中国科学: 信息科学 2019 年 第 49 卷 第 9 期: 1138–1158
SCIENTIA SINICA Informationis
论文

基于双向哈希链表的异构内存页迁移机制

裴颂文[1,2,4*], 姬燕飞[1], 沈天马[1], 刘海坤[3]

1. 上海理工大学光电信息与计算机工程学院, 上海 200093
2. 复旦大学管理学院, 上海 200433
3. 华中科技大学计算机学院, 武汉 430074
4. 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
* 通信作者. E-mail: swpei@usst.edu.cn

收稿日期: 2018–09–08; 接受日期: 2018–11–17; 网络出版日期: 2019–09–03

上海市浦江人才 (批准号: 16PJ1407600)、中国博士后科学基金 (批准号: 2017M610230)、国家自然科学基金重点项目 (批准号: 61332009)、国家自然科学基金面上项目 (批准号: 61775139)、上海市自然科学基金 (批准号: 15ZR1428600) 和计算机体系结构国家重点实验室开放课题 (批准号: CARCH201807) 资助项目

摘要 随着大数据技术的快速发展, 大规模访问存储器的需求随之剧增, 导致访问动态随机访问存储器 DRAM 的高耗能问题越来越突出. 大容量、低能耗的非易失性内存 NVM 技术逐渐成熟, 有望被广泛应用于异构内存计算机系统. 基于访问内存页的历史记录, 本文针对异构内存系统提出了一种双向哈希链表的异构内存页迁移机制 (THMigrator), 将频繁访问的内存页从 PCM 或 STT-RAM 迁移到 DRAM, 并用能效分析模型 (EEAM) 评估了异构内存系统的能效. 实验结果表明, THMigrator 迁移机制比采用多级队列迁移机制 MQMigrator 的系统计算性能提升了 9.3%, 系统平均能效比提升了 17%; THMigrator 比采用随机迁移机制 CoinMigrator 的系统平均能效比提升了 26%.

关键词 页迁移, 双向哈希链表, 异构系统, 非易失性内存, 迁移方法

1 引言

随着大数据、云计算、机器学习和人工智能等高新技术的飞速发展, 需要实时处理的数据呈指数级增长. 然而, 由于 DRAM 突出的动态功耗和刷新功耗, 导致访问 DRAM 内存系统的功耗占整个计算机系统运行功耗的 30%~40% [1,2]. 大数据时代下访问内存系统的高能耗问题日益突出, 如何降低能耗是提高整机系统性能的关键因素.

非易失性存储器 NVM 所存储的信息在电源掉电后依然能长时间保存, 是当前以 DRAM 为主流的易失性存储器的有效补充. 近年来, 国内外主要研究机构对 DRAM 和非易失性存储单元构建的异构内存系统取得了大量的研究成果. IBM 研究院采用 PCM 在每个存储单元上实现存储 3 字节容量

引用格式: 裴颂文, 姬燕飞, 沈天马, 等. 基于双向哈希链表的异构内存页迁移机制. 中国科学: 信息科学, 2019, 49: 1138–1158, doi: 10.1360/N112018-00246
Pei S W, Ji Y F, Shen T M, et al. Migration mechanism of heterogeneous memory pages using a two-way Hash chain list (in Chinese). Sci Sin Inform, 2019, 49: 1138–1158, doi: 10.1360/N112018-00246

表 1 DRAM, PCM 和 STT-RAM 性能对比

Table 1 DRAM, PCM and STT-RAM performance comparison

Storage/memory	Reciprocal density	Read speed	Write speed	Read power/mW	Write power/mW	Endurance
DRAM	$4-6F^2$	Slow	Slow	Medium	Medium	10^{16}
PCM	$4-12F^2$	Slow	VerySlow	Medium	High	10^8-10^9
STT-RAM	$6-50F^2$	Fast	Slow	Low	High	4×10^{12}

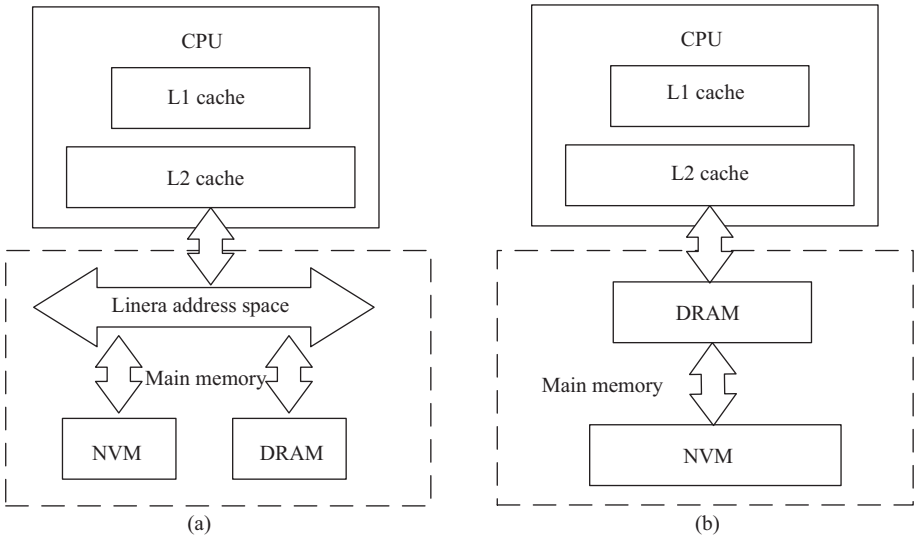


图 1 异构内存架构

Figure 1 Structure of heterogeneous memory system. (a) Flat memory architecture; (b) hierarchical memory architecture

的数据, 有望应用到物联网等电子产品. Yole Développement 最新报告中指出新兴非易失性存储器将越来越多用于多种领域: 工业和运输、企业存储、智能卡、手机和大容量存储. 全球新兴的非易失性存储器市场将从 2012 年的 2.09 亿美元增长至 2018 年的 20 亿美元, 相当于每年增长 46%. 表 1 从工艺尺寸、能耗和读写性能等方面比较了 DRAM, PCM 和 STT-RAM 的特性 [3~6]. 根据表 1 可知, NVM 相比 DRAM 具有非易失性、断电仍可保存数据、工艺制程低、功耗低等优点. 随着非易失性存储器 NVM 价格的不断走低, 加上自身优良的特性, DRAM 和 NVM 组成的异构内存系统成为解决降低内存能耗的首选方案 [4,5]. PCM 读操作延迟是 DRAM 读操作延时的 2 倍以上, 其写操作延迟达到 DRAM 写操作延时的 40 倍 [1,6]. 但是, PCM 具有几乎可以忽略的静态能耗, 这使得将 PCM 应用到内存系统上具有节约内存能耗的巨大潜力. 因此, 在传统的 DRAM 内存系统中引入 NVM 构成异构内存系统, 一方面可以充分利用 NVM 能耗低、容量大等特点, 另一方面又可以有效利用 DRAM 读写操作延时短速度快的优势, 从而构建新一代低能耗、大容量的异构内存系统 [7].

一般地, DRAM 和 NVM 组成的异构内存系统有两种组织形式: 线性统一编址的内存结构 (水平结构) 和以 DRAM 作为缓存功能的内存结构 (垂直结构). 如图 1(a) 所示, 在线性统一编址的异构内存架构中, DRAM 和 NVM 处于同等地位, 无主次之分, 而图 1(b) 中, 在以 DRAM 为缓存的异构内存架构中, DRAM 作为 NVM 的缓存, 以 NVM 为主存储器, 从而结合 DRAM 访问速度快, NVM 容量大的特性.

本文的研究工作是针对第 2 种异构内存系统架构, 将 DRAM 作为 NVM 主存储器的缓存模块.

由于 PCM 写延时长的问题[8,9], 导致系统能耗增加, 通过降低写 PCM 或 STT-RAM 的次数来减少异构内存系统的综合访问延时, 从而降低系统总体能耗是本文的主要目标. 本文的主要贡献如下:

(1) 提出了一种基于双向哈希链表的异构内存页迁移机制 (THMigrator). 采用双向哈希链表跟踪内存页历史访问轨迹, 利用哈希特性实现数据操作复杂度最低.

(2) THMigrator 根据历史访存信息, 将访问计数超过阈值的内存页从 NVM 迁移到 DRAM, 从而减少对 PCM 的写操作, 降低系统总延时.

(3) 建立了能效分析模型 (energy efficiency analysis model, EEAM). EEAM 模型综合考虑了执行时间和能效因素, 并对内存页迁移的能效比进行了量化分析.

2 相关工作

以 PCM, STT-RAM 为代表的新型存储器在如何减少 PCM 写操作次数, NVM 和 DRAM 组成异构内存, 均衡 PCM 磨损, 用 B+ 树优化异构数据库等方面取得了大量成果. NVM 可以解决现代移动设备、大数据计算、物联网等领域访问存储器的高能耗、长延迟问题. 但是, NVM 写延时高于 DRAM, 又会增加能耗. 对于异构内存系统, 为了减少 NVM 写操作的次数, 可将 NVM 中频繁访问的内存页迁移到 DRAM[10,11], 以提高对内存系统的访问效率. 基于异构内存系统的内存页迁移机制研究非常重要.

软硬件协同迁移机制的研究中, Dhiman 等[12] 设计基于内存控制器的硬件单元, 进行中断操作, 并将访问信息管理为不同的 PCM 内存页, 操作系统内存管理模块负责将 PCM 中频繁访问的内存页迁移到 DRAM. Yoon 等[13] 提出了基于行缓冲区的局部感知缓存机制, 该机制跟踪 PCM 中最近使用的行缓冲未命中计数器, 并用 DRAM 预测缓存即将被频繁访问但行缓冲器未命中的行. 该机制采用了预测机制缓存了潜在的未命中行, 但忽视了行缓冲区的局部性问题. Liu 等[14] 使用 TLB 结合动态阈值调整技术提出了软硬件协同的页迁移机制.

基于冷热页面迁移机制的研究中, Ramos 等[15] 提出用多级队列结构存储内存页信息. 当读写操作计数器达到预定阈值时, NVM 内存页被迁移到 DRAM. Park 等[16] 假定连续两次内存访问会使 NVM 内存页变热, 而连续两次没有被访问则会使 DRAM 内存页变冷. 根据内存页的冷热度, 迁移热页面到 DRAM. Seok 等[17] 提出了用 4 个 LRU 队列保存内存页的读写计数值, 按照内存页在队列中的顺序将频繁访问的页面从 NVM 迁移到 DRAM[18].

动态自适应迁移机制的研究中, Mai 等[19] 提出了动态内存页迁移机制, 该机制不仅可以预测内存页的访问次数, 并且全面评估了多种迁移机制的成本与收益之间的关系. 选择收益最大的候选内存页进行迁移, 但是该机制没有考虑访存的能耗. Sungho 等[20] 提出基于自适应分类 (adaptive-classification CLOCK) 算法的内存页迁移机制. 通过跟踪应用程序的读/写访问路径确定读/写密集型模式, 减少不必要的或迁移效益低的迁移操作. DualStack[21] 根据历史访存信息和内存访问的局部性原理来动态管理内存页, 将对 DRAM 内存页进行密集型的写操作, 对 PCM 内存页进行密集型的读操作, 通过这种方式来降低内存页在混合存储模块之间的迁移次数. Wu 等[22] 提出的具有动态迁移和降级功能的 Rank-Aware DRAM 能效管理方法, 对异构内存能效研究亦有借鉴意义. Pei 等[23,24] 建立一种异构并行多核系统的能效模型和动态任务调度算法, 降低工作负载的执行总时间, 提高异构系统加速比. 并通过减少数据准备阶段的访存和通信开销改善异构系统的综合性能.

从软硬件协同的迁移机制、基于冷热页面的迁移机制到动态自适应的迁移机制, 迁移机制不断优化升级, 同时也会带来一些新的问题. 以上页迁移机制虽然在一定程度上提高访问内存的系统性能, 但

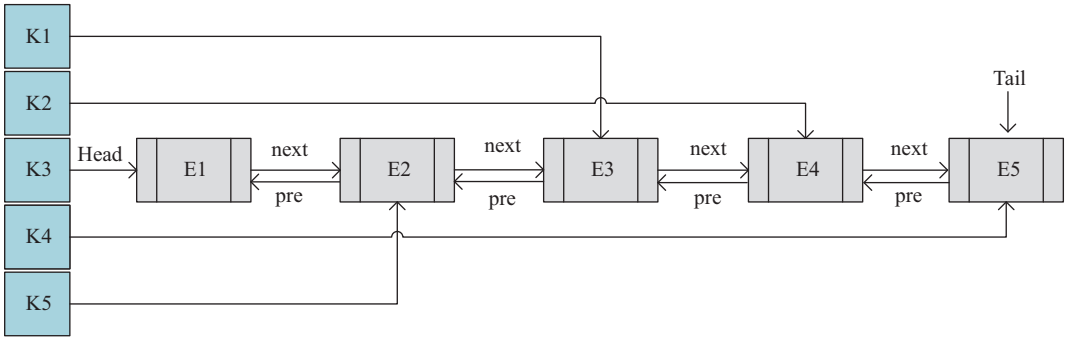


图 2 (网络版彩图) 双向哈希链表结构

Figure 2 (Color online) Structure of two-way Hash chain list

是存在以下缺陷. (1) 不同迁移机制缺乏时间复杂度的分析, 使用复杂的结构或算法可能引起时间成本的开销. (2) 缺乏页迁移的成本开销和系统效益的评估方法, 这些机制采用的静态阈值无法适用于不同的工作负载, 因此页迁移的成本可能大于收益. (3) 以启发式搜索算法为核心的动态调整阈值算法可能由于其自身的局部性, 计算出局部最优值, 最终陷入局部性最优 “陷阱”.

基于对以上异构内存系统 [25~28] 的迁移机制的访存效率和能效分析研究, 考虑到多级队列 [26] 迁移机制 (MQMigrator) 使用的多队列结构存在查找复杂度高和存储代价大、时间开销大等问题, 本文对垂直型异构内存系统提出了一种基于双向哈希链表的内存页迁移机制 (THMigrator). 利用双向哈希链表降低内存页查找复杂度和提高内存页查找的速度, 并降低了页表的存储开销. 本文还建立了 EEAM 模型, 量化分析了访存能效比.

3 双向哈希链表迁移机制

THMigrator 迁移机制通过构造双向哈希链表记录内存页的访问踪迹, 并根据访存踪迹计算内存页的访问计数值, 将计数值超过阈值的内存页从 NVM 迁移到 DRAM.

3.1 双向哈希链表结构

我们研究了一个重要而通用的数据结构, 即双向哈希链表 (two-way Hash chain list), 将双向哈希链表结构用于异构内存系统的内存页迁移机制的设计中, 提高系统的性能, 降低系统的整体能耗. 哈希表无处不在并广泛用于主存和缓存中构造索引和查找表. 在传统存储器技术 (即 DRAM 和 SRAM) 上设计哈希表主要考虑两个性能参数, 包括空间利用和请求延迟 [18]. 在 NVM 上设计哈希表时, 由于 NVM 的固有不对称特性、写入耐久性 [27] 和写延时过长, 还应该考虑第 3 个重要参数 [28], 即 NVM 写入的数量. 我们使用了一种经济高效的哈希表, 双向哈希链表, 因为该链表在进行插入、替换和查找数据时的时间复杂度为 $O(1)$. 哈希表是普通数组概念的推广, 对于普通数组, 可以直接寻址, 通过数字下标直接访问到值, 而无需遍历数组取值, 所以哈希表可以在 $O(1)$ 的时间内访问任意位置. 因此可达到查找时间复杂度为 $O(1)$, 时间复杂度远低于多级队列结构. 采用基于双向哈希链表的结构进行内存页迁移机制的优化, 复杂度和存储代价远低于多级队列结构, 双向哈希链表的结构如图 2 所示, 双向哈希链表是有 K (K1~K5) 和 E (E1~E5) 组成的键值对, 其中, E1~E5 表示内存页结点, 保存了内存页的历史访存信息, 内存页结点通过前向指针和后向指针连接其他内存页结点, K1~K5 为双向哈希链表的键 (K), 每个 K 的值为内存页编号, 键值对通过内存页编号进行关联, 可实现内存页的快速查找.

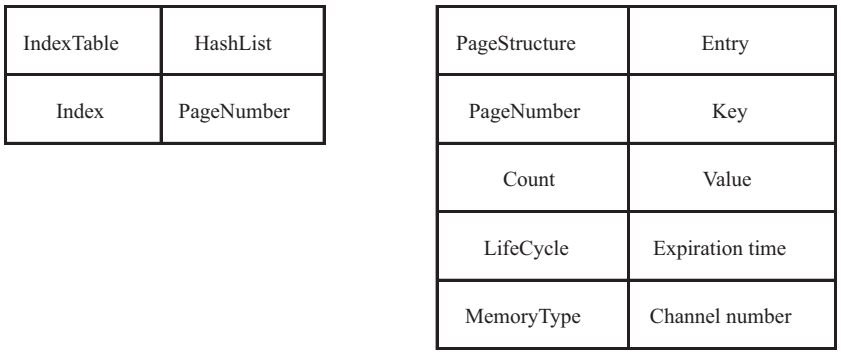


图 3　双向哈希链表索引及条目结构

Figure 3　Structure of HashList and entry table

每个结点 Entry 记录内存页的访问次数 value, 初始值为 0, 当该内存页再次被访问时, value 加 1, 内存页的生存周期为 expiration time, 使用生存周期 (时间) 作为度量来减少空间的需求, 超过生存周期的内存页要从双向哈希链表中删除, expiration time 的计算方法是当请求访问内存页时, 内存页的生存周期等于当前系统时间加上内存页的预定生命值 (life_time), 该内存页迁移之前检查当前的系统时间是否大于 expiration time, 若大于, 则从双向哈希链表中删除该内存页. 内存页页编号为 key, 以及内存页所属内存类型为 channel number. channel number 指定该内存页的位置是在 DRAM 或 NVM 存储器中. 每个结点 Entry 都有前项结点 pre Entry 和后项结点 next Entry, 通过页编号索引内存页结点的哈希实现方式, 可实现以数组下标访问值的形式快速访问到内存页信息. 双向哈希链表索引及条目具体结构如图 3 所示.

3.2　THMigrator 迁移机制

根据双向哈希链表中记录的内存页历史访存信息, 统计内存页的历史访问计数值, 若某个内存页的访问计数值超过给定的阈值且在 NVM 中, 则将该内存页迁移到 DRAM. THMigrator 迁移机制的基本思想如下:

步骤 1. 根据内存页的历史访存信息构造 HashList 链表. 若当前请求访问的内存页 page_i 不在 HashList 里时, 将 page_i 插入 HashList 头部. 否则, 将 page_i 的访问计数值 value 加 1. 双向哈希链表 HashList 始终保证将最近访问过的内存页放置于链表头部.

步骤 2. 判断 page_i 的 value 是否超过给定的阈值, 若 value 超过阈值, 则将 page_i 迁移到候选迁移表 MigrationMap 中. HashList 和 MigrationMap 通过内存页的页编号进行关联.

步骤 3. 候选迁移表 MigrationMap 中, 没有超过生存周期的内存页按照先进先出的原则, 将相应的物理页从 NVM 迁移到 DRAM. 基于双向哈希链表的内存页迁移机制的伪代码实现方法如算法 1 所示.

THMigrator 迁移机制的好处在于使用双向哈希链表数据结构存储内存页的信息, 将内存页操作的复杂度降到最低 $O(1)$, 不仅使得内存页信息易于增删改查, 而且大大降低了时间成本; 其次, THMigrator 迁移机制将频繁访问的内存页从 NVM 迁移到 DRAM, 避免了 NVM 写延迟带来的高能耗问题; 再者, 基于 THMigrator 迁移机制, 我们建立了普适性的 EEAM 模型, 量化分析 THMigrator 迁移机制在能效方面的作用, THMigrator 为 EEAM 模型提供了有效的输入.

Algorithm 1 Migration algorithm of two-way Hash chain list

Input: $page_i$ is a memory page, and channelNumber for different memory type (DRAM or NVM);

/*Initialize variables*/
Initialize HashList, MigratorMap, lifeTime, Threshold, etc.;
Primary iteration: Request access memory page;
Request($page_i$); /*Request access to the i-th memory page*/
if isLocationHashMap($page_i$) **then**
    $page_i \rightarrow$ value $\Leftarrow page_i \rightarrow$ value + 1; /*the value of $page_i$ add 1*/
    moveToHead($page_i$); /*Move $page_i$ to the head of HashList*/
else
    setHead($page_i$); /*Set $page_i$ to the head of HashList*/
end if
if $page_i \rightarrow$ value > Threshold **then**
    MigratorMap.insert($page_i$); /*Insert $page_i$ into MigratorMap*/
    HashList.remove($page_i$); /*Remove $page_i$ from HashList*/
end if
if LifeTime($page_i$) > currentTime **then**
    MigratorMap.remove($page_i$); /*Remove memory pages that exceed the life time*/
end if
if InMigratorMap($page_i$)&&!Migratored($page_i$) **then**
    startMigrator($page_i$); /*Migrate $page_i$ from NVM to DRAM*/
    MigratorMap.remove($page_i$); /*Remove memory pages that exceed the life time*/
end if

Output: True/False. /*Output memory page migration is successful or failure*/

3.3 开销分析

THMigrator 迁移机制的页迁移开销主要体现在以下两个方面:

(1) 双向哈希链表的数据结构存在一定的硬件开销. 双向哈希链表存储开销大小和使用的 NVM 大小成正比 [12]. 本文实验时设计了 4 个 channel, 包括 1 个 channel 为 4 GB 的 DRAM 和 3 个 channel 分别是容量为 8 GB 的 NVM. 内存页大小为 4 KB, 双向哈希链表中每个 Entry 条目的大小为 32 B. 采用直接映射方式, 根据正比例 [12] 关系可以推算出存储双向哈希链表所需的内存开销为 64 MB, 占内存总容量为 28 GB 的异构存储器容量的 0.2%. 因此, 双向哈希链表的存储开销对于 GB 级别的存储器, 额外存储开销占比较低.

(2) 对双向哈希链表中内存页信息结点的增删改查等操作, 会产生额外的时间和额外的能耗开销. 由于对双向哈希链表操作的时间复杂度是 $O(1)$, 并对每个内存页根据页编号建立索引表, 可进行快速查找. 对存储结构优化后, 可以进一步降低内存页更新操作的时间开销和能耗开销. 在实验部分, 本文通过具体的实验数据对比分析, 详细分析了额外开销和内存性能增益的关系及对异构系统整体性能的影响.

4 EEAM 模型

针对 DRAM 和 NVM 组成的垂直结构异构内存架构, 本文建立了一种基于效益的 EEAM 模型. EEAM 模型依赖于 THMigrator 机制, 综合考虑时间和能效因素, 以内存页为单位, 总体效益 benefit 包括读写 NVM 内存页 (R1/W1) 所需的时间和能效 [29], 读写 DRAM 内存页 (R2/W2) 所需的时间和

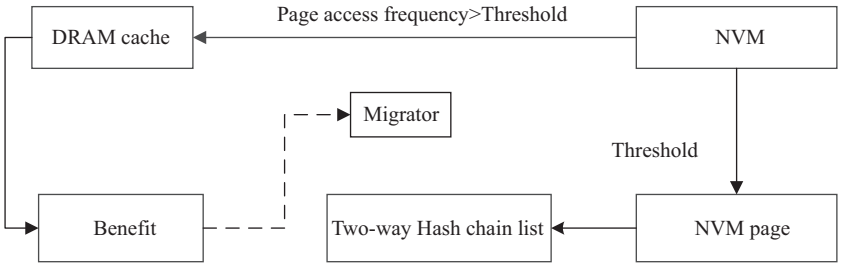


图 4　基于双向哈希链表页迁移算法的能效分析模型

Figure 4　Energy efficiency analysis model based on two-way Hash chain list page migration mechanism

能效, 以及内存页迁移所需的额外时间和能效. 本文的页迁移算法是基于双向哈希链表的, 迁移访问计数值超过给定阈值 Threshold 的内存页. 面向页迁移效益评价的 EEAM 模型如图 4 所示.

假定 DRAM 用 M_0 表示, NVM 用 M_1 表示, DRAM 和 NVM 组成的异构内存可表示为集合 $\{M_0, M_1\}$. 参考 Mai 等 [19] 的研究成果, 定义了以下变量.

定义1　n 个内存页构成的集合可定义为 $\{\text{page}_0, \ldots, \text{page}_i, \ldots, \text{page}_{n-1}\}$, 其中 $0 \leqslant i \leqslant n-1$.

定义2　$E_r(M_0)$ 表示读取一个内存页到 DRAM 所需能量, $E_w(M_0)$ 表示写入一个内存页到 DRAM 所需能量. $E_r(M_1)$ 表示读取一个内存页到 NVM 所需能量, $E_w(M_1)$ 表示写入一个内存页到 NVM 所需能量.

定义3 (能效比 IPJ)　能效比 (ratio of energy efficiency) 是指单位焦耳能量所能执行完成的机器指令数或任务量, 是评估计算机系统单位时间和单位能耗的条件下, 计算机系统整体工作性能的量化指标. 可以反映计算机系统的节能效率. 能效比的量化计算公式可以表示为

$$\text{IPJ} = \text{Number of Instructions}/(\text{Watt} \times \text{Second}) \quad \text{or} \quad \text{IPJ} = \text{Number of Instructions}/\text{Joule}.$$

由 IPJ 公式可知, 能效比越高, 单位能量下某台计算机完成的平均指令数或任务量越多, 亦为某台计算机完成单位指令执行所消耗的平均能量越低.

假定内存系统空闲时间和空闲期所需的能耗忽略不计. $T_r(M_0)$ 表示读取一个 DRAM 内存页所需的时间, $T_w(M_0)$ 表示写入一个内存页到 DRAM 所需时间. $T_r(M_1)$ 表示读取一个 NVM 内存页所需的时间, $T_w(M_1)$ 表示写入一个内存页到 NVM 所需的时间, W_j 表示第 j 个访问周期. 因此, 在周期 W_j 期间内 (所有的访存计数在下一个周期 W_{j+1} 开始处清零) 内存页 page_i 驻留在 M_0 或 M_1 所需的总访问时间和总能量, 可根据内存页 page_i 的访问计数值以及读取和写入 M_0 或 M_1 所需的时间和能量来计算. 内存页 page_i 驻留在 M_0 所需的总访问时间:

$$\text{DT}_{rw}(i) = N_r(i) \times T_r(M_0) + N_w(i) \times T_w(M_0), \tag{1}$$

内存页 page_i 驻留在 M_1 所需的总访问时间:

$$\text{PT}_{rw}(i) = N_r(i) \times T_r(M_1) + N_w(i) \times T_w(M_1), \tag{2}$$

其中 $N_r(i)$, $N_w(i)$ 为内存页 page_i 读写访问的计数值. 内存页 page_i 驻留在 M_0 所需的总能量:

$$\text{DE}_{rw}(i) = N_r(i) \times E_r(M_0) + N_w(i) \times E_w(M_0), \tag{3}$$

内存页 page_i 驻留在 M_1 所需的总访问时间:

$$\text{PE}_{rw}(i) = N_r(i) \times E_r(M_1) + N_w(i) \times E_w(M_1), \tag{4}$$

内存页从 NVM 迁移到 DRAM 的过程中, 需要先从 NVM 中读取需要迁移的内存页 page_i, 然后将相应的内存页 page_i 写入到 DRAM. 内存页 page_i 迁移需要的时间开销为

$$T_m(i) = T_r(M_1) + T_w(M_0), \tag{5}$$

内存页迁移需要的能量为

$$E_m(i) = E_r(M_1) + E_w(M_0), \tag{6}$$

内存页 page_i 从 NVM 迁移到 DRAM 时间效益为

$$\text{TB}(i) = \text{PT}_{rw}(i)/\text{DT}_{rw}(i) + T_m(i), \tag{7}$$

内存页 page_i 从 NVM 迁移到 DRAM 能量效益为

$$\text{EB}(i) = \text{PE}_{rw}(i)/\text{DE}_{rw}(i) + E_m(i), \tag{8}$$

内存页 page_i 从 NVM 迁移到 DRAM 的总效益为

$$B(i) = \text{TB}(i) \times \text{EB}(i), \tag{9}$$

假设 NVM 中有 m 个内存页, DRAM 中有 $n-m$ 个内存页, 则没有内存页迁移时的总效益可进一步表示为

$$B = \left(\sum_0^m \text{PT}_{rw}(i) \Big/ \sum_0^{n-m} \text{DT}_{rw}(i)\right) \times \left(\sum_0^m \text{PE}_{rw}(i) \Big/ \sum_0^{n-m} \text{DE}_{rw}(i)\right), \tag{10}$$

若有 k 个内存页迁移到 DRAM, 则 DRAM 中有 $n-m+k$ 个内存页, 异构系统有内存页迁移时的时间效益 T' 和能量效益 E' 评估值为

$$T' = \sum_0^{m-k} \text{PT}_{rw}(i) \Big/ \sum_0^{n-m+k} \text{DT}_{rw}(i) + \sum_0^k T_m(i), \tag{11}$$

$$E' = \sum_0^{m-k} \text{PE}_{rw}(i) \Big/ \sum_0^{n-m+k} \text{DE}_{rw}(i) + \sum_0^k E_m(i), \tag{12}$$

k 个内存页迁移时的总效益 B' 表示为

$$B' = T' \times E', \tag{13}$$

k 个内存页迁移对比无内存页迁移的总能效增益 ΔB 表示为

$$\Delta B = B - B', \tag{14}$$

k 个内存页迁移对比无内存页迁移的总能效增益率 η 表示为

$$\eta = B'/B. \tag{15}$$

通过以上分析可知, 合理有效的内存页迁移机制可以降低异构内存系统的访问时间和能量开销. 量化的效益评估方法为优化迁移算法提供了依据.

表 2 EEAM 参数配置

Table 2 Configuration of the EEAM

Memory	Read latency	Write latency	Read energy	Write energy	Read speed	Write speed
DRAM	3 μs (4 KB)	3 μs (4 KB)	0.8 J/GB	1 J/GB	1.09 GB/s	1 GB/s
PCM	3 μs (4 KB)	64 μs (4 KB)	1.2 J/GB	6 J/GB	400 MB/s	100 MB/s

表 3 模拟器环境配置

Table 3 Environment configuration of the simulator

Configuration parameter	Value
CPU	2.0 GHZ + TimingSimpleCPU
Mempry	NVMainMemory
L1 cache	32 KB instruction cache + 32 KB data cache
L2 cache	256 KB instruction cache + 256 KB data cache
Memory structure	DRAM + PCM (1:3)
Bus	64 bit
Operation mode	SE mode

5 实验与结果

本文采用 GEM5 和 NVMain 构建基于异构内存的多核处理器模型系统, 针对 SPEC CPU2006 基准测试程序集测试了 THMigrator 迁移机制, 对比多种迁移机制分析了系统的计算性能、访存性能和能效.

5.1 实验环境

GEM5 是用于计算机系统架构的全系统模拟器, NVMain [30] 是 NVM (PCM, STT-RAM) 和 DRAM 异构内存模拟器, 可模拟易失性存储器和非易失性存储器. 以 DRAM 作为 PCM 缓存的垂直型异构内存, GEM5 SE 模式下进行仿真实验的参数配置如表 2 所示.

GEM5 和 NVMain 组成的全系统模拟器中, CPU 类型为 TimingSimpleCPU, CPU Clock 为 2 GHZ. 内存类型是 NVMainMemory, 支持非易失性存储器, 异构内存设置在由 DRAM 和 NVM 组成的异构内存系统的配置文件中, 一级缓存大小为 32 KB, 二级缓存大小为 256 KB. 采用 SPEC CPU2006 基准测试程序 [31] 进行测试, 根据不同的基准测试程序会微调参数. 本文选取多种基准测试程序, 全面检验基于双向哈希链表迁移机制的系统性能和 EEAM 模型的普适性. 实验环境配置如表 3 和 4 所示. 其中, EEAM 模型的相关参数设置和模拟器的配置, 来源于国内外关于 DRAM 和 PCM 的读写速度和能效的量化研究结果 [7, 22, 23, 32].

内存页大小设置为 4 KB, 读一个 4 KB 的 DRAM 内存页需要的能量为 3051 nJ, 读延迟时间为 3 μs. 写一个 4 KB 大小的 DRAM 内存页的能量为 3815 nJ, 写延迟时间为 3 μs. 读一个 4 KB 大小的 PCM 内存页的能量为 4577 nJ, 读延迟时间为 3 μs, 写一个 4 KB 大小的 PCM 内存页的能量为 22880 nJ, 写延迟时间为 64 μs. GEM5 和 NVMain 模拟器自身集成的 Trace 工具可以帮助我们以文本的形式输出程序运行时的日志信息, 在程序代码中通过添加一些关键数据的打印操作, 使用 AddStat 方法可以将运行时的数据输出到日志信息中, 通过这种输出日志的跟踪方法可以获得读写操作的数量和分布以及迁移内存页的数量等信息.

表 4 测试程序配置

Table 4 The configuration of benchmarks

Benchmarks	The number of instruction
bzip2	10000000
gcc	10000000
leslie3d	10000
mcf	10000
calculix	10000000
cactusADM	10000
sjeng	10000000
hmmer	10000000
milc	10000
povray	10000000
soplex	10000000

实验中的异构内存系统有 4 个通道, 第 1 个通道是容量为 4 GB 的 DRAM, 其他 3 个通道为容量为 8 GB 的 PCM 或者 STT-RAM. 总线位宽是 64 bit, 设备通信位宽是 8 bit, DRAM 和 PCM 的带宽比为 2:1. DRAM 的每个 Bank 中的 Rank 的平均带宽为 630 MB/s, DRAM 的带宽占用为 5041 MB/s, PCM 的每个 Bank 中的 Rank 平均带宽为 320 MB/s. PCM 的带宽占用为 2564 MB/s. DRAM 和 PCM 的带宽占用会随着容量大小改变而变化 [33]. 由于 PCM 写延迟较长, 导致内存带宽降低, 在后续的工作中我们将研究带宽对异构内存系统的影响, 通过改变 DRAM 的内存容量和内存时钟频率来调整带宽, 从而设计出一种更高效的自适应内存页迁移机制.

5.2 实验分析

本文通过对比基于冷热区的 MQMigrator 迁移机制和 GEM5, NVMain 模拟器自带的随机迁移机制 (CoinMigrator), 3 种迁移机制在多种非易失性存储器 (PCM 或 STT-RAM) 和 DRAM 组成的异构内存系统中的作用效果, 来验证 THMigrator 机制的有效性. 实验中采用 3 种异构内存系统迁移机制, 分别为 THMigrator, MQMigrator 和 CoinMigrator, 两种不同非易失性存储器分别为 PCM 和 STT-RAM 组成的异构混合内存系统 (DRAM+PCM, STT-RAM+DRAM), 从多个方面验证 THMigrator 机制的性能. CoinMigrator 机制的核心思想是随机产生一个概率值 P, 若 P 超过 0.5, 并且要访问的内存页在 PCM 或 STT-RAM 中, 则执行内存页的迁移操作, 优点是迁移条件判断简单快速, 缺点是根据随机产生的概率值进行迁移具有一定的盲目性. MQMigrator 机制将缓存划分为多个队列, 每个队列对应不同的访问优先级, 访问优先级的设置是根据内存页的访问次数计算出来的, 访问计数值大的内存页所在队列的优先级高, MQMigrator 机制降低了 “缓存污染” 带来的问题, 并且命中率比 CoinMigrator 高, 缺点是 MQMigrator 机制需要维护多个队列, 且需要维护每个数据的访问计数和访问时间, 复杂度远高于 CoinMigrator 迁移机制. THMigrator 机制是这两种机制的折中, 许多内存页缓存算法如 CLOCK-Pro, LIRS 和 LRU 等性能表现良好 [20], 但这些算法只考虑到具有统一访问延迟和无限耐力的主存, 不能直接适用于采用 PCM 或 STT-RAM 混合主内存系统. 因此, 在 LRU 替换算法的基础上, 我们提出了新的迁移机制 (THMigrator), THMigrator 采用双向哈希链表结构优化内存页存储, 通过分析内存页的历史访存信息, 迁移频繁访问的内存页, 避免随机迁移机制的盲目性, 并且双向哈希链表结构无需维护多个队列, 在命中时不需要通过遍历链表, 找到命中的内存页索引, 然后将内存页移到头部, 而是通

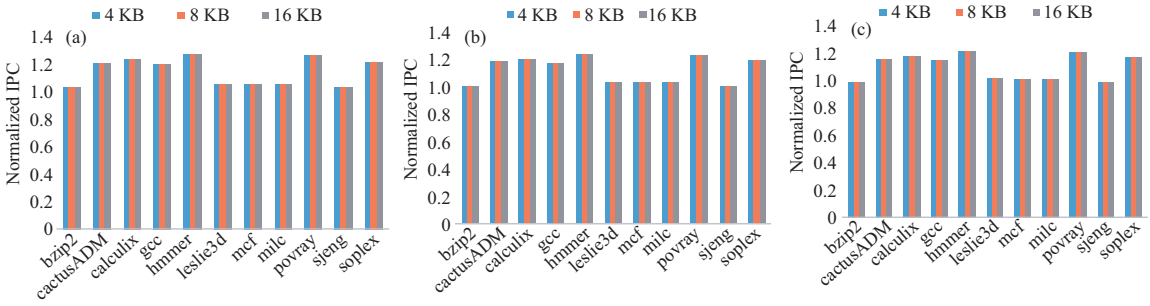


图 5 (网络版彩图) 内存页的归一化 IPC (DRAM + PCM 内存模型)

Figure 5 (Color online) Normalized IPC for different size of pages (DRAM+PCM memory model). (a) THMigrator; (b) MQMigrator; (c) CoinMigrator

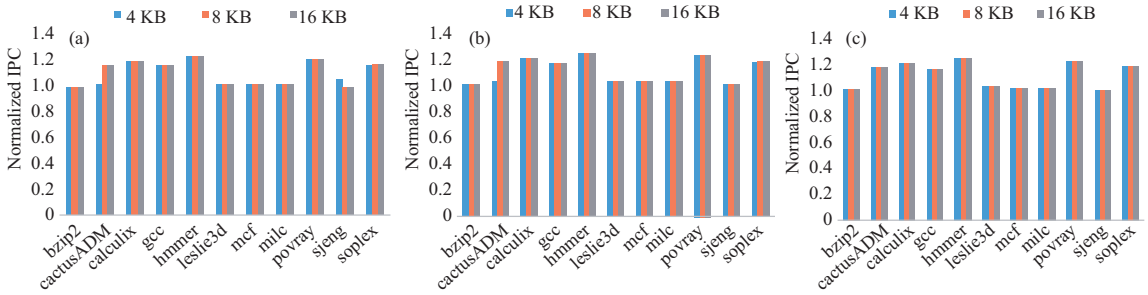


图 6 (网络版彩图) 内存页的归一化 IPC (DRAM + STT − RAM 内存模型)

Figure 6 (Color online) Normalized IPC for different size of pages (DRAM+STT-RAM memory model). (a) THMigrator; (b) MQMigrator; (c) CoinMigrator

过哈希的特性, 可以直接用下标的方式索引内存页, 复杂度降低到 $O(1)$. 本节通过具体的实验数据来评估 THMigrator 机制的效果, 全面分析单个周期内执行的指令数 IPC (instructions per cycle)、DRAM cache 利用率、能效、内存访问延迟、命中率、执行时间、迁移时间轨迹等多项指标, 综合评估系统性能.

实验中, 我们设置了不同大小的内存页 4, 8, 16 KB, 目的是验证内存页大小是否是影响系统能效的一个关键因素. 并在多种不同的 NVM 上进行实验, 实验中使用 PCM 和 STT-RAM 非易失性存储器, 因为 PCM 的技术相对成熟, STT-RAM 容量大, 在扩大共享 LLC (最后一级缓存) 方面取得良好的效果 [33, 34]. 通过对比 CoinMigrator 算法和 MQMigrator 算法, 验证内存页大小对系统整体性能的影响. 图 5 显示了 DRAM 和 PCM 组成的异构内存系统中使用不同的迁移算法, 设置不同大小的内存页, 计算 IPC. 由图 5 可知, CoinMigrator 算法使用不同的内存页, 对 IPC 没有任何影响, IPC 不会随着内存页的大小增大而增大, 而是保持不变. MQMigrator 算法使用不同大小的内存页, 除了 cactusADM 测试程序外, 其他基准测试程序也没有随着内存页大小的变大而增大, THMigrator 算法也表现出相同趋势. 接着, 在 DRAM 和 STT-RAM 组成的异构内存系统上使用不同的迁移算法, 设置不同大小的内存页, 计算单个周期内执行的指令数, 由图 6 可得, 3 种迁移机制中设置大小不同的内存页并不会影响到 IPC, IPC 并没有随着内存页大小的改变而改变.

图 7 为 DRAM 和 PCM 组成的异构内存系统中, 设置大小不同的内存页, 不同迁移机制的单位时间内执行的指令数数据, CoinMigrator 算法中不同大小内存页对指令率影响较大, 图 7(c) 中, 8 KB 大小的内存页的归一化指令率是 4 KB 大小的内存页归一化指令率的 1.01 倍, 16 KB 大小的内存页

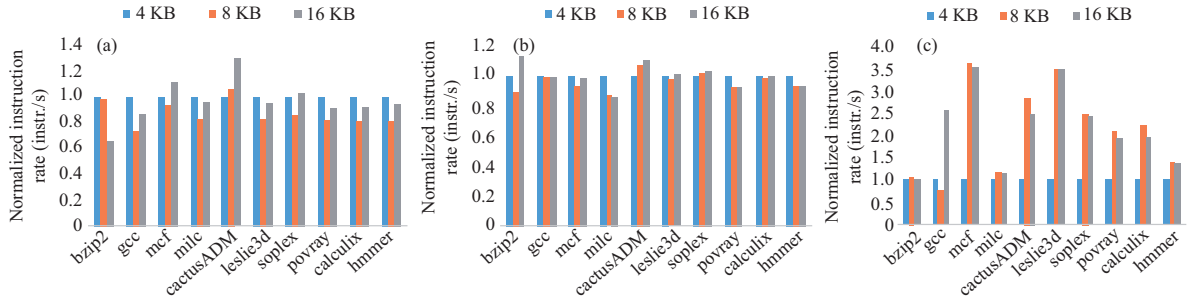


图 7 (网络版彩图) 内存页的归一化指令率 (DRAM + PCM 内存模型)

Figure 7 (Color online) Normalized instruction rate for different size of pages (DRAM+PCM memory model). (a) THMigrator; (b) MQMigrator; (c) CoinMigrator

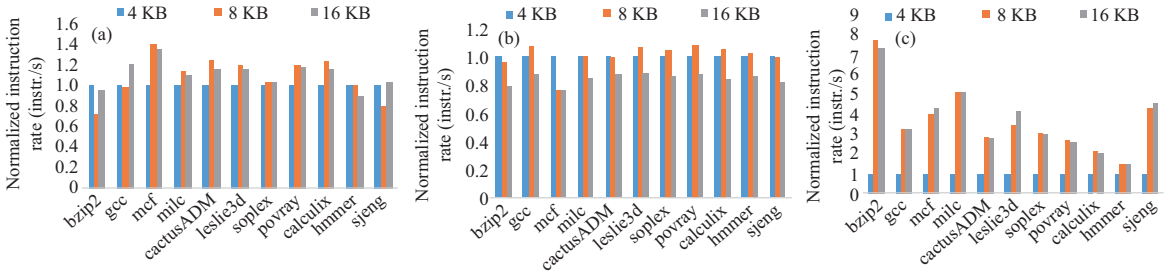


图 8 (网络版彩图) 内存页的归一化指令率 (DRAM + STT − RAM 内存模型)

Figure 8 (Color online) Normalized instruction rate for different sizes of pages (DRAM+STT-RAM memory model). (a) THMigrator; (b) MQMigrator;(c) CoinMigrator

的归一化指令率是 4 KB 大小的内存页归一化指令率的 1.07 倍. 图 7(a) 中, 8 和 16 KB 的内存页的指令率分别比 4 KB 的内存页的归一化指令率减少了 12.12%, 3.02%. 由图 7(b) 可知, 8 KB 的内存页和 16 KB 的内存页的指令率分别比 4 KB 的内存页的归一化指令率减少了 3.60%, 0.26%. 除了 CoinMigrator 算法在大的内存页下指令率增加外, THMigrator 和 MQMigrator 算法在大的内存页下指令率均略有减少.

图 8 为 DRAM 和 STT-RAM 组成的异构内存系统中, 设置不同大小的内存页, 不同迁移机制的单位时间内执行的指令数数据, CoinMigrator 算法中不同大小内存页对指令率影响较大, 由图 8(c) 可知, 8 KB 大小的内存页的归一化指令率是 4 KB 大小的内存页归一化指令率的 2.62 倍, 16 KB 大小的内存页的归一化指令率是 4 KB 大小的内存页归一化指令率的 2.68 倍. 图 8(a) 中, 8 KB 的内存页和 16 KB 的内存页的指令率分别比 4 KB 的内存页的归一化指令率高出 7.74%, 10.26%. 图 8(b) 中, 8 KB 的内存页的指令率比 4 KB 的内存页的归一化指令率高出 0.5%, 16 KB 的内存页的指令率比 4 KB 内存页的指令率减少了 15.94%. 对于 MQMigrator 算法, 内存页大小增大到 16 KB 时, 指令率反而下降, 可见, 不是所有的迁移机制都适合增大内存页取得更高的指令率.

通过实验再次比较了内存页大小的 DRAM cache 的利用率、能效、访问延迟、写访问命中率等性能指标, 发现内存页大小对这些指标的影响较小, 通过以上实验分析, 可以发现内存页大小对 IPC 和指令率有一定的影响, CoinMigrator 迁移算法的指令率受内存页大小的影响比较大, IPC 在不同的内存页下变化还是很小的.

图 9~11 对比分析了 THMigrator, MQMigrator 和 CoinMigrator 机制下的归一化 IPC、归一化的

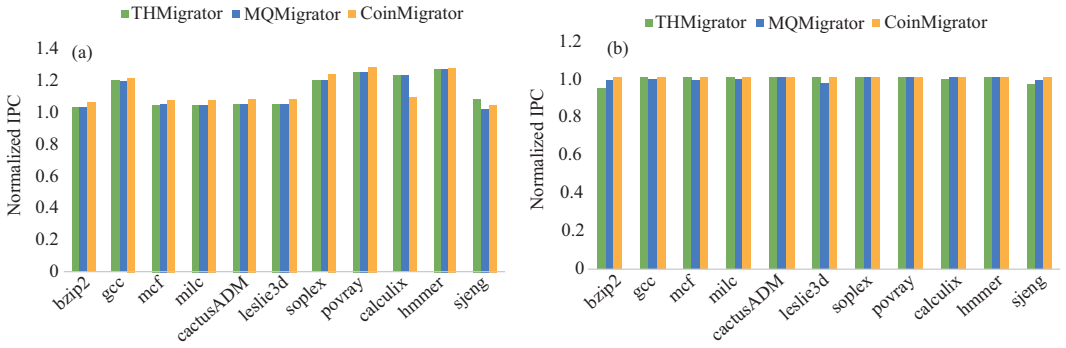


图 9 (网络版彩图) 归一化的 IPC

Figure 9 (Color online) Normalized IPC. (a) DRAM+PCM heterogeneous memory system; (b) DRAM+STT-RAM heterogeneous memory system

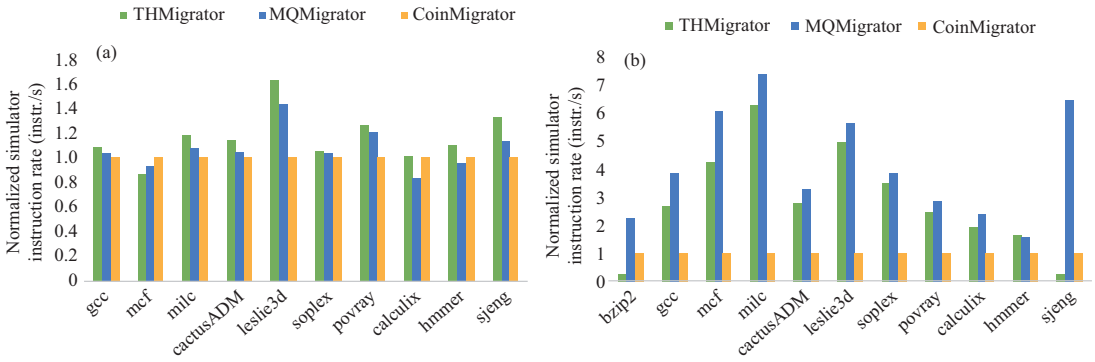


图 10 (网络版彩图) 指令率

Figure 10 (Color online) Average instruction rate. (a) DRAM+PCM heterogeneous memory system; (b) DRAM+STT-RAM heterogeneous memory system

指令率 (异构内存系统单位时间内执行的指令数) 和 DRAM cache 利用率. 图 9(a), 10(a) 和 11(a) 采用 DRAM 和 PCM 组成的异构内存系统, 图 9(b), 10(b) 和 11(b) 采用 DRAM 和 STT-RAM 组成的异构内存系统. STT-RAM 和 PCM 相比, 具有更低的读写能耗、读写延迟和更高的访问速度, 更强的耐久性.

由图 9(a) 可知, calculix 基准测试程序使用 THMigrator, MQMigrator 和 CoiMigrator 迁移机制的归一化 IPC 分别是 1.22, 1.22 和 1.08, 图 9(b) 中, calculix 使用 THMigrator, MQMigrator 和 CoinMigrator 迁移机制的归一化的 IPC 分别是 0.99, 1.00 和 1.00, THMigrator 和 MQMigrator 机制在多种基准测试程序下每个时钟周期内平均执行的指令条数差异很小, 不超过千分之一, THMigrator 和 CoinMigrator 机制的 IPC 差异不到千分之四.

我们接着观察了异构内存系统单位时间 (s) 内执行的指令数. 图 10(a) 中 gcc 使用 THMigrator 算法比使用 MQMigrator 算法的指令率提高了 5%, 比用 CoinMigratior 算法的指令率提高 7.9%, sjeng 基准程序使用 THMigrator 算法比 MQMigrator 算法的指令速率提高了 19%, 比用 CoinMigratior 算法的指令速率提高了 32.5%, 综合 10 种基准测试程序的测试结果, THMigrator 比 MQMigrator 的平均指令率提高了 9.3%, THMigrator 比 CoinMigratior 的平均指令率提高了 16.4%. THMigrator 迁移算法在 PCM 和 DRAM 组成的异构内存系统上, 单位时间内执行的指令数有显著提高, 我们接着测试了在

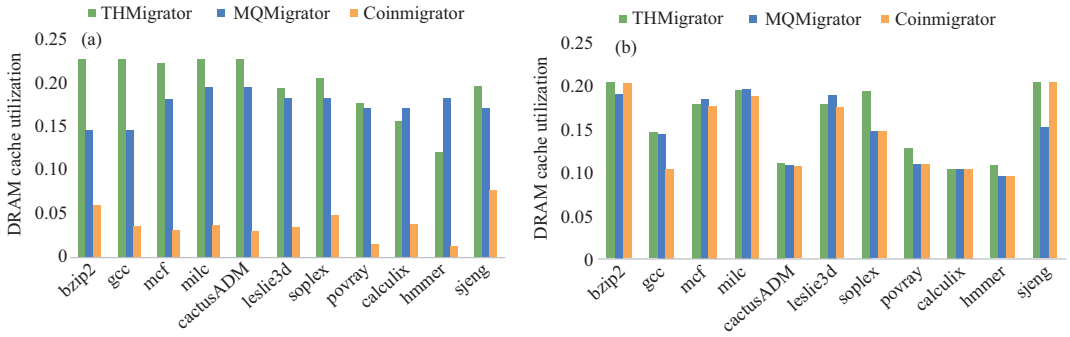


图 11 (网络版彩图) DRAM cache 利用率

Figure 11 (Color online) DRAM cache utilization. (a) DRAM+PCM heterogeneous memory system; (b) DRAM+STT-RAM heterogeneous memory system

STT-RAM 和 DRAM 组成的异构内存系统上 THMigrator 算法的性能, 由图 10(b) 可知, THMigrator 比 CoinMigratior 的平均指令率提高了 0.82 倍, MQMigrator 比 CoinMigratior 的平均单指令率提高了 2.15 倍.

我们接着分析了 DRAM cache 的利用率, 从图 11(a) 可知, bzip2 基准测试程序使用 THMigrator, MQMigrator 和 CoinMigrator 机制的 DRAM cache 利用率分别是 0.22, 0.14 和 0.06, CoinMigrator 机制的 DRAM cache 利用率比较低, THMigrator 机制平均高出 CoinMigrator 机制的 6.5 倍, THMigrator 平均高出 MQMigrator 的 15%. 图 11(b) 中, THMigrator 机制的 DRAM cache 利用率高出 CoinMigrator 机制的 10.6%, 高出 MQMigrator 的 8.87%, 无论是 PCM 还是 STT-RAM 和 DRAM 组成的异构内存系统, THMigrator 算法都有显著的性能提升. 但由于 PCM 和 STT-RAM 材料性质本身存在的差异, THMigrator 在写延迟高的 PCM 上效果较为明显.

CoinMigrator 随机迁移机制仅根据随机概率迁移内存页, 降低了 DRAM cache 的利用率和命中率, THMigrator 和 MQMigrator 则是根据内存页的历史访存信息, 迁移频繁访问的内存页, 将不经常访问的冷页面迁移到 DRAM, MQMigrator 根据内存页的访问频率值, 将内存页划分为多个优先级不同的队列, 优先将访问次数多的内存页迁移到 DRAM, 降低了 “缓存污染” 问题, 但是 MQMigrator 需要维护多个队列, 并且需要维护每个内存页的访问时间, 复杂度比较高, MQMigrator 需要定时扫描所有的队列, 代价也是比较高的, THMigrator 机制采用双向哈希链表优化存储内存页信息结点, 对双向哈希链表的更新操作时间远低于多级队列, 并且无需维护多个队列, 虽然存在 “缓存污染” 问题, 但由于其简单和代价小, 实际应用中反而应用的更多. THMigrator 通过过滤掉冷的 PCM 内存页, 以最低的复杂度和最小的代价将频繁访问的 PCM 内存页迁移到 DRAM 中, 增加了 DRAM 的热度, 从而高效地利用 DRAM cache, 提升了 DRAM cache 的利用率, 减少了随机选取内存页进行迁移的盲目性和不确定性.

图 12(a), 13(a), 14(a) 和 15(a) 采用 DRAM 和 PCM 组成的异构内存系统, 图 12(b), 13(b), 14(b) 和 15(b) 采用 DRAM 和 STT-RAM 组成的异构内存系统, PCM 和 STT-RAM 都是常用的非易失性存储器. 图 12 对比分析不同迁移机制的能效. 由图 12(a) 可知, calculix 基准程序使用 THMigrator 迁移机制比 CoinMigrator 迁移机制的能效降低了 56%, 比 MQMigrator 算法能效降低了 5%. 分析 11 种基准测试程序可知采用 THMigrator 比采用 CoinMigrator 的平均能效降低 26%, 比采用 MQMigrator 的平均能效降低 17%. 分析图 12(b) 可知, THMigrator 算法比采用 CoinMigrator 算法的平均能效降低 5.28%.

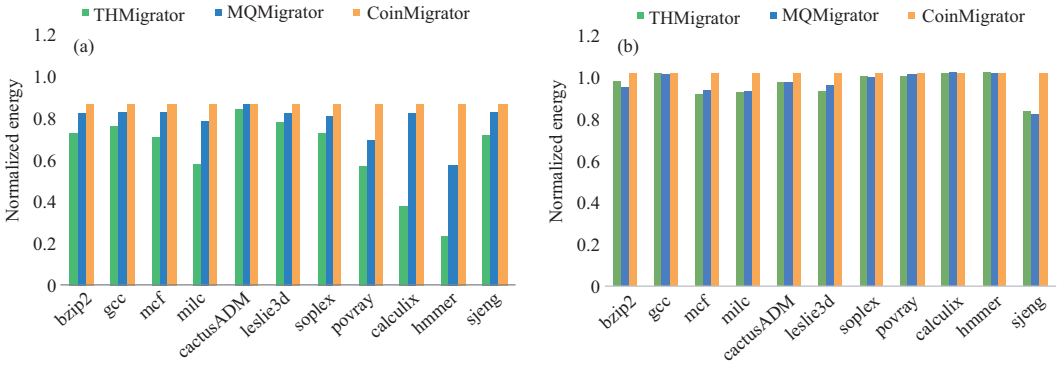


图 12　(网络版彩图) 归一化能效

Figure 12　(Color online) Normalized energy efficiency. (a) DRAM+PCM heterogeneous memory system; (b) DRAM+STT-RAM heterogeneous memory system

图 13(a) 分析了平均内存访问时延, bzip2 使用 THMigrator, MQMigrator 和 CoinMigrator 算法的内存访问时延分别是 27.80, 27.73, 35.76 ns. gcc 使用 THMigrator, MQMigrator 和 CoinMigrator 算法的访问时延分别是 28.30, 46.90, 36.85 ns. 分析 11 种基准测试程序, THMigrator 比 MQMigrator 平均降低 27.5% 的时延, 比 CoinMigrator 平均降低 27.4% 的时延. 图 13(b) 中 THMigrator 比 CoinMigrator 平均降低 6.03% 的内存访问时延.

在异构内存系统中, DRAM cache 能耗大约占总能耗的 13%, THMigrator 在两种异构内存系统中能效有显著降低, 总能效主要来自 DRAM 读写操作所需的能效, 内存页迁移所需的能效和 PCM 读写操作所需的能效, PCM 和 STT-RAM 具有比 DRAM 更低的读写能效, 在 PCM 和 DRAM 组成的异构内存系统中, DRAM 读写操作所需的能效会降低, PCM 由于写延迟高会带来写能效高的问题, 但我们采用有效的 THMigrator 迁移机制, 将 PCM 中频繁访问的内存页迁移到 DRAM, 避免了 PCM 写操作带来的高能效问题. THMigrator 具有比 MQMigrator 更快的迁移速度和更低的迁移代价, 相同时间内可以迁移出更多的内存页到 DRAM, 大量减少这些页面在 PCM 中写操作带来的写延迟问题, 并且 DRAM 具有更高的访存性能, 频繁访问的内存页在 DRAM 中, 可以快速读写, 总的延迟就会降低, 提高了访问异构内存的效率, 而 CoinMigrator 机制不能保证将 PCM 或者 STT-RAM 中频繁访问的内存页迁移到 DRAM, 这些频繁访问的内存页驻留在 PCM 或者 STT-RAM 中带来更多的延迟, 由于 STT-RAM 读写操作延迟小于 PCM, 所以 THMigrator 在 STT-RAM 和 DRAM 组成的异构内存中延迟降低的幅度比较少.

图 14 分析了不同迁移机制下的写操作命中率. 由图 14(a) 可知 THMigrator 算法在不同的基准测试程序下写操作命中率平均达到 95%, CoinMigrator 算法在不同的基准测试程序下写操作命中率平均达到 91.46%, MQMigrator 算法在不同的基准测试程序下写操作命中率平均达到 95%. THMigrator 比 CoinMigrator 机制平均写操作命中率提高了 4.06%. 由图 14(b) 可知, 在 povray, calculix 和 hmmer 基准测试程序下, 3 种机制的写操作命中率达到 99% 以上, 具有很高的命中率. THMigrator 比 CoinMigrator 的命中率平均提高了 0.89%, 由于 CoinMigrator 算法在迁移的过程中是根据概率随机迁移的, 具有一定的盲目性, 没有考虑内存页使用的频繁性, 而 THMigrator 和 MQMigrator 算法在迁移过程中, 有效利用内存页的历史访问信息, 充分考虑内存页的使用频度, 将访问计数值超过阈值的内存页迁移到 DRAM 缓存中, 频繁访问的内存页可能再次被访问, 增加了缓存命中率, 避免了随机迁移的盲目性, 所以 MQMigrator 和 THMigrator 算法比 CoinMigrator 具有更高的命中率. 图 15 对比不同基准测试程

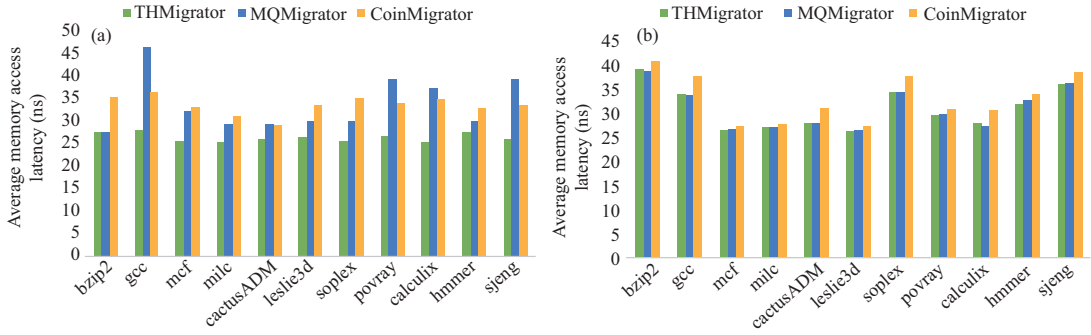


图 13 (网络版彩图) 平均内存访问时延

Figure 13 (Color online) Average latency of accessing memory. (a) DRAM+PCM heterogeneous memory system; (b) DRAM+STT-RAM heterogeneous memory system

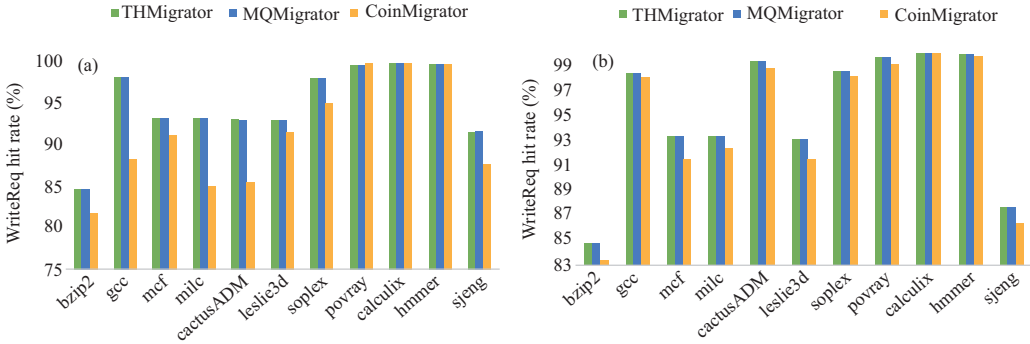


图 14 (网络版彩图) 写操作命中率

Figure 14 (Color online) The hit-rates of write access. (a) DRAM+PCM heterogeneous memory system; (b) DRAM+STT-RAM heterogeneous memory system

序在不同迁移机制下的执行时间. 由图 15(a) 可知, cactusADM 基准测试程序使用 THMigrator 迁移机制比使用 MQMigrator 迁移机制总执行时间降低 2%, 分析多种基准测试程序可知 THMigrator 比 MQMigrator 机制平均执行时间降低 2%. 图 15(b) 中, THMigrator 机制下总执行时间比 MQMigrator 机制下总执行时间降低了 10.68%, 比 CoinMigrator 机制下总执行时间降低 8.95%. THMigrator 机制在 STT-RAM 和 DRAM 组成的异构内存系统中, 节省的时间效果比较明显. THMigrator 机制相比 MQMigrator 机制, 在迁移过程中都是将访问频次超过阈值的内存页进行迁移, 不同的是 THMigrator 机制采用双向哈希链表结构, 具有更低的操作复杂度 ($O(1)$), MQMigrator 机制的复杂度较高, 因为 MQMigrator 要维护多个队列, 查找一个内存页需要遍历整个多级队列, 所以, 相同时间内 THMigrator 可以有效迁移出更多的内存页, 使得驻留在 PCM 中的内存页减少, 从而减少频繁访问的内存页对 PCM 读写操作的次数. 由于 PCM 写操作延时高, 所以迁移出的内存页可以降低 PCM 写操作的总时间. 从数据的存储结构角度分析, 由于 THMigrator 采用双向哈希链表结构存储数据, 在构建链表时根据内存页编号建立索引表, 使得内存页结点在后续的增加、删除、修改和查询操作中时间复杂度相对较低; 而 MQMigrator 采用多级队列存储数据, 时间复杂度远高于双向哈希链表结构, 所以总执行时间比 THMigrator 更高.

图 16~18 分别为 THMigrator, MQMigrator 和 CoinMigrator 机制下每个周期内内存页迁移到 DRAM 的时间轨迹图. 采用的异构内存是 PCM 和 DRAM, 迁移时间以无迁移所需时间为基准进行归

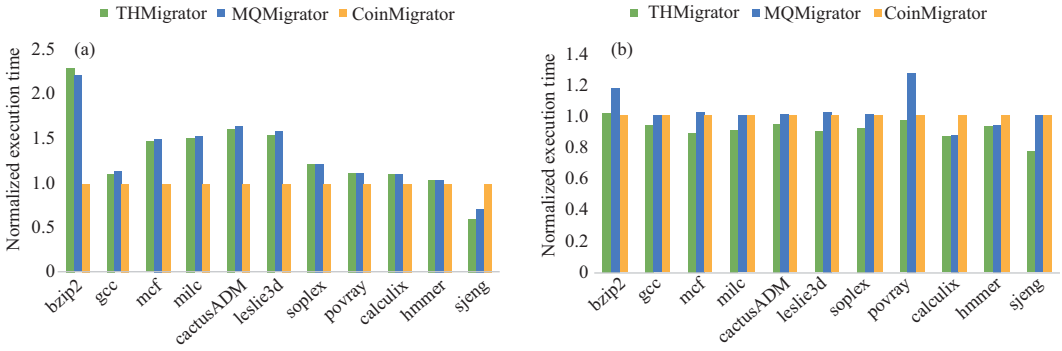


图 15 (网络版彩图) 归一化的执行时间

Figure 15 (Color online) The normalized execute time. (a) DRAM+PCM heterogeneous memory system; (b) DRAM+STT-RAM heterogeneous memory system

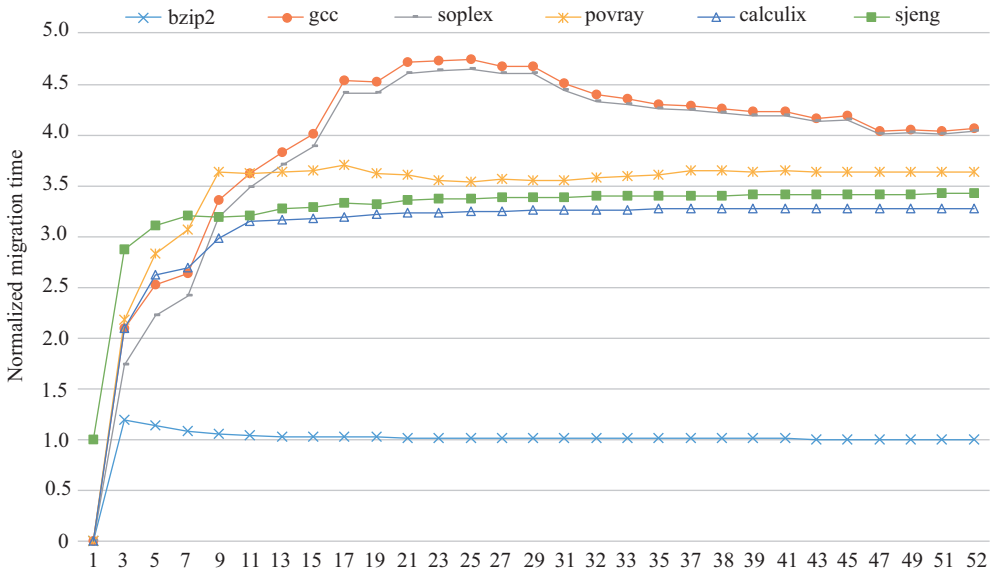


图 16 (网络版彩图) THMigrator 机制下内存页迁移到 DRAM 归一化的时间

Figure 16 (Color online) The normalized time of migration with THMigrator mechanism

一化, 由于 THMigrator 算法迁移的是访问频次值超过阈值的内存页, 而 MQMigrator 算法是根据内存页的访问频次值设置多级队列, 根据队列的优先级进行迁移操作的, THMigrator 算法比 MQMigrator 算法迁移操作简单, 迁移了更多的内存页到 DRAM, 导致 THMigrator 比 MQMigrator 平均高出 32% 的迁移时间. 但迁移出更多内存页却在一定程度上减少 PCM 写延时带来的额外时间和能量开销. THMigrator 比 MQMigrator 平均内存访问时延降低 27.5%, 总的执行时间降低 2%. THMigrator 和 CoinMigrator 相比, 平均迁移时间降低了 15.9%, 由于 THMigrator 迁移算法是根据内存页的访问频繁程度进行迁移的, 避免了 CoinMigrator 迁移的随机性和盲目性, THMigrator 合理的迁移算法可以减少迁移时间.

通过以上实验分析, THMigrator 机制对异构内存的访问时延、能效、DRAM cache 利用率、命中率、执行时间、迁移时间等方面表现出良好的性能提升. 在内存异构系统 (PCM+DRAM 和 STT-

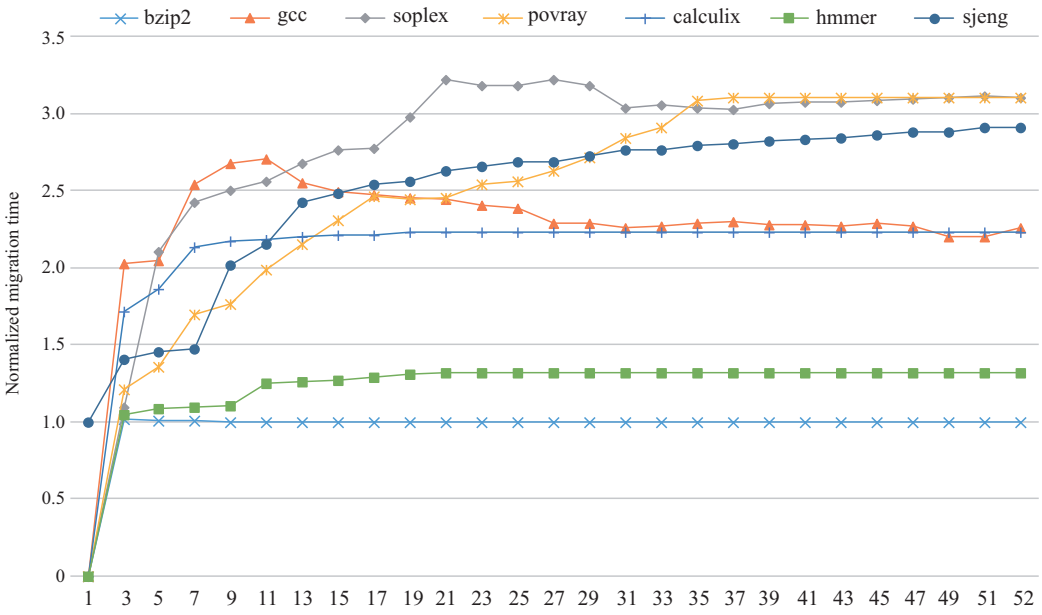


图 17 (网络版彩图) MQMigrator 机制下内存页迁移到 DRAM 归一化的时间

Figure 17 (Color online) The normalized time of migration with MQMigrator mechanism

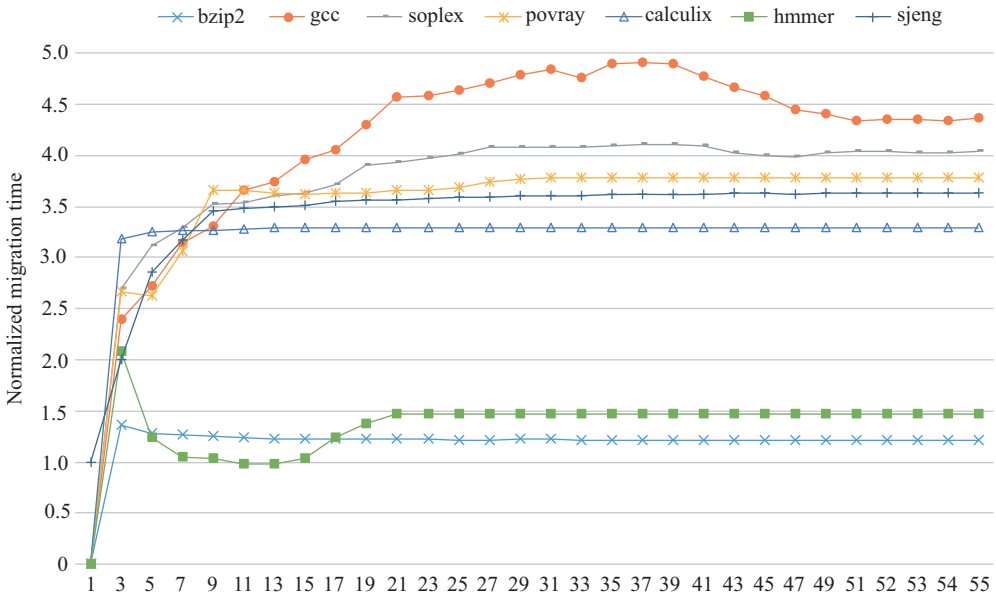


图 18 (网络版彩图) CoinMigrator 机制下内存页迁移到 DRAM 归一化的时间

Figure 18 (Color online) The normalized time of migration with CoinMigrator mechanism

RAM+DRAM) 上进行实验, 对比分析了 3 种迁移机制 (THMigrator, MQMigrator 和 CoinMigrator), 设置大小不同的内存页 4, 8, 16 KB, 从实验系统平台、迁移机制和内存页大小多方面测试 THMigrator 迁移机制的有效性, 全面对比和评估增强 THMigrator 的普适性和泛化性. THMigrator 迁移机制从底层的内存页存储结构用双向哈希链表进行优化, 到内存页的迁移过程用 THMigrator 机制升级, 至

基于效益评估的 EEAM 模型改进, 并分析了不同的非易失性存储器 PCM 和 STT-RAM, 有效提高了 NVM 和 DRAM 组成的异构内存整体性能. 提高了多核异构计算和异构内存构成的高性能的双异构计算机系统的数据预取效率, 提高了双异构计算机系统的整体计算性能.

6 总结

本文主要研究异构内存系统的页迁移机制及相关的时间开销和能效开销, 提出了基于双向哈希链表的异构内存页迁移机制 THMigrator, 建立了 EEAM 模型. 通过实验对比 MQMigrator 和 CoinMigrator 在 NVM 和 DRAM 组成的异构内存系统中的性能, 实验结果表明 THMigrator 和 EEAM 对异构内存系统的访存性能和效能有着显著的提高.

后续工作将 EEAM 能效模型进行优化, 使其对预测迁移提供决策支持. 评估 DRAM 和 NVM 混合系统架构中, 带宽对能耗的影响. NVM 虽然显著降低了访存能耗, PCM 由于写延迟较长, 导致内存带宽降低, 通过改变 DRAM 的内存容量设置和内存时钟频率来调整带宽, 从而设计出一种高效能的自适应数据迁移机制. THMigrator 迁移算法和基于软硬件协同的迁移算法, 动态自适应迁移算法等其他迁移算法做对比, 全面评估 THMigrator 算法的性能. 对于 EEAM 模型, 影响能效的因素有迁移算法、阈值设置、迁移时间、内存访问时延和异构内存架构设计等多种因素, 将来的工作中可以融合多种因素建立综合 EEAM 模型, 最后, 随着机器学习和人工智能的飞速发展, 采用卷积神经网络算法加速预测异构内存地址空间的访问踪迹, 提高多核异构计算单元和异构内存系统构成的高效能双异构计算机系统的数据预取效率和访存性能, 增强双异构计算机系统的综合性能.

参考文献

1 Zhang D Z. Research and implementation of a simulation system for PCM/DRAM-based hybrid memory. Dissertation for Master Degree. Hefei: University of Science and Technology of China, 2017 [张德志. 基于 PCM 和 DRAM 的混合主存仿真系统研究与实现. 硕士学位论文. 合肥: 中国科学技术大学, 2017]

2 Lefurgy C, Rajamani K, Rawson F, et al. Energy management for commercial servers. Computer, 2003, 36: 39–48

3 Wu Y, Fu Y J, Chen W W, et al. Efficient mechanism of hybrid memory placement and erasure code. Comput Sci, 2017, 44: 57–62 [吴炀, 付印金, 陈卫卫, 等. 一种高效的混合内存布局机制与编码技术. 计算机科学, 2017, 44: 57–62]

4 Mao W, Liu J N, Tong W, et al. A review of storage technology research based on phase change memory. Chinese J Comput, 2015, 38: 944–960 [冒伟, 刘景宁, 童薇, 等. 基于相变存储器的存储技术研究综述. 计算机学报, 2015, 38: 944–960]

5 Mittal S, Vetter J S, Li D. A survey of architectural approaches for managing embedded DRAM and non-volatile on-chip caches. IEEE Trans Parallel Distrib Syst, 2015, 26: 1524–1537

6 Li Y, Chen Y R, Jones A K. A software approach for combating asymmetries of non-volatile memories. In: Proceedings of ACM/IEEE International Symposium on Low Power Electronics and Design, 2012. 191–196

7 Shu J W, Lu Y Y, Zhang J C, et al. Research progress on non-volatile memory based storage system. Sci Technol Rev, 2016, 34: 86–94 [舒继武, 陆游游, 张佳程, 等. 基于非易失性存储器的存储系统技术研究进展. 科技导报, 2016, 34: 86–94]

8 Jin P Q. Big data storage management based on new storage. Big Data Res, 2017, 3: 70–82 [金培权. 基于新型存储的大数据存储管理. 大数据, 2017, 3: 70–82]

9 Liu T. Parallel program scheduling for hybrid memory computing. Dissertation for Master Degree. Wuhan: Huazhong University of Science and Technology, 2015 [刘涛. 异构内存环境下并行程序调度优化系统. 硕士学位论文. 武汉: 华中科技大学, 2015]

10 Zhang J B. An energy management for hybrid memory based on write frequency of pages. Dissertation for Master Degree. Wuhan: Huazhong University of Science and Technology, 2015 [张进宝. 一种基于页面热度的异构内存能耗管理机制. 硕士学位论文. 武汉: 华中科技大学, 2015]

11 Khouzani H A, Yang C M, Hu J T. Improving performance and lifetime of DRAM-PCM hybrid main memory through a proactive page allocation strategy. In: Proceedings of the 20th Asia and South Pacific Conference and Design Automation Conference (ASP-DAC), 2015. 508–513

12 Dhiman G, Ayoub R, Rosing T. PDRAM: a hybrid PRAM and DRAM main memory system. In: Proceedings of the 46th Annual Design Automation Conference, 2009

13 Yoon H B, Meza J, Ausavarungnirun R, et al. Row buffer locality aware caching policies for hybrid memories. In: Proceedings of the 30th International Conference on Computer Design (ICCD), 2012. 337–344

14 Liu H K, Chen Y J, Liao X F, et al. Hardware/software cooperative caching for hybrid DRAM/NVM memory architectures. In: Proceedings of the International Conference on Supercomputing, 2017

15 Ramos L E, Gorbatov E, Bianchini R. Page placement in hybrid memory systems. In: Proceedings of the International Conference on Supercomputing, 2011. 85–95

16 Park K H, Park S K, Hwang W, et al. Resource management of manycores with a hierarchical and a hybrid main memory for MN-mate cloud node. In: Proceedings of the 8th World Congress on Services (SERVICES), 2012. 301–308

17 Seok H, Park Y, Park K H. Migration based page caching algorithm for a hybrid main memory of DRAM and PRAM. In: Proceedings of ACM Symposium on Applied Computing, 2011. 595–599

18 Pagh R, Rodler F F. Cuckoo hashing. J Algorithm, 2004, 51: 122–144

19 Mai H T, Park K H, Lee H S, et al. Dynamic data migration in hybrid main memories for in-memory big data storage. ETRI J, 2014, 36: 988–998

20 Kim S, Hwang S H, Kwak J W. Adaptive-classification CLOCK: page replacement policy based on read/write access pattern for hybrid DRAM and PCM main memory. Microprocessors MicroSyst, 2018, 57: 65–75

21 Zhang Z, Fu Y J, Hu G Y. DualStack: a high efficient dynamic page scheduling scheme in hybrid main memory. In: Proceedings of International Conference on Networking, Architecture, and Storage (NAS), 2017

22 Wu D H, He B S, Tang X Y, et al. RAMZzz: rank-aware DRAM power management with dynamic migrations and demotions. In: Proceedings of International Conference on High Performance Computing, Networking, Storage and Analysis (SC), 2012

23 Pei S W, Zhang J, Xiong N, et al. Performance-energy efficiency model of heterogeneous parallel multicore system. In: Proceedings of the 6th International Conference on Green Computing Conference and Sustainable Computing Conference (IGSC), 2015

24 Pei S W, Zhang J G, Jiang L H, et al. Evaluating the overhead of data preparation for heterogeneous multicore system. KSII Trans Int Inform Syst, 2016, 10: 3231–3244

25 Liu D, Zhang J B, Liao X F, et al. Simulator for hybrid memory architecture. J East China Norm Univ (Nat Sci), 2014, 5: 133–140 [刘东, 张进宝, 廖小飞, 等. 面向混合内存体系结构的模拟器. 华东师范大学学报, 2014, 5: 133–140]

26 Zhou Y Y, Philbin J, Li K. The multi-queue replacement algorithm for second level buffer caches. In: Proceedings of the General Track: 2001 USENIX Annual Technical Conference, 2001. 91–104

27 Lee B C, Ipek E, Mutlu O, et al. Architecting phase change memory as a scalable dram alternative. ACM SIGARCH Comput Architect News, 2009, 37: 2–13

28 Zuo P F, Hua Y. A write-friendly and cache-optimized hashing scheme for non-volatile memory systems. IEEE Trans Parallel Distrib Syst, 2018, 29: 985–998

29 Hassan A, Vandierendonck H, Nikolopoulos D S. Software-managed energy-efficient hybrid DRAM/NVM main memory. In: Proceedings of the 12th ACM International Conference on Computing Frontiers, 2015

30 Poremba M, Xie Y. NVMain: an architectural-level main memory simulator for emerging non-volatile memories. In: Proceedings of IEEE Computer Society Annual Symposium on VLSI, 2012. 392–397

31 Henning J L. SPEC CPU2006 benchmark descriptions. SIGARCH Comput Archit News, 2006, 34: 1–17

32 Chen S, Gibbons P, Nath S, et al. Rethinking database algorithms for phase change memory. In: Proceedings of the 5th Biennial Conference on Innovative Data Systems Research (CIDR'11), 2011. 21–31

33 Zhao J S, Xie Y. Optimizing bandwidth and power of graphics memory with hybrid memory technologies and adaptive data migration. In: Proceedings of International Conference on Computer-Aided Design (ICCAD), 2012. 81–87

34 Gao L, Wang R, Xu Y L, et al. SRAM- and STT-RAM-based hybrid, shared last-level cache for on-chip CPU-GPU heterogeneous architectures. J Supercomput, 2018, 74: 3388–3414

Migration mechanism of heterogeneous memory pages using a two-way Hash chain list

Songwen PEI[1,2,4*], Yanfei JI[1], Tianma SHEN[1] & Haikun LIU[3]

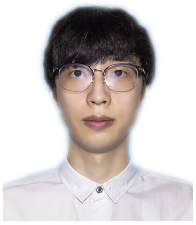
1. *School of Optical-Electrical and Computer Engineering, University of Shanghai for Science and Technology, Shanghai* 200093*, China*;
2. *School of Management, Fudan University, Shanghai* 200433*, China*;
3. *School of Computing, Huazhong University of Science and Technology, Wuhan* 430074*, China*;
4. *State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100190*, China*
* Corresponding author. E-mail: swpei@usst.edu.cn

Abstract With the rapid development of big data technologies, the requirements of tremendously accessing memory are increasing, therebyresulting in a serious problem of high consumption power by accessing traditional DRAM memory. However, large volume and low power dissipation of NVM are becoming increasingly mature, showing potential as an alternative main memory in a heterogeneous memory system. In terms of the historical traces of accessing memory pages, we propose a two-way Hash chain list-based migration mechanism (THMigrator) for a heterogeneous memory system. The THMigrator can migrate memory pages with high frequency from PCM or STT-RAM to DRAM. Moreover, we evaluated the energy efficiency of a heterogeneous memory system by the proposed energy efficiency analysis model. The experimental results show that the performance of computation and the average ratio of energy efficiency supported by the THMigrator are improved by 9.3% and 17%, respectively, compared with that of an MQMigrator. Moreover, the average ratio of energy efficiency supported with the THMigrator increased by 26% compared with that of a CoinMigrator.

Keywords page migration, two-way Hash chain list, heterogeneous systems, non-volatile memory, migration algorithm

Songwen PEI was born in 1981. He received his Ph.D. degree in Computer Architecture from Fudan University in 2009. He is currently an Associate Professor with the University of Shanghai for Science and Technology, Shanghai, China, and a Research Fellow with Fudan University. He was a Research Scientist with the University of California, Irvine, USA, from July 2013 to August 2015 and the Queensland University of Technology, Brisbane, Australia, in 2017. His current research interests include heterogeneous multicore system, cloud computing, and big data.

Yanfei JI was born in 1992. She received her B.S. degree in Computer Science from Luoyang Normal University in 2016. She is currently a graduate student in Computer Architecture at the University of Shanghai for Science and Technology. Her main research interests include memory migration in heterogeneous systems, machine learning, artificial intelligence, and neural network acceleration.



Tianma SHEN was born in 1994. He received his B.S. degree from Shanghai Maritime University in 2017. He is currently a graduate student of Computer Science and Engineering Department at the University of Shanghai for Science and Technology. His main research interests include deep learning, image processing, and natural language processing.

Haikun LIU was born in 1981. He received his Ph.D. degree from Huazhong University of Science and Technology, China. He is an Associate Professor in School of Computer Science and Technology, HUST. His current research interests include in-memory computing, virtualization technologies, cloud computing, and distributed systems.