软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2022,33(2):751−769 [doi: 10.13328/j.cnki.jos.006370]

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

内存体系划分技术的研究与发展*

邱杰凡[1], 华宗汉[1], 范 菁[1], 刘 磊[2]

[1](浙江工业大学 计算机科学与技术学院, 浙江 杭州 310023)

[2](计算机体系结构国家重点实验室 Sys-Inventor Lab (中国科学院 计算技术研究所), 北京 100190)

通信作者: 刘磊, E-mail: lei.liu@zoho.com; liulei2010@ict.ac.cn; https://liulei-sys-inventor.github.io/

摘 要: 在多核计算机时代, 多道程序在整个共享内存体系上的"访存干扰"是制约系统总体性能和服务质量的重要因素. 即使当前内存资源已相对丰富, 但如何优化内存体系的性能、降低访存干扰并高效地管理内存资源, 仍是计算机体系结构领域的研究热点. 为深入研究该问题, 详述将"页着色(page coloring)"内存划分技术应用于整个内存体系(包括 Cache、内存通道以及内存 DRAM Bank), 进而消除了并行多道程序在共享内存体系上的访存干扰的一系列先进方法. 从 DRAM Bank、Channel 与 Cache 以及非易失性内存(non-volatile memory, NVM)等内存体系中介质为切入点, 层次分明地展开论述: 首先, 详述将页着色应用于多道程序在 DRAM Bank 与通道的划分, 消除多道程序间的访存冲突; 随后是将页着色应用于在内存体系中 Cache 和 DRAM 的"垂直"协同划分, 可同时消除多级内存介质上的访存干扰; 最后是将页着色应用于包含 NVM 的混合内存体系, 以提高程序运行效率和系统整体效能. 实验结果表明, 所提内存划分方法提高了系统整体性能(平均 5%–15%)、服务质量(QoS), 并有效地降低了系统能耗. 通过梳理和总结, 较为全面地展现了内存体系划分技术的核心思想、关键技术、应用架构及发展脉络. 对未来优化内存体系性能、服务器性能及服务质量相关的工作提供了参考.

关键词: 页着色; 共享内存; 访存干扰; 内存划分

中图法分类号: TP303

中文引用格式: 邱杰凡, 华宗汉, 范菁, 刘磊. 内存体系划分技术的研究与发展. 软件学报, 2022, 33(2): 751–769. http://www.jos.org.cn/1000-9825/6370.htm

英文引用格式: Qiu JF, Hua ZH, Fan J, Liu L. Evolution of Memory Partitioning Technologies: Case Study through Page Coloring. Ruan Jian Xue Bao/Journal of Software, 2022, 33(2): 751−769 (in Chinese). http://www.jos.org.cn/1000-9825/6370.htm

Evolution of Memory Partitioning Technologies: Case Study through Page Coloring

QIU Jie-Fan[1], HUA Zong-Han[1], FAN Jing[1], LIU Lei[2]

[1](School of Computer Science and Technology, Zhejiang University of Technology, Hangzhou 310023, China)

[2](Sys-Inventor Lab, State Key Laboratory of Computer Architecture (Institute of Computing Technology, Chinese Academy of Sciences), Beijing 100190, China)

Abstract: This paper shows the technical evolution of memory partitioning on shared memory systems using the cases of page coloring. Multiple co-running applications incur memory contentions across the entire memory hierarchy in the age of multicore, negatively impacting the overall system performance and the QoS. Researchers are always seeking new solutions to aid these problems. The paper presents the well-known and well-developed technology—"Page coloring" and summarizes its evolution since 1990. This paper introduces how to leverage page coloring on Cache, DRAM/NVM Banks in main memory systems, and memory channels. On the Bank level, concurrent applications are mapped to a different group of Banks, thus eliminating inter-application memory conflicts. In terms of Cache and DRAM Banks, this paper introduces the "vertical partitioning" to mitigate memory conflicts across the multiple levels at the

* 基金项目: 国家自然科学基金(62072432, 61502452, 61502427); 国家重点研发计划(2018YFB1402800); 浙江省自然科学基金(LY20F020026)

收稿时间: 2020-07-29; 修改时间: 2020-12-30; 采用时间: 2021-04-30; jos 在线出版时间: 2021-10-20

memory hierarchy at the same time. Finally, this study shows how to use page coloring on systems equipped with hybrid NVM systems. The experimental results show that the memory partitioning technologies through page coloring bring significant performance benefits and QoS.

Key words: page coloring; shared memory; memory conflict; memory partitioning

多核计算机系统已成为时代的主流. 适用于早期单核处理器操作系统(operating system, OS)的核心机制(如内存管理等), 在面对日趋复杂的计算环境与内存架构以及多样化的计算需求时, 已很难建立从数据到内存体系的最佳映射, 造成大量的线程间访存互扰和冲突, 进而导致计算机吞吐量(throughput)、服务质量(quality of service, QoS)与理想状态相去甚远[1−6].

以被广泛使用的 Linux 操作系统为例, 其内存管理机制采用了近乎“盲目”的随机物理页面分配机制. 即: 在出现“页缺失”时, 从伙伴系统(buddy system)[2,7,8]中随机取出一个(或地址连续的多个)物理页面, 通过 OS 在页表中建立虚实映射. 这种随机分配方式直接导致数据在物理内存介质(DRAM Bank)上映射的位置是不确定的[3]; 这种不确定性导致被频繁访问的页面可能会被随机地分配到同一个物理内存 Bank 中, 造成 Bank 上的访存冲突[3]. 与此同时, 在片上 Cache 中也极有可能产生在 Cache Set 上的数据分布不均[4], 引发较多的 Cache 缺失, 影响系统的总体性能. 总之, 由于现有的内存管理与分配机制未考虑数据在物理介质上的运行时分布状态, 导致整个内存体系的资源利用率始终处于较低的状态[2,5,9,10].

内存管理机制的缺陷由来已久, 研究人员有针对性地开展了一系列研究. 其中, 具有里程碑意义的工作是 SUN 公司于 1985 年前后, 在 MIPS 操作系统中引入的“页着色(page-coloring)”技术. 其核心思想是: 在建立虚实映射时, 让虚拟地址的低位和物理地址的低位满足一定的函数关系(如相等、固定偏移等), 使得读入 Cache 之后的数据分布不会出现明显的不均.

在 20 世纪 90 年代, 页着色技术发展迅速, 其中较有代表性的成果来自 Richard 等人[11]的研究, 他们发现: 页着色技术会导致不同地址空间(如数据段、代码段、栈等)出现在同一个 Cache Bin 中(一个 Bin 包含一组 Cache Set), 进而造成访存竞争和干扰, 使得 Cache 利用率下降. 为了解决该问题, 他们提出了 Bin Hopping 和 Best Bin 改进经典的页着色技术, 以 Cache Bin 为单位, 让连续的页面映射到连续的 Cache Bin 中, 减少在 Cache 上的冲突. 此方法提出之后, 被学术界和工业界广泛接受, 从已公开的资料来看, Windows 操作系统的内存管理以及 VMware 虚拟机资源管理体系均已采用相关页着色技术.

进入 21 世纪后, 在采用多核处理器的系统中, 来自多个计算单元的访存请求会引发在整个共享内存体系(包括 Cache 和 DRAM Bank)上的冲突和互扰[2,10,12], 造成内存资源利用率不高, 进一步拉大了计算速率与访存速率之间的差距. 此时, 页着色技术再一次走进人们的视野, 被用于消除/降低多道程序、线程之间在 Cache 上的访存互扰. 其核心思想是: 利用物理页帧号(page frame number, PFN)中的 Cache Set 的索引位, 将整个物理地址空间以页为单位划分为不同“颜色”的区域, 每种颜色代表一段物理地址空间, 并对应一组 Cache Set. 在物理页分配的过程中, 依据颜色映射数据, 即: 相同颜色的页帧将会分配到相同的 Cache 区域; 不同颜色的页面在映射之后, 不会存在重叠的 Cache 资源, 从而实现不同程序在 Cache 上的划分, 消除了程序间在 Cache 上的相互竞争和干扰. 鉴于此方法的有效性以及在工业界应用的潜力, 在随后的几年, 又出现了一系列有代表性的工作[13,14].

程序之间的访存干扰同样存在于内存系统的 DRAM Bank 中. 虽然业界设计了很多访存调度机制[15,17−21,35], 但是这些方法可能需要复杂的硬件逻辑, 调度算法较为固化, 性能在不同应用场景中的差异较大, 且较难部署和大规模应用(需要大规模流片). 本文将介绍一种 DRAM Bank 划分机制, 该机制首次成功地将页着色技术应用于对 DRAM Bank 的划分[3](在某些计算平台上, 主存地址映射中存在若干地址位能够直接索引 Bank[3]), 进而消除了程序间在主存上的访存干扰, 提高了 Bank 的并发度和响应速率.

在此基础上, 本文进一步引入了内存通道(channel)划分, 在消除程序间主存 Bank 上相互干扰的同时, 也降低了程序间在内存通道上的访存竞争[22].

上文介绍的内存划分机制均为“水平”的方式, 即仅划分 Cache 或仅划分 DRAM Bank 的单一层次的内存资

源. 本文通过对大量真实的工作集运行情况的深入分析, 介绍了首次发现的内存体系“垂直”优化空间的全过程[2], 亦即通过地址映射中能够同时索引 DRAM Bank 和 Cache Set 的地址位协同划分多级存储器, 进而获得更大的性能收益. 垂直划分也可通过页着色完成, 为访存优化提供了新方向和实践方法. 随着内存体系的发展, 非易失性存储单元(non-volatile memory, NVM)以其高密度、大容量和低能耗等优点逐步走进人们的视野, 并广泛应用于主流服务器. 但 NVM 并未完全替代 DRAM, 这两种内存材质将长期并存构成混合内存体系.

近年来, 研究者为充分发挥 DRAM 与 NVM 的特点, 开展了许多针对 DRAM-NVM 混合内存架构的研究[9,10,23−28]. 本文介绍了一种最新的面向 DRAM-NVM 的内存管理机制[9], 是利用页着色技术的最新成果. 该研究工作不但利用内存体系中的地址位索引(Bank id、Cache set id、Channel id)进行着色, 还对物理页面所在的内存材质进行着色, 区分页面属性, 并进行高效的访存行为监测、页面迁移、数据排布等管理操作.

本文后继内容将梳理、剖析“页着色”技术的发展历史及近年来新的研究工作, 促发更多的关于现代操作系统内存资源管理机制的设计思路的思考. 本文第 1 节介绍“页着色”技术的历史与发展. 第 2 节对基于“页着色”的 DRAM 划分方法进行总结和介绍, 包含 DRAM Bank 单一层次的划分和 DRAM Bank 与通道两个层次的划分. 第 3 节总结上述技术在 Cache 以及 DRAM 内存体系中的“垂直”划分. 第 4 节介绍当前“页着色”技术在新兴 DRAM-NVM 混合内存体系中的发展与应用. 第 5 节总结“页着色”技术过去的发展思路, 并展望内存划分技术未来的发展趋势.

1 “页着色”技术的发展历史与演进

中央处理器(CPU)处理数据的速率要远比由 DRAM 组成的主存系统排出数据的速率高很多. 通常, 一条访存请求需要 50–100 个时钟周期才能完成, 而在一个时钟周期内, 处理器能执行若干条指令. 这意味着处理器绝大多数的时间都在等待主存的响应, 因此, 内存瓶颈制约了处理器性能的发挥. 这种处理器与主存之间的处理能力的差异即是“内存墙”[29]. 为了缓解“内存墙”, 高速缓存(Cache)技术被提出. 由静态随机存取存储器(static random-access memory, SRAM)构成的 Cache 响应速度大于主存, 缓存一部分近期被访问过的数据(不同的缓存算法缓存的数据有所不同). 对于应用程序, 处理器先从 Cache 中读取数据, 而不是从主存中读取数据, 显著提高了系统性能. 但是, 系统设计者却发现: 由于从虚页到实页的“任意映射(arbitrary mapping)”, 数据块在采用物理地址索引的 Cache 中的分布通常是不均匀的. 这导致系统性能通常会产生很大的波动, 即使是同一道程序的多次运行, 执行时间也可能会发生明显的变化. 为了解决此类问题, 1985 年, MIPS 操作系统引入了“页着色(page coloring)”技术, 这是有文献可查的第一次正式出现的“页着色”这一技术词汇[1,11,30]. 研究人员设计了新的页面虚实转换的算法: 在映射过程中, 让物理页面的“低几位(low order bits)”与虚拟页面的“低几位”满足一定的函数关系(如固定偏移距离等). 以某几个地址位为依据, 将不同的虚、实页面对应不同的“颜色”(例如: 低 3 位为 000 的页面为“红色”, 001 的页面为“蓝色”). 因此, 从主存到 Cache 的页面映射不再是随机、任意和无序的, 而是满足某种“颜色”的对应关系, 从而保证了 Cache 中不会出现明显的数据分布不均匀的情况.

然而, 在多核时代, 由于多道程序并发执行, 每道程序发出的访存请求会在 Cache 上产生访存竞争, 导致 Cache 命中率降低, 从而使得计算机整体性能下降. 为了解决这一问题, 有研究者再次诉诸于页着色技术, 通过给每个程序分配不同颜色的页面, 使得不同程序仅使用分给它对应颜色的一部分 Cache 资源, 以此消除程序间在 Cache 上的访存竞争. 具体做法是: 当在程序运行过程中发生“页缺失”时, OS 为该程序分配仅属于其所对应颜色的物理页面, 因此实现了不同程序在 Cache 上的隔离, 消除了彼此之间的干扰. 如图 1 所示, 由 16 路组相连的 8MB 的 LLC(last level Cache)可以与 8GB 主存构成地址映射关系. 其中, PFN 中的 12–18 位为 Cache Set 索引位. 通过对 Cache Set 位中第 12 位和第 13 位的着色, 就可以实现 4 个程序在 Cache 上的隔离.

上文中提到的页着色仍然是静态的. 静态的页着色是指页着色不会在程序执行过程中更改映射. 虽然静态的页着色易于实现且有较低的开销, 但无法满足程序对 Cache 资源的动态变化的需求[31,32]. 页面迁移[33]可使静态页着色技术支持动态“重着色”[33], 进而根据运行时实际需求调整页面分配及着色规则. “重着色”技术

的核心是“页面迁移(page migration)”，该技术对 OS 系统内存管理机制影响深远.

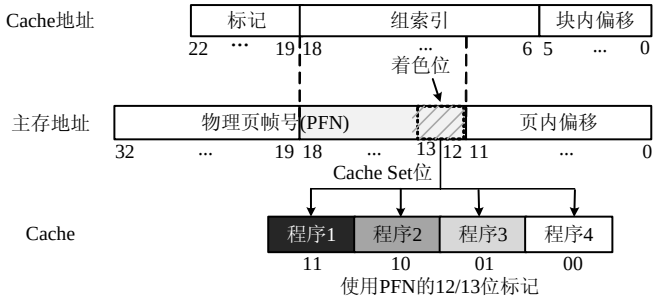


图 1 基于页着色技术的 Cache 划分原理图

此外，研究人员设计了一系列编译制导的页着色方法[16,34]. 研究人员意识到: 通过编译技术，将具有时间局部性(temporal locality)的代码和数据映射至不同的 Cache Set，可以提升数据缓存(data Cache)和指令缓存(instruction Cache)的性能[16]. 他们使用软硬件结合的方法对 Cache 中的页面重排，提高了 Cache 命中率. 在软件层，利用编译器预先为代码页和数据页提供颜色映射，随后，OS 使用这种映射指导物理页的分配. 在硬件层，向页表缓存(TLB)添加一个页面重新映射字段，控制是否将页面重新映射到 Cache 中的不同“颜色”的空间(由物理地址索引表示). 国内也有研究者[36]将页着色技术应用于 Cache 划分. 该研究通过动态内存分配的方式，在不修改程序源码与系统内核的前提下，实现动态的 Cache 划分. 通常，这类研究的不足之处在于，需要为每个程序单独划分 Cache，如果工作集中程序数量增加，系统性能提升将在短时间内达到上限，甚至出现负收益.

工业界对页着色技术产生了浓厚的兴趣，并分别从降低地址映射开销以及提高 Cache 使用率的角度出发做出了大量的尝试. SUN 早期版本的 UNIX 操作系统里就采用了页着色技术. 20 世纪 90 年代，页着色的优势还常常体现在降低地址转换时候的硬件开销上. 因为在采用页着色的机器上，虚拟页和物理页在若干个低位上已经建立了对应关系，因此这种设计减轻了负责地址转换的内存管理单元(memory management unit, MMU)的设计开销. VMware 考虑到 Guest OS 可能使用页着色技术，曾经在虚拟机监控系统 ESX Hypervisor 中包含了提供页着色优化的兼容版本. 该版本可以在为多个 Guest OS 分配物理页面的时候，采用页着色技术来匹配 Guest OS 所申请页面的颜色. 换言之, Hypervisor 尽可能满足 Guest OS 对颜色的需求，以最大限度地保留客户系统的优化手段.

在上述研究工作中，页着色仅应用于对 Cache 的优化/划分，而未考虑将其应用到其他内存资源划分. 本文介绍了一种 DRAM Bank 层次上的划分机制: BPM(Bank partitioning mechanism). 该机制把主存 DRAM Bank 均分成几组，每个程序只占用一组 DRAM Bank，程序之间不交迭，因此在根源上消除了程序间在同一 Bank 上的访存干扰；本文进一步介绍了对内存通道的划分(BPM/BPM+)[22]，以此消除程序在内存通道上的竞争，提高系统整体吞吐量. 最后，本文介绍了一种基于页着色技术的“垂直”划分方法[2]以及“非对称”的垂直划分机制[5]，并介绍了将这一系列的方法应用于对 DRAM-NVM 混合内存体系架构上的实践经验[9]. 本文对近年有代表性的 LeiLiu 等人的相关工作进行了梳理汇总(见表 1)，下文将详细介绍若干关键性的工作.

表 1 页着色技术近 10 年来的主要代表性工作

年份	工作	优化的存储器	创新
2012	BPM[3]	DRAM Bank	首次，在真机上将页着色应用于 DRAM Bank 的划分
2014	BPM/BPM+[22]	DRAM Bank	在 BPM[3]的基础上引入内存通道划分
2014	HVR[2]	Cache, DRAM	利用同时索引 Cache Set 与 DRAM Bank 的地址位“垂直”划分多级存储器
2016	Curve-VP[5]	Cache, DRAM	在垂直划分[2]基础上，设计了非对称的“垂直”划分
2019	Memos[9,37]	Cache, DRAM-NVM 混合内存	对包含 Cache、DRAM Bank、NVMBank 的整个混合内存体系的划分

2 基于页着色技术的 DRAM Bank 划分

在多核时代, 程序间的冲突不仅发生在 Cache 上, 同样发生在 DRAM Bank 与内存通道上. 有学者将页着色技术应用于 DRAM Bank 与通道的划分上, 用软件方法消除、降低多道程序间在主存系统上的访存干扰[3,22].

2.1 访存调度算法

在多核共享内存的计算机系统中, 当多个程序的访存请求落在同一个 Bank 上时, 将引发 Bank 上的访存冲突, 导致行缓冲命中率下降, 进而造成访存性能的下降, 并影响系统的整体性能[3].

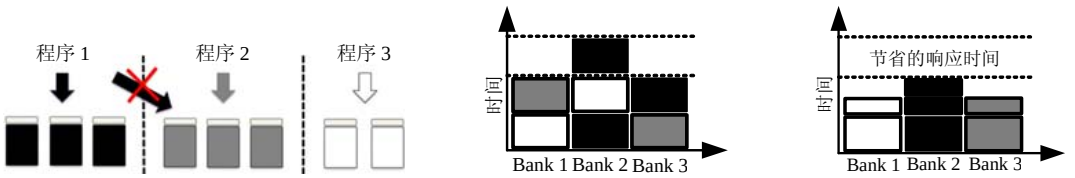
目前被工业界广泛使用的内存调度算法的基础是 FR-FCFS(first ready-first come first service), 该算法简洁有效且易于实现. 其核心思想是: 在先来先服务(first come first service, FCFS)算法的基础上, 增加了对 DRAM 行缓冲的局部性(row buffer locality)的考量, 即: 在打开某个 Bank 的行缓冲(row buffer)之后, 尽可能地让调度窗口中潜在能够命中该行的后继请求优先于其他的访存请求. 虽然这种以最大化行缓冲利用率为目的的调度方式易于实现, 但在公平性和吞吐量方面的表现却不甚理想[15].

国内外学者提出了很多先进的内存控制器中的调度算法[10,15,17−20,35], 但这些算法均存在着诸多不确定因素. 例如, 较为先进的线程簇内存调度(thread cluster memory scheduling, TCM)算法[15]不仅需要引入额外的至少 4K 的存储空间, 且调度时需要复杂的硬件逻辑支持. 虽然 TCM 调度算法在性能上有较好的表现, 但芯片设计与生产的复杂性阻碍了该算法的实际应用. 因此, 虽然当前先进的调度算法在大多数场景下能够缓解内存的访存干扰问题, 但仍难以在根源上彻底消除线程间的访存干扰.

2.2 DRAM Bank划分(Bank partitioning mechanism, BPM)

通过对主存地址映射的研究, 研究人员发现, 在主存地址映射中存在着若干地址位能够直接索引 DRAM Bank[3,38]. 例如, Intel i7-860 处理器在配置 8 GB DDR3 内存、64 个 DRAM Bank(每个 Bank 125MB)时, 其地址映射的第 13 位–第 15 位和第 21 位、第 22 位可以索引 Bank 的地址位. 这启发了研究者使用页着色技术消除程序间在 DRAM Bank 上的访存干扰问题. 即: 通过页着色技术, 对上述 5 个 Bank 索引位着色, 即可将系统中所有的 DRAM Bank 最多划分为 2^5=32 组, 让每个程序使用且仅使用其中一组(或若干组)Bank.

至此, 一种从根源上消除进程间访存干扰的机制 BPM(Bank partitioning mechanism)[3]被提出. 如图 2(a)所示: 在 BPM 中, 程序 1 只能访问其对应的 Bank 而不能越界访问属于程序 2 的 Bank, 有效消除了程序间的访存干扰, 程序自身的行缓冲局部性也得以保证. 值得注意的是: 在每个 Bank 组内部, 访存行为仍然是“交替”的. 在运行时可以明显看到: 不采用 Bank 划分, 不同程序的访存请求被“交替”分配到任意 Bank 上, 引发了行缓冲颠簸, 因此会导致部分访存请求的时延较长, 如图 2(b)所示; 相反, 采用 Bank 划分时, 程序之间的访存干扰被消除, 每个程序的行缓冲局部性有所提高, 故访存请求的响应时延相对减小, 如图 2(c)所示. BPM 既有利于系统的整体性能, 同时也提升了系统的服务质量(QoS). BPM 是有资料可查的首次在真实的多核服务器上实现的主存 DRAM Bank 划分机制, 为后来很多内存优化相关的工作奠定了基础.



(a) 在 DRAM Bank 上的着色 (b) 不采用 Bank 划分运行效果 (c) 采用 Bank 划分的运行效果

图 2 在 DRAM Bank 上的页着色技术(BPM)及在是否启用 BPM 情况下的性能对比

另一方面, 为了获得更高的性能提升, 为每个程序所分配的 Bank 的数量是需要考虑的重要问题. 以 SPECCPU 2006[39]中的程序作为测试用例, 图 3 显示了不同 Bank 容量下单个程序运行性能的变化[3]. 可以看出: 对 90%的程序来说, 16 个 Bank(每个 Bank 容量 125MB)即能满足程序的需要, 分配更多的 Bank 并不能显

著提高程序性能. 这说明程序对 Bank 数量的需求是有限的, 满足容量需求后, 分配更多的 Bank 对提高程序性能不起作用. 因此, 对于 SPECCPU 2006 中的程序, 由于每个程序的内存足迹(MemoryFootprint)较小, 在划分 Bank 时, 直接使用均分的方法也能满足需求[22].

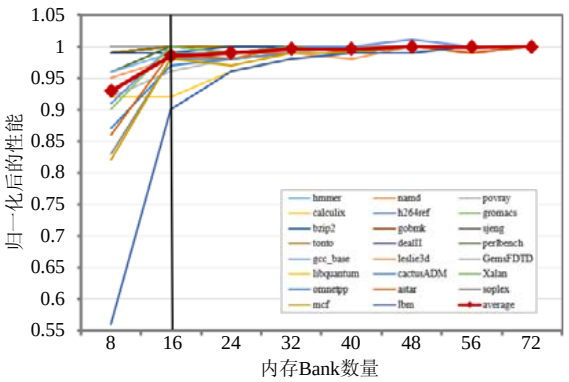


图 3 单个程序分配 Bank 的数量与其性能之间的关系[3]

为了验证 Bank 划分的有效性, 实验中使用加权加速比(weighted speedup, WS)来衡量系统的吞吐量, 而公平性(QoS)用工作集中性能下降最大的程序来表示(max slowdown)[15], 具体表达式为

$$Weighted\ \ Speedup = \sum \frac{运行时间_{alone}}{运行时间_{shared}},$$

$$\text{Max } Slowdown = \text{MAX}\left\{\sum \frac{运行时间_{alone}}{运行时间_{shared}}\right\}.$$

实验在 10 组随机产生的工作集中开展(每组工作集包含若干个 SPECCPU 2006 中的程序)[3], 对采用和不采用 BPM 情况下系统的吞吐量和公平性分别进行测量, 结果如图 4 所示: 使用 Bank 划分后, 系统的吞吐量均有提升, 平均提升 4.8%, 最高达 6%; 在公平性方面, 最高能达到 16%的提升, 但存在对部分工作集 QoS 下降的情况. 可见, BPM 能够带来的性能提升是可观的. 当前, 计算机性能已接近极限, 能够提高平均 1%的性能都是有意义的重大改进. 详细的性能分析可见 LeiLiu 等人的工作[2−5,9]. 采用静态的 Bank 划分方法后, 每个程序能使用的 Bank 数量是固定的. 但在有些情况下, 随着程序的运行某个程序的内存足迹增大, 对 Bank 需求增加时, 静态 Bank 分配将会导致现有分配不足的情况. 为了解决这一问题, 研究人员提出了动态的 Bank 划分方法[22]. 其核心思想是: 根据运行时的信息, 实时动态地调整各个程序划分的 Bank 数目. 基于文献[3,22]提出的动态划分机制, 针对不同的优化目标, 内存控制器中的访存调度方法可以是多样化的. 实验结果表明: 在大内存足迹负载下, 动态的 Bank 划分机制在提升系统性能方面有更好的表现.

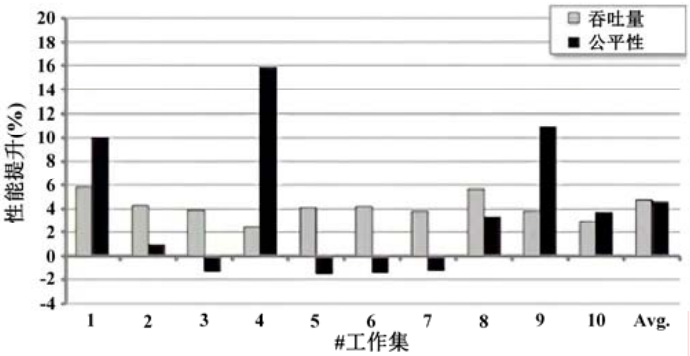


图 4 BPM 带来的性能提升

为了增加系统的总体带宽, 现代的计算机普遍采用多通道技术. 例如, DDR3-1600 的峰值带宽是 12.8 GB/s, 双通道系统的理论峰值带宽即为 25.6 GB/s. 对于主流的多通道计算机, 通常以 Cache line(64B)为单位“交替”访问, 将访存请求平均分布在若干通道, 这样可以达到充分利用带宽的目的. 但是, 如图 5 所示(不同颜色的色块代表来自不同程序的访存请求), 盲目地交替访存实际上会加剧多道程序在通道内的竞争, 同时也可能会引发程序在 DRAM 上更多的冲突.

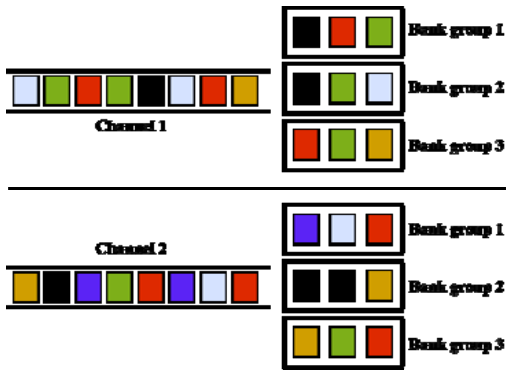


图 5 程序在通道上“交替”访存

2.3 通道划分(BPM+)

当前, 主流平台(如 Intel i7 系列)的 BIOS 为用户提供了可配置的选项, 可控制是否在通道上执行“交替”访问的策略. 如果取消“交替”访问, 并利用页着色技术对“通道位”进行着色, 这意味着可以通过页着色技术来控制程序访存在通道上的分布, 从而实现通道划分技术.

在传统的 DRAM Bank 划分的基础上, 一种消除通道间访存干扰的机制(BPM+)[22]被提出. 该方法通过对通道进行划分, 让每个程序仅使用一条通道, 并在通道内部使用 Bank 划分. 如图 6 所示: BPM+不仅消除了程序在 DRAM Bank 上的竞争, 还缓解了程序间在通道上的竞争, 所以获得了进一步的性能提升. 同样, 在配置 Intel i7-860 处理器、8 GB DDR3 内存、64 个 DRAM Bank(每个 Bank 125MB)、双通道服务器上, 采用 BPM+的性能平均要比仅采用 BPM 时提高接近两个百分点.

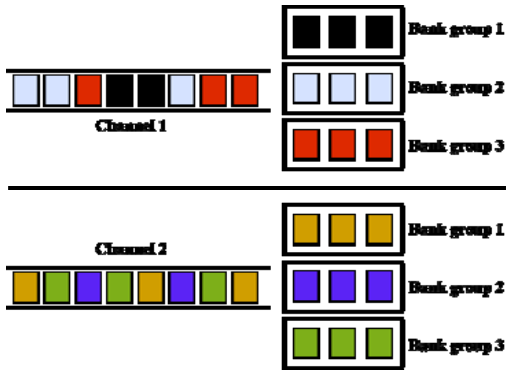


图 6 使用 BPM+的效果图

3 基于页着色技术的垂直划分

面对日趋复杂的计算需求以及逐渐加深且相互影响的内存层次结构, 仅在单一某个内存体系层次上进行划分、优化等操作不能最大化系统性能. 例如高速缓存 Cache 和主存 DRAM Bank, 它们之间存在着密切关联, 性能相互影响, 如果仅在一个层次上使用划分技术, 则欠缺了对整个内存体系性能统一的考量, 并不能很好

地解决应用程序之间在 Cache 与 DRAM Bank 上同时发生访存干扰的问题. 在此背景下, 本节将介绍基于页着色的多级内存体系“垂直”协同划分机制的发现、原理及应用.

3.1 仅划分Cache和仅划分DRAM Bank的性能分布

Liu 等人[2,5]对 200 余组由 SPECCPU 2006[39]中的程序组成的多道程序工作集展开了测试和详细分析. 在 200 余组工作集中(这些工作集包括 4 道或 8 道程序不等), 分别独立执行仅 Cache 划分和仅 DRAM Bank 划分, 可以得到如图 7 所示的性能分布结果图. 图中每一个点代表一个工作集, x 轴是仅使用 DRAM Bank 划分得到的性能提升, y 轴是仅使用 Cache 划分得到的性能提升.

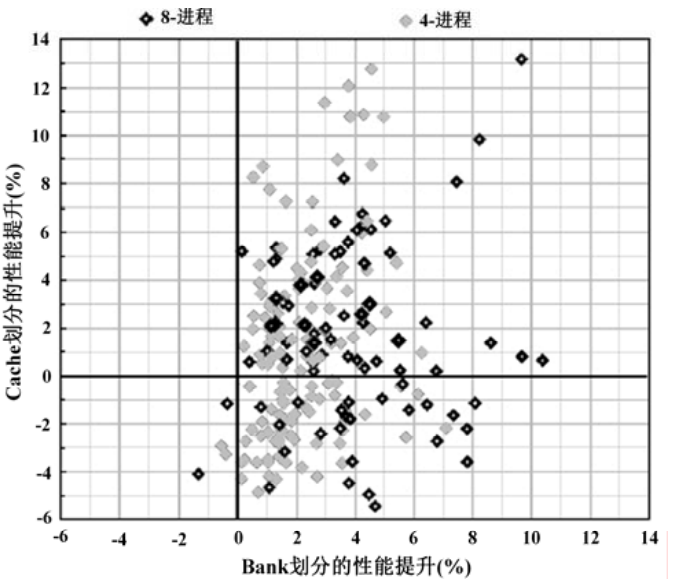
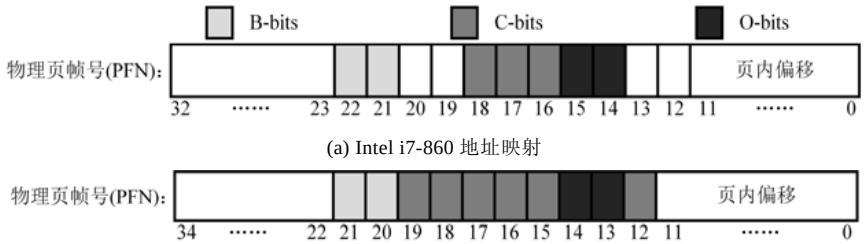


图 7 Cache 划分和 DRAM Bank 划分的性能变化图

从图 7 中可以很清晰地看到, 大多数点都分布在第 1 象限和第 4 象限. 第 1 象限的点表示针对这些工作集, 分别执行 Cache 划分和 DRAM Bank 划分能有效提升系统性能. 第 4 象限的点表示针对这些工作集, 执行 DRAM Bank 划分可以有效提升系统性能, 但执行 Cache 划分会降低系统总体性能. 从图中可以看出: 在大多数情况下, 对 DRAM Bank 进行独立划分可以有效提高系统性能, 而 Cache 划分则只对部分工作集有效. 这些实验具有丰富的启发意义: 第一, 能否设计一种机制, 针对第 1 象限中的工作集, 可以将两种划分机制带来的好处累加在一起? 第二, 很明显可以看出, Cache 划分是工作集敏感的, 因此在实际应用中要根据工作集特点调整划分算法; 第三, Bank 划分普遍能对系统性能带来正反馈, 是一种积极的优化策略, 但问题是性能普遍来讲提高得并不多, 平均只有 4%左右.

Liu 等人经过深入调研发现: 在主流多核计算机系统的地址映射中, 除了包含能够索引 Bank 的地址位(Bank bits, B-bits)和索引 Cache 的地址位(Cache bits, C-bits)之外, 还存在一类 DRAM Bank 和 Cache 重合的地址位(overlapped bits, O-bits), 能够同时索引 DRAM Bank 和 Cache set. 这表明, 对 O-bits 着色就能够同时协同划分内存体系中的 DRAM Bank 和 Cache Set. 这种划分方式因为贯穿了内存体系, 即被称为“垂直”划分[2,5]. 主流服务器的地址映射可被软件检测获得或由合作芯片设计厂商提供, 这种地址映射为在通用计算机系统上实现“垂直”划分提供了可能[38].

图 8 展示了两种通用 CPU 对应 PFN 的构成, 其中, 图 8(a)所示为 2.80 GHz 的 Intel i7-860 处理器搭载采用 16 路组相联的 LLC(8MB 容量)和 DDR3 主存(8G 容量)时系统的地址映射, 图 8(b)所示为 2.40 GHz 的 Intel Xeon 5600 处理器搭载采用 16 路组相联的 LLC(12MB 容量)和 DDR3 主存(32GB 容量)时系统地址映射.



(a) Intel i7-860 地址映射

(b) Intel Xeon-5600 地址映射

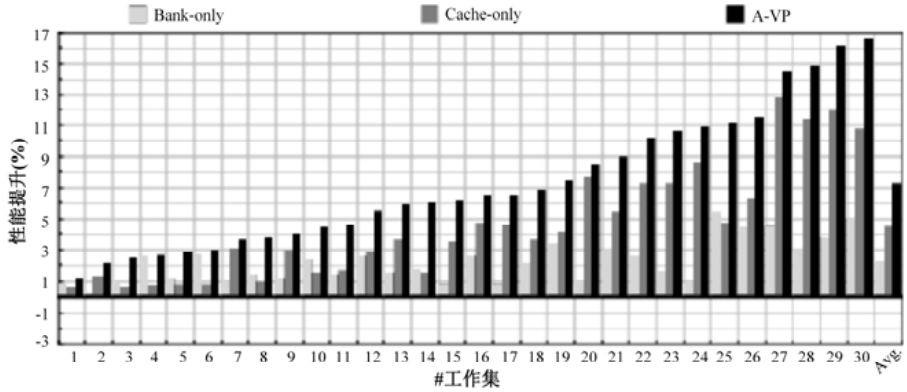
图 8　地址映射中索引不同资源的索引位

文献[2,5]中介绍了除了单独使用 O-bits 进行的“垂直”划分(a vertical partitioning, A-VP)外, 当工作集中的程序数大于 O-bits 能划分的最大颜色数时, 例如 2 位 O-bits 能划分 4 种颜色, 当工作集中的程序数大于 4 时, 还可以使用 O-bits 与 B-bits 或 C-bits 结合的方式, 这样可以实现偏向于 DRAM Bank 划分的“垂直”划分(B-VP)或偏向于 Cache 划分的“垂直”划分(C-VP). 表 2 展示了两种水平划分机制和 3 种垂直划分机制的综合情况.

表 2　两种水平划分机制和 3 种垂直划分机制的综合情况

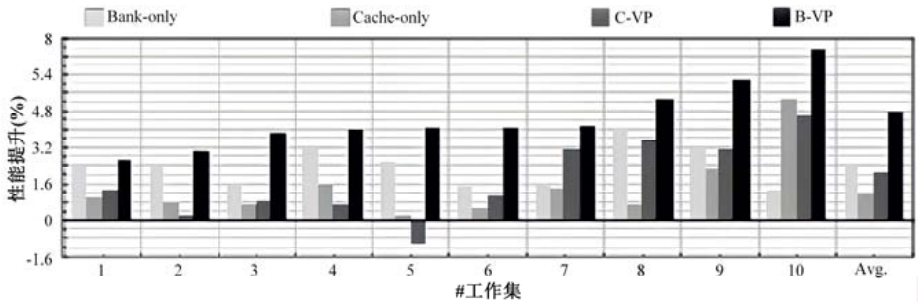
策略	着色位	描述	目标系统
Cache-only	C-bits {16–18}	LLC→8 groups	4/8 核
Bank-only	B-bits {21,22}	Banks→4 groups	4 核
Bank-only	B-bits {21,22} O-bits {15}	LLC→2 groups Banks→8 groups	8 核
A-VP	O-bits {14,15}	LLC→4 groups Banks→4 groups	4 核
B-VP	B-bits {22}+ O-bits {14,15}	LLC→4 groups Banks→8 groups	8 核
C-VP	C-bits {16}+ O-bits {14,15}	LLC→8 groups Banks→4 groups	8 核

合理选择划分策略可以最大化性能提升. 从图 7 所示第 1 象限的工作集中随机选择 50 个工作集, 分别对它们进行 Bank-only 划分、Cache-only 划分、A-VP 划分、B-VP 划分以及 C-VP 划分. 结果显示: “垂直”划分(A-VP)技术是普遍有效的, 形成了性能累加的效果, 最高的性能提升可达到 16.8%; 而单独 DRAM Bank 划分和单独 Cache 划分只有 11.7%和 5.9%的性能提升, 如图 9(a)所示. 对于图 9(b)所示的工作集, 采用 B-VP 获得的性能提升比 C-VP 和仅划分 Bank 或 Cache 获得的性能提升要多. 对于图 9(c)所示的工作集, 采用 C-VP 获得的性能提升比 B-VP 和仅划分 Bank 或 Cache 获得的性能提升要多. 综合来看, 由于累加了多级内存划分的效果, VP 的效果要优于单层次内存划分, 但同样也存在工作集敏感性的问题. 后文将进一步分析这一问题.

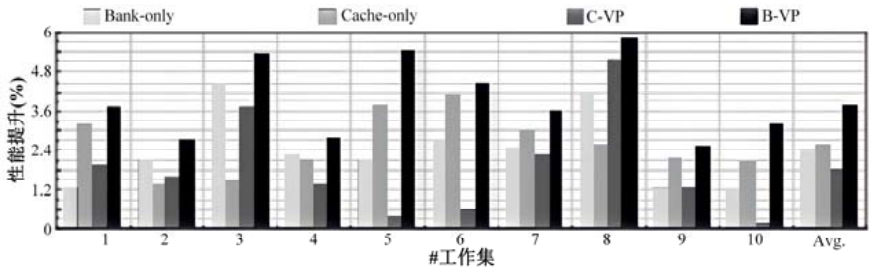


(a) A-VP 性能最佳工作集

图 9　A-/B-/C-VP 的性能提升[2]



(b) B-VP 性能最佳工作集



(c) C-VP 性能最佳工作集[2]

图 9 A-/B-/C-VP 的性能提升(续)[2]

3.2 “划分”和“融合”规则

从图 7 可以看出: 部分工作集是对 Cache 划分敏感的, 即 Cache 划分只对部分工作集有效, 而部分工作集采用 Cache 划分后性能反而有所下降. 因此, 在启用垂直划分时, 应考虑工作集对 Cache 资源的敏感性. 在前期的探索中, 研究者分别以动态在线或静态离线的采样方式获取单个程序运行参数(如带宽、局部性、Cache 缺失率等), 并建立性能模型, 最后利用机器学习方法预测任务级或访存级的调度. 这类方法通过预先建立性能模型, 可以有效降低调度开销, 但也存在着不可忽视的问题. (1) 建模的准确性和复杂度; (2) 采样的开销. 通常, 预测结果的准确度依赖于模型的复杂度以及采样的频率, 这也意味着将增加额外的系统开销[2]. 此外, 确定性能最佳的内存划分策略需要准确地预测工作负载中每个程序的访存特征及其对每个分配策略的反应, 然而, 由于每个程序的资源需求、访存特征差异较大, 一旦工作负载中的应用程序发生变化, 则对当前划分策略的影响就会较大.

基于上述原因, 为了减少系统总体开销, 并提高策略的适用性, Liu 等人提出了一种基于分类的 Cache 划分机制[2], 即: 依据程序对 Cache 资源的敏感性进行分类, 再对不同类的程序采用不同“划分”和“融合”策略. 首先, 依据应用程序对 Cache 资源的敏感度不同, 将程序分为 4 类. (1) 对 LLC(last level Cache)需求量高的 LLCH; (2) 对 LLC 需求量中等的 LLCM; (3) 对 Cache 资源不敏感且带宽需求小的 CCF(core Cache fitting); (4) 对 Cache 资源不敏感但有很高带宽需求的 LLCT(LLC thrashing, “刷 Cache”), 如图 10 所示. 在运行过程中, LLCT 类的程序在 Cache 上对其他程序干扰较大(造成 LLC 污染), 在 Cache 上将其与其他类程序划分开有利于提高系统的整体性能; LLCH 和 LLCM 类的程序对 Cache 的需求较大, 所以在 Cache 上不应该限制该类程序的使用(减少少量的 LLC 容量就会引发较大的性能下降). 在对大量真实工作集测试、分析、总结的基础上, 本文总结了 3 条“划分”规则.

- 规则 1. 包含 LLCT 类程序的工作集, 采用 A-VP/C-VP 划分能获得最大的性能提升;
- 规则 2. 不包含 LLCT 类程序、但包含 LLCH 类程序的工作集, 采用 Bank-only 划分能获得最大的性能提升;
- 规则 3. 不包含 LLCT 和 LLCH 类程序、但包含 LLCM 类程序的工作集, 采用 A-VP/B-VP 划分能获

得最大的性能提升.

以及两条“融合”规则:

- 规则 4. LLCH 和 LLCM 类程序可以共享划分的颜色;
- 规则 5. LLCT 和 CCF 类程序可以分别融合在一块较小的 Cache 资源里.

这是因为 LLCH 和 LLCM 类程序对于 Cache 资源的需求量较大, 而现有的最近最少使用算法(least recently used, LRU)能够较好地处理这两类程序产生的访存冲突, 并不会降低系统整体性能. 而 LLCT 和 CCF 类程序对于Cache的需求量较小, 采用完全划分的方式并不会提升系统整体性能, 反而会降低资源使用率, 给予它们最小的 Cache 资源就能够满足多道程序的需求.

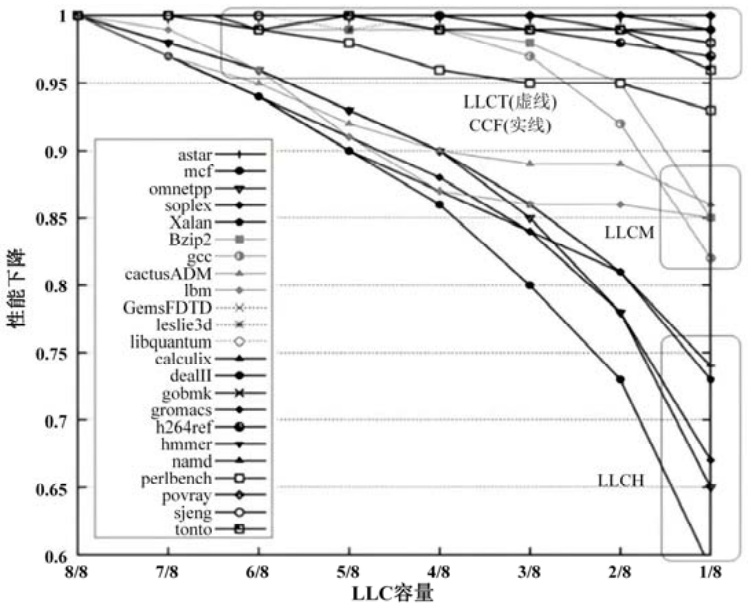


图 10 Cache 容量与性能变化的关系

3.3 非对称的“垂直”划分架构

页着色技术的核心是调整内存储器体系中的数据排布. 垂直划分技术由 Liu 等人在 ISCA-2014 上提出[2], 并在实践中不断演变. 随后, 一种“非对称”的“垂直”划分机制[5]被提出, 该机制应用得更为灵活、可靠, 数据排布方式更为简洁、高效, 在实际部署应用过程中开销更低. 以前文所述的 Intel i7-860 的地址映射为例, 图 11 展示了相关“非对称”的“垂直”划分情况.

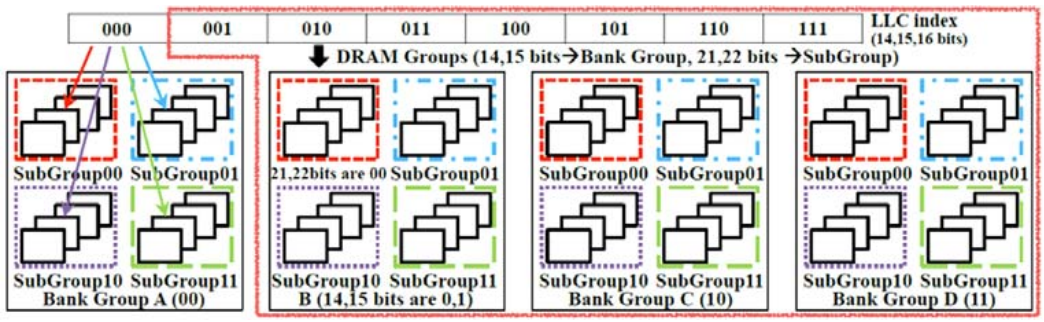


图 11 非对称的“垂直”划分

首先, 在 Cache 上划分一个较小的区域(占整个 Cache 的 1/8), 即图 11 中“000”的区域. 工作集中所有的

LLCT 类程序共享该区域, 剩余的 Cache 容量让 LLCH、LLCM 和 CCF 类的程序共享, 虽然该划分方法与第 4.2 节的规则相悖, 但研究发现, 现在多级 Cache 的设计使得 CCF 类程序的访存请求在一级缓存和二级缓存中能够得到响应, 所以将 CCF 类程序在 LLC 上的划分和 LLCH 及 LLCM 类程序的划分合并在一起并不会造成性能的明显下降. 此外, 研究表明[5]: 从服务器总体性能的角度来讲, 给予 CCF 类程序一块单独的 LLC 区域会造成 LLCH 和 LLCM 类程序的性能下降, 而这部分下降并不能通过 CCF 类程序的性能提升得到弥补.

其次, 在 DRAM Bank 上, 通过 O-bits(14 位、15 位)可以将 DRAM Bank 划分为 4 组, 再通过 B-bits(21 位、22 位)可以将这 4 组划分为 16 个子组. 这使得在 Cache 中共享的程序可以在 DRAM Bank 上进行划分, 从而消除多道程序在 DRAM Bank 上的访存冲突. 如图 11 所示: O-bits 为“00”的 Bank 组被划分给 LLCT 类程序, 多个 LLCT 类程序在该 Bank 组中可以通过 B-bits 再次划分, 消除彼此在 DRAM Bank 上的访存冲突; 同理, 也可将 O-bits 为“01”“10”“11”的 DRAM Bank 组划分给 LLCH、LLCM 和 CCF 类程序, 再通过 B-bits 进一步划分给各程序, 消除在 DRAM Bank 上的访存冲突. 此外, 在 LLC 上共享 000 区域的若干程序在 DRAM Bank 上依然可以被划分开, 虽然按照需求在 LLC 上没有划分, 但在主存 Bank 上的相互干扰依然可被消除. 具体内容请参考 Liu 的论文[5].

综合来看, 采用“非对称”的多层次协同的“垂直”划分更能发挥出计算机潜在的性能. 在实践中, 这种方式比之前的 VP 表现得更好(无论在页面迁移开销、系统易用度、灵活度, 还是在系统吞吐量、服务质量提升等方面均表现优良); 在最优的情况下, 超过经典的 Linux“伙伴系统”的性能 21%. 更多细节请参考文献[5].

4 基于页着色的 DRAM-NVM 混合内存管理

本节介绍的是页着色技术在新内存体系中的应用, 是对该技术在新时代的发展与演进. 目前, 以 NVM 为代表的新内存材质已经逐步商用. NVM 与 DRAM 相比, 具有大容量, 高存储密度、低能耗以及非易失性的优点. 但 NVM 写入延迟高、寿命与 DRAM 相比较低等缺点也限制了其单独作为内存系统的可行性. 为此, 研究者提出了 DRAM-NVM 混合内存系统. 针对这种新内存体系, 本文将介绍如何将页着色技术应用于对异构混合内存的管理, 并通过设计混合内存访存行为监控器、双层伙伴系统以及页面迁移引擎等机制, 形成一整套内存管理体系.

4.1 页着色技术在DRAM-NVM存储架构中的应用

目前, DRAM-NVM 混合内存体系存在两种不同的组织架构.

- 一种是“垂直”管理架构, 将 DRAM 与 NVM 置于不同内存层级, 速度较快的 DRAM 被作为 NVM 的缓存[10,23,24]. 在这种架构中, 需要由专用的硬件逻辑控制 NVM 和 DRAM 之间的数据移动;
- 另一种是 DRAM 和 NVM“水平”管理架构, 将 DRAM 与 NVM 置于内存层次结构中的同一层级, 且依赖 OS 等系统软件管理页面和迁移数据[23−25,40−44]. 相比于“垂直”管理架构, “水平”架构能够提供更大的物理内存空间. 此外, 设计合理的面向“水平”内存体系的内存管理机制, 可使得水平内存架构性能优于垂直架构. 在“水平”架构下, 内存管理机制的主要挑战是: 数据排布的软件控制、数据迁移以及最大化内存体系的利用率.

针对“水平”架构, 已有研究者[24]提出了面向 NVM 的管理策略, 该策略在运行时动态地监测每个内存页的写操作, 并将写密集型页从 NVM 迁移到 DRAM(NVM 的写操作速率较低且存在写寿命等问题). 有研究者[26,41,42,44]改进了页面迁移算法、管理策略等机制, 并依据页面的热度(即被访问频率)对页面分类, 减少了页面迁移的频次. 实际上, 现有针对 DRAM-NVM 混合内存管理架构的研究普遍存在着访存行为采样开销过高的问题. 此外, 这些研究更加注重主存架构, 忽视了主存与 Cache 间的相互影响的联动关系, 以至于使用这些策略得到的通常为次优解. 在上述 DRAM-NVM 混合内存管理研究的基础上, Liu 等人[9]尝试将页着色应用到 DRAM-NVM 混合内存管理. 总的思路是: 通过将事件采样与页表遍历结合起来, 以较低的监控采样频率精确地获取页面的热度; 利用 NVM/DRAM 所处的地址空间对页面着色(位于不同材质上的页面为不同颜色), 同时利用如前文所述的 Cache/Bank/Channel index bits 将页面与 Cache, Bank 和通道等绑定, 在能够控制不同

内存材质之间数据分布的同时, 也可利用页着色消除多道程序之间在内存体系上的相互干扰, 进而有效提高了包含 NVM 在内的整个内存体系的资源利用率.

以 Inteli7-860 平台为例, 如图 12 所示: 在 64 位系统中, 每页的大小为 4KB, 第 0 位至第 11 位是页面偏移, 第 32 位作为通道位被使用(每条通道仅连接一种内存材质, NVM 或 DRAM). 通过为第 32 位着色(0 或 1), 可以控制使用哪条内存通道, 即哪个内存段(DRAM 或 NVM)容纳该页. Cache Set 的颜色由 PFN 的第 15 位至第 18 位构成(这些位同样可以索引内存 Bank 中的一些行), 其大小是 LLC 中的一个 Slab(8MB LLC 总容量的 1/16). 所以, 可以利用这些位实现对 Cache 的划分及分配. 此外, 对于 NVM 和 DRAM 通道中的内存 Bank 资源, 可以利用 Bank 索引位(图 12 中的第 12 位至第 14 位和第 20 位、第 21 位)实现对 Bank 划分分配. 通常情况下, 第 20 位、第 21 位作为一个组合来指定一个 Bank Group 的颜色. 一个 Bank Group 由 8 个内存 Bank 组成, 内存管理系统可以通过使用多个 Bank Group 颜色来为某个程序分配多个 Bank Group. 因此, 借助页着色技术, 可以实现从 DRAM-NVM 到 Cache 的完整内存体系划分. 在通道划分方面, 可以根据页面的读/写特征选择 DRAM 或者 NVM 来映射页面, 以求最大限度地利用 DRAM 和 NVM 通道提供的总带宽. Liu 等研究者[9]将“热”页(被频繁访问的页面或者流式页面), 特别是那些热写页面, 分配至 DRAM 中. “冷”页, 以读操作为主的热页(NVM 的读速率和 DRAM 相仿), 则被保存在 NVM 中以节省能耗, 并为“热”页留出 DRAM 空间.

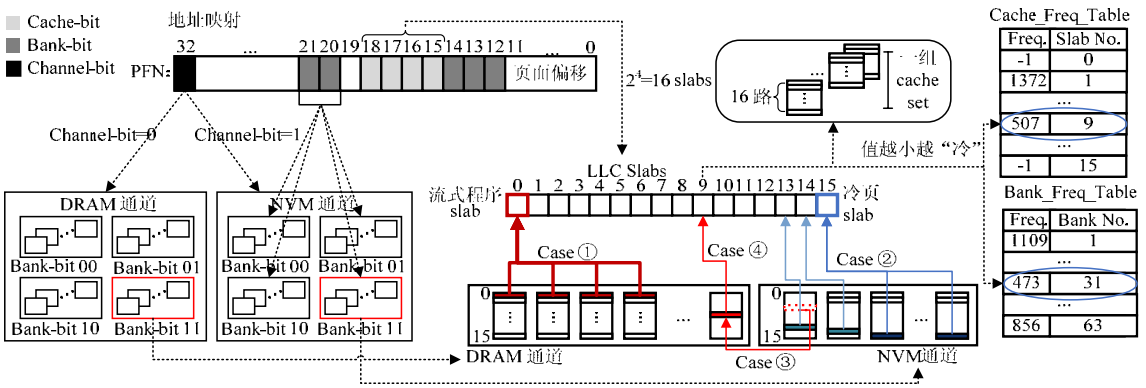


图 12 混合内存管理在整个内存体系上的应用架构

在带宽调度方面, 新型 DRAM-NVM 混合内存管理系统的设计目标是最大化 DRAM 和 NVM 的组合带宽(即最大化数据传输速率), 以此避免 NVM 带宽的明显下降(并且这部分下降不能通过增加 DRAM 带宽来弥补). 如后文图 14 所示: 在活动期, 内存管理系统将 10 000 个“热”页从迁移队列中迁移到 DRAM, 然后监视 DRAM 和 NVM 通道上产生的带宽. 当 DRAM 带宽增加幅度小于前一个周期时, 系统将在下一个周期减少迁移页面的数量(即默认为前一次迁移量的 1/5). 此外, 如果发现 NVM 带宽的减少幅度比 DRAM 带宽的增加幅度大, 那么它将在下一个周期停止将页面迁移到 DRAM; 移动更多的页面不会提高带宽利用率, 因为 DRAM 带宽接近饱和, 但这会进一步降低 NVM 带宽, 损害整体带宽. 最后, 内存管理系统会将一些页面迁移回 NVM, 以弥补 NVM 在带宽上的损失. 具体设计细节请参见文献[9,37].

在 Cache 与内存 Bank 的划分方面, 将 LLC Slab 分为 3 部分, 分别供流式程序、冷页面和热页面使用. 通过 Cache/Bank 访存频率记录表, 记录 Cache Slab 和内存 Bank 利用率; 将页面放置在低利用率的 Cache Slab 和内存 Bank 中, 以提升内存系统利用率, 减少“热”区域访存冲突. 若目标存储区域已被占用, 则选择其他利用率低的 Slab. 通过跟踪“冷”页面, 并将“冷”页面迁移至 NVM. 图 12 详细展示了整个内存管理体系, 设计细节参见文献[9,37].

4.2 面向DRAM-NVM混合内存管理系统的运行逻辑

DRAM-NVM 混合内存管理系统如图 13 所示, 由混合内存监控器(hybrid memory monitor, HyMM)、双层伙伴系统(two-tiered buddy system, TBS)以及数据迁移引擎(data migration engine, DME)这 3 个主要的模块构

成. 通过 HyMM 识别访问频率较高的读/写页面(“热”页)、流式页面以及访问频率较低的页面(“冷”页). 双层伙伴系统是对 Linux 经典的内存管理伙伴系统的扩展, 系统中的页面按照前文所述着色规则排布[9], 已着色的页面与通道、NVM/DRAM Bank 等绑定. “冷”页面因为被访问频率低, 会被暂时保存在 NVM 中; “热”页面会被移动到更快速的 DRAM 中. 最后, 利用数据迁移引擎控制“冷”/“热”页面在 DRAM 与 NVM 间的移动.

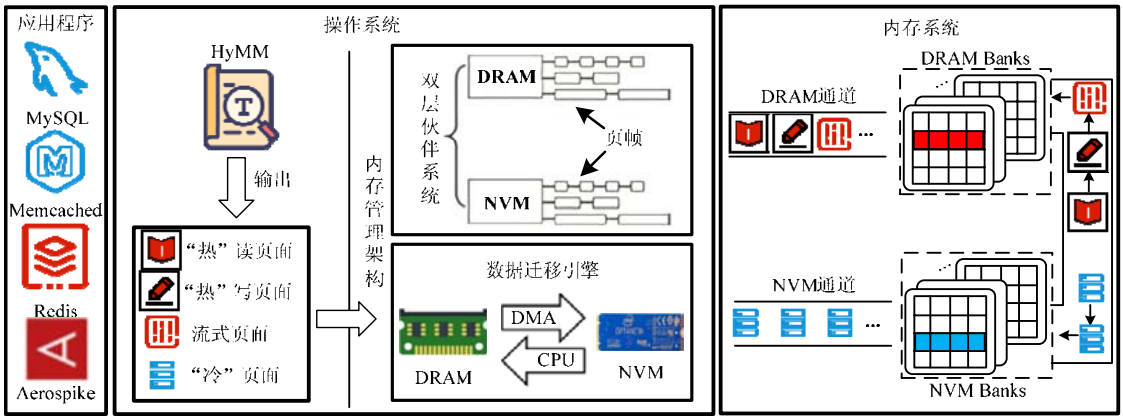


图 13 DRAM-NVM 内存管理的关键系统组件

在先前的工作[2,5,28]中, 研究者通过对页表项(PTE/access_bit/dirty_bit)采样的方法判断桌面级应用程序的内存访问模式(冷热页面所在区域以及读写模式). 但该方法在应对占用大量内存以及内存访问模式多变的工作负载则显得捉襟见肘, 例如: 利用 PTE 采样判断冷热页面区域需要不断地遍历 PTE 的 access_bit 位, 产生较大开销. 为了解决该弊端, HyMM 模块将事件采样(TLBMiss)和页表遍历结合起来, 可以有效提高识别精度并降低采样开销[9].

HyMM 的监控范围包括 TLB 的命中率/缺失率、PTE 的 dirty_bit 位以及 access_bit 位. 如图 14 所示的一个实例, HyMM 一次完整的检测由准备期与活动期构成. 在初始状态时, 所有程序的数据都预先存储在 NVM 中. HyMM 持续监控 TLB 命中情况 5s, 以确定“热”页面区域与“冷”页面区域. 随后, 通过监控 dirty_bit 位判断页面的读写模式. 上述两个步骤在准备期完成, 持续时间大约为 21s. 在准备期, HyMM 通过分析存储在 NVM 中的数据页, 收集 TLB 未命中数较大的页面(热页面), 并对这些数据页排序, 为接下来的迁移做准备. 之后是 20s 的活动期, 在该时期, 页面根据排序高低被动态迁移到 DRAM 中(热页面被移动到 DRAM 中). 迁移后, HyMM 会以较低的频率地扫描 PTE 的 access_bit 位, 以监控 DRAM 中的页面的冷热程度. HyMM 在运行逻辑中会不断循环. 更详细设计请参见文献[9].

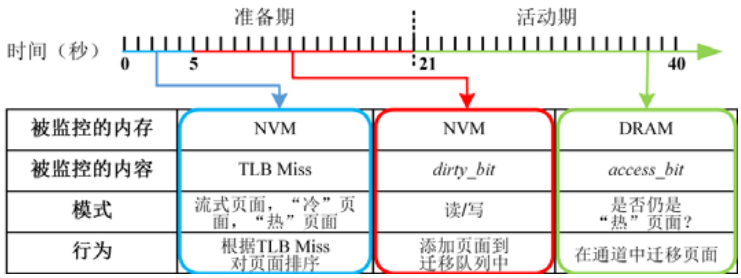


图 14 HyMM 调度逻辑示意图

双层伙伴系统(TBS)用于分配能够映射到内存体系中指定位置物理页面. 如前文所述, TBS 使用 PFN 中的通道位(channel bit), 将所有物理页面分配到两个子伙伴系统中. 这两个子伙伴系统分别对应管理 NVM 与 DRAM 中的页面. 两个子系统仍可以使用其他索引位分配资源. 实际上, PFN 中有 9 位可用于颜色控制(如图

12 所示), 即, 共有 512 种颜色用于多样化的内存资源分配.

在数据迁移引擎(DME)中, 应用两种不同的数据迁移方法: 当NVM的页面向DRAM迁移时, 使用有锁页面迁移[9], 目的是保证数据迁移的一致性(因为有大量的热页面将被迁移); 当 DRAM 的页面向 NVM 迁移时, 使用基于 DMA 的无锁迁移(因为加锁操作的开销较大, 而从 DRAM 中替换出的页面通常是冷页面)[9,45]. 有锁页面迁移是在OS内核原生页面复制方法的基础上实现的[46], 因此, 该迁移过程需要CPU参与. 当"冷"页面被移出 DRAM 时, 由于这种页面在迁移的过程中被修改的概率较小, 所以针对"冷"页面迁移, 可以使用基于 DMA 的无锁迁移, 如果发现页面在此过程中"脏"了, 则可回滚[9].

在系统验证过程中, 采用划分通道的方式[9,22,47]将内存地址空间划分为 DRAM 段和 NVM 段, 模拟 DRAM-NVM 内存架构. 实验平台是一台配置 Intel i7-860/2.8GHz CPU 和 DDR3 内存的服务器. 在工作集预热完成后, 研究者采用 PIN 工具[48]采集工作集的运行参数, 并将这些参数作为输入, 传入一个开源的 x86 多核混合内存模拟器[49,50]. 为了更好地模拟 DRAM 与 NVM 的存储架构, 使用 Dinero IV[51]和 DRAMsim2[52]分别优化了该模拟器对 Cache 的模拟和包含 NVM 配置的内存模拟. 此外, OS 运行时的开销(包括页表项更新开销和页面迁移开销等)以及使用 HyMM 的采用开销也被记录量化之后输入模拟器. 该模拟器也分别拥有 DRAM 和 NVM 内存控制器, 它们均采用 FR-FCFS 调度策略. 混合内存管理系统中, NVM, DRAM 及 Cache 的详细参数见表 3.

表 3 NVM、DRAM 和 Cache 的参数[9]

存储器	参数
1 级 Cache	指令 Cache: 32 KB; 数据 Cache: 32 KB; 缓存块(Cache block): 64 B
2 级 Cache	数据 Cache: 256 KB; 缓存块: 64 B
3 级 Cache	数据 Cache: 8 MB; 缓存块: 64 B
DRAM 系统	行寻址到列寻址延迟时间: 5 个时钟周期; 内存行地址控制器预充电时间: 5 个时钟周期; 写恢复延迟时间: 6 个时钟周期; 读能耗: 1.17 pJ/bit; 写能耗: 0.39 pJ/bit; 待机功率: 1 W/GB; 刷新功率: 0.032 W/GB; Bank: 512 MB
NVM 系统	行寻址到列寻址延迟时间: 22 个时钟周期; 内存行地址控制器预充电时间: 60 个时钟周期; 写恢复延迟时间: 6 个时钟周期; 耐擦写能力: 107; 读能耗: 2.47 pJ/bit; 写能耗: 16.82 pJ/bit; Bank: 4 GB

图15显示了在不同NVM容量配置下的5种应用(包括云计算应用)的执行时间. 在实验过程中, 使用4 GB容量的 DRAM 与不同容量的 NVM 组成异构混合内存系统, 验证基于"页着色"的混合内存管理的性能. 实验的基线(baseline)是仅使用 DRAM、不在内存架构中使用 NVM 的情况.

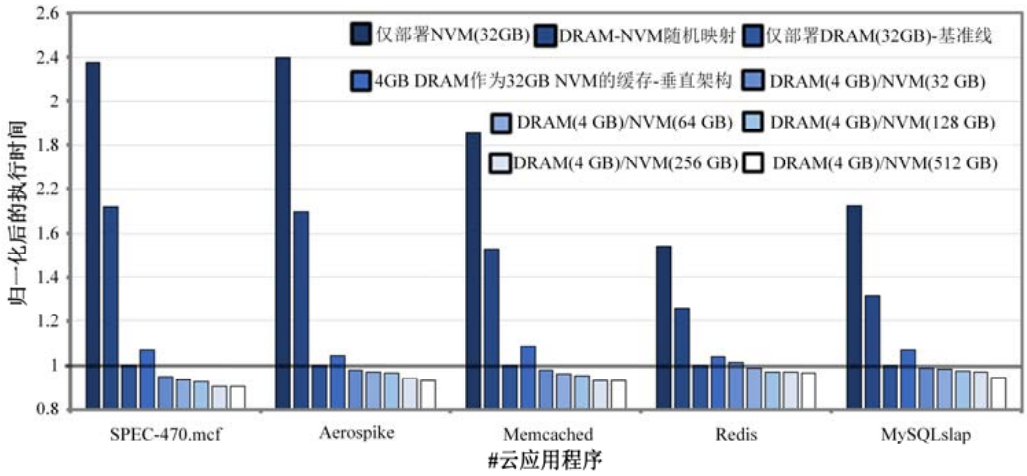


图 15 云计算应用负载的执行时间

从图 15 中可以看出: 仅配备 NVM 的系统由于 NVM 较高的写延迟, 性能较差. 在 DRAM-NVM 混合内

存系统中, 如果内存页随机映射到 DRAM 或 NVM 中, NVM 的高延迟也会导致整体性能下降. 4GB DRAM 作为 NVM 缓冲的“垂直”架构系统也被用作对比实验[25], 该系统的性能接近基准系统(系统仅使用 DRAM 的情况). 这是由于大多数热页面在运行时可以通过硬件逻辑迁移到 DRAM 中, 然而, 由于页面迁移以及硬件采样的开销, 该系统比基准系统的执行时间长 4%左右. “水平”架构内存管理系统的性能基本接近完全使用 DRAM 的情况, 这是一个好的现象, 证明了本文论述的技术的优势. 原因在于以下几点: (1) 平均 83.2%的“热”页在运行时迁移到 DRAM 中, 因此“水平”异构内存架构系统的总延迟与纯 DRAM 系统大致相同; (2) “水平”架构可以使用 NVM 通道向 CPU 传输数据, 与仅使用 DRAM 相比, 具有更高的总带宽, 并且随着 NVM Bank 数量增多, Bank 并行度越高, 能提供的带宽越大, 也就越有优势; (3) HyMM 和数据迁移引擎的开销较低. 这些性能指标的提升与内存体系中数据划分、排布优化有直接关系.

图 16 展示了将页着色技术应用在整个内存体系时产生的性能优势. 这里展示了 3 种情况下的对比: 第 1 种情况是仅划分缓存; 第 2 种情况是在缓存划分的基础上对内存 Bank 划分(也就是前文提到的 VP), 但并未将内存通道映射纳入优化范畴; 第 3 种情况是在缓存、内存 Bank 和内存通道这 3 个层面上的整体划分优化. 对比基线是未经修改的 Linux 原始内核. 第 3 种情况的平均性能提升比第 2 种情况高出 7.2%, 比基线高出 23%. 此外, 由于引入了对内存 Bank 的管理, 第 2 种情况的平均性能提升略高于第 1 种情况. 在第 3 种情况中, 通过指定特殊页面进入某条特定通道(例如将流式访问的页面限定访问某一条通道, 而不是交替访问所有通道), 能减少如流式程序等对其他程序的干扰. 这说明通过合理地使用页着色技术, 实现对内存资源的精细化管理, 可以有效提升系统的吞吐量. 此外, 工作集 1 至工作集 10 中的测试程序数量从 4 升至 8, 系统带来的性能提升仍保持稳定增长. 这说明在干扰越严重的工作集中, 应用“水平”架构内存管理系统能获得的性能收益越高. 更详尽的内容请参见文献[9].

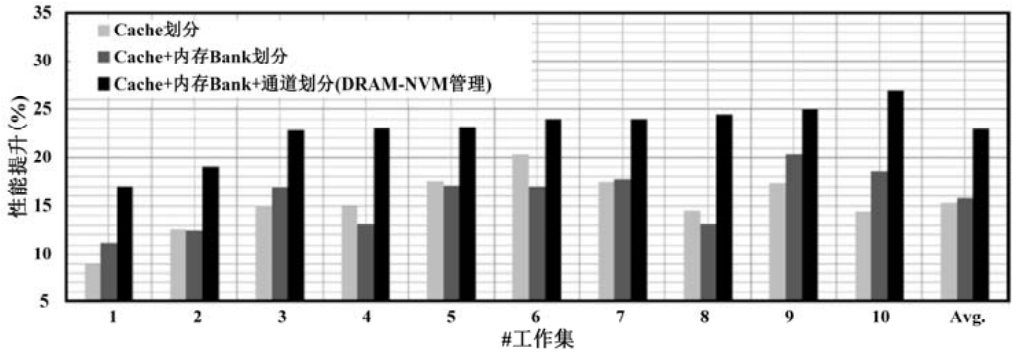


图 16 系统整体吞吐量的提升[9]

有参考文献[4,53]表明: 为了保证关键程序的服务质量, 可以采用“多核变单核”的简单处理方法, 即让高性能多核服务器只运行一道程序, 如果存在 N 个需要保证服务质量的程序, 那么需要 N 台服务器. 在这种情况下, 资源的有效利用率不到 10%. 这对于数据中心这样的大规模计算环境来说, 会造成极其严重的算力资源浪费, 无论与“低碳”环保的全球理念, 还是利益至上的商业目标, 都是严重冲突的. 而基于“页着色”的内存体系划分系列技术可以对数据中心的相关优化提供丰富的参考知识体系. 通过内存划分, 可以隔绝多道程序在数据中心的访存冲突, 从而提高程序的服务质量, 同时还可以提高服务器的资源利用率.

5 总 结

本文梳理了内存划分技术的发展, 介绍了有代表性的基于页着色技术内存体系划分机制. 经典的页着色技术仅是用在划分 Cache, 而本文涉及的代表工作创新的将页着色用在了整个内存体系(memory hierarchy), 拓展了内存体系优化、操作系统设计的维度, 系国内科研人员 Liu 首创的成果. 在多核时代, 多道程序在 DRAM 上的访存冲突制约了计算系统整体性能的提升. 为此, 页着色技术被应用于 DRAM 划分, 即从 OS 软

件层面出发, 为不同的应用程序分配不同的内存通道与 DRAM Bank, 以达到有效消除多道程序在 DRAM 中的访存冲突, 提高系统的整体性能的目的. 随后, “垂直”划分方法被提出. 其通过合理利用 PFN 上特殊的地址位(O-Bits), 实现了 Cache 与 DRAM 的协同划分. 最后, 本文介绍了如何将页着色技术应用于当前包含 NVM 的混合内存架构中, 并揭示了这项技术的潜力, 展示了内存划分技术的应用架构和前景.

随着技术的进步, Intel 等公司推出了一系列新的内存划分技术, 例如 CAT 技术等. 底层技术时刻在发生着变化, 内存划分的核心思想一直也在向前演进. 本文的作者相信: 通过阅读本文, 不但能对内存划分技术有较清晰的认识, 同时也能对整个内存体系架构有深刻的理解. 在未来的工作中, 如何将机器学习应用于内存资源划分, 是一个值得探讨的问题. 利用更加复杂的运行参数作为应用程序分类的依据, 构建分类模型, 实现更加多样化的程序分类. 并根据每类的特点, 有针对性地实现对内存资源的划分, 以进一步提高系统性能. 然而, 在这一过程中, 采集更加复杂的运行时参数势必增加额外的开销; 并且, 运行时参数与系统平台的配置密切相关, 会导致已构建的机器学习分类模型针对不同平台移植性较差. 如何解决这两方面问题, 将是未来研究的工作重点.

References:

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

[2] Liu L, *et al*. Going vertical in memory management: Handling multiplicity by multi-policy. In: Proc. of the 2014 Int’l Symp. on Computer Architecture. New York: IEEE, 2014. 169−180.

[3] Liu L, *et al*. A software memory partition approach for eliminating bank-level interference in multicore systems. In: Proc. of the 2012 Parallel Architectures and Compilation Techniques. New York: IEEE, 2012. 367−375.

[4] Lin J, Lu Q, Ding X, Zhang Z, Zhang X, Sadayappan P. Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems. In: Proc. of the 2008 Int’l Symp. on High-performance Computer Architecture. New York: IEEE, 2008. 367−378.

[5] Liu L, *et al*. Rethinking memory management in modern OS kernel for multicore systems: Horizontal, vertical, or random? IEEE Trans. on Computers, 2016, 65(6): 1921−1935.

[6] Cortez E, Bonde A, Muzio A, *et al*. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In: Proc. of the 2017 Symp. on Operating Systems Principles. New York: ACM, 2017. 153−167.

[7] Knowlton KC. A fast storage allocator. Communications of the ACM, 1965, 8(10): 623−624.

[8] Tanenbaum AS, Bos H. Modern Operating Systems. 4th ed., New Jersey: Pearson Education, Inc., 2015. 758−764.

[9] Liu L, *et al*. Hierarchical hybrid memory management in OS for tiered memory systems. IEEE Trans. on Parallel and Distributed Systems, 2019, 30(10): 2223−2236.

[10] Mutlu O. Main memory scaling: Challenges and solution directions. In: More than Moore Technologies for Next Generation Computer Design. Cham: Springer, 2015. 127−153.

[11] Kessler RE, Hill MD. Page placement algorithms for large real-indexed Caches. ACM Trans. on Computer Systems, 1992, 10(4): 338−359.

[12] Qureshi MK, Patt YN. Utility-based Cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In: Proc. of the 2006 Int’l Symp. on Microarchitecture. Los Alamitos: IEEE Computer Society, 2006. 423−432.

[13] Kim S, Chandra D, Solihin Y. Fair cache sharing and partitioning in a chip multiprocessor architecture. In: Proc. of the 2004 Int’l Conf. of Parallel Architectures and Compilation Techniques. Los Alamitos: IEEE Computer Society, 2004. 111−122.

[14] Nesbit KJ, Laudon J, Smith JE. Virtual private Caches. In: Proc. of the 2007 Int’l Symp. on Computer Architecture. New York: ACM, 2007. 57−68.

[15] Kim Y, Papamichael M, Mutlu O, Harchol-Balter M. Thread cluster memory scheduling: Exploiting differences in memory access behavior. In: Proc. of the 2010 Int’l Symp. on Microarchitecture. Los Alamitos: IEEE Computer Society, 2010. 65−76.

[16] Sherwood T, Calder B, Emer J. Reducing cache misses using hardware and software page placement. In: Proc. of the 1999 Int’l Conf. on Supercomputing. New York: ACM, 1999. 155−164.

[17] Mutlu O, Moscibroda T. Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems. In: Proc. of the 2008 Int'l Symp. on Computer Architecture. Piscataway: IEEE, 2008. 63−74.

[18] Moscibroda T, Mutlu O. Memory performance attacks: Denial of memory service in multi-core systems. In: Proc. of the 2007 USENIX Security Symp. Berkeley: USENIX, 2007. 257−274.

[19] Mutlu O, Moscibroda T. Stall-time fair memory access scheduling for chip multiprocessors. In: Proc. of the 2007 Int'l Symp. on Microarchitecture. Los Alamitos: IEEE Computer Society, 2007. 146−160.

[20] Muralidhara SP, Subramanian L, Mutlu O, Kandemir M, Moscibroda T. Reducing memory interference in multicore systems via application-aware memory channel partitioning. In. Proc. of the 2011 Int'l Symp. on Microarchitecture. Los Alamitos: IEEE Computer Society, 2011. 374−385.

[21] Jeong MK, Yoon DH, Sunwoo D, Sullivan M, Lee I, Erez M. Balancing DRAM locality and parallelism in shared memory CMP systems. In: Proc. of Int'l Symp. on High-performance Computer Architecture. New York: IEEE, 2012. 1−12.

[22] Liu L, *et al*. BPM/BPM+: Software-based dynamic memory partitioning mechanisms for mitigating DRAM Bank-/channel-level interferences in multicore systems. ACM Trans. on Architecture and Code Optimization, 2014, 11(1): 1−28.

[23] Marinella M. The future of memory. In: Proc. of the 2013 IEEE Aerospace Conf. New York: IEEE, 2013. 1−11.

[24] Dhiman G, Ayoub R, Rosing T. PDRAM: A hybrid PRAM and DRAM main memory system. In: Proc. of the 2009 ACM/IEEE Design Automation Conf. New York: IEEE, 2009. 664−669.

[25] Qureshi MK, Srinivasan V, Rivers JA. Scalable high performance main memory system using phase-change memory technology. In: Proc. of the 2009 Int'l Symp. on Computer Architecture. New York: ACM, 2009. 24−33.

[26] Zhang WY, Li T. Exploring phase change memory and 3D die-stacking for power/thermal friendly, fast, and durable memory architectures. In: Proc. of the 2009 Int'l Conf. on Parallel Architectures and Compilation Techniques. Los Alamitos: IEEE Computer Society, 2009. 101−112.

[27] Lee S, Bahn H, Noh SH. CLOCK-DWF: A write-history-aware page replacement algorithm for hybrid pcm and dram memory architectures. IEEE Trans. on Computers, 2014, 63(9): 2187−2200.

[28] Zhang X, Dwarkadas S, Shen K. Towards practical page coloring-based multi-core cache management. In: Proc. of the 2009 ACM European Conf. on Computer Systems. New York: ACM, 2009. 89−102.

[29] Hennessy JL, Patterson DA. Computer Architecture: A Quantitative Approach. 5th ed., Amsterdam: Elsevier Science, Inc., 2011. 129−131.

[30] Taylor G, Peter D, Farmwald M. The TLB slice—A low-cost high-speed address translation mechanism. In: Proc. of the 1990 Int'l Symp. on Computer Architecture. Los Alamitos: IEEE Computer Society, 1990. 355−363.

[31] Chen JB. Memory behavior of an X11 window system. In: Proc. of the USENIX Winter 1994 Technical Conf. Berkeley: USENIX, 1994. 189−200.

[32] Mogul JC, Borg A. The effect of context switches on cache performance. ACM SIGPLAN Notices (ACM Special Interest Group on Programming Languages), 1991, 26(4): 75−84.

[33] Romer TH, Lee D, Bershad BN, Chen JB. Dynamic page mapping policies for cache conflict resolution on standard hardware. In: Proc. of the 1994 Symp. on Operating Systems Design and Implementation. Berkeley: USENIX, 1994. 255−266.

[34] Bugnion E, Anderson JM, Mowry TC, Rosenblum M, Lam MS. Compiler-directed page coloring for multiprocessors. SIGPLAN Notices (ACM Special Interest Group on Programming Languages), 1996, 31(9): 244−255.

[35] Subramanian L, Lee D, Seshadri V, Rastogi H, Mutlu O. The Blacklisting Memory Scheduler: Achieving high performance and fairness at low cost. In: Proc. of the 2014 Int'l Conf. on Computer Design. Seoul: IEEE, 2014. 8−15.

[36] Zhang LD, Wang R, Liu Y, Qian DP. Dynamic partitioning for shared cache of multicore processors with page coloring. Chinese Journal of Computers, 2014, 37(7): 1478−1486 (in Chinese with English abstract).

[37] Liu L, *et al*. Memos: A full hierarchy hybrid memory management framework. In: Proc. of the 2016 Int'l Conf. on Computer Design. New York: IEEE, 2016. 368−371.

[38] Mi W, Feng XB, Xue JL, Jia YC. Software-hardware cooperative DRAM bank partitioning for chip multiprocessors. In: Proc. of the 2010 IFIP Int'l Conf. on Network and Parallel Computing. Berlin: Springer-Verlag, 2010. 329−343.

[39] Standard Performance Evaluation Corporation. 2006. http://www.spec.org/cpu2006

[40] Liu HK, Chen YJ, L XF, Jin H, He BS, Zheng L. Hardware/Software cooperative caching for hybrid DRAM/NVM memory architectures. In: Proc. of the 2017 Int'l Conf. on Supercomputing. New York: ACM, 2017. 1–10.

[41] Ryoo JH, John LK, Basu A. A case for granularity aware page migration. In: Proc. of the 2018 Int'l Conf. on Supercomputing. New York: ACM, 2018. 352–362.

[42] Ramos LE, Gorbatov E, Bianchini R. Page placement in hybrid memory systems. In: Proc. of the 2011 Int'l Conf. on Supercomputing. New York: ACM, 2011. 85–95.

[43] Chen J, Liu HK, Wang XY, Zhang Y, Liao XF, Jin H. Largepage supported hierarchical DRAM/NVM hybrid memory systems. Journal of Computer Research and Development, 2018, 55(9): 2050–2065 (in Chinese with English abstract).

[44] Li G, Chen L, Hao XR. Linux memory management algorithm based on hybrid memory architecture PDRAM. Microelectronics & Computer, 2014, 31(5): 14–20 (in Chinese with English abstract).

[45] Herrell RW, Morrissey TP. User scheduled direct memory access using virtual addresses. Int Cl: G06F15/00U.S.Patent 5301287, 1994.

[46] Bovet DP, Cesati M. Understanding the Linux Kernel. 3rd ed., New York: O'Reilly Media, Inc., 2005. 294–350.

[47] Muralidhara SP, Subramanian L, Mutlu O, Kandemir M, Moscibroda T. Reducing memory interference in multicore systems via application-aware memory channel partitioning. In: Proc. of the 2011 Int'l Symp. on Microarchitecture. New York: ACM, 2011. 374–385.

[48] Pin 2.14. https://software.intel.com/sites/landingpage/pintool/docs/71313/Pin/html

[49] Utility-based hybrid memory management simulator. 2017. https://github.com/CMU-SAFARI/UHMEM

[50] Li Y, Ghose S, Choi J, Sun J, Wang J, Mutlu O. Utility-based hybrid memory management. In: Proc. of the 2017 Int'l Conf. on Cluster Computing. New York: IEEE, 2017. 152–165.

[51] DineroIV trace-driven uniprocessor cache simulator. http://pages.cs.wisc.edu/~markhill/DineroIV/

[52] DRAMSim2. http://www.eng.umd.edu/~blj/dramsim/

[53] Petersen W. 2010. http://web.cse.ohio-state.edu/~zhang.574/OS-cache-software_intel_2010.pdf

附中文参考文献:

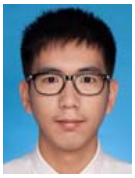
[36] 张栌丹, 王锐, 刘轶, 钱德沛. 基于页着色的多核处理器共享 Cache 动态分区. 计算机学报, 2014, 37(7): 1478–1486.

[43] 陈吉, 刘海坤, 王孝远, 张宇, 廖小飞, 金海. 一种支持大页的层次化 DRAM/NVM 混合内存系统. 计算机研究与发展, 2018, 55(9): 2050–2065.

[44] 李功, 陈岚, 郝晓冉. 基于 PDRAM 混合内存架构的 Linux 内存管理算法. 微电子学与计算机, 2014, 31(5): 14–20.

邱杰凡(1984－), 男, 博士, 副教授, CCF 专业会员, 主要研究领域为物联网, 计算机系统结构, 嵌入式系统.

范菁(1969－), 女, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为服务计算, 人工智能.



华宗汉(1995－), 男, 学士, CCF 会员, 主要研究领域为计算机系统结构, 嵌入式系统.

刘磊(1981－), 男, 博士, 教授, 博士生导师, CCF 专业会员, 主要研究领域为计算机系统结构.