2]
[1](*State Key Laboratory of Computer Architecture* (*Institute of Computing Technology*, *Chinese Academy of Sciences*), *Beijing* 100190)
[2](*Beijing Key Laboratory of Mobile Computing and New Terminals* (*Institute of Computing Technology*, *Chinese Academy of Sciences*), *Beijing* 100190)
[3](*University of Chinese Academy of Sciences*, *Beijing* 100049)

Abstract Currently, the processors improve system performance by increasing the number of cores and simultaneously running threads. However, increasing the number of processor cores and threads which share the memory system will decrease the memory row-buffer hit rate (RBHR), causing more memory power consumption and longer memory access latencies. We design and develop a fine-grained victim row-buffer (VRB) memory system to solve this problem. VRB mechanism provides an additional row-buffer (VRB) which temporarily stores the expelled data due to the row-buffer (RB) conflict for a possible access in the near future. This mechanism mitigates the multi-threaded interference phenomenon and increases the reuse ratio of row-buffer data in DRAM and avoids unnecessary accesses of the array of cells, thus some row activations, precharge operations and data transmission activities can be reduced. VRB can improve system performance and power consumption while incurring minor hardware complexity. Through full-system cycle-accurate simulations of many threads applications, we demonstrate that VRB mechanism achieves an up to 17.6% (8.7% on average) system-level throughput improvement, an up to 142.9% (51.4% on average) RBHR improvement, and saves an up to 17.6% (9.2% on average) power consumption compared with an 8-core Intel Xeon server.

Key words DRAM architecture design; row buffer (RB); power consumption; multi-threaded; victim row-buffer (VRB) mechanism

摘 要 目前处理器通过持续增加核数和同时执行的线程数来提高系统性能.但是,增加共享内存的处理器核数和线程数会使得存储器中的行缓存(row-buffer, RB)命中率下降,造成存储器访问功耗增加和访存延迟增加.设计并开发了一种细粒度的 victim row-buffer(VRB)内存机制系统来解决此问题.

收稿日期:2014-11-04;**修回日期**:2015-05-28
基金项目:国家"九七三"重点基础研究发展计划基金项目(2011CB302501);国家自然科学基金项目(61020106002,61221062);NSFC 与香港 RGC 合作项目(61161160566);"核高基"国家科技重大专项基金项目(2013ZX0102-8001-001-001)
通信作者:刘志勇(zyliu@ict.ac.cn)

VRB 机制提供附加的行缓存(VRB),暂时缓存由于行缓存(RB)冲突而从行缓存(RB)逐出的数据,以备后续可能的访问.这种机制缓解了多线程冲突,增加了 DRAM 中行缓存数据的重用率,避免了不必要的内存数据阵列的访问、行激活和预充电、数据传输等电路动作,可以通过少量的硬件代价提高内存系统的性能,并节约系统的功耗消耗.通过时序精确的全系统模拟器实验,对比 8 核的 Intel Xeon 处理器,所提出的 VRB 机制可以达到最高 17.6%(平均 8.7%)的系统级吞吐率改善、最高 142.9%(平均 51.4%)的行缓存命中率改善以及最高 17.6%(平均 9.2%)的系统功耗改善.

关键词 DRAM 结构设计;行缓存;功耗消耗;多线程;VRB 机制

中图法分类号 TP302

在现代多核处理器系统中,DRAM 被多个处理器核所共享,当多个核同时产生独立的内存请求流时,交错的以及交互的访存模式经常会破坏和干扰 DRAM 系统中的行缓存局部性(row-buffer hit rate, RBHR),从而降低系统的整体性能. Udipi 等人[1]的研究显示,RBHR 在单核系统中平均超过 60%,而在 16 核系统中却降低至 35%,其原因主要来自于不同线程的访存请求对行缓存的干扰. Sudan 等人[2]在他们的研究中同样观察到了类似的 RBHR 趋势.

这种现象将给整个计算系统带来严重的问题,例如:1)大量的内存页将得不到重用,即使线程本身具有较高的行缓存局部性;2)行缓存冲突会带来额外的内存命令,如行激活(Activate)、预充电(Precharge),从而降低内存带宽的有效利用;3)行缓存数据被经常地换入和换出,导致严重的功耗消耗.

在过去 10 年中,行缓存的冲突问题已存在广泛的调查研究.其中一类有效的技术是地址映射方法,许多方法[3-6]按照线程对内存资源的不同需求,进行 DRAM Bank 或者 cache 上的划分,来达到缓解访存干扰的目的.然而这往往限制了每个线程的内存容量,导致较差的内存/缓存利用率,从而降低性能,尤其是当线程对内存的需求随时间变化时.与此同时,传统的 DRAM 存储器按位成本(cost-per-bit)考虑的设计并没有改变.这种粗粒度的内存结构虽然可以降低行缓存失效率,但行缓存干扰现象随着处理器核数/线程数的增多进一步严重,粗粒度的访存会加剧行缓存冲突带来的影响,并导致过高的系统功耗.许多微结构技术[7-9]通过动态或静态的方法来改变内存访问的粒度大小,减少行缓存干扰现象.这些技术本质上是通过改变 DRAM 的同步操作模式,来产生更细粒度的异步操作模式.虽然可以减少访存的干扰机会、提高整个系统的吞吐率,但这往往会导致内存访问延迟的增加,并限制单个线程的访存性能.

本文要解决如下 2 个重要的问题:1)多个线程交错的存储器访问方式会干扰 DRAM 中行缓存的数据复用率,从而导致性能下降(延迟增加)和功耗增加;2)粗粒度的行缓存数据访问进一步加剧了线程间访存干扰的代价.这促使我们设计并开发了一种新的 DRAM 系统——victim row-buffer(VRB)内存系统,来降低系统内存延迟并提高内存带宽的利用率. VRB 机制提供附加的行缓存(VRB)暂时缓存由于行缓存(row buffer, RB)冲突而从行缓存(RB)逐出的数据,以备后续可能的访问,提高 DRAM 行缓存数据的重用率.同时细粒度的数据存取有效地减少了不必要的数据传输,提高了整个系统的有效内存带宽.

本文的贡献主要包括:

1) 提出了一种新的 VRB 内存系统,可以提高 DRAM 行缓存的数据复用率,减轻线程间访存干扰产生的负面影响.

2) 提出了一种细粒度的 DRAM 数据访问机制,可以减少不必要的数据传输,有效地提高存储器带宽的利用率.

3) 设计和实现了一种存储器访问协议,使得上述 VRB 机制可以有效地运行,存储器的内存总线带宽可以被充分地利用.

与 8 核 Intel Xeon 服务器基准系统相比较,实验结果表明,VRB 机制可以改善系统整体吞吐率,最高达 17.6%(平均 8.7%);提高行缓存命中率,最高达 142.9%(平均 51.4%),并节省系统功耗 9.2%.

1 DRAM 系统结构

1.1 内存结构和组织方式

如图 1 所示,一个基于 DRAM 的内存芯片包含若干个 Bank 存储体,每个存储体是最小的可并行访问的内存结构,这被称作 Bank 级并行性[10].每个

存储体由二维的 DRAM 单元阵列(the array of DRAM cells)构成,它由多个行和多个列组成.列方向上的各个单元由纵向的位线(bitline)所连接,行方向上的各个单元由横向的字线(wordline)所连接.每个存储体上所有的数据读写操作都需要经过行缓存(RB)来完成.

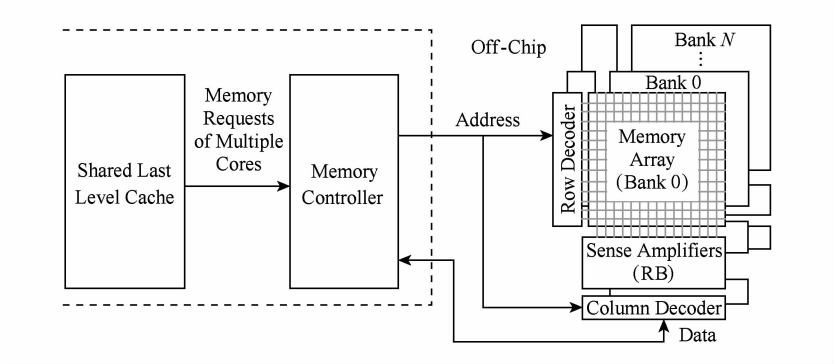

Fig. 1　Conventional DRAM organization.

图 1　传统 DRAM 的组织结构

1.2　访存数据读写过程

为了处理一个访存请求,内存控制器发送如下 3 种指令到相应的 Bank 上,每个具体的指令会在 Bank 上激发一系列的事件,即 Activate、读/写(Read/Write)和 Precharge:

1) Activate.从数据阵列中激活一个整行并把数据存入行缓存.在一个 DRAM 行被激活之前,Bank 必须处于空闲状态(precharged state).该激活指令的目的是将 DRAM 阵列中的存储单元数据读入到行缓存中,然后将数据恢复到 DRAM 阵列的存储单元中.

2) Read/Write.激活命令发出之后,内存控制器便可以发出包含列地址的读或写命令.读命令(Read)从行缓存通过数据总线将数据返回给内存控制器,并将所需数据从 DRAM 芯片放置到数据总线上;写(Write)命令是从内存控制器传输数据到具体 Bank 的行缓存中,注意相对于读操作,写操作需要额外的时间将数据恢复到存储单元中.

3) Precharge.为了要激活一个新的行,内存控制器必须首先将 Bank 重置到预充电状态.Precharge 命令发出之后,行缓存以及位线被置为预充电状态.

另外,不同的 DRAM 命令具有不同的延迟时间.如果一个命令在未被完全处理的情况下而发送另外的命令,可能出现不确定的行为.为了防止这种情况的发生,内存控制器在发送命令到具体的 Bank 上时,必须服从一系列的时序限制.这些限制定义了一个依赖于其他已发送命令的命令何时被调度.表 1 总结了 Activate,Precharge,Read,Write 命令之间最重要的时序限制.

Table 1　DDR3-1333 DRAM Timing Constrains

表 1　DDR3-1333 DRAM 的时序限制

Parameter	Value	Description
t_{CK}/ns	1.5	DRAM Clock Tick
t_{CL}/ns	15	Column Address Strobe Delay
t_{RCD}/ns	15	Row Address to Column Address Delay
t_{RP}/ns	15	Row Precharge Time
t_{RAS}/ns	36	Row Active Time
t_{RC}/ns	51	Row Cycle Time
t_{RFC}/ns	111	Refresh Row Cycle Time
t_{FAW}/ns	30	4-Page Activates Time Window
t_{REFI}/us	7.8	Refresh Interval
t_{RTP}/ns	7.5	Read to Precharge Delay
t_{WTR}/ns	7.5	Write to Read Delay
t_{RTR}/ns	1.5	Rank to Rank Delay
t_{BURST}	8 Data Beats	Burst Length
t_{RRD}/ns	6	Back-to-back Row Activations to Any Bank

2　VRB 系统设计

2.1　行缓存冲突问题

通常,处理机一次访存操作所传送的数据是一个缓存块(cache block).内存中数据被组织成多个“页”,而一个页就是 DRAM 单元陈列中的一行(row),它的容量要远远高于一个 cache 块.比如一个页为 8 KB,而一个二级 cache 的 cache 块包含 64 B.当处理机发出一个访存请求后,内存是把一个 cache 块的数据返回给处理机,这个数据是从行缓存传送到总线上去的.如果被访问的数据已经在行缓存中(即行缓存命中(row-buffer hit)),则这一个 cache

块的数据就直接从行缓存中传到总线上;若被访问的数据不在行缓存中(即行缓存不命中(row-buffer miss)),则就要重新激活 DRAM 单元阵列中包含所需的数据的一个行并把它读入到行缓存中.需要重新激活一行并把它读入行缓存中的操作时间可能不同(由于 DRAM 需要动态刷新以及上述的各种命令的时序限定),但行缓存不命中(需要重新激活一行)时的访问时间都要大大长于行缓存命中时的访问时间,而且要耗费更多的重新激活一行单元的能量.通常的例子,当行不命中时,激活一个 8 KB 的 DRAM 行并把数据预取到行缓存中,然后再从行缓存中把一个 64 B 的 cache 块数据放到总线上,这将产生额外的 Precharge, Activate 协议命令操作.同时连续的 Activate 命令受到 t_{RC}(行周期)的时序限制,它包含 t_{RAS}和 t_{RP}的时间之和.在最坏的情况下,当 N 个不同线程的请求访问相同 Bank 的不同行时,Bank 必须每次激活一个新的行,然后 Precharge 该行准备为其他请求服务.因此,最后一个请求需要经历 $N\times t_{RC}$的时间延迟,这将产生成百上千纳秒的延迟,而行命中时则只是直接把一个 64 B 的 cache 块放到总线上.我们把访问数据在行缓存中不命中的原因称为行缓存冲突,多核处理器、多个线程的同时执行均加重了行缓存冲突.

2.2 VRB 机制

本文在引言介绍了 2 种缓解线程访存干扰的微结构技术,然而它们很少同时减少 DRAM 的延迟和缓解访存干扰问题,主要将重点放在容忍访存延迟,整体上提高系统吞吐能力.本文提出一种 VRB 技术,用于缓解由于行缓存冲突而产生的访问延迟增长和访问能耗增大问题.VRB 的 DRAM 机制是通过提供附加的行缓存(VRB)暂时缓存由于行缓存(RB)冲突而从行缓存(RB)逐出的数据,以备后续可能的访问,与此同时我们在 VRB 中提供细粒度段操作机制,使得 DRAM 内存系统具有更大的并发操作能力.

2.2.1 VRB 的组织

图 2 展示的是 VRB 的组织结构.该存储器阵列的结构基本保持不变,包含多个物理 Bank 以及每个 Bank 中存在一个行缓存.VRB 的实现兼容 DRAM 工业标准并使用与 DRAM 类似的协议命令和接口.传统的内存操作分为行缓存的前端操作(Read, Write)和 DRAM 的 Bank 后端操作(Activate, Precharge).我们添加了 2 个命令,以支持引入 VRB 的操作.引入操作这些缓冲区的命令如下:

1) 预取(Prefetch)指令.它是用来传输行缓存中的数据到 VRB.当行缓存中的数据由于行缓存冲突被换出时,在进行预充电之前,此命令将被发出.一旦数据存储在 VRB 中,列存取便可以进行;而在传统 DRAM 中,重用这部分数据是不可能的.

2) 恢复(Restore)指令.当 VRB 中的数据被修改并且 VRB 数据将被替换时,需要发送 Restore 命令,用于把 VRB 中的数据恢复到 DRAM 的存储阵列(the array of cells)中.一旦数据恢复到 DRAM 的存储阵列中,VRB 缓存的数据即可被新的行缓存数据所替换.

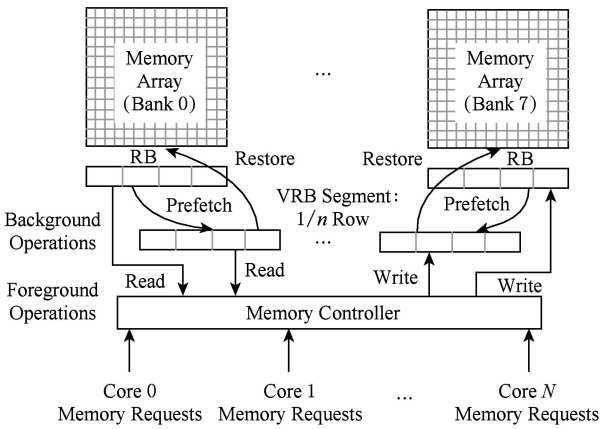

Fig. 2 VRB memory system.

图 2 VRB 内存系统

通过提供独立的 VRB 缓存,DRAM 中的列数据访问不仅可以在行缓存中进行,也可以在行缓存和/或 VRB 缓存区中进行.

2.2.2 细粒度的数据访问机制

DDRx 接口高度依赖于数据的空间局部性,出于成本考虑并不断增大每次数据存取的访问粒度.不管处理器端需要多大的数据,DRAM 端都会取得固定大小的数据量(4~8 KB),这被称作一次 DRAM 的行激活,并返回小部分数据 (64 B).当程序受到线程干扰的影响而导致空间局部性降低时,粗粒度的内存访问会降低存储器带宽的有效利用,并产生过高的功耗消耗.Yoon 等人[7]测试了同时运行的多线程程序对高速缓存块(cache block)的干扰影响,并分析了一个 64 B 高速缓存块数据在被替换之前被访问到的数据量,结果显示大多数应用程序所访问的数据不足 50%.另外,Rixner[11]也发现,较大的行缓存(RB)大小,在数据被换出之前仅重复使用其中较小部分的数据,所以粗粒度的存储器系统在访存局部性受到线程干扰的情况下,会加重浪费带宽使用并带来额外的系统功耗.在多核和多线程时代,

只有细粒度的内存访问才可以缓解内存系统中的访存干扰问题,获得更高的系统吞吐率.

本文在 VRB 上通过构建若干个段(segment)来打破这种粗粒度的内存访问,以减少行缓存冲突所带来的带宽浪费和功耗消耗影响.如图 2 所示,VRB 的段大小限制为四分之一的行缓存大小(分段越多,粒度越细).其中每个 VRB 段和行缓存按照段粒度进行全相连,暂时保留行缓存中任意一段的被替换数据,以备后续可能的访问.通过提供细粒度的段机制并结合 VRB 的使用策略,VRB 内存系统可以缓解多线程的干扰问题,并提供更大机会的访存并行度,提高内存带宽的使用和降低功耗影响.这不同于以往工作在异步模式下的窄数据总线技术,虽然提高了系统的整体吞吐率,却限制了单线程的性能,而我们的机制并不限制单线程性能.

2.2.3　VRB 协议调度策略

图 3 展示了 VRB 内存系统的协议调度过程.当最后一级高速缓存未命中发生时,内存控制器首先检查判断寄存器,确定所需要的数据是否已在 VRB 中.如果读出或写入请求在所有 VRB 缓存段中均未命中,内存控制器将像传统 DRAM 那样进行协议命令调度.我们改变了 DRAM 的协议命令调度过程,以处理行缓存冲突问题.例如,当线程 1 和线程 2 发生线程干扰时,在线程 1 的行数据被预充电之前,内存控制器将向对应的 Bank 发送 Prefetch 命令,以保存受到干扰的一个行数据段到 VRB 的缓存当中(注意:如果 VRB 的缓存段在之前被修改了,在预取步骤之前需要发送 Restore 命令进行恢复处理);随后预充电线程 1 的行缓存数据,并激活线程 2 的行数据.由于线程本身的局部性,当下次线程 1 继续进行列数据访问时,线程干扰所导致的行缓存颠簸现象将不会发生,读或写命令可以立即发送到 VRB 的缓存段中,经过读取延迟或写入延迟之后,输出或输入的数据便可以直接发送到数据总线上,这增加了 Bank 中数据的复用率.

另外一个好处是:VRB 缓存区和 DRAM 中的 Bank 是独立的,访存协议命令的前端操作和后端操作可以被并行执行.例如相同 Bank 中的 VRB 段操作(Read/Write),可以与行缓存的后端操作(Activate/Precharge)并行执行,也可以与其他 VRB 段的前端操作(Read/Write)并行执行,这在传统的 DRAM 结构中是不可能完成的.与此同时,读取到 VRB 缓存中的数据,如果没有对其进行写操作,是不需要恢复到存储阵列中的,因为 DRAM 的存储阵列已经在预充电阶段完整保存了数据.总之,行缓存中的数据在被暂存到 VRB 缓存后,就可以进行列访问操作,而不必关心 Bank 所处的当前状态是什么,这大大提高了 DRAM 的并行性.

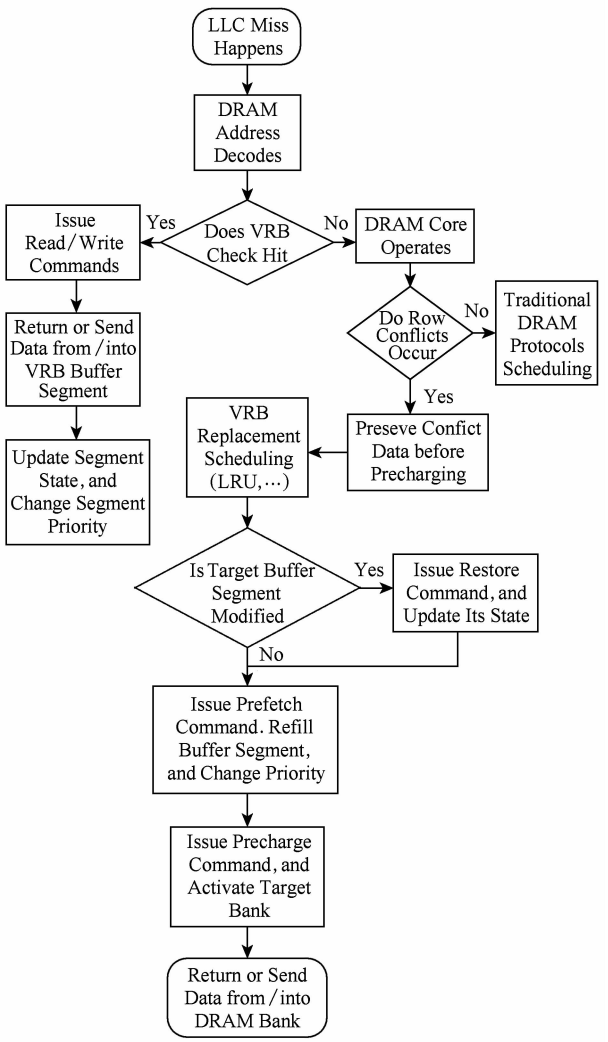

Fig. 3　VRB protocol scheduling.

图 3　VRB 协议调度过程

2.2.4　VRB 的段替换策略

内存控制器必须实现一个 VRB 缓存区的段选择算法,因为 Bank 内的一个 VRB 缓存区可以存储 Bank 中的任一行数据.内存控制器必须选择一个 VRB 缓存段来存储行缓存中的一段数据,并且一个修改过的 VRB 缓存段必须恢复到存储阵列的行中,才能进行下一次的重填操作.本文介绍 3 种策略使用 VRB 缓存区:1)循环(round robin, RR)策略;2)最近最少使用的(least recently used, LRU)策略;3)未经修改(unmodified favorite, UF)策略.其中,RR 策略只是简单地利用了 VRB 缓存段的数量,而没有考虑到局部性的问题;LRU 策略类似于 cache

缓存的使用,这需要额外的资源跟踪使用所有缓冲段;而 UF 策略主要倾向于尽量减少额外的 Restore 协议开销,因为未修改过的缓存段可以直接被覆盖,而修改过的段在进行预取操作之前需要恢复到 DRAM 的存储阵列中. 与此同时,这 3 种策略都可以在 DRAM 没有其他内存访问操作的时候自动进行 VRB 段的恢复操作. 在本文中的实验中,我们实现了一个采用 LRU 策略的 VRB 缓存方案.

总体而言,一个集成了 VRB 机制的内存控制器可以缓解多线程的访存干扰问题,以增加单个线程对行数据的复用率. 细粒度的缓存段在可访问实体数量上以及前端和后端操作上,提供了更多的并行性机会. 进一步而言,VRB 内存机制在不同的多程序应用环境中,还可以针对特定线程或目标进行优化. 因为 VRB 的缓存区是被绑定到每个 Bank 上的,内存控制器可以对特定 Bank 采用不同的替换策略来实现加速某些应用的目的. 另外,VRB 还可以结合内存调度优化技术,进一步改善整个存储器系统的性能.

3 实验方法

为了评估 VRB 机制,我们使用 MARSSx86 全系统模拟器[12]模拟了一个 8 核英特尔至强服务器系统,并对 x86 流水线结构进行了完整和时序准确的模拟. 通过对处理器进行全系统模拟,我们可以捕获虚拟内存和操作系统内核之间的交互过程,反映了多线程程序对行缓存干扰的真实过程. 另外我们修改了时序精确的 DRAMSim2 模拟器[13],以实现在第 2 节中描述的 VRB 内存系统,并详细模拟了 DDR3-1333 的 DRAM 芯片模型(MT41J256M4-125E),同时我们在每组实验中采用 FR-FCFS 的内存控制器调度策略. 表 2 列出了实验所使用的模拟参数,包括在内存模拟器中所有组成部分的详细参数. 通过 2 个时序精确的模拟器,本文创建了一套准确的服务器计算模型,可以完成从启动操作系统、运行应用程序,乃至所有访存操作与高速缓存和 DRAM 存储器系统进行交互的完整过程.

Table 2 Full-System Simulators Configuration

表 2 全系统模拟器配置

Category	Parameter	Configuration
CPU	ISA	Intel x86
	CMP Size and Frequency	8-core, 3.4 GHz
	Re-Order-Buffer	64 Entry
	Fetch, Dispatch, Execute, Retire	Maximum 4 per Cycle
	L1 I-cache	32 KB / 4-way, Private, 2-cycle
	L1 D-cache	32 KB / 8-way, Private, 4-cycle
	L2 Cache	256 KB / 8-way, Private, 6-cycle
	L1 and L2 Cache Block Size	64 B
	Coherence Protocol	Snooping MESI
DRAM	DRAM Device	DDR3-1333 Timing Parameters, 8 Banks
	DIMM Configuration	2 DIMMS, 2 Rank/DIMM, 64b Channel
	DIMM-Level Row-Buffer Size	8 KB
	Active Row-Buffers per DIMM	4
	Total DRAM Capacity	4 GB(128 MB/Device, 8 Devices, 2 DIMMs, 2 Channels)

3.1 DRAM 地址映射

为了模拟过程的精确性,我们探测了一个英特尔至强 5645 的内存控制器,并采用其地址映射作为基准系统. 下面描述内存系统是如何将一个物理地址映射到 DRAM 存储单元的. 如图 4 所示,英特尔采用 8 KB 作为其行缓存的页面大小,以及采用页面交错的数据布局. 连续的缓存块(64 B)被保存在不同的通道上,但具有相同的 Rank、Bank 以及行地址. 对于一个 32 b 的物理地址来说,低位的 3 b 被作为字节地址,3～12 b 以及 15 b 提供列地址;16～19 b 以及 22～31 b 为行地址;最重要的 14 b,20 b,21 b 表示 Bank 的地址位,其中 2 个较高位被排除在缓存

地址之外,这降低了 Zhang 等人[14]所描述的 Bank 存储体冲突的机会.一个操作系统页面(4 KB)被分配在了不同通道上的相同 Rank 和 Bank 地址的芯片设备上.对于 4 KB 的操作系统页面大小,一个 Bank 中的行缓存包含了 4 个 OS 页面,并在每个 Bank 存储体中拥有 2 KB 的操作系统页面大小.

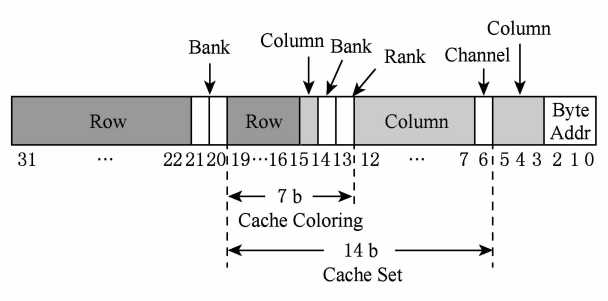

Fig. 4　Intel Xeon5645 Nehalem address mapping.

图 4　Intel Xeon5645 Nehalem 地址映射

3.2　工作集访存特征

我们从 SPEC CPU2006 测试集[15]中混合组成了多线程的工作负载,并使用 gcc 4.1.2 编译器版本以及-O3 优化选项对每个程序进行编译.在全系统模拟过程中,我们利用 System V 的 IPC 机制来同步所有线程,以跳过线程的初始化数据阶段.

表 3 是我们测得的 SPEC2006 工作集程序的内存特征参数,包括每个时钟周期的指令数目(instructions per cycle, IPC)、每千条指令高速缓存未命中数目(misses per kilo instructions, MPKI)、行缓存的命中率(row-buffer hit rate, RBHR)以及 DRAM 内存带宽的使用情况.另外,根据程序的访存特征,可以将程序细分成 4 类:

1) 访存密集型(memory intensive, MI).它指的是计算指令较少,而访存指令较多.而内存带宽大小对程序性能影响较大,因此这类应用也被称为带宽敏感型.

2) 访存非密集型(memory non-intensive, MNI).它指的是计算指令较多,而访存指令较少,对内存带宽要求不高.但访存延迟大小对程序性能影响较大,因此这类应用被称为延迟敏感型.

Table 3　SPEC2006 Benchmarks Memory Characteristics

表 3　SPEC2006 基准测试程序访存特征

Benchmark	IPC	RBHR/%	MPKI	Bandwidth/MBps	Category
458. sjeng	0.353 589	63.27	55.17	6 037.47	MIMF
433. milc	0.858 515	14.01	0.01	2.05	MNIMNF
470. lbm	0.547 758	16.73	25.51	3 787.21	MIMNF
429. mcf	0.289 269	1.88	57.64	3 105.75	MIMNF
462. libquantum	1.573 999	57.13	5.77	2 791.82	MNIMF
450. soplex	0.481 508	18.57	39.15	3 029.04	MIMNF
445. gobmk	0.907 04	62.20	11.68	3 239.54	MIMF
473. astar	0.493 111	5.27	17.36	1 805.76	MIMNF
482. sphinx3	0.661 544	3.97	12.93	1 418.39	MIMNF
401. bzip2	0.854 772	18.60	6.33	1 401.42	MNIMNF
436. cactusADM	0.641 377	8.07	11.31	1 245.16	MIMNF
454. calculix	1.325 885	39.41	2.40	841.06	MNIMF
403. gcc	0.724 465	23.23	6.26	928.15	MNIMF
400. perlbench	1.425 184	31.78	2.13	700.58	MNIMF
456. hmmer	0.798 176	23.29	2.71	622.23	MNIMF
483. xalancbmk	0.987 964	17.25	2.39	494.83	MNIMNF
453. povray	1.043 949	24.51	2.18	515.35	MNIMF
435. gromacs	1.173 049	41.44	1.30	335.87	MNIMF
447. dealII	1.165 379	25.92	0.72	218.67	MNIMF
471. omnetpp	1.800 494	38.78	0.38	173.33	MNIMF
464. h264ref	0.663 275	38.89	0.67	105.73	MNIMF
444. namd	1.096 478	14.80	0.06	11.93	MNIMNF

3) 访存友好型(memory friendly, MF). 它指的是程序的访存局部性较好,行缓存命中率较高的应用程序.

4) 访存非友好型(memory non-friendly, MNF). 它指的是程序的访存局部性较差,行缓存命中率较低的应用程序.

表 3 根据程序的不同访存特征将 22 个应用程序混合组成 10 组测试套,每组测试套均包含以上 4 类不同程序,这展现了在未来多核多线程应用中(例如云计算)一种常见的组合场景. 我们将 10 组测试套运行在目标机器为 8 核英特尔至强服务器的全系统模拟器上,以评估 VRB 内存系统在不同内存特征组合下的性能和功耗表现. 表 4 列出了这 10 种不同访存类型的程序组合.

Table 4 SPEC CPU2006 Benchmark Combinations

表 4 SPEC CPU2006 测试程序组合

Combinations	Core-0	Core-1	Core-2	Core-3	Core-4	Core-5	Core-6	Core-7
Com1	433. milc	458. sjeng	445. gobmk	462. libquantum	450. soplex	458. sjeng	403. gcc	401. bzip2
Com2	458. sjeng	462. libquantum	464. h264ref	401. bzip2	403. gcc	444. namd	450. sphinx3	429. mcf
Com3	470. lbm	445. gobmk	447. dealII	462. libquantum	483. xalancbmk	436. cactusADM	456. hmmer	471. omnetpp
Com4	450. sphinx3	429. mcf	435. gromacs	456. hmmer	470. lbm	429. mcf	444. namd	454. calculix
Com5	445. gobmk	447. dealII	436. cactusADM	458. sjeng	471. omnetpp	445. gobmk	429. mcf	433. milc
Com6	470. lbm	458. sjeng	403. gcc	453. povray	450. sphinx3	473. astar	445. gobmk	453. povray
Com7	473. astar	445. gobmk	470. lbm	462. libquantum	462. libquantum	400. perlbench	483. xalancbmk	458. sjeng
Com8	454. calculix	470. lbm	470. lbm	462. libquantum	433. milc	400. perlbench	450. soplex	483. xalancbmk
Com9	473. astar	401. bzip2	462. libquantum	471. omnetpp	458. sjeng	458. sjeng	450. soplex	483. xalancbmk
Com10	436. cactusADM	435. gromacs	483. xalancbmk	429. mcf	401. bzip2	464. h264ref	444. namd	445. gobmk

4 结果与分析

本节的实验结果部分将在带宽、延迟、系统性能和功耗开销方面对 VRB 内存系统和英特尔至强基准内存系统进行比较. 在实验中,内存通道数配置为 2,Rank 数目为 2,Bank 数目为 8,内存芯片采用 64b 位宽的 DDR3-1333 芯片组,VRB 内存系统采用 8×1 KB 的参数配置.

4.1 系统性能改善

图 5 显示,VRB 机制相对于基准系统显著提高了内存系统的带宽利用,达到了最高 17.7%(平均 15%)的带宽提升. 本文将带宽的大幅改善归结为以下 4 个方面的原因:

1) 更高的行缓存命中率. 从图 6 可以看出,相对于英特尔至强服务器内存系统,VRB 系统显著改善了多线程混合应用的行缓存命中率,达到了最高 142.9%(平均 51.4%)的 RBHR 改善. VRB 的段缓存暂时保存了被其他线程逐出的数据,而后续的线程利用了这部分数据,从而减少了内存系统额外的激活和预充电操作,改善了内存带宽的有效利用.

2) 更高的 Bank 级并行度. VRB 缓存独立于 Bank 存储体,可以进行独立的读写操作,有效增加了可以并行处理的访存数量,提高 Bank 级并行度.

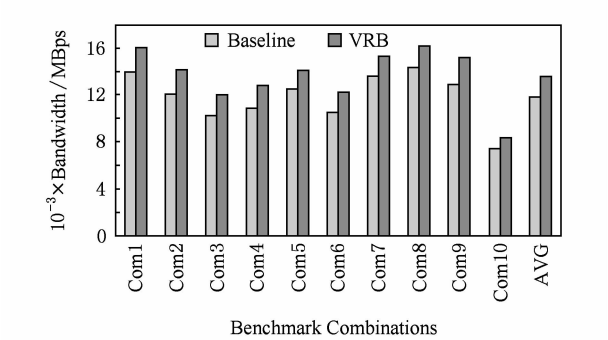

Fig. 5 Bandwidth improvements (VRB vs Baseline).

图 5 VRB 相对于 Intel 基准系统的带宽改善

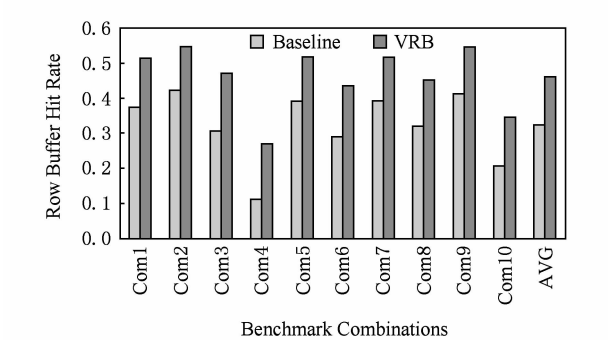

Fig. 6 RBHR improvements (VRB vs Baseline).

图 6 VRB 相对于 Intel 基准系统的 RBHR 改善

3) 更细粒度的数据传输. 传统的 4～8 KB 的行缓存被全相连映射到更细粒度的 VRB 段缓存上,多个段缓存可以充分利用 DRAM 内部较宽的数据总线,完成数据的前端(Read/Write)和后端操作(Activate/Precharge/Prefetch/Restore),进一步增加了访存协议命令的并行操作能力.

4) 更有效的使用命令总线. RB 行缓存和 VRB 段缓存共享命令总线,可以大幅提高命令总线的利用效率.

图 7 展示了 VRB 机制相对于英特尔基准内存系统的延迟改善情况,可以看出,最高可以达到 30.6% 的延迟改善,而平均也有 23.8% 的改善. 存储器访问延迟时间主要分为 2 部分:DRAM 核心(DRAM core)的访问延迟和队列延迟时间[1,16]. 在大部分的 DRAM 芯片设计中,核心访问延迟是基本不变的;而队列延迟在内存控制器中根据不同的调度方式以及微结构技术会有所不相同. VRB 内存机制,通过缓解多线程的行缓存冲突问题,减少了队列中访存请求在行缓存不命中情况下产生的额外协议命令,从而降低了访存请求在队列中的停留时间,加速了 DRAM 内存的处理速度.

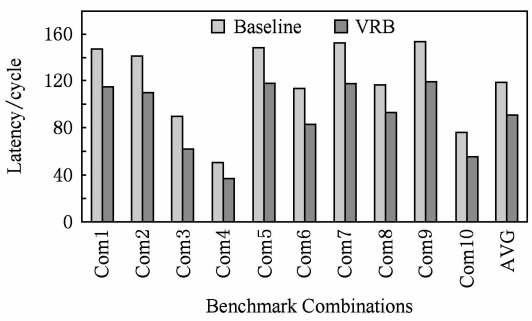

Fig. 7　Latency improvements (VRB vs Baseline).

图 7　VRB 相对于 Intel 基准系统的延迟改善

总体而言,实验证明了我们的 VRB 机制相对于 Intel 的基准系统,获得了非常高的性能改善. 如图 8 所示,VRB 实现了最高 17.6%(平均 8.7%)的整体性能改善. 通过本节的实验结果和分析表明,VRB 可以使内存控制器有效提高由于线程干扰所降低的 DRAM 行缓存命中率,这是在传统 DRAM 结构中无法完成的;VBR 机制不仅能缓解多线程应用程序的内存竞争问题,减少不必要的协议开销,同时也为 DRAM 的前端操作和后端操作提供了大量的并行机会,有效提高了 DRAM 的资源利用效率.

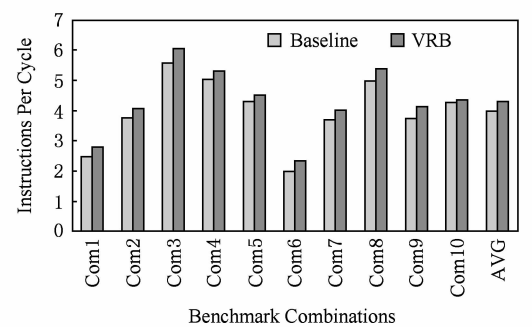

Fig. 8　IPC improvements (VRB vs Baseline).

图 8　VRB 相对于 Intel 基准系统的 IPC 改善

4.2　功耗和面积开销

DRAM 的功耗主要是由背景功耗、激活功耗、预充电功耗、突发和刷新功耗所组成[17]. VRB 机制主要通过减少多线程行缓存干涉现象降低 DRAM 的功耗,干扰现象会导致额外的预充电和激活过程,并带来额外的功率消耗. 在 DRAM 系统中,激活功率是 DRAM 功率消耗最主要的组成部分[1],因为它必须从 DRAM 存储阵列中激活一整行数据到行缓存中. VRB 之所以可以降低 DRAM 的功耗,是因为缓解了多线程应用干扰造成 Bank 上的行缓存冲突现象,减少了激活和预充电过程. 我们通过 DRAMSim2 的功耗模型测量了 VRB 的内存系统. 实验结果如图 9 所示,内存控制器在 FC-FRFS 的调度和开页策略下,VRB 内存系统可以节省最高 17.6%(平均 9.2%)的功耗消耗. 本文实验所采用的 DDR3 芯片的 Bank 存储体大小为 128 MB,而增加一个 8×1 KB 的 VRB,面积几乎可以忽略不计.

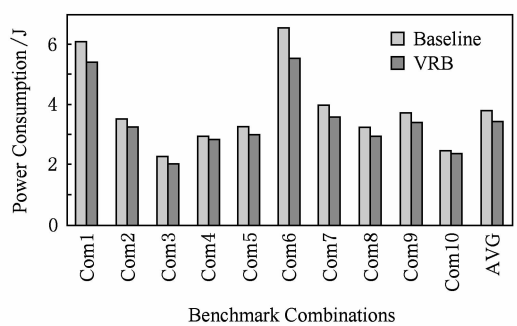

Fig. 9　Power consumption improvements (VRB vs Baseline).

图 9　VRB 相对于 Intel 基准系统的功耗改善

5　相关工作

关于多线程访存竞争的问题,学术界和工业界存在大量的相关工作,本文主要将其分为缓存/DRAM 分区、内存控制器调度以及 DRAM 的微结构技术.

5.1 缓存和 DRAM 分区

在高速缓存上,Liu 等人[18]提出采用 XOR 地址映射的方法来减少缓存中的数据冲突问题,Song 等人[19]进一步在众核平台上采用 XOR 技术来减少共享缓存的线程干扰问题. Qureshi 等人[6]设计了一种分配缓存资源的方法,通过专门硬件来监控应用程序的资源使用情况,并据此作出分配决定. 其他工作[5,20]采用硬件分析的方法对工作负载进行分类,然后选择适当的调度策略完成资源使用. 另外还有操作系统级的内存利用率监测[21-22]方法也被提出,用于提供资源管理方面的辅助信息.

在 DRAM 主存级别上,Mi 等人[3]提出在 Bank 存储单元上进行划分,并使用页着色和异或高速缓存映射技术,以减少多处理器芯片上的线程干扰问题. Liu 等人[4]也使用页着色技术在 OS 的内存管理中对进程进行着色,以缓解多个程序对 DRAM 上 Bank 的冲突问题. Park 等人[23]采用随机分配算法来分散多线程工作负载的页到不同的 Bank 上,以避免冲突的发生. Muralidhara 等人[24]提出另一个分区的方法,他们的目标是内存通道而不是内存 Bank.

5.2 内存控制器调度

许多内存调度策略已被提出,旨在提高 CMP 系统中的访存公平性和系统吞吐率. STFM(stall-time fair memory)调度算法[25]基于每个线程的停顿时间来完成内存控制器的调度决策,以实现公平性. 一种 Bank 并行度感知的调度方法[10]被提出,以达到系统公平性和吞吐量之间的平衡目的. 而 ATLAS(adaptive per-thread least-attained-service memory)调度算法[26],是通过提高最少被服务到的线程的访存优先级来提高系统的最大化吞吐率,但该算法以牺牲公平性为代价. TCM(thread cluster memory)调度算法[27]将线程分成 2 个独立的集群,并对每个集群采用不同内存请求的调度策略,所提出的方案在吞吐率和公平性 2 方面都会得到增强. 这些以前的解决方案,都是在获取线程的访存行为并通过调度的方式来缓解线程间的干扰问题.

5.3 DRAM 微结构技术

Sub-Ranking 技术是将传统的 Rank(64 b)位宽改变成更窄(8 b,16 b 或 32 b)Sub-Ranks 的技术. Ware 等人[28]提出 Threaded Memory Module 技术,内存控制器通过芯片使能信号来控制这些单独的 Sub-Rank,该技术通过增加 Bank 的数量来获得内存系统更大的 Bank 级并行度. Zheng 等人[29]提出 Mini-Rank 内存系统,他们并没有改变内存控制器模块的接口,而是使用一个模块端的控制器来访问 Sub-Ranks,目标是节省功耗而非性能. Ahn 等人[30]提出了多核 DIMM(multicore DIMM, MC-DIMM)内存,他们的工作类似与 Threaded Memory Module 技术,并在全系统环境评估了该技术在多线程应用程序下的功耗表现. 尽管分区 DRAM 的 Rank 变为更小的 Rank 子集增加了并行性,但每个 Sub-Rank 的数据总线变窄导致传输一个 64 B 的缓存块时需要额外的时间.

Cached DRAM 技术[31]在 2000 年之前被广泛提出,其通过在 DRAM 芯片上增加一个附加的 SRAM 高速缓冲存储器,来存储最近访问过的数据. 虚通道内存(virtual channel memory, VCM)[32]方法是这一技术的典型代表,在 20 世纪 90 年代末由 NEC 公司第 1 次提出. 在 VCM 中的每一个 Rank 上,有 16 个或 32 个通道缓冲区,每个通道预取行缓存的一段数据,以此提供更快的访问和获得更多的并发性. 虽然这列方法类似于本文所提出的 VRB 机制,但 Cached DRAM 仅在 SRAM 缓存命中时才能提供更大的并行性,而在缓存失效时不得不串行化访问 DRAM 的 Bank 存储体. 相比之下,我们的方案只保留被替换出的行缓存数据,并且可以与行缓存进行并发访问.

6 结束语

本文提出了一种新的 VRB 内存系统,通过缓解多核、多线程处理器环境下的行缓存干扰问题,来改善系统性能和功耗消耗. VRB 缓存可以暂存由于行缓存冲突被替换出的行数据,以提高行缓存数据的复用率,并减少线程间的访问干扰所带来的额外的访存协议开销. 通过为每个 Bank 存储体引入独立的细粒度 VRB 缓存段,不仅增加了行缓存数据的复用率,还可以增加 DRAM 存储体操作的并行度(前端操作和前端操作,前端操作和后端操作). 与此同时,为了有效地利用 VRB 机制,我们设计了兼容 DRAM 工业标准的访存协议来运行上述机制,以达到充分利用 DRAM 存储器的资源和带宽的目的. 通过全系统模拟环境的实验和分析,VRB 内存系统相对于英特尔的至强服务器基准内存系统,可以有效地提高系统性能,并减少功耗消耗. 总体而言,在面向多核、多线程并行执行的计算平台上,VRB 内存系统是一种可以有效缓解 DRAM 行缓存冲突的内存系统.

参 考 文 献

[1] Udipi A N, Muralimanohar N, Chatterjee N, et al. Rethinking DRAM design and organization for energy-constrained multi-cores [C] //Proc of the ACM SIGARCH Computer Architecture News. New York: ACM, 2010: 175-186

[2] Sudan K, Chatterjee N, Nellans D, et al. Micro-pages: Increasing DRAM efficiency with locality-aware data placement [C] //Proc of the ACM SIGPLAN Notices. New York: ACM, 2010: 219-230

[3] Mi W, Feng X, Xue J, et al. Software-hardware cooperative DRAM bank partitioning for chip multiprocessors [C] //Proc of the Network and Parallel Computing. Berlin: Springer, 2010: 329-343

[4] Liu L, Cui Z, Xing M, et al. A software memory partition approach for eliminating bank-level interference in multicore systems [C] //Proc of the 21st Int Conf on Parallel Architectures and Compilation Techniques. New York: ACM, 2012: 367-376

[5] Lin J, Lu Q, Ding X, et al. Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems [C] //Proc of the High Performance Computer Architecture. Piscataway, NJ: IEEE, 2008: 367-378

[6] Qureshi M K, Patt Y N. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches [C] //Proc of the 39th Annual IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2006: 423-432

[7] Yoon D H, Jeong M K, Sullivan M, et al. The dynamic granularity memory system [C] //Proc of the 39th Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2012: 548-559

[8] Zhang G, Wang H, Chen X, et al. Heterogeneous multi-channel: Fine-grained DRAM control for both system performance and power efficiency [C] //Proc of the 49th Annual Design Automation Conf. New York: ACM, 2012: 876-881

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

[10] Mutlu O, Moscibroda T. Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems [C] //Proc of the ACM SIGARCH Computer Architecture News. New York: ACM, 2008: 63-74

[11] Rixner S. Memory controller optimizations for Web servers [C] //Proc of the 37th Annual IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2004: 355-366

[12] Patel A, Afram F, Chen S, et al. Marss: A full system simulator for multicore x86 CPUs [C] //Proc of the 48th Design Automation Conf. New York: ACM, 2011: 1050-1055

[13] Rosenfeld P, Cooper-Balis E, Jacob B. DRAMSim2: A cycle accurate memory system simulator [J]. IEEE Computer Architecture Letters, 2011, 10(1): 16-19

[14] Zhang Z, Zhu Z, Zhang X. A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality [C] //Proc of the 33rd Annual ACM/IEEE Int Symp on Microarchitecture. New York: ACM, 2000: 32-41

[15] Henning J L. SPEC CPU 2006 benchmark descriptions [J]. SIGARCH Computer Architecture News, 2006, 34(4): 1-17

[16] Jeong M K, Yoon D H, Sunwoo D, et al. Balancing DRAM locality and parallelism in shared memory CMP systems [C] //Proc of the 18th Int Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2012: 1-12

[17] Micron. Calculating memory system power for DDR, Tn-46-03 [R]. Boise, ID: Micron Technology, 2001

[18] Liu Zhiyong, Li Enyou, Qiao Xiangzhen. Cache memory system address mapping technology and device: China, CN1217505[P]. 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: 中国, CN1217505[P]. 1999-05-26)

[19] Song F, Fan D, Liu Z, et al. Efficient address mapping of shared cache for on-chip many-core architecture [C] //Proc of the Euro-Par 2010-Parallel Processing. Berlin: Springer, 2010: 280-291

[20] Jaleel A, Najaf-Abadi H H, Subramaniam S, et al. Cruise: Cache replacement and utility-aware scheduling [C] //Proc of the 17th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2012: 249-260

[21] Ding X, Wang K, Zhang X. SRM-buffer: An OS buffer management technique to prevent last level cache from thrashing in multicores [C] //Proc of the 6th Conf on Computer Systems. New York: ACM, 2011: 243-256

[22] Zhang X, Dwarkadas S, Shen K. Towards practical page coloring-based multicore cache management [C] //Proc of the 4th ACM European Conf on Computer systems. New York: ACM, 2009: 89-102

[23] Park H, Baek S, Choi J, et al. Regularities considered harmful: Forcing randomness to memory accesses to reduce row buffer conflicts for multi-core, multi-bank systems [C] //Proc of the 18th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2013: 181-192

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

[25] Mutlu O, Moscibroda T. Stall-time fair memory access scheduling for chip multiprocessors [C] //Proc of the 40th Annual IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2007: 146-160

[26] 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 Int Symp on High Performance Computer Architecture (HPCA). Piscataway, NJ: IEEE, 2010: 1-12

[27] Kim Y, Papamichael M, Mutlu O, et al. Thread cluster memory scheduling: Exploiting differences in memory access behavior [C] //Proc of the 43rd Annual IEEE/ACM Int Symp on Microarchitecture (MICRO). Piscataway, NJ: IEEE, 2010: 65-76

[28] Ware F A, Hampel C. Improving power and data efficiency with threaded memory modules [C] //Proc of the Int Conf on Computer Design. Piscataway, NJ: IEEE, 2006: 417-424

[29] Zheng H, Lin J, Zhang Z, et al. Mini-rank: Adaptive DRAM architecture for improving memory power efficiency [C] //Proc of the 41st Annual IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2008: 210-221

[30] Ahn J H, Jouppi N P, Kozyrakis C, et al. Improving system energy efficiency with memory rank subsetting [J]. ACM Trans on Architecture and Code Optimization, 2012, 9(1): 1-28

[31] Hidaka H, Matsuda Y, Asakura M, et al. The cache DRAM architecture: A DRAM with an on-chip cache memory [J]. IEEE Micro, 1990, 10(2): 14-25

[32] NEC. 64m-bit virtual channel SDRAM data sheet, M13022EJBV0DS00 [R]. Tokyo: NEC, 1998

Gao Ke, born in 1983. PhD. His main research interests include high performance computer architecture and memory systems.

Fan Dongrui, born in 1979. PhD, professor. His main research interests include low-power architecture and micro-architecture.

Liu Zhiyong, born in 1946. PhD, professor, PhD supervisor. His main research interests include algorithm, parallel processing and memory systems.