

ISMB:多核系统中利用 Bank 分区实现共享库隔离

杨虎斌,李嘉翔,陈玉聪,刘 刚,张红涛,周 睿,周庆国
(兰州大学 信息科学与工程学院,甘肃 兰州 730000)

摘 要:动态随机存取存储器 DRAM 一直以来以其低功耗、高性价比和良好的扩展性等优点作为计算机内存的最佳选择。为了提高内存的访问速度,DRAM 中的每个 Bank 都有一个行缓冲区,它可以有效地提升局部性良好的应用程序的性能。然而在多核系统中,DRAM 被系统中的所有 Core 共享,因此对内存的并发访问会导致 Bank 行缓冲区冲突问题的产生,从而导致内存访问延迟的增大。共享库作为一种共享资源,使 Bank 行缓冲区冲突问题更加严重。虽然目前有一些基于 DRAM Bank 分区技术的解决方案可以有效缓解由进程访问私有内存导致的 Bank 行缓冲区冲突问题,但是这些解决方案无法解决访问共享库引起的 Bank 行缓冲区冲突问题。该文提出了一种在多核系统中利用 Bank 分区实现共享库隔离的方案(ISMB)。ISMB 使运行在同一个 Core 上的进程只能访问属于该 Core 的共享库的副本,因此 ISMB 消除了共享库导致的 Bank 行缓冲区冲突问题。对比实验结果表明,ISMB 能够有效地提升系统隔离性能,在使用 ISMB 的情况下,SPEC CPU2006 基准测试程序的减速率最大可降低 26.3%。

关键词:共享库;Bank 分区;隔离;动态随机存取存储器;Bank 行缓冲区冲突

中图分类号:TP316

文献标识码:A

文章编号:1673-629X(2023)02-0017-07

doi:10.3969/j.issn.1673-629X.2023.02.003

ISMB: Isolation of Shared Libraries in MultiCore Systems via Bank Partitioning

YANG Hu-bin, LI Jia-xiang, CHEN Yu-cong, LIU Gang, ZHANG Hong-tao,
ZHOU Rui, ZHOU Qing-guo

(School of Information Science and Engineering, Lanzhou University, Lanzhou 730000, China)

Abstract: Dynamic Random Access Memory (DRAM) has always been the best choice of memory device due to its low power consumption, high performance-price ratio and good scalability. To improve memory access speed, each Bank in DRAM has a row buffer that can effectively improve the performance of programs with good locality. However, the DRAM memory is shared by all Cores on MultiCore systems, so concurrent access to memory can lead to Bank row buffer conflict, resulting in increased memory access latency. As a shared resource, shared library makes the problem worse. Although there are some solutions based on DRAM Bank partitioning technology that can effectively alleviate the Bank row buffer conflict caused by the access to private data, the conflict caused by accessing the shared libraries has not been completely solved. In this paper, we propose ISMB, a new mechanism which realizes the isolation of shared library on MultiCore system by using Bank partition technology. ISMB allows all processes running on the same Core to access the copies of shared libraries that belong to this Core, so ISMB can eliminate the Bank row buffer conflicts caused by shared libraries. The experimental results demonstrate that ISMB is effective in improving the performance isolation, the slowdown ratios of the SPEC CPU2006 benchmarks can be reduced by up to 26.3% by using ISMB.

Key words: shared library; Bank partitioning; isolation; DRAM; Bank row buffer conflict

0 引言

动态随机存取存储器 (Dynamic Random Access Memory, DRAM) 被广泛地应用于嵌入式设备、个人

电脑和服务器^[1]。在现代计算机内存体系结构中,当处理器核(Core)通过内存控制器访问 DRAM 中的数据时,需要将线性的物理地址转换为由通道

收稿日期:2022-11-15

修回日期:2022-12-28

基金项目:国家重点研发计划资助(2020YFC0832500);兰州大学中央高校基本科研业务费专项资金资助(lzujbky-2022-kb12);国家自然科学基金(61402210)

作者简介:杨虎斌(1988-),男,硕士,研究方向为内存优化;通信作者:周庆国(1973-),男,博士,教授,研究方向为嵌入式实时系统和系统安全。

(Channel)、DIMM、Rank、Bank、行(Row)和列(Column)组成的多维的 DRAM 地址^[2]。为了提高访问内存的速度,在每个 Bank 中都有一个行缓冲区(Row Buffer)^[3-4],它一次可以存储 Bank 中的一行数据。特别地,应用程序的局部性越好,行缓冲区起到的效果就越好。

在多核平台上,DRAM 由系统中所有的 Core 共享,因此,只有在不同 Core 上运行的进程并发访问同一个 Bank 中的同一行数据,或者并发访问不同 Bank 中的数据时,才不会发生行缓冲区冲突。但是,当不同 Core 上运行的进程并发访问同一个 Bank 中不同行中的数据时,会引发行缓冲区冲突,从而导致该行缓冲区的频繁刷新。因此,行缓冲区冲突导致的内存访问延迟使整个系统的性能下降。

目前已有一些解决方案利用 DRAM Bank 分区技术缓解这个问题^[5-13]。这些解决方案的主要设计思想是利用 Bank 分区技术将 DRAM 中指定的 Bank 分配给指定的进程,因此,不同的进程只能访问位于自己的 Bank 中的物理内存,从而可以有效地缓解 Bank 行缓冲区冲突。然而,由于共享库是一种进程间的共享资源,它们被加载到内存后,通常随机地分布在不同的 DRAM Bank 中。因此,虽然基于 DRAM Bank 分区技术的解决方案可以有效地缓解由进程访问私有内存导致的 Bank 行缓冲区冲突问题,但是,无法解决访问共享库引起的 Bank 行缓冲区冲突问题。

该文提出了一种在多核系统中利用 DRAM Bank 分区技术实现共享库隔离的方案(Isolation of Shared Libraries in MultiCore Systems via Bank Partitioning, ISMB)。ISMB 为每个 Core 提供了一个共享库的副本,并使运行在相同 Core 上的所有进程共享只属于该 Core 的共享库的副本。

具体地,ISMB 首先将共享库的副本分别加载到位于对应的 Core 的 Bank 中的物理页面中,然后利用 Bank 分区技术使运行在同一个 Core 上的所有进程只能访问属于该 Core 的共享库的副本。因此,ISMB 消除了共享库导致的 Bank 行缓冲区冲突问题,从而提升了系统整体的性能。

从表 1 中可以看出,与其他没有考虑共享库的 Bank 分区方案相比,ISMB 的创新性在于:在涉及共享库的两种情况下,ISMB 都可以提供有效的隔离。例如,在情况 2 下,假设为 Core 0 和 Core 1 分别分配的是 Bank 0 和 Bank 1,则运行在 Core 0 上的应用程序的二进制可执行代码被加载到 Bank 0 中,而运行在 Core 1 上的共享库的代码被加载到 Bank 1 中,因此,当 Core 0 和 Core 1 并发访问内存时,不会导致 Bank 行缓冲区冲突。

表 1 ISMB 与其他不考虑共享库的 Bank 分区机制在共享库隔离方面的对比

情况	Core 0	Core 1	ISMB	其他 ¹
1	E ²	E	Yes ⁴	Yes
2	E	S ³	Yes	No ⁴
3	S	S	Yes	No

注:1. 不考虑共享库的 Bank 分区机制。

2. E 表示正在 Core 0 上运行的是应用程序的二进制可执行代码。

3. S 表示正在 Core 1 上运行的是共享库的代码。

4. Yes 表示可以为运行在不同 Core 上的代码提供 Bank 隔离;No 表示不可以。

对比实验结果表明,ISMB 能够有效地提升系统隔离性能。与未使用 ISMB 的 Linux 相比,在高负载的情况下,所有 SPEC CPU2006 基准测试程序的减速率平均降低了 3.9%,最大降低了 26.3%。在混合负载的情况下,减速率平均降低了 6%,最大降低了 15.7%。

1 相关工作

1.1 DRAM 控制器调度算法

目前已有一些研究^[14-16]通过优化 DRAM 控制器调度算法来解决多核平台上 Bank 行缓冲区冲突问题,从而实现提高系统吞吐量或公平性的目的。虽然 DRAM 控制器调度算法优化方案可以提高系统吞吐量或公平性,但是这些方案有以下三个缺点:第一,涉及到内存控制器硬件的修改;第二,这些方案的有效性受到诸多因素的限制,例如调度算法缓冲区的大小等等;第三,当运行在不同 Core 上的多个进程共享 DRAM 时,优化后的 DRAM 控制器调度算法可能会导致饥饿现象的产生^[11]。因此,利用优化 DRAM 控制器调度算法来解决多核平台上 DRAM Bank 行缓冲区冲突问题,所能达到的效果有限。

1.2 DRAM Bank 分区技术

DRAM Bank 分区技术是通过软件的方式缓解多核平台上 Bank 行缓冲区冲突问题的机制。基于 DRAM Bank 分区的方案^[5-13]主要分为静态分区和动态分区两类。其中,Bank 静态分区技术对 DRAM 中的 Bank 进行静态分区,并将指定的 Bank 分配给指定的进程,以减少 Bank 的行缓冲区冲突,从而提高系统性能。Bank 动态分区技术首先会根据进程对 Bank 数量的需求,动态地对 DRAM 中的 Bank 进行分区,然后在进程运行的不同阶段,为其分配数量不等的 Bank。

1.3 共享库隔离机制

目前已有一些为共享库提供隔离机制的解决方案,这些解决方案通常是将共享库独立地运行在隔离环境中,例如不同的上下文环境^[17]或独立的虚拟

机^[18-19]中。在这些解决方案中,调用共享库中函数需要进行上下文环境或虚拟机的切换,因此,在函数调用很频繁的情况下,这会造成很大的系统开销。除此之外, Kim 等人提出了一种称为选择性共享的策略^[20]。该策略首先为每个共享库创建 $n+2$ 个副本,其中 n 表示系统中 Core 的数量;然后将 n 个副本分别由运行在 n 个 Core 上的高优先级实时任务共享访问;其次,将另外 2 个副本分别由运行在所有 Core 上的低优先级实时任务和非实时任务共享访问。由于低优先级实时任务和非实时任务分别共享的 2 个共享库仍然可以导致 Bank 行缓冲区冲突问题,因此,该策略没有彻底地解决共享库引发的 Bank 行缓冲区冲突问题。

2 设计与实现

ISMB 的核心设计思想是:首先,为每个 Core 创建一个共享库的副本;然后,利用 Bank 分区技术使运行在某 Core 上的所有进程只能访问该 Core 对应的共享库副本。

如图 1 所示,假设为 Core 0 和 Core 1 分别分配的是 Bank 0 和 Bank 1,则运行在 Core 0 上的进程只能访问 Bank 0 中的进程的私有内存和 Core 0 的共享库副本的共享内存。因此,ISMB 可以同时消除进程访问私有内存和共享库导致的 Bank 行缓冲区冲突问题,从而提升系统的整体性能。

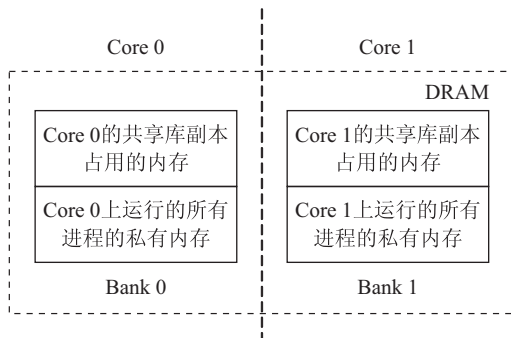


图 1 ISMB 的共享库隔离示意图

2.1 为 Core 分配 DRAM Bank

在 Linux 内核中, ISMB 为系统中所有的 Core 维护了一个空闲物理页面链表数组 `pcpu_list`, 数组中链表的数量等于系统中 Core 的数量。除此之外, ISMB 将系统中的所有 DRAM Bank 进行分区, 并为每个 Core 分配了一组固定的 Bank, 因此, `pcpu_list` 数组中每个链表存放的空闲物理页面位于该链表对应的 Core 的 Bank 中。例如, 为 Core 0 分配的是 Bank 0, 则 `pcpu_list[0]` 链表中存放的空闲物理页面都位于 Bank 0 中。当运行在 Core 0 上的进程在运行过程中发生缺页异常, 进入异常处理程序后, ISMB 通过修改 Linux 内核函数 `_rmqueue`, 实现了直接从 `pcpu_list[0]` 链表而

不是 Linux 伙伴系统中为进程申请空闲物理页面的操作。

图 2 是从 `pcpu_list[0]` 链表中为运行在 Core 0 上的进程申请空闲物理页面的流程。如果 `pcpu_list[0]` 链表不为空, 那么 ISMB 直接从 `pcpu_list[0]` 链表中取出第一个空闲物理页面后, 返回该物理页面。否则, ISMB 首先从 Linux 伙伴系统中申请一个空闲物理页面, 然后将其插入 `pcpu_list[0]` 链表。最后, 从 `pcpu_list[0]` 链表中取出空闲物理页面后, 返回该物理页面。在这个过程中, 若从 Linux 伙伴系统申请的空闲物理页面不位于 Bank 0 中, 则把这些物理页面分别插入适当的 `pcpu_list` 链表, 从而可以减少在将来申请空闲物理页面的时间开销。

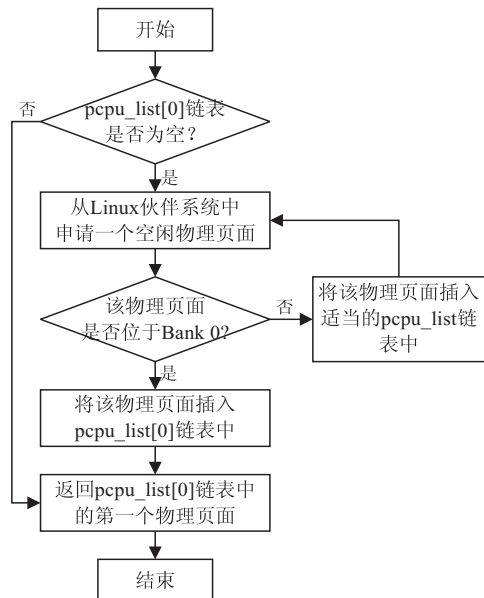


图 2 从 `pcpu_list[0]` 链表中申请空闲物理页面的流程

2.2 DRAM Bank 映射

为了把从 Linux 伙伴系统中申请的空闲页面插入适当的 `pcpu_list` 链表, 需要知道将物理地址转换为 DRAM 地址的映射信息, 从而确定一个物理页面位于哪个 DRAM Bank。对于没有公开地址映射信息的体系结构(例如 Intel), 可以利用逆向技术获取这些地址映射信息^[21-27]。目前已有的逆向技术主要分为两类: 基于软件的方法和基于硬件的方法。

由于文中的实验平台基于 AMD 架构, 并且该架构在其架构手册中明确地公开了 DRAM 地址映射信息, 因此, 通过查询 AMD 架构手册可知: 物理地址中的 Rank 和 Bank 位的信息分别存放在“F2x[1, 0][6C:60] DRAM CS Mask Registers”和“F2x[1, 0]80 DRAM Bank Address Mapping Register”两个寄存器中^[28-29]。如表 2 所示, 文中使用的实验平台有 1 个通道, 每个通道有 2 个 DIMM, 每个 DIMM 有 2 个 Rank, 每个 Rank 有 8 个 Bank, 因此, 系统中总共有 32 个

Bank,即在物理地址中,总共有 5 个比特位用来表示 Bank 的编号。

图 3 显示了文中使用的实验平台的 DRAM Bank 映射信息。由图可知,比特位 16~17 用于表示 Rank 的编号,比特位 13~15 用于表示 Bank 的编号。综上,ISMB 使用物理页面的起始地址中的比特位 13~17,来确定物理页面所在的 Bank 的编号。例如,一个物理页面的起始地址中的比特位 13~17 全为 0,表示该物理页面位于 Bank 0 中。

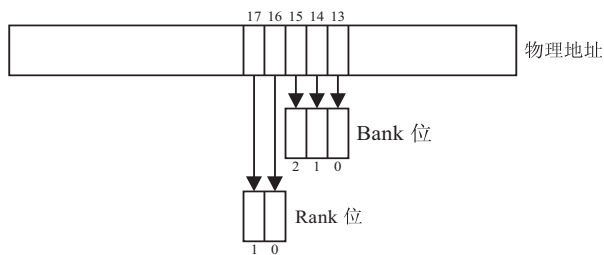


图 3 文中实验平台的 DRAM Bank 映射信息

2.3 共享库隔离

在已经对 DRAM Bank 进行分区,并将指定的 Bank 分配给指定 Core 的情况下,为了使进程只能访问属于运行该进程的 Core 的共享库的副本,ISMB 需要将属于某个 Core 的共享库的副本加载到位于该 Core 的 Bank 的物理页面。

ISMB 使用了一种简单有效的方法^[20]对共享库进行 Core 间的隔离。首先,在磁盘上为每个 Core 创建一个用于存放共享库副本的目录,并将共享库分别复制到每个目录中。然后,在运行绑定在指定 Core 上的应用程序前,将该 Core 对应的存放共享库副本的目录添加到环境变量 LD_LIBRARY_PATH。故该应用程序在运行过程中,只能使用该 Core 对应目录中的共享库副本。因此,在已经将指定 Bank 分配给指定 Core 的情况下,属于某 Core 的共享库副本只能被加载到位于该 Core 的 Bank 的物理页面,并被该 Core 上运行的所有进程共享。

3 实验与评估

3.1 实验环境

文中使用的实验平台的参数如表 2 所示,如 2.2 节所述,实验平台上总共有 32 个 Bank。由于物理内存的总大小为 16 GB,因此,每个 Bank 的大小为 512 MB。在所有实验中,为每个 Core 静态地分配 8 个 Bank。操作系统使用 Ubuntu 18.04,其中 Linux 内核版本为 5.3。此外,在所有实验过程中,通过禁用不相关的服务(桌面服务和网络服务)以提高实验精度。

在文中实验中,使用 SPEC CPU2006^[30]作为实验基准测试程序,表 3 对每个 SPEC CPU2006 基准测试

程序使用的共享库进行了全面的统计。使用减速率(slowdown ratio)作为评估 ISMB 实现的性能隔离程度的指标^[12],其定义如下:

表 2 实验平台参数

参数	配置
Processor	AMD Phenom II X4 920 Desktop, 4 Cores, 2.8 GHz
L1 I/D caches	64 KB/64 KB
L2 cache	512 KB
L3 cache	6 MB
DRAM Memory	1 Channel, 2 DIMMs/Channel, 2 Ranks/DIMM, 8 Banks/Rank, 16 GB DDR3 DIMM module 1 600 KHz

表 3 SPEC CPU2006 基准测试程序的
共享库使用统计信息

序号	基准测试程序	基准测试程序使用的共享库
1	401. bzip2	
2	403. gcc	libc. so
3	470. lbm	
4	458. sjeng	
5	400. perlbench	
6	433. milc	
7	435. gromacs	
8	445. gobmk	
9	456. hmmer	libc. so, libm. so
10	462. libquantum	
11	464. h264ref	
12	470. lbm	
13	482. sphinx3	
14	444. namd	
15	447. dealII	
16	450. soplex	
17	453. povray	libc. so, libm. so, libgccs. so libstdc++. so
18	471. omnetpp	
19	473. astar	
20	483. xalancbmk	
21	410. bwaves	
22	416. gamess	
23	434. zeusmp	
24	436. cactusADM	libc. so, libm. so, libgcc s. so
25	437. leslie3d	libstdc++. so, libgfortran. so
26	454. calculix	libquadmath. so
27	459. GemsFDTD	
28	465. tonto	
29	481. wrf	

$$\text{slowdown ratio} = \frac{\text{IPC}^{\text{alone}}}{\text{IPC}^{\text{shared}}}$$

其中, $\text{IPC}^{\text{alone}}$ 表示在无负载的情况下, SPEC CPU2006 基准测试程序运行时, 每个时钟周期执行的指令条数 (Instructions Per Clock, IPC); $\text{IPC}^{\text{shared}}$ 表示在有负载的情况下的 IPC。减速率的值越小, 说明隔离性能越好。

3.2 高负载的隔离性评估

实验中将 SPEC CPU2006 基准测试程序中的访存密集型 (Memory intensive) 程序 470. lbm 作为负载^[12], 来评估 ISMB 实现的性能隔离程度。实验过程如下: 首先, 在没有负载的情况下, 在 Core 0 上运行所有 SPEC CPU2006 基准测试程序, 并获取它们的 IPC 作为 $\text{IPC}^{\text{alone}}$; 然后, 在 Core 1~3 上分别运行三个 470. lbm 基准测试程序作为负载, 在 Core 0 上运行所有

SPEC CPU2006 基准测试程序, 并获取它们的 IPC 作为 $\text{IPC}^{\text{shared}}$; 最后, 利用公式 (1) 计算出 SPEC CPU2006 基准测试程序的减速率。

图 4 显示了将 470. lbm 作为负载的情况下, 所有 SPEC CPU2006 基准测试程序的规范化减速率。从图中可以看出, 在使用 ISMB 时, 大部分基准测试程序的性能较好, 除了如 471. omnetpp 之类的少数基准测试程序, 这些基准测试程序的性能下降的原因主要是 Bank 数量减少导致的性能下降抵消了 Bank 分区技术带来的性能提升。实验结果表明, 在使用 ISMB 的情况下, SPEC CPU2006 基准测试程序的减速率平均降低了 3.9%, 最大降低了 26.3% (470. lbm 基准测试程序, 规范化减速率为 0.737)。

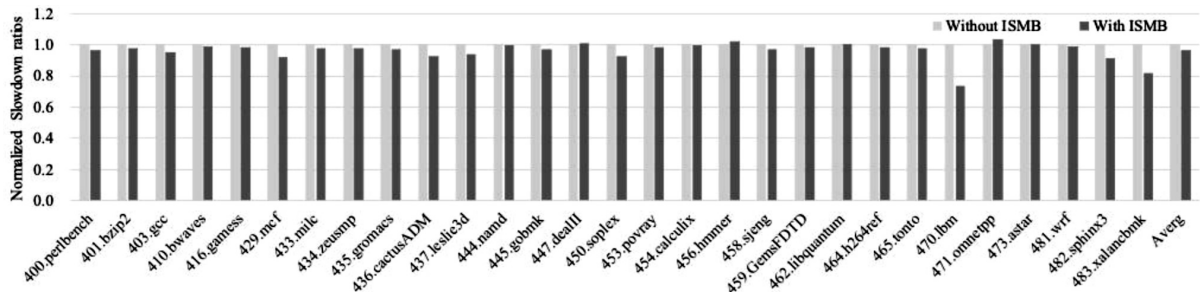


图 4 以 470. lbm 作为负载的 SPEC CPU2006 基准测试程序的规范化减速率

3.3 混合负载的隔离性评估

为了模拟更加真实的运行环境, 如表 4 所示, 从所有 SPEC CPU2006 基准测试程序中随机选择 10 组基准测试程序, 每个测试组包含 4 个基准测试程序。每个测试组的实验过程如下: 首先, 将测试组中的 4 个基

准测试程序分别同时运行在 4 个 Core 上, 并获取各自的 $\text{IPC}^{\text{shared}}$; 然后, 结合上一小节实验中获取的 $\text{IPC}^{\text{alone}}$, 计算出每个 SPEC CPU2006 基准测试程序的减速率; 最后, 计算出这 4 个基准测试程序的平均减速率, 作为测试组的减速率。

表 4 SPEC CPU2006 基准测试程序组

组号	SPEC CPU2006 基准测试程序
Mix1	445. gobmk, 464. h264ref, 473. astar, 470. lbm
Mix2	462. libquantum, 450. soplex, 471. omnetpp, 400. perlbenc
Mix3	437. leslie3d, 470. lbm, 410. bwaves, 401. bzip2
Mix4	444. namd, 453. povray, 433. milc, 450. soplex
Mix5	410. bwaves, 403. gcc, 482. sphinx3, 470. lbm
Mix6	465. tonto, 458. sjeng, 462. libquantum, 456. hmmer
Mix7	416. gamess, 434. zeusmp, 481. wrf, 462. libquantum
Mix8	459. GemsFDTD, 437. leslie3d, 470. lbm, 483. xalancbmk
Mix9	403. gcc, 470. lbm, 462. libquantum, 471. omnetpp
Mix10	454. calculix, 459. GemsFDTD, 470. lbm, 450. soplex

图 5 显示了 10 个测试组在使用 ISMB 的情况下, 每个测试组的规范化减速率。实验结果表明, 在混合负载的情况下, ISMB 的使用使 SPEC CPU2006 基准测试程序的减速率平均降低了 6%, 最大降低了 15.7%

(Mix8 测试组, 规范化减速率为 0.843)。因此, 与未使用 ISMB 相比, ISMB 可以有效地提升系统的隔离性能。

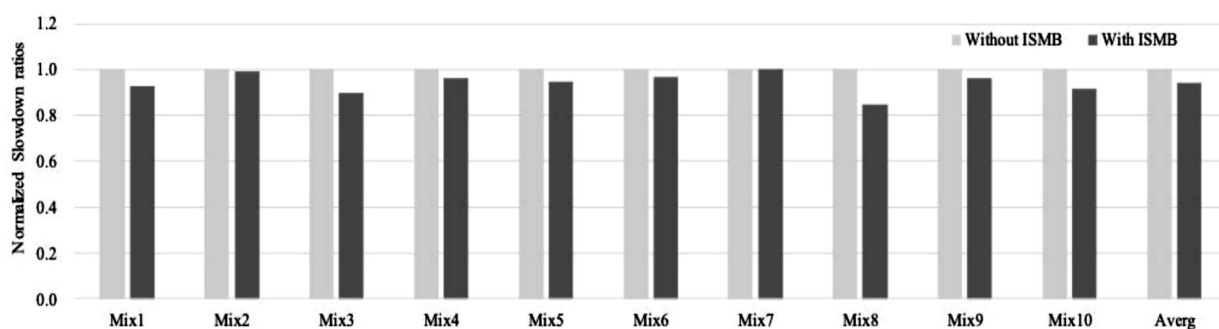


图 5 SPEC CPU2006 基准测试程序组的规范化减速率

4 结束语

该文提出的在多核系统中利用 DRAM Bank 分区技术实现共享库隔离的方案 (ISMB) 消除了共享库导致的 Bank 行缓冲区冲突问题,从而有效地提升了系统的整体性能。对比实验结果表明,与未使用 ISMB 的 Linux 相比,ISMB 能够有效地提高系统隔离性能。特别地,在高负载的情况下,SPEC CPU2006 基准测试程序的减速率最大降低了 26.3%,在混合负载的情况下减速率最大降低了 15.7%。提出的 ISMB 机制是基于静态 Bank 分区技术实现的,在未来的研究工作中,计划采用动态 Bank 分区技术来实现 ISMB 机制。从而在保证 ISMB 隔离性的同时,进一步提升系统的整体性能。

参考文献:

- [1] 任智源,杨伦,柴凯中. RAM 的现状与发展方向[J]. 电子元件与信息技术,2022,6(4):1-4.
- [2] 邱杰凡,华宗汉,范菁,等. 内存体系划分技术的研究与发展[J]. 软件学报,2022,33(2):751-769.
- [3] 王得利,高德远,王党辉,等. 存储器行缓冲区命中预测研究[J]. 计算机科学,2010,37(6):297-302.
- [4] LIU Wenjie,ZHOU Ke,HUANG Ping,et al. RBC:a memory architecture for improved performance and energy efficiency[J]. Tsinghua Science and Technology,2021,26(3):347-360.
- [5] FANG J, LU J, CAI M. Bank partitioning based adaptive page policy in multi-core memory systems[C]//2015 14th international symposium on distributed computing and applications for business engineering and science (DCABES). Guiyang:IEEE,2015:240-243.
- [6] FANG J, WANG M, WEI Z. A memory scheduling strategy for eliminating memory access interference in heterogeneous system[J]. The Journal of Supercomputing,2020,76(4):3129-3154.
- [7] IKEDA T, KISE K. Application aware dram bank partitioning in cmp[C]//2013 international conference on parallel and distributed systems. Seoul:IEEE,2013:349-356.
- [8] JEONG M K, YOON D H, SUNWOO D, et al. Balancing DRAM locality and parallelism in shared memory CMP systems[C]//2012 IEEE international symposium on high-performance comp architecture. New Orleans:IEEE,2012:1-12.
- [9] KIM H, DE NIZ D, ANDERSSON B, et al. Bounding memory interference delay in COTS-based multi-core systems[C]//2014 IEEE 19th real-time and embedded technology and applications symposium (RTAS). Berlin:IEEE,2014:145-154.
- [10] LIU Y, LU J, TONG D, et al. Locality-aware bank partitioning for shared DRAM MPSoCs[C]//2017 22nd Asia and South Pacific design automation conference (ASP-DAC). Chiba:IEEE,2017:770-775.
- [11] XIE M, TONG D, HUANG K, et al. Improving system throughput and fairness simultaneously in shared memory CMP systems via dynamic bank partitioning[C]//2014 IEEE 20th international symposium on high performance computer architecture (HPCA). Orlando:IEEE,2014:344-355.
- [12] YUN H, MANCUSO R, WU Z P, et al. PALLOC: DRAM bank-aware memory allocator for performance isolation on multicore platforms[C]//2014 IEEE 19th real-time and embedded technology and applications symposium (RTAS). Berlin:IEEE,2014:155-166.
- [13] LIU L, CUI Z, XING M, et al. A software memory partition approach for eliminating bank-level interference in multicore systems[C]//2012 21st international conference on parallel architectures and compilation techniques (PACT). Minneapolis:IEEE,2012:367-375.
- [14] SUBRAMANIAN L, LEE D, SESHADRI V, et al. The blacklisting memory scheduler: achieving high performance and fairness at low cost[C]//2014 IEEE 32nd international conference on computer design (ICCD). Seoul:IEEE,2014:8-15.
- [15] KIM Y, HAN D, MUTLU O, et al. ATLAS: a scalable and high-performance scheduling algorithm for multiple memory controllers[C]//The 16th international symposium on high-performance computer architecture. Bangalore:IEEE,2010:1-12.
- [16] KIM Y, PAPAMICHAEL M, MUTLU O, et al. Thread clus-

- ter memory scheduling; exploiting differences in memory access behavior [C]//2010 43rd annual IEEE/ACM international symposium on microarchitecture. Atlanta: IEEE, 2010: 65–76.
- [17] WU Y, SATHYANARAYAN S, YAP R H, et al. Codejail: application-transparent isolation of libraries with tight program interactions [C]//2012 European symposium on research in computer security. Pisa: Springer, 2012: 859–876.
- [18] GOONASEKERA N A, WILLIAM C, COLIN F. LibVM: an architecture for shared library sandboxing [J]. Software: Practice and Experience, 2015, 45 (12): 1597–1617.
- [19] QIANG W, CAO Y, DAI W, et al. Libsec: a hardware virtualization-based isolation for shared library [C]//2017 IEEE 19th international conference on high performance computing and communications; IEEE 15th international conference on smart city; IEEE 3rd international conference on data science and systems (HPCC/SmartCity/DSS). [s. l.]: IEEE, 2017: 34–41.
- [20] KIM N, CHISHOLM M, OTTERNESS N, et al. Allowing shared libraries while supporting hardware isolation in multicore real-time systems [C]//2017 IEEE real-time and embedded technology and applications symposium (RTAS). Pittsburgh: IEEE, 2017: 223–234.
- [21] PESSL P, GRUSS D, MAURICE C, et al. DRAMA: exploiting DRAM addressing for cross-CPU attacks [C]//2016 25th USENIX security symposium (USENIX security 16). [s. l.]: USENIX, 2016: 565–581.
- [22] 张文文. DRAM 地址线位数及容量识别方法 [J]. 数字化用户, 2018, 24 (37): 251.
- [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]//2013 Proceedings of the eighteenth international conference on architectural support for programming languages and operating systems. [s. l.]: [s. n.], 2013: 181–192.
- [24] XIAO Y, ZHANG X, ZHANG Y, et al. One bit flips, one cloud flops: cross-vm row hammer attacks and privilege escalation [C]//2016 25th USENIX security symposium (USENIX security 16). [s. l.]: USENIX Association, 2016: 19–35.
- [25] WANG M, ZHANG Z, CHENG Y, et al. Dramdig: a knowledge-assisted tool to uncover dram address mapping [C]//2020 57th ACM/IEEE design automation conference (DAC). San Francisco: IEEE, 2020: 1–6.
- [26] HASSAN M, KAUSHIK A M, PATEL H. Reverse-engineering embedded memory controllers through latency-based analysis [C]//2015 21st IEEE real-time and embedded technology and applications symposium. Seattle: IEEE, 2015: 297–306.
- [27] HELM C, AKIYAMA S, TAURA K. Reliable reverse engineering of intel dram addressing using performance counters [C]//2020 28th international symposium on modeling, analysis, and simulation of computer and telecommunication systems (MASCOTS). [s. l.]: IEEE, 2020: 1–8.
- [28] PAN X, GOWNIVARIPALLI Y J, MUELLER F. Tintmallo: reducing memory access divergence via controller-aware coloring [C]//2016 IEEE international parallel and distributed processing symposium (IPDPS). Chicago: IEEE, 2016: 363–372.
- [29] PAN X, MUELLER F. Controller-aware memory coloring for multicore real-time systems [C]//2018 proceedings of the 33rd annual ACM symposium on applied computing. [s. l.]: ACM, 2018: 584–592.
- [30] HENNING J L. SPEC CPU2006 benchmark descriptions [J]. ACM SIGARCH Computer Architecture News, 2006, 34 (4): 1–17.
- while summarizing: multi-task learning for multi-hop QA with evidence extraction [C]//Proceedings of the 57th ACL. Florence: ACL, 2019: 2335–2345.
- [50] TANG Y X, NG H T, TUNG A K H. Do multi-hop question answering systems know how to answer the single-hop sub-questions? [J]. arXiv:2002.09919, 2020.
- [51] CAO X, LIU Y. Coarse-grained decomposition and fine-grained interaction for multi-hop question answering [J]. arXiv:2101.05988, 2021.

(上接第 16 页)

science question answering [J]. arXiv:2105.11776, 2021.

- [47] MIN S, ZHONG V, ZETTLEMOYER L, et al. Multi-hop reading comprehension through question decomposition and rescoring [C]//Proceedings of the 57th ACL. Florence: ACL, 2019: 6097–6109.
- [48] MIN S, WALLACE E, SINGH S, et al. Compositional questions do not necessitate multi-hop reasoning [C]//Proceedings of the 57th ACL. Florence: ACL, 2019: 4249–4257.
- [49] NISHIDA K, NISHIDA K, NAGATA M, et al. Answering