XOR 缓存:压缩的催化剂

潘哲文威斯康星大
学麦迪逊分校 美国麦迪逊

zhewen.pan@wisc.edu

约书亚·圣米格尔威斯康星大学麦迪逊分
校 美国麦迪逊市 jsanmiguel@wisc.edu

摘要现代计算
系统为缓存分配了大量资源,尤其是末级缓存 (LLC)。我们观察到,通过利用当今系统中常见的私有缓存和包含机制带来的冗余,存在尚未开发的压缩潜力。我们引入了 XOR Cache ,通过 XOR 压缩来利用这种冗余。与传统的缓存架构不同,XOR Cache 存储行对的按位异或值,通过一种行间压缩的形式将存储的行数减半。当与其他压缩方案结合使用时,XOR Cache可以通过对值相近的行进行异或运算,从而降低压缩前数据的熵,进一步提高行内压缩率。

评估结果表明,与更大的未压缩缓存相比， XOR Cache 可以节省1.93 倍的 LLC 面积和1.92 倍的功耗,性能开销为2.06% ,同时能量延迟积降低 26.3%。

CCS 概念 ·计算机
系统组织→多核架构。

关键词

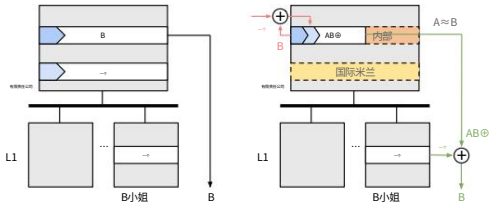
缓存层次结构、缓存压缩、低功耗架构

ACM 参考文献格式： Zhewen Pan 和
Joshua San Miguel。2025 年。XOR 缓存 :压缩的催化剂。第 52 届国际计算机体系结构研讨会 (ISCA 25) 论文集，2025 年 6 月 21 日至 25 日,日本东京。ACM ,美国纽约州纽约,共 14页。https ://doi.org/10.1145/3695053.3730995

1 简介

当今的计算系统将数十到数百[39] MB的SRAM 专用于缓存,这占据了芯片面积的很大一部分,例如 AMD 的 Zen3 的 32 MB L3 缓存占据了芯片面积的 40% 左右[38]。此外,这些系统的功耗也会激增,进一步降低了整体能效。由于数据集大小的增长和内存墙问题,对缓存层次结构中的资源的需求将继续增加。然而,尽管容量更大,但大型缓存并不一定意味着更好的性能;此外,它们的代价是高访问延迟,通常为数十个周期。鉴于其对资源的要求,这些因素加在一起可能使传统的大型缓存在未来的系统中效率低下。为了弥补这种效率差距,我们寻求更好的方法来优化缓存层次结构,在保持性能的同时减少缓存占用空间和功耗。

ACM ISBN 979-8-4007-1261-6/25/06 https://
doi.org/10.1145/3695053.3730995



(a)传统（b)XOR 缓存图 1:高级概览。与传统缓存不同， XOR 缓存存储的是线对的按位 XOR。

缓存压缩[3, 8–10, 16, 24, 41, 42, 45–47, 49, 51, 54]是一项颇具前景的研究方向。缓存可以在插入时压缩缓存行,并在访问时解压缩。这种方法减少了缓存占用,而访问延迟的开销仅为几个额外的周期。有效的缓存压缩利用数据压缩性,在显著降低硬件成本的同时,实现了更大缓存的优势。

我们介绍了 XOR Cache,这是一种新的压缩 LLC 架构,它利用了当今系统内存层次结构所固有的跨多个缓存的冗余。更具体地说， XOR Cache 利用了由于包含和私有缓存而产生的冗余。首先,以前的缓存压缩范例仅利用单个缓存级别内的值冗余来获取压缩机会,而 XOR Cache 则专注于控制由于高级和低级缓存之间的包含而产生的冗余。事实证明,包含式缓存层次结构会引入大量数据冗余,从而降低低级缓存的有效容量1 [15, 26, 37, 40, 48 , 55]。不幸的是,除了放宽严格包含性这一简单的解决方案之外,这种数据重复一直被忽视。为了弥补这一差距,XOR Cache 将曾经被认为是缺点的东西（由于包含而导致的冗余)转化为尚未开发的可压缩性。

其次,XOR Cache 利用私有缓存带来的冗余,通过其一致性协议支持的私有缓存之间的转发来解压缩数据。XOR Cache通过利用这两种冗余形式实现了高效的压缩。

图1展示了 XOR Cache 的高层概览。与按原样存储行的传统缓存不同,在 XOR Cache 中,一个包含式缓存行（例如行 A)由于已存在于 L1 中,因此可以与较低层级的另一个缓存行（例如行 B)进行异或运算,形成单行A⊕B ,如粉色箭头所示。在 LLC 访问中,我们可以简单地将异或后的行A⊕B转发到较高层级,并执行另一个异或运算来反转压缩,如绿色箭头所示。需要注意的是,L1 中的行永远不会进行异或运算,因此 L1 命中不会造成额外的延迟。这样做有两个新的好处：

1我们将距离处理器较远的缓存称为低级缓存。

1) 对两行数据进行异或运算,可以节省 LLC 中的一行存储空间； 2) 当两行数据值相似时,进行异或运算可以降低熵值,使其更易于压缩,并促进其他压缩方案的有效性。本研究的目标是利用异或缓存带来的增强压缩性来减少 LLC 的面积和功耗。

1.1缓存层次结构中的冗余（行间压缩）

除非严格设置为排他,否则 LLC 通常包含所有缓存行（即包含式 LLC)或部分缓存行（即非包含非排他 (NINE) LLC），这些缓存行存在于更高级别的缓存中。这种重复是传统缓存压缩方法中缺失的环节,因为它们通常仅利用单个缓存级别内的冗余。然而,它可以为跨缓存级别边界的压缩创造额外的机会。为了利用这些机会， XOR Cache 在插入时,会在插入的缓存行与另一个选定的缓存行之间执行按位异或运算,并将结果存储在LLC 数据阵列中。通过这样做,它有效地将两个缓存行共置在一个物理插槽中,从而实现 2:1 的压缩率。当至少有一条原始缓存行在更高级别共享以保持恢复能力时,压缩后的缓存行对可以保持压缩状态,我们称之为最小共享不变量。仅利用 XOR 压缩,我们可以实现最佳情况下2 的压缩率,从而将 LLC 数据阵列的大小缩小一半。我们将其表示为一种行间压缩形式（图1b 中的 inter）。

1.2 XOR压缩的协同作用（行内压缩）

XOR 压缩可以独立于之前的缓存压缩方案工作;然而,当我们仔细选择 XOR 候选行时,它和其他方案之间存在协同作用。当组合使用时,XOR Cache 可以提高（催化)其他方案的压缩率。例如,在图1b 中,当选定的行 A 与 B 相似时,即A≈B,XOR 后的行A⊕B可以表现出更低的熵并进一步压缩。这种压缩率的提升是通过利用 XOR 行内的相似性来实现的,这表示为一种行内压缩形式,在图中标记为 intra。使用 XOR 压缩,首先对每对缓存行进行 XOR 运算,然后我们可以对 XOR 后的数据应用现有的基线压缩方案。

如图2所示,我们通过在运行各种基准测试时分析 LLC 快照来量化这种协同作用。为了进行此分析,我们假设同一存储体或同一集合中的任意两条线都可以进行异或运算,而无需施加最小共享不变量（第1.1 节）。作为示例,我们展示了XOR Cache 与BΔI [45]和位平面 (BPC) [30]压缩（它们是线内缓存压缩方案)以及Thesaurus [24] （作为线间压缩方案)的协同作用的潜力。BΔI [45]利用缓存线内值的低动态范围,并使用单个基值和具有较小大小的增量数组对线进行编码。BPC [30]应用三种转换来提高数据压缩率,并使用游程编码和频繁模式压缩来压缩转换后的数据。Thesaurus [24]使用局部敏感哈希动态地聚类缓存线,并根据质心压缩线。

XOR 对选择策略（简称 XOR 策略)决定了哪两条缓存线应该相互进行 XOR 运算,这是 XOR Cache 设计空间的关键所在。我们来考虑三种假设的 XOR 策略,分别是 randBank、idealSet 和 ideal- Bank。对于 randBank,一条线路会随机选择同一 Bank 中的另一条线路,并与其进行 XOR 运算。对于 idealSet,一条线路会将同一集合中的所有线路视为XOR 候选,并选择最理想的 XOR 运算。

产生最易压缩异或行的候选。例如,在图3 中,集合 0 和路 0 处的行可能与同一集合中的任何行（即蓝色框中的行)进行异或。这里,集合 0 中理想的异或候选是路 2 处的行,因为它们只有一位差异。同样,对于 idealBank,一条行会考虑同一bank中的所有行,并与产生最小大小的行进行异或。在图3 中,集合 0 和路 0 处的行可以与橙色框中的任何行进行异或。集合 1 和路 3 处的行是最理想的,因为它具有完全相同的值。如图2 所示,最左侧的几组条形表示未进行异或的基准压缩比;其他三组条形表示使用上述三种异或策略的异或缓存的压缩比。首先,这些行相互进行异或,然后所选的基准压缩方案进一步压缩经过异或处理的行。

我们得出以下观察结果:首先,两种基于搜索的XOR 策略（即 idealBank 和 idealSet)比与值无关的随机 XOR 策略实现了更高的压缩比。

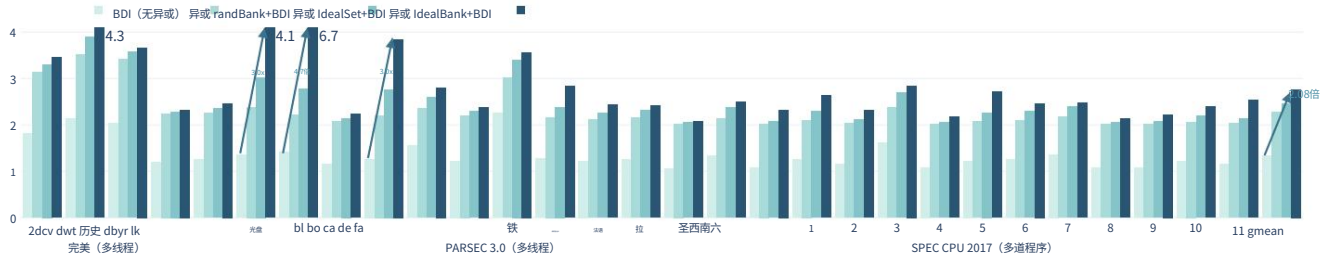
这表明在选择 XOR 候选时考虑值相似性的重要性。其次,idealBank 的性能优于 idealSet,因为前者的搜索范围明显更大,因此更有可能找到理想的 XOR 候选。然而,idealBank 的实现成本也可能明显超过 idealSet。虽然基于搜索的策略并不代表实际实现（我们将在3.2节中讨论实际实现），但它展示了XOR Cache 的上限压缩率。更具体地说， idealBank XOR 压缩会在整个 LLC 库中搜索最佳候选,它可以将BΔI、BPC 和同义词库的压缩率平均分别提高2.08 倍、 2.09 倍和2.02 倍,最高可分别提高4.7 倍、 3.0 倍和4.6 倍,这表明XOR Cache 具有催化压缩的潜力。

1.3 挑战要充分发挥 XOR Cache 的潜力,必须应对两大挑战。第一个挑战在于设计能够实现协同效应的 XOR 策略。设计空间十分充足,从随机到值感知,每种策略都有其在复杂性、有效性和性能之间的权衡取舍。第二个挑战涉及一致性协议的设计。与以往的研究不同,XOR 压缩跨越了单个缓存级别的边界;因此,一致性协议需要进行重大的重新设计。

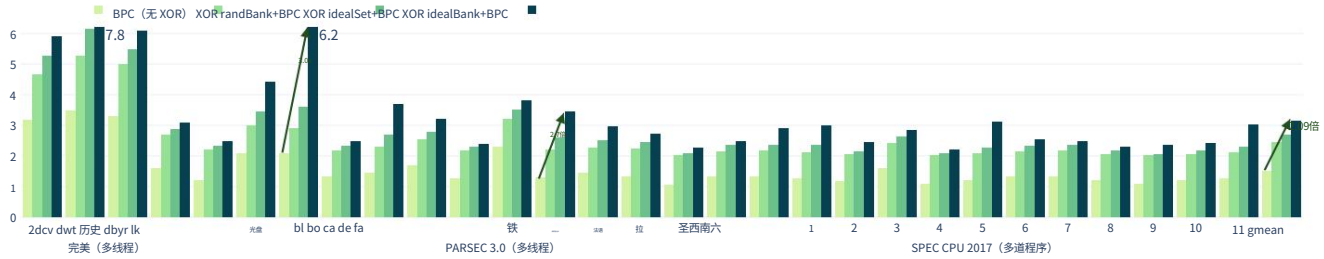
除了保持一致性之外,协议还需要确保未压缩的数据值始终可恢复。

本研究提出了一种压缩的 LLC 架构 XOR Cache,它能够在保持性能的同时大幅缩减数据阵列。通过探索 XOR Cache 的设计空间,我们展示了XOR 与其他方案之间的协同作用如何提升压缩比。本研究的贡献如下：

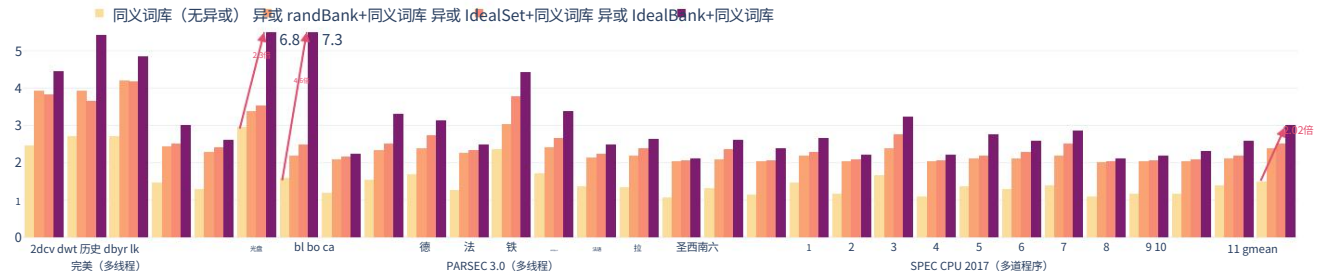
(1)我们提出了一种新颖的缓存压缩方案,该方案利用了由于包含和私有缓存而产生的冗余,而这些在先前的工作中尚未得到充分研究。



(a)异或+BΔI



(b)异或+BPC



(c)XOR+同义词库

图 2:LLC 分析的压缩比。(a)显示 XOR 与BΔI 的压缩比;(b)显示
与 BPC 进行异或;(c) 显示与 Thesaurus 进行异或的压缩率。一个缓存行可以随机地与另一个缓存行进行异或
同一银行（randBank）,或者搜索整个集合/银行以找到最小化数据存储的最佳候选者（idealSet/idealBank）。

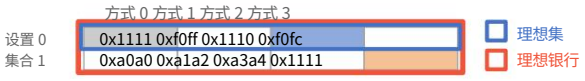


图 3: IdealSet和idealBank异或策略。示例
LLC 组是 4 路组相联的,包含 2 个集合。
表示搜索范围;阴影线代表理想
对集合 0 和路径 0 处的行的候选进行异或。

(2)我们表明 XOR 压缩可以协同提升
与其他缓存压缩方案结合时的压缩率。

(3)我们使用 Ruby 内存模型实现 XOR Cache
在 gem5 模拟器[36]中,并在全系统中进行评估
在多个基准套件上进行模拟。

本文的其余部分安排如下。第2部分
提供缓存概念的背景并总结缓存
压缩研究。第3节介绍了 XOR 压缩和
XOR策略。第4节和第5节讨论了XOR Cache对数据转发和架构实现
的一致性协议支持。
第6节介绍了我们的评估方法和实验
结果。第7节讨论了相关工作,第8节进行了总结。

2 背景

我们提供了现代缓存层次结构中的包含策略和缓存一致性协议设计选择
的背景,然后
低功耗和压缩缓存工作原理总结。

2.1 现代缓存层次结构

缓存层次结构通常分为包含式、NINE 式和
排他性,各有优缺点[6, 15, 26, 37, 40, 48, 55]。
在包容性缓存层次结构中,较高级别的所有行
缓存必须保留低级缓存中的行子集。在
相比之下,独占缓存层次结构中的较低级别缓存
不能包含任何更高级别的行。NINE 层次结构
位于两者之间,更高级别的线路可能会或可能不会
出现在较低层级。传统上,包容性层级
由于其所需的侦听带宽较低而被广泛采用,并且
一致性协议的低复杂度自然提供
包容性。然而,维持包容性面临两大挑战:反向失效和低有效容量。虽然一些

工作包括针对前者提出的解决方案[15, 26],

后者较少受到关注。严格包含缓存通常采用保持高层缓存较小、低层缓存较大的平凡解决方案,以限制数据重复量。

最近,缓存层级结构趋向于 NINE结构[4, 6, 25, 31, 53, 56],这种结构不存在反向失效问题,并且在容量调整方面也更加灵活。无论如何,无论是包含式缓存层级结构还是 NINE 缓存层级结构,都不可避免地会因为包含而包含冗余的缓存行,从而导致有效容量缩减。

2.2 缓存一致性

本节总结了与 XOR Cache 功能相关的传统缓存一致性的设计选择。

2.2.1 驱逐通知。所有核心共享的 LLC 通常是系统中的一致性点。因此,它需要一个目录结构来存储一致性状态,例如共享者列表和下面缓存行的所有者。目录跟踪的最关键信息是共享者列表。它是每行包含该行所在高级缓存 ID 的列表。高级缓存在读取请求时添加到共享者列表中,并在驱逐通知时从列表中删除。此共享者列表可能由于以下原因而不精确,具体取决于一致性协议和目录组织实现。首先,某些实现采用不完整的目录,例如有限指针或粗位向量。出于可伸缩性原因,这些目录仅跟踪共享者的子集[40]。其次,不精确也可能源于一致性协议。在干净驱逐时,高级缓存可能支持静默驱逐,即选择退出驱逐通知并静默删除干净行,以避免驱逐时过多的通信。请注意,这种不精确的共享者信息不会损害正确性,因为列表可能包含误报共享者,但绝不会包含误报共享者。然而,它确实会影响性能和成本,因为不必要的失效请求和确认会发送到误报共享者,也会从误报共享者那里收到[22]。在XOR缓存中,为了确保可恢复性,如果两条缓存行中至少有一条是共享的,则缓存行对可以保持压缩状态。我们称之为最小共享者不变量。因此,我们采用完整的位向量目录实现,并且我们的缓存一致性协议会在干净的驱逐时发送显式通知。

2.2.2 升级通知。某些协议在独占状态下也会选择不接收升级通知。一条唯一共享的线路在读取时可以直接提升为独占状态。当所有者稍后修改该线路时,它可以默默地将其升级为已修改状态,因为可以保证不存在其他共享者。当线路从共享状态转换为修改状态时,XOR Cache 受益于显式升级通知,因为 LLC 会被告知其副本可能已过时,不应继续进行异或操作。

2.3 低功耗缓存架构先前的研究探索了多种降低缓存功耗

的方法。降低泄漏功耗的技术包括动态关闭死块[1, 27]或缓存路[5],以及将冷块置于低功耗休眠状态[23]。其他策略则侧重于通过在阈值电压附近运行缓存[20]或采用混合单元设计[28] 来降低整体功耗。为了解决动态功耗问题,先前的研究建议使用小型过滤缓存[33]来更高效地处理频繁访问。此外,通过仅存储重用块[4]或通过缓存压缩[32]来减小缓存大小。

其中,压缩技术尤为有前景,它利用数据冗余实现压缩[3, 7–10 , 16, 18, 21, 24, 29 , 30, 35, 41–47, 49–51, 54]。压缩缓存保留了更大缓存的优势,同时显著降低了硬件成本。为了实现缓存压缩,设计采用了解耦的标签数据阵列结构。有些设计还需要额外的硬件结构来存储元数据[24, 54]。根据压缩粒度,压缩算法可分为行内压缩和行间压缩。行内压缩捕获单个内存或缓存行内的值相似性。 [3, 8, 30, 45, 54]。

这些方法通常基于字典,以子行粒度匹配预先定义的公共值或模式。跨行方法[24, 29, 41, 44, 46, 47, 49]将多个相似的行压缩在一起,并只存储该行的一个副本以及其他元数据。通常,哈希函数会根据地址或数据值选择压缩在一起的行。

3 XOR 压缩我们解释了我们提出的
XOR 压缩的工作原理,并解决了第 1.3节中提到的第一个挑战。

3.1压缩和解压缩算法我们的XOR压缩实现采用了简单且对称的压缩和解压缩算法。如1.1节所述， XOR Cache存储了行对的按位异或结果。

访问时,异或后的行会与原始的两条行中的一条执行另一次按位异或运算。因此,压缩和解压缩是完全对称的,因为与给定输入的异或是一个自逆函数。此外,压缩器和解压缩器硬件极其简单。假设有64B 缓存行,它们仅仅是一个长度为 512 的异或门阵列。由于它们是简单的按位运算,因此可以嵌入到缓存控制器中,甚至更靠近 SRAM 单元[2, 51]。实际上,任何形式的可逆计算都与这种类型的压缩兼容。在本文中,我们选择异或,因为如前所述,它简单且对称。我们只考虑双向异或,即对两条行进行异或,其他可逆函数的探索留待以后再说。

2

3.2 XOR 策略:寻找 XOR 候选者我们可以采用机会主义XOR 策略,即
只要有任何候选者（即独立线路)可用,就允许进行 XOR 压缩。

这样可以最大化 XOR 压缩的行间压缩率。或者,我们可以采用协同XOR 策略,更有选择性地执行XOR 压缩。如 1.2节所述,如果允许相似的行进行 XOR 运算,则得到的⊕行可能表现出较低的熵并包含许多零,从而实现进一步的行内压缩。我们可以通过检查图 4 中 bodytrack 基准测试中的两条行的具体示例来看到这一点。它们非常相似,只有几个位差异,即汉明距离较小。单独来看,它们的行内压缩率有限,但当作为一对进行 XOR 运算时,XOR 后的行的熵可以显著降低。

然而,关键的挑战在于识别彼此相似的线对。如图2所示,对整个 bank（即idealBank)进行穷举搜索,实现了显著的压缩率提升。

2给定一个算子 ◦及其逆算子 ◦算子 ◦是可逆的,如果我们可以 $^{-1}$, 和= ◦ $^{-1}$. 给定 A 和 C, 通过 ◦唯一地确定 B

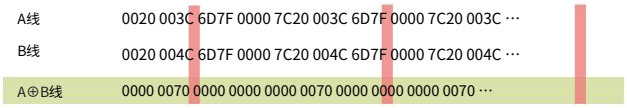


图 4:PAESEC3.0 套件中 bodytrack 基准测试中的两条相似线 A 和 B。异或后的线A⊕B具有低en-

購買。

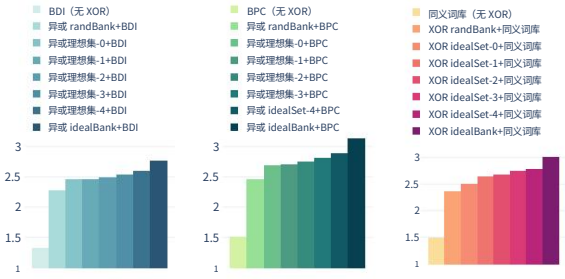


图 5: idealSet压缩率对空间值局部性影响的敏感性研究。idealSet -X中的 X表示向MSB 方向移位的索引位数。

从基线开始。然而,它过于理想化,而且硬件成本过高。idealSet 由于合适候选范围较小,协同作用较弱,因为一个集合包含的行比整个库少得多。在图 5 的敏感性研究中,我们还考虑了 idealSet 的变体,其中我们利用空间值局部性,并通过将缓存行地址中的索引位向 MSB 方向移动 1-4 位来有意创建更多相似的候选者。我们发现,移动索引位可以将集合内的穷举搜索平均提高5.47% 和11.66%,最高分别提高28.32% 和4.04 倍。结果,即 idealSet-1,2,3,4,表现出更显著的协同作用,缩小了理想压缩比和更现实压缩比之间的差距。

作为一种实际实现,我们考虑一种基于映射表的协同异或 (XOR) 策略,用于在可控的硬件复杂度下查找存储体中的相似行。该映射表与先前重复数据删除研究[24, 46, 49] 中的映射表类似,它是一种额外的间接层,可以有效地实现哈希表。映射函数应用于缓存行以生成映射值,该映射值作为缓存行值的签名,并用作映射表的索引。映射表的实现和映射函数的选择将在5.1.3节中讨论。

4 异或缓存一致性本节介绍一致性协议,以解

决1.3节中讨论的第二个挑战。虽然将异或操作后的行作为状态对来推理一致性可能更自然,但为了简洁起见,我们在此以解耦的状态转换来描述该协议。执行异或压缩（4.2节)后,异或缓存依赖其一致性协议执行解压缩以服务数据请求（4.3节）,并执行反异或操作以维护数据可恢复性（4.4节）。我们将在4.5节讨论行间依赖性及其对死锁的影响。

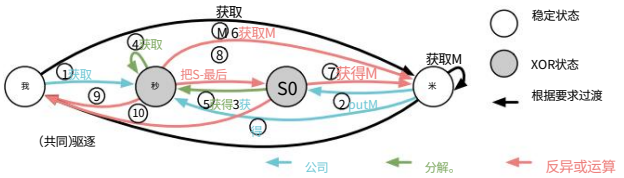


图 6:LLC 在稳定状态之间的转换。I代表无效； S代表共享； M代表修改； S0是共享者数量为零时的特殊S状态;压缩、解压缩和解异或边缘分别以蓝色、绿色和红色表示。

4.1 假设在本文中,我们基于

MSI 提出了 XOR Cache 协议,尽管这些概念可以推广到其他协议。图6展示了LLC 在稳定状态之间的转换。除了三个状态之外,

为了清晰起见,我们将状态（即已修改、已共享和无效)标记为另一个状态 Shared0 。它是Shared的特定场景,即当行在私有缓存中没有共享者时。请注意， Shared0状态与MESI 协议中的Exclusive状态根本不同,因为Shared0行如果不发送显式写入请求则没有写入权限。我们的实现假设采用混合包含式缓存层次结构,其中干净行保持包含,而脏行强制排除,因为它们的所有者位于较高级别,而较低级别不直接处理脏行上的请求。表1 阐明了这些假设,其中S行同时分配目录和 LLC 条目,而M行仅分配目录条目。当提升到M 级别时,行会重新分配其在 LLC 中的条目。S0行是那些在较高级别没有共享者的行;因此,它们只分配 LLC 条目。请注意,这不会带来更多瞬态状态,因为S0是 MSI 中已经存在的状态;我们只是为了在描述中清晰起见而将其突出显示。该目录跟踪准确的共享者信息,需要显式驱逐和升级通知。在我们的工作中,我们不假设支持静默升级,因此不考虑独占状态。在接下来的章节中,我们将通过讨论XOR缓存特有的三种转换类型来剖析该协议:压缩、解压缩和解异或,如图6中蓝色、绿色和红色突出显示的那样。

表 1:存储中的相干稳定状态和映射。

状态	无效共享已修改Shared0		
目录		✓	✓
有限责任公司		✓	✓

4.2 压缩插入行时,XOR Cache

会尝试将其与数据数组中的现有行进行异或运算。插入的行可以是:1) 请求未命中时从内存中读取的行,在图 6 中标记为 1 ;2)由于脏数据驱逐而从更高层级写入的行,即 2 ； 3)降级时写入数据更新,即3 。压缩不需要一致性改变;我们仅强调在压缩结束时可能发生一致性改变的转换。

4.3 解压缩当getS请求（图 6 中

的4 和5)到达 LLC 进行异或操作时,XOR Cache 需要对其进行解压缩,以便为

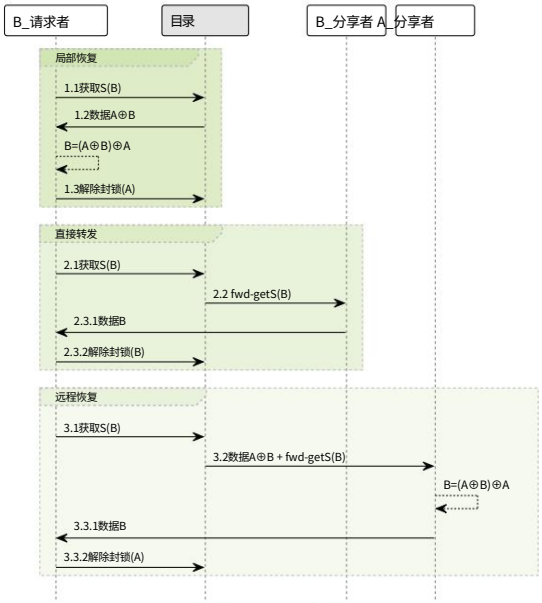


图 7:A 和 B 进行异或运算的三种转发情况。
从上到下分别是本地恢复、直接转发、远程恢复。

请求。为了进行解压缩,XOR Cache 将请求转发到更高层级。如表 2 所示,数据转发有三种情况,处理它们的序列图如图7 所示。我们继续使用与图1相同的场景,其中B 行发生未命中,即 getS(B),而 LLC 保持 A⊕B。

表 2:当请求者发出getS(B) 时的转发选择,假设 A 和 B 在 LLC 中被异或。

案件	A 状态（共享/ B 状态（共享/ B 请求者 分享0)	Shared0) 共享 A 行	
局部恢复	共享	-	✓
直接转发	-	共享	
远程恢复	共享	共享0	

在第一种情况下,A 至少有一个共享者,而 B 的请求者恰好是其中之一,LLC 会将数据A⊕B转发给B 的请求者。然后,请求者对A⊕B及其本地 A 副本执行另一个按位异或运算,以检索需求数据 B。我们将这种情况称为本地恢复。在第二种情况下， B 的请求者不共享 A,但 B 在更高级别至少有一个共享者,LLC 会将对 B 的请求转发给B 的共享者之一。然后,B 的共享者提供数据。请注意,这种情况不需要任何异或运算,而是作为常规的缓存到缓存转发来处理。我们将这种情况称为直接转发。

第三种情况是,B 的请求者同样不共享 A,并且B 不再有任何共享者。在这种情况下,A 必须至少有一个共享者,因此 LLC 会将 B 的请求连同数据A⊕B转发给 A 的共享者。A 的共享者读取其本地的 A 副本，对A⊕B和 A执行另一个按位异或运算,以远程检索B,然后将其发送回 B 的请求者。我们将这种情况称为远程恢复。

在这些情况下,向目录提供数据的私有缓存会发送解锁控制消息（1.3、2.3.2 和 3.3.2） ,告知其数据服务已完成。这些消息对于完全解锁缓存控制器的实现至关重要,无需承担任何网络排序要求。在基线阶段,从 M 状态降级和升级到M 状态时也同样需要解锁消息。对于直接转发和远程恢复,当存在多个共享者时，我们会随机选择转发者。

有两个目录到缓存的消息（图7 中的 1.2 和 3.2)包含地址 A 和 B。我们假设这些数据包额外占用 8 个字节。请注意,由于最小共享者不变量的存在,A 和 B 同时处于S0状态的情况是不可能的。此不变量通过强制执行必要的异或运算来保证,这将在下文的 4.4 节中讨论。

4.4 取消异或4.4.1 何时

取消异或。在以下三种情况下， XOR Cache 偶尔需要对一对数据进行取消异或,否则任何一对数据会进入不可恢复状态。首先,当行 B 与 A 进行异或后升级为“已修改”状态（在图6 中用6 和7表示)时,B 的写入者预计会更新其值,这会导致 LLC 副本可能过时,从而无法恢复。因此,在getM请求中,异或后的行 A⊕B 需要取消异或。其次,XOR Cache 要求,如果异或对中至少有一行具有至少一个共享者（最小共享者不变量） ,则异或后的行可以保持压缩状态。

因此,在最后一个putS请求从S转换到S0 （即8)时,需要进行非异或运算。第三,在处理从S和S0（即9 和10)驱逐时,需要进行非异或运算来恢复原始数据并选择性地写回内存,如果 1)由于标签空间不足,两条线路中只有一条正在执行驱逐;或2)由于数据空间不足,两条线路都在执行共同驱逐,并且至少其中一条是脏的。4.4.2非异或运算实现。注意,触发非异或运算的线路（比如 B)可能处于S或S0状态； 6 、 8和9 是S状态触发的,而7 和10 是S0状态触发的。这两种情况需要不同的非异或运算实现。要进行非异或运算,LLC 必须通过发出特殊的写回请求来获取更高级缓存中的一条原始线路。如果触发线路 B 处于S状态（ 6 、 8和9 ）,则写回请

求将发送到触发线路自己的共享器。它可以直接在协议中实现,因为它只涉及一个地址。否则,当触发线路 B 处于 S0 （ 7 和10)时， A 和 B 都会参与其中。配对线路 A 转换为瞬态并充当检索数据以执行非异或运算的代理。这实际上在 A 和 B 之间建立了依赖关系,因为对 B 的请求可能会触发 A 的状态转换。这种线路间依赖性对死锁自由的影响将在第 4.5 节中讨论。

4.4.3 避免不受控制的扩展。

在其他压缩缓存方案中,对某一行的更新可能会导致数据大小发生变化,从而可能由于数据阵列空间不足而引发不同数量的 LLC 驱逐,从而导致性能问题。

在XOR Cache中,由于非XOR操作,除了共同驱逐的情况外,数据膨胀也会发生。尤其是当XOR压缩与其他行内压缩方案结合使用时,数据膨胀的可能性会更大。如果牺牲品恰好是经过XOR操作的数据块,则两条行都应该执行驱逐操作,即共同驱逐。

假设 A 行与 B 行进行异或运算,触发了通过

上述操作。然后,恢复的行B和可选的行A3在行内压缩后分别尝试重新插入。

此数据扩展可能会导致另一个异或操作对 C 和 D 同时被驱逐。如果 C 和 D 中任何一个为脏数据,则执行反异或操作。但是,这保证不会导致进一步扩展,因为恢复后的C 和 D 行仅占用事务缓冲区空间,而不是实际的缓存空间。因此,异或操作后的数据块驱逐永远不会导致扩展,因此它不会导致进一步的数据驱逐,并且保证会被吸收。

4.5 无死锁为了证明XOR Cache的一

致性协议是正确且实用的,本节证明了以下两个性质:1） 4.5.1节中请求之间没有循环依赖（无协议死锁）； 2)4.5.2节中它的实现不需要任何额外的虚拟网络（无虚拟网络死锁）。

4.5.1 消除请求之间的循环依赖。如第 4.4 节所述， XOR Cache 通过 unXORing 引入了行间依赖性。
为了验证死锁自由度,我们将模型检查与分析相结合。我们利用模型检查工具 Murphi [19]验证单地址情况下的死锁自由度。为了避免组合爆炸问题,我们采用类似于[34] 的分析方法评估多地址情况下的死锁自由度。我们采用非阻塞私有缓存控制器和阻塞 LLC 控制器实现。在我们的控制器实现中,只有 LLC 绑定的请求才会被阻塞。回想一下,在 getM(B) 的情况下,异或对中的 S 状态线 A 充当其配对线 B 的代理,向 A 的共享者发送写回请求。作为对写回请求的回复， A 的共享者应发送 LLC 绑定的写回数据响应。

请注意,此响应不能被 LLC 绑定请求以外的消息阻止,因为线路 A 不能处于其他阻止瞬态4 ,因此 XOR Cache 建立的依赖关系不会导致请求之间的循环依赖。

,

4.5.2 无需额外的虚拟网络 (VN)。对于单个地址,两个虚拟网络（一个专用于 LLC 绑定请求,另一个专用于其余消息)足以保证虚拟网络不会发生死锁。由于 XOR Cache 引入了新的依赖关系,通过强制 LLC 绑定请求（例如getM)不能与此 LLC 绑定写回响应位于同一虚拟网络中,足以确保 VN 不会发生死锁。

单个地址的虚拟网络分配已经满足了这一点。因此,我们认为 XOR Cache 的一致性协议是无死锁的,更重要的是,它的实现不需要在基线上添加任何额外的虚拟网络。

综上所述,XOR Cache 的一致性协议支持在 LLC 访问异或操作的线路时进行解压缩（第4.3 节） ,并支持为维护数据可恢复性而进行反异或操作（第 4.4 节）。该协议需要增加 18.8% 的瞬态状态来实现解压缩和反异或操作。此外,还需要一对额外的消息（第 4.5 节中的写回请求和响应)以及一个转发的 getS请求（图 7 中的fwd-getS ） ,这占消息支持开销的 18.2%。

[3]如果 getM 触发了 unXORing ,则触发的行不会被重新插入,因为脏行会被强制排除。为了避免过度扩展,在最后一个putS(A)请求中,我们可以选择仅插入 B 并删除 A,而无需进行反向失效。

否则,它首先就会导致非异或运算,或者 A 不会与 B 进行异或运算

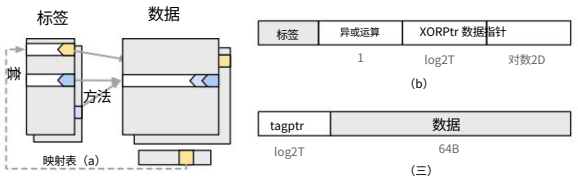


图 8:XOR Cache 组织。a) 解耦的标签数据存储和映射表;b) 标签条目;c) 数据条目;灰色块与未压缩的基线相同;T 是标签条目的数量;D 是数据条目的数量。

5 XOR缓存架构

本节概述了 XOR Cache 的存储组织以及它如何处理缓存操作。

5.1 组织为了允许一个bank
中的任意两条线进行XOR操作,XOR Cache采用了解耦的标签数据组织,并使用映射表来识别XOR候选,如图8a所示。

5.1.1 标签数组。标签数组以链表形式管理,每个标签条目如图8b 所示。XORed 是一个 1 位指示符,指示该行是否与对应行进行了异或运算。XORptr 指向其对应行的标签条目。需要注意的是,在我们的实现中,每行只能有一个对应行,因此 XOR Cache 只需要一个标签指针,而不是传统情况下的两个。我们将对非成对行的异或运算的探索留待以后研究。DataPtr 指向数据数组中的对应条目。

5.1.2 数据数组。如图8c 所示,数据数组条目存储指向标签数组条目的反向指针,即 tagptr。我们对数据数组使用随机替换策略。需要注意的是,当与其他压缩方案结合使用时,XOR Cache 需要支持可变的数据条目大小。我们使用类似于 Thesaurus 中的 8B 分段数据数组和每个集合的起始映射,将标签条目中存储的序数索引转换为实际的段 ID。为简洁起见,此处省略详细信息,详情可参见[24]。我们假设数据压缩发生在驱逐、扩展和收缩之后,类似于先前的研究。

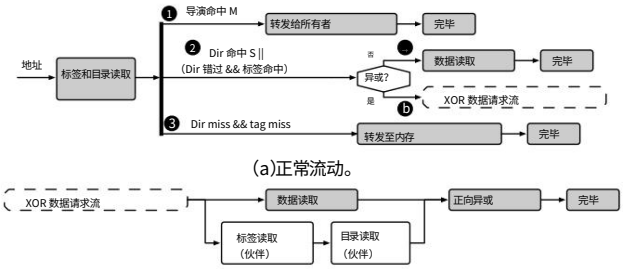
5.1.3 映射表和映射函数从高层次上讲,映射表是一个用于识别相似 XOR 候选的小型存储结构。
如图 8a 所示,映射表包含独立缓存行的标签指针。它使用映射值进行索引,该映射值是将映射函数应用于数据后生成的。在插入缓存行时,首先计算映射值并访问映射表。如果命中,即存在有效的异或候选,则触发异或压缩,并清除映射表条目。否则,该缓存行将作为未压缩的缓存行插入,映射表将为独立缓存行分配一个条目。我们将在5.2.5节详细讨论异或缓存的插入流程。映射表作为一种额外的间接层引入,类似于[24, 47, 49]。

我们将四个哈希函数视为我们的映射函数候选。
其中两个是基于随机投影（LSH-RP)的局部敏感哈希函数（类似于[24]）和位采样（LSH-BS）。另外两个基于字节标记。对于缓存行中的每个字节,如果该字节为0x0,则生成布尔标签0,否则生成1。字节标记可以有效地在字节级别捕获缓存行的稀疏性。对于基线字节标记（BL） ,生成的

(一)SPEC2017。 (b)PARSEC 3.0。

(c)完美。 (d)字节级熵尺度。

图 9:每个 8 字节字的平均字节级熵。



(a)正常流动。

(b)XOR缓存流。正向XOR指的是表2中的转发情况。

图 10:数据请求流程。关键路径以灰色表示。

字节标签首先被置换,然后被异或折叠到映射中
值。稀疏字节标记(SBL)仅考虑字节子集
在行中,即每个 8 字节字的最高有效 6 个字节。
这种稀疏版本的字节标记利用了这样一个事实:单词中低阶位表现出最高的熵[29, 44] ,如下所示
在图9 中。因此,排除低位字节理想情况下可以减少
地图值中的噪声。对于所有四个地图函数,
映射值位可以作为参数变化。我们在一个
敏感性研究见第6.2节,并讨论覆盖范围和
精度权衡与变化的地图值位数。由于
地图表每个唯一地图值只需要一个条目,我们
将映射值保持在较少的位数中,可以实现
作为直接映射结构以最小化开销。

5.2 操作

与其他压缩缓存一样,XOR Cache 执行解压缩
在处理数据请求时,这将在第 5.2.1 节中讨论。
5.2.2 和5.2.3节重新讨论了必要的异或运算情况。最后,
5.2.5节描述了XOR Cache的插入流程。

5.2.1 读取。如图10a 所示,到达的读取请求
LLC 首先在目录中执行并行查找,然后
标签数组。如果地址命中目录,则该行位于
M或S状态。在M状态1 下,读取请求执行正常
流程转发给所有者。在S状态2 ,即当我们有一个
在 S 状态下目录命中,该行可以是独立的,即a ,
或与另一条线进行异或,即b ,由 1 位异或表示
标签条目中的字段。对于情况a ,我们执行正常流程
通过从数据数组中检索数据。在后一种情况下,
标签条目有一个由 XORPtr 指向的有效伙伴。这会触发
XOR Cache的数据请求流程如图10b所示。读取
XOR 数据可以立即从 DataPtr 开始
请求行的标签条目。同时,请求还需要
访问异或运算后的伙伴的一致性元数据来决定
接下来的第4.3节将讨论三种转发情况。
首先,我们按照XORPtr访问XOR伙伴的标签条目。
然后,在目录中进行第二次查找以检索

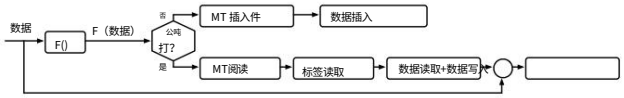


图 11:插入流程(偏离关键路径)。F() 表示
地图功能。

合作伙伴的一致性元数据。回想一下表2中
根据请求线路和合作伙伴的状态,这
读取访问在三种不同情况下处理。它可以被服务
通过发送异或后的线路,搭载在伙伴身上
已经存在于请求者中,即本地恢复。或者,
可以转发给请求线路对应的共享者
直接转发。请注意,在这种情况下,由于请求命中
目录,保证线路不处于S0状态,因此远程
不需要恢复。
如果地址在目录中丢失但在标签数组中命中,即
同样,情况2 ,线路处于S0状态。同样,它可以是独立的,也可以是
异或运算。对于异或运算后的S0行b ,也会触发第二次查找,
并且该请求要么搭载在现有的合作伙伴身上,即
本地恢复,或者转发给伙伴的共享者,即远程
恢复。最后,在 LLC 读取未命中3 时,请求被转发
按照正常操作流程写入内存。数据返回后
从记忆中,触发第 5.2.5节中的插入流程。

5.2.2 写入和升级。当对I或M线发出写入请求时,
照常转发给内存或所有者。在其他情况下,
当一个写入请求到达一个XORed行时,XOR缓存必须
对异或后的对进行反异或。这涉及一个额外的写回跳
更高级别的缓存到 LLC。由于 XOR Cache 强制排除
在脏行上,它将被从标签和数据数组中逐出,并且
插入到目录中。

5.2.3 写回。干净的写回请求,即putS很短
无数据消息,允许目录跟踪精确的共享者
列表。在异或行的最后一个写回请求中,它会查找其
合作伙伴的一致性元数据来确定是否触发了 unXORing。
否定确认作为回复发送出去,并且私人
缓存应该重试干净的写回以及干净的数据。脏
写回操作(即putM)的处理方式与插入操作类似,这将
将在 5.2.5 节中讨论。该行执行 XOR 压缩
如果存在候选人。

5.2.4 驱逐。当同时驱逐一个异或行时,反异或操作是
需要执行脏写回内存。两个标签都被驱逐
通过反向和正向指针。

5.2.5 插入。插入数据时,XOR Cache 会尝试找到
映射表中可用的候选对象。插入
可能是由于按需获取而来自内存,也可能来自私有缓存
由于写回。如图11 所示,此插入流程脱离了
关键路径。在基于映射表的实现中,映射值是
通过将 map 函数应用于返回的数据来计算。
然后使用映射值来索引映射表。在映射表上
命中,它读取标签指针,然后读取数据,按位执行
异或,并将异或后的数据插入数据数组中以替换
原始数据。相反,如果不存在候选,即在映射表上
未命中,标记指针插入到映射表中,数据插入,
并且标签已更新。对于这两行,标签数组中的 DataPtrs 指向
到异或数据条目,并且 XORPtrs 相互指向。

表3:硬件系统配置。

中央处理器	4核,3GHz x86-64
L1I	32KiB,4 路,4 循环,64B 行,LRU,私有
L1D	32KiB,4 路,4 循环,64B 行,LRU,私有
L2	256KiB,8 路,9 个周期,64B 行,LRU,私有
L3	每组 1MiB,16 路,40 周期, 64B 线,LRU,共享,4 个组
记忆	双通道DDR4-2400

表4:各银行有限责任公司的存储明细。其他指
同义词库中的基础缓存和 XOR Cache 中的映射表。

包容性		混合包容					独家的	
压缩		联合国-公司	生物量	这-斯姆	BPC	XOR Un- +BΔI补偿。		生物量
标签	#条目	16384					13312	
	入口尺寸	32b	49b	49b	49b	63b	32b	49b
	大小（KiB）	64	98	98	98	126	52	79.63
数据	#条目	16384 12288 10240 10240				6144	13312 10240	
	入口尺寸	512b				512+40b	512b	
	大小（KiB）	1024	768	640	640	414	832	640
额外的	#条目	-		512	-	128	-	
	入口尺寸	-		512b	-	14b	-	
	大小（KiB）	-		三十三	-	0.22	-	
总大小（KiB） 1088			866	770	738	540.22	884	719.63

6 评估

XOR Cache 通过 XOR 压缩实现有效压缩。通过 XOR 压缩,XOR Cache 可
以减少面积和功耗，
与未压缩缓存相当的性能。
本节的目标是评估 XOR Cache 的压缩性（第6.2和6.3节)及其在
减少面积和功耗方面的有效性
（第6.4 节）。我们最后展示其性能开销（第6.5 节）
以及能量延迟积的改进（第6.8 节）。

6.1 方法论

6.1.1 模拟器和模拟系统。我们模拟XOR缓存
使用 gem5 中的 Ruby 模型的一致性协议
模拟器[36]。我们使用 CACTI 7.0 [11]来评估面积、功耗和
内存结构的延迟并使用 Synopsys 设计编译器
采用32nm工艺的压缩器硬件合成。表3列出了
模拟硬件系统配置为3级
缓存层次结构类似于[24]。它代表一个具有高
LLC 与 MLC 的尺寸比,即 4:1,这是一种悲观的配置
由于 XOR 压缩机会有限,因此不适合 XOR Cache。
6.1.2 LLC 配置。表4列出了
LLC 用于 XOR Cache 和其他 3 级缓存层次结构
基线,除非另有规定。未压缩的基线
每家银行 LLC 为 1 MiB（表3）;压缩基线包括
行内压缩的BΔI和 BPC LLC,以及行间压缩的同义词库 LLC。同义词库
[24]动态聚类缓存
使用局部敏感哈希对行进行压缩
质心。我们的 XOR 缓存基于异或压缩与行内BΔI压缩协同作用。我们压缩数据

根据图 2 中我们分析的几何平均压缩比,所有压缩缓存的数组大小都会减小。
具体来说，
我们对BΔI使用1.3 倍小的数据阵列,对 BΔI 使用1.5 倍小的数据阵列
用于同义词库和 BPC 的数组。XOR Cache 与BΔI采用2.5×
较小的数据阵列。映射表直接映射,包含 128 个条目

表 5:SPEC CPU 2017 基准随机混合。

跑步	基准测试
1	505.mcf_r、541.leela_r、510.parest_r、503.bwaves_r
2	520.omnetpp_r、548.exchange2_r、500.perlbench_r、557.xz_r
3	511.povray_r、521.wrf_r、538.imagick_r、549.fotonik3d_r
4	502.gcc_r、544.nab_r、519.lbm_r、527.cam4_r
5	523.xalancbmk_r、525.x264_r、508.namd_r、554.roms_r
6	531.deepsjeng_r,507.cactuBSSN_r,521.wrf_r,538.imagick_r
7	525.x264_r、500.perlbench_r、549.fotonik3d_r、519.lbm_r
8	503.bwaves_r、541.leela_r、557.xz_r、508.namd_r
9	527.cam4_r、511.povray_r、507.cactuBSSN_r、505.mcf_r
10	520.omnetpp_r、554.roms_r、502.gcc_r、531.deepsjeng_r
11	523.xalancbmk_r、544.nab_r、510.parest_r、548.exchange2_r

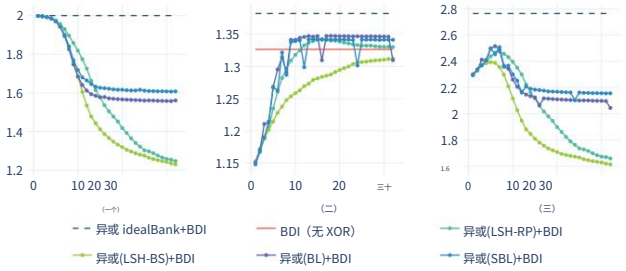


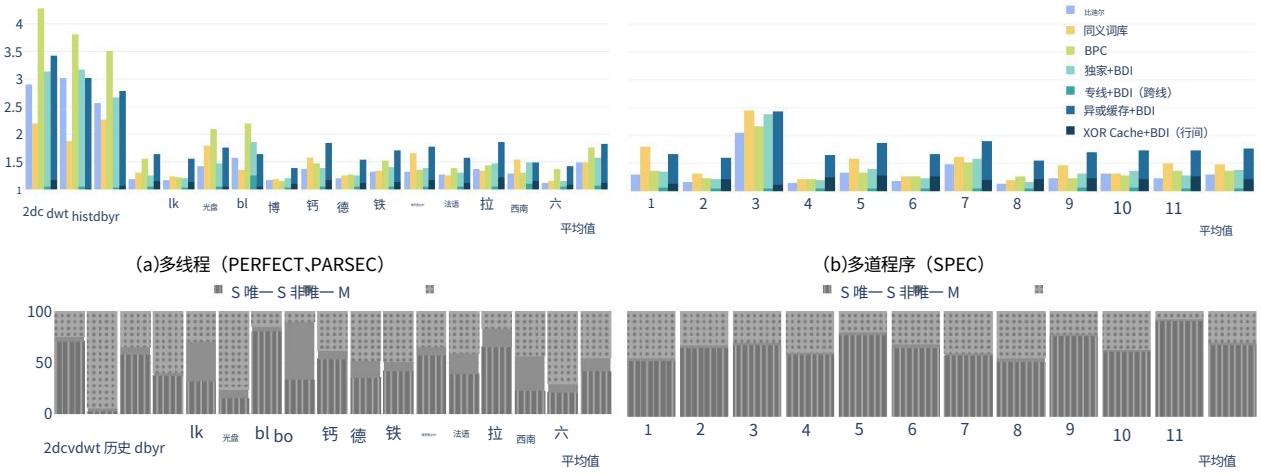
图 12:具有四个映射函数的比较比率（第5.1.3 节）。
（a)线间竞争比率；(b)线内竞争比率；(c)总计
比较比率。x轴是地图值位数。

（第5.1.3 节）。除了这些混合包容性 LLC 之外,我们还包括
具有相同有效能力的独家有限责任公司。5我们还包括
独家 LLC,其顶部应用了 BΔI。
我们悲观地假设 LLC 延迟统一为 40 个周期,
尽管数据阵列较小,延迟可能会更低。
我们假设 BΔI 的延迟为 1 个周期,同义词库的延迟为 5 个周期,位平
面解压的延迟为 7 个周期。对于 XOR Cache 的（解)压器,
合成的异或门阵列仅产生 0.12 ns 延迟,因此我们
假设执行按位异或运算与
读取。请注意,我们还将转发延迟建模为 XOR 的一部分
除上述解压缩器延迟外,还有解压缩。
6.1.3 基准测试。我们评估了 gem5 中的三个基准测试套件
全系统模拟。对于多线程工作负载,我们包括
PERFECT [12] openmp 版本,针对图像处理
工作负载,以及具有类似大数据集的 PARSEC 3.0 [13] 。对于 PERFECT和
PARSEC 3.0,我们模拟了整个感兴趣的区域。
我们纳入了 SPEC CPU 2017 [14]参考工作负载,用于多程序评估。每次运
行,我们都会启动一个随机混合
4 个 SPEC 速率基准,每个基准都有一个副本,如表5 所示。
我们快进 100B 指令并使用
遵循每个核心的 1B 详细说明。

6.2 XOR压缩协同作用

在我们基于映射表的 XOR Cache 实现中,我们使用
带有映射函数的小映射表,用于根据值相似性选择异或候选值。我们对 LLC 快
照进行分析,以
比较四个映射函数（LSH-RP、LSH-BS、BL 和 SBL）

5我们根据S0线路的比例作为基准来确定专属LLC的大小。



(a)多线程（PERFECT、PARSEC） (b)多道程序（SPEC）

(c)多线程（PERFECT、PARSEC） (d)多道程序（SPEC）

图 13:压缩比分析。

在第 5.1.3节中介绍过。图12展示了三个基准测试套件的行间和行内几何平均压缩比。X 轴表示映射值位数， Y 轴表示压缩比。此外,还包含了XOR idealBank 加BΔI和基线BΔI （不加 XOR)的压缩比以供参考。对于所有四个映射函数,我们都看到了覆盖率和准确率之间的权衡。

覆盖率。如图 12a 所示,随着映射值位数的增加,所有四个映射函数的异或压缩覆盖率均有所下降。这是因为唯一映射值的数量呈指数增长,在匹配的 bin 中找到候选值的概率降低。因此,行间异或压缩的机会减少了。

准确度。然而,如图12b 所示,随着行内压缩比的提高,识别相似候选行的准确度也会提高,这是因为异或后的行对之间的值相似度增加。这是因为使用的映射函数值位数越多,同一 bin 中出现的错误相似候选数就越少。LSH-BS 需要超过 30 位才能实现行内压缩比协同,而 LSH-RP 则需要 12位。BL 和 SBL 早在 7 位时就实现了与BΔI相似的行内压缩比;然而,SBL 保持了更高的行间压缩比,因为它有效地消除了字中高熵位的噪声。

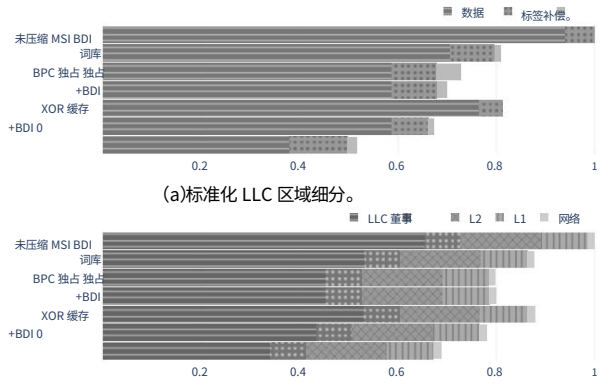
权衡。为了平衡异或压缩覆盖率和类似候选选择的准确性,我们需要选择合适的映射值位数。图12c显示,对于 BL 和 SBL,平衡行间和行内压缩比的最佳点出现在 7 位左右。考虑到我们分析结果的所有基准,在其余评估中,我们使用 7 位 SBL,其平均压缩比约为2.5 ,这证明了我们选择使用 2.5 倍小的数据阵列作为 BΔI 的异或缓存是合理的。

6.3 压缩率提升图13展示了 XOR Cache 和基准测试的压缩率分析。图13a和13b分别展示了每个基准测试的压缩率相对于未压缩的 LLC 进行了比率标准化。我们做出以下观察。首先,具有BΔI的 XOR Cache 在所有基准测试中均匀地提高了基线BΔI压缩比,证明了 XOR 压缩的有效性。其次,它实现了比具有 BΔI 的独占 LLC更高的压缩比。请注意,独占 LLC（没有BΔI)也具有大于 1 的压缩比。这是因为它不存储任何S状态行,而不像基线存储包含行。然而,这个比率很低（1.06 倍） ,因为基线层次结构中的 LLC 和私有缓存（4:1 比率)之间的冗余度已经很有限（表 3）。

对于具有 BΔI 的独占 LLC ,由于包含而不存在冗余。相反,XOR Cache 利用这种冗余实现有效压缩。第三,XOR Cache 的平均压缩率也高于 Thesaurus 和 BPC。Thesaurus 在 PERFECT 上的压缩率较低,而 BPC在利用同质数据的值相似性方面表现良好。对于多程序 SPEC 工作负载,由于字和缓存行级别的值相似性有限,两个基准的压缩率均较低。

XOR Cache 仅通过 XOR 压缩实现的行间压缩比以深蓝色阴影标记。在实际设置中,它小于 2 的上限比率。原因有三方面。首先， LLC 和私有缓存之间的冗余有限。在我们评估的配置（表3)中,私有缓存行最多只占LLC 中缓存行的四分之一,绝大多数 LLC 行处于S0状态。它们被异或的机会有限,因此限制了行间压缩比。其次,由于存在修改行,行间压缩率受到限制。请注意,即使我们的设置强制排除LLC 中的M行,它们也会与S行争夺私有缓存空间,即M行越多,私有缓存中的S行越少。回想一下,XOR Cache 利用私有缓存中的共享数据,即由于包含而产生的冗余来实现压缩率； M行的存在限制了我们实现的压缩比。

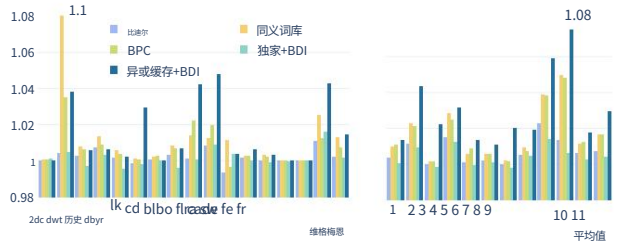
第三,工作负载可能在数据和指令方面在核心之间进行广泛的共享;这些私有缓存行映射到同一组



(a)标准化 LLC 区域细分。

(b)规范化的缓存层次结构功率细分。

图 14:标准化面积和功率细分。



(a)多线程（PERFECT、PARSEC） (b)多道程序（SPEC）

图 15:标准化性能开销。(a)显示多线程运行的标准运行时间；(b)显示多程序运行的 CPI 的标准几何平均值。

LLC 中的S条缓存线。在极端情况下,所有私有缓存可能共享完全相同的 N 条缓存线;LLC 仅有 N 条S 条缓存线,其余大多数缓存线最终为S0。同样， S0缓存线的异或机会有限,因为它们只能与S条缓存线（最小共享不变量)进行异或。S0和S条缓存线数量的不平衡会限制异或压缩性。我们通过检查图13c和13d所示的私有缓存线分布,验证了上述最后两个假设。所有有效的私有缓存线分为M 条、 S 条非唯一缓存线和S条唯一缓存线； S条非唯一缓存线是指具有多个共享者的缓存线,而S 条唯一缓存线是指专用于一个私有缓存的缓存线。基于最后两个原因,XOR压缩机会（图13a和13b 中的深蓝色区域)与S 条唯一线（底部带有垂直线的深灰色区域)的权重成正比,但黑洞（bl)除外,因为私有缓存占用空间较小,因此未得到充分利用。

例如,dwt 的压缩率较低,是因为超过90% 的私有缓存行处于M状态（顶部带有点的浅灰色区域）。
由于PERFECT 和 PARSEC 3.0 的多线程特性,它们通常具有更多共享（中间的S非唯一区域）。因此,与多程序方案相比,它们的行间压缩率较低。

要点:与通过私有缓存消除冗余的独占式 LLC 不同,XOR Cache 拥抱冗余,并通过协同 XOR 压缩将其驯服,从而提高压缩率。当两者与BΔI线内方案结合使用时,XOR Cache在多线程和多程序工作负载下的压缩率比缺乏这种协同作用的独占式基准方案高出16.2%和27.8%。此外,与线内和线间方案相比,XOR Cache 与BΔI结合使用时,在多线程工作负载下的压缩率也分别高出23.1%（BΔI）、 4.5%（BPC)和23.4% （Thesaurus） ,在多程序工作负载下的压缩率分别高出34.9% （BΔI）、 28.5% （BPC)和18.4% （Thesaurus） 。

6.4 面积和功率改进我们在图14a和14b中获得了假设
32nm 技术的标准化 LLC 面积和缓存层次结构功率细分。

6.4.1 面积。为了支持 XOR 压缩,XOR Cache 仅需要0.01 的额外面积。虽然将标签和数据阵列分离会增加元数据的边际开销,但压缩带来的数据阵列大小的减少占主导地位。Comp. 包括压缩器，

映射表和 XOR Cache 的映射函数逻辑,并包括其他基线的压缩器和存储（例如,基本缓存）。

要点： XOR Cache 的LLC 面积比未压缩的LLC 面积小1.93 倍,比BΔI、Thesaurus和 BPC 的 LLC 面积小1.56 倍、 1.41 倍和1.35 倍,比带有BΔI 的 Exclusive LLC 面积小 1.30 倍。

6.4.2 功耗。图14b展示了缓存层级结构的归一化功耗细分。除了 LLC 和私有缓存之外,我们还使用了 [52] 中的模型,计算了网络动态功耗。

使用 XOR 压缩后,由于本地和远程恢复而产生的 XOR Cache 的额外私有缓存访问仅占总私有缓存访问的1.99% 。增加的活动增加了动态功耗的开销。然而,由于私有缓存的过滤效应,泄漏功耗仍然在总 LLC 功耗贡献中占主导地位。由于额外的转发消息， XOR Cache 产生了23.4%以上的网络流量,这转化为增加的动态网络功耗。虽然这看起来很多,特别是在横向扩展系统中,但随着新兴的基于 chiplet 的系统中网络带宽的扩展趋势[17],我们预计额外的流量不会转化为显著的带宽开销。此外,这种开销仍然低于 Exclusive LLC,后者为维持严格的排他性而增加了 24.6% 的流量。

结论:尽管增加了额外的活动,与未压缩的缓存相比,XOR Cache 仍然实现了显著的1.92 倍LLC 功耗降低和1.46 倍缓存层次结构功耗降低。

6.5 性能开销为了量化 XOR Cache 的性能开销,我们在

多线程基准测试中使用了标准化的运行时间;对于多道程序基准测试,我们取每个核心在 1B 条详细模拟指令上的 CPI 几何平均值。图15显示了XOR Cache 和四个基线（表3)的性能开销,这些开销已标准化为未压缩的 MSI 基线。BΔI在关键路径上增加了固定的 1 个周期解压缩延迟;由于解压缩器复杂,BPC 和 Thesaurus 的解压缩延迟更高,并且 Thesaurus 还存在基线缓存未命中。XOR Cache 表现出缓慢的

在多线程工作负载下下降了1.45%,在多程序工作负载下下降了 2.95%。这两组工作负载之间的差异

有两个原因:1)多道程序工作负载通常压缩性较低;2)更多的 LLC 命中（15%)遵循远程恢复解压路径（第4.3 节） ,这是

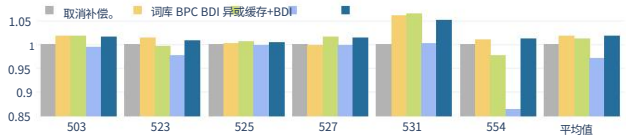


图 16:Iso 存储性能。

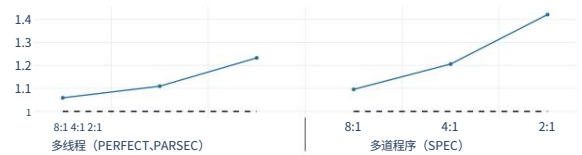


图 17:标准化行间压缩比的几何平均值。X 轴表示 LLC 与私有缓存大小之比。

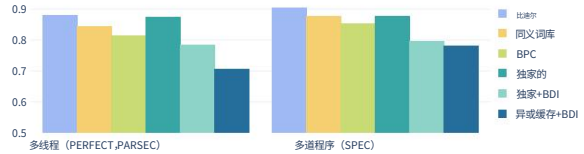


图 18:归一化能量延迟积。

在多道程序工作负载中,XOR Cache 是三者中最慢的。总体而言,XOR Cache 的性能开销为 2.06%。

6.6 案例研究:Iso-Storage 性能虽然我们的主要目标是减少 LLC 面积和功耗,但我们还是针对三个压缩基线对 XOR Cache 的 iso-storage 性能进行了案例研究。我们运行 4 核多程序工作负载,每个核心都运行一份 SPEC基准测试。图16突出显示了对 LLC 大小最敏感的工作负载子集,其中使用2× LLC 时性能差异超过 3% 。XOR Cache 比未压缩的 iso-storage LLC 平均加速1.78%,最高加速 5.22% ,高于BΔI （-2.89%）、Thesaurus （1.75%)和 BPC （1.28%）,这要归功于其更高的压缩比。在所有工作负载中,XOR Cache 的加速效果适中,为0.21%,但在所有压缩方案中仍然是最高的。

6.7 敏感性研究6.7.1 核心数量。除了 4 核结果外,我们还提供了一组额外的 8 核多线程6结果。将 XOR Cache 扩展至 8 核时,网络流量开销仅为 18.7% ,性能开销仅为 1.55%,而 4 核多线程情况下的流量开销为 18.3%,性能开销为 1.45%。

6.7.2 LLC 大小。图17展示了 LLC 大小对行间压缩率的敏感性,并进行了另外两组实验,其中LLC 大小分别翻倍和减半,分别对应 8:1 和 2:1 的 LLC与私有缓存大小比。如第 6.1.1节和第 6.3 节所述,由于 XOR Cache 利用 LLC 和私有缓存之间的冗余来执行 XOR 压缩,因此,随着 LLC 与私有缓存大小比的降低,XOR Cache在多线程和多程序设计情况下的压缩机会更多。

6.8 总结:能量延迟积图18通过测量能量延迟积 (EDP) 总结了我们的评估,并将其标准化为未压缩的 MSI 基线。

由于有效的压缩,XOR Cache在性能损失不大的情况下显著节省了功耗和面积。因此， XOR Cache 的 EDP 是所有 Cache 中最低的，比未压缩的基准低26.3% 。

7 相关工作大量研究利用行间压缩性来处理异构数据。重复数据删除[49]使用基于启发式异或折叠的哈希来识别相同的行,并仅存储唯一的行,从而消除了缓存行级别的冗余。MORC [41]提出利用基于日志的缓存的时间值局部性来压缩行。同义词库[24]使用局部敏感哈希来动态聚类缓存行,并根据质心压缩行。

几项行间压缩研究表明,字中的高位表现出低熵,而低位表现出高熵。BCD [44]建议对内存行内的低熵位进行重复数据删除。EPC [29]基于类似的观察,存储了一组低熵位的频繁模式。XOR Cache通过其 map 函数利用了这种类似的观察。Wang 等人[51]也提出了一种缓存压缩方案,通过使用 SRAM 内位线计算对多行进行异或运算。然而,它不像XOR Cache 那样以包含冗余为目标。Bunker [46]和 Doppelgänger [47]缓存分别提出利用基于地址和值的相似性。它们适用于图像处理工作负载的近似计算。Zippads [50]超越了缓存行的粒度,压缩相同类型的对象。 SC2 [9, 10]基于值统计构建霍夫曼编码,并利用字级冗余。XOR Cache作为一项新的行间压缩工作,融入了缓存压缩领域。与之前的行间压缩研究[24, 46, 49]类似， XOR Cache 也采用基于哈希表的方法来捕获行间相似性。然而,其目标是将每两行配对,以创建结构化稀疏性,从而促进行内压缩。

8 结论在本文中,我们介绍了 XOR Cache,它利用由于包含和私有缓存而产生的数据冗余来同时执行有效的行间和行内压缩。与传统缓存不同,XOR Cache 存储行对的按位 XOR 结果,从而有效地将一对行配置在单个槽中。当与其他压缩方案结合使用时,XOR Cache 可以通过利用 XOR 数据值的可压缩性来进一步提高压缩率。我们解决了 XOR Cache 中的两个关键挑战:基于值相似性设计 XOR 策略和实现缓存一致性协议。评估结果表明,由于XOR Cache 的有效压缩,它将 LLC 面积减少了1.93倍,功耗降低了1.92 倍,性能（平均开销为2.06%)与更大的未压缩缓存相当,同时能量延迟积降低了 26.3%。

致谢:感谢所有审稿专家的宝贵反馈。本研究由威斯康星校友研究基金会和美国国家科学基金会资助,资助编号:CNS-2045985。

6由于内存有限,大多数 8 核多程序 SPEC 运行无法完成。

参考

[1] Jaume Abella、Antonio González、Xavier Vera 和 Michael FP O Boyle。 2005年。IATAC:一种用于关闭二级缓存行的智能预测器。ACM 架构代码优化汇刊 2, 1（2005 年 3 月）,55–77。doi :10.1145/1061267.1061271 [2] Shaizeen Aga、Supreet Jeloka、Arun Subramaniyan、Satish Narayanasamy、David Blaauw 和 Reetuparna Das。 2017。计算缓存。在高性能计算机体系结构国际研讨会上。

[3] Alaa R. Alameldeen 和 David A. Wood。2004 年。频繁模式压缩:一种基于重要性的 L2 缓存压缩方案。技术报告。威斯康星大学麦迪逊分校,计算机科学系。

[4]豪尔赫·阿尔贝里西奥、巴勃罗·伊巴涅斯、维克多·维纳尔斯和何塞·M·拉贝里亚。 2013.重用缓存:缩小共享末级缓存的大小。在国际微架构研讨会上。

[5] DH Albonesi. 1999. 选择性缓存方式:按需缓存资源分配。在MICRO-32中。第32届ACM/IEEE国际微架构研讨会论文集。248–259页。doi :10.1109/MICRO.1999.809463

[6] Mohammad Alian、Siddharth Agarwal、Jongmin Shin、Neel Patel、Yifan Yuan Yuan、Daehoon Kim 和 Ren Wang。2022 年。IDIO:基于服务器处理器的网络驱动入站网络数据编排。国际微架构研讨会论文集。

[7] Alexandra Angerd、Angelos Arelakis、Vasilis Spiliopoulos、Erik Sintorn 和Per Stenström。2022 年。GBDI:利用全局基超越基-增量-立即压缩。2022 年 IEEE 高性能计算机架构 (HPCA) 国际研讨会。1115–1127。doi :10.1109/HPCA53966.2022.00085 [8] Angelos Arelakis、Fredrik Dahlgren 和 Per Stenstrom。2015 年。HyComp:一种用于选择特定数据类型压缩方法的混合缓存压缩方法。载于第 48 届国际微架构研讨会（夏威夷威基基）（MICRO-48）论文集。美国计算机协会,纽约,纽约州,第 38-49 页。doi :10.1145/2830772.2830823

[9] Angelos Arelakis 和 Per Stenstrom。2014 年。SC2:一种统计压缩缓存方案。载于第 41 届计算机架构国际研讨会（ISCA）（美国明尼苏达州明尼阿波利斯市）（ISCA 14)论文集。IEEE 出版社,第 145–156 页。

[10] Angelos Arelakis 和 Per Stenstrom。2014 年。SC2:一种统计压缩缓存方案。SIGARCH 计算建筑新闻 42, 3（2014 年 6 月）,145–156。doi :10.1145/2678373.2665696

[11] Rajeev Balasubramonian、Andrew B. Kahng、Naveen Muralimanohar、Ali Shafiee和 Vaishnav Srinivas。2017 年。CACTI 7:创新片外存储器互连探索新工具。ACM 架构代码优化汇刊 14,2,第 14 篇文章（2017 年 6 月）,共 25 页。doi :10.1145/3085572

[12] Kevin Barker、Thomas Benson、Dan Campbell、David Ediger、Roberto Gioiosa、 Adolfy Hoisie、Darren Kerbyson、Joseph Manzano、Andres Marquez、Leon Song、 Nathan Tallent 和 Antonino Tumeo。2013 年。PERFECT（嵌入式计算技术的能效革命)基准套件手册。太平洋西北国家实验室和佐治亚理工学院研究机构。http : //hpc.pnnl.gov/projects/PERFECT/。

[13] Christian Bienia、Sanjeev Kumar、Jaswinder Pal Singh 和 Kai Li。2008 年。PARSEC 基准套件:特性描述和架构含义。国际并行架构与编译技术会议论文集。

[14] James Bucek、Klaus-Dieter Lange 和 Jóakim v. Kistowski。2018 年。SPEC CPU2017:下一代计算基准。2018 年 ACM/SPEC国际性能工程会议（德国柏林）(ICPE 18)配套论文。美国计算机协会,纽约,纽约州,美国,41-42。doi :10.1145/3185768.3185771

[15] Mainak Chaudhuri,2021 年,《零包含受害者:将核心缓存与包含性末级缓存驱逐隔离》。2021 年 ACM/IEEE 第 48 届计算机体系结构国际研讨会（ISCA）。

[16] Xi Chen, Lei Yang, Robert P. Dick, Li Shang, and Haris Lekatsas. 2010. C-pack:一种高性能微处理器缓存压缩算法. IEEE Trans. 超大规模积分系统。18,8（2010年8月）,1196-1208。doi :10.1109/TVLSI.2009。2020989

[17] Grigory Chirkov 和 David Wentzlaff。2023 年。 《后摩尔定律时代,如何实现封装内互连的带宽扩展》 。第 37 届国际超级计算大会（美国佛罗里达州奥兰多)论文集（ICS 23）。美国纽约州纽约市计算机协会,410-422。

[18] Esha Choukse、Mattan Erez 和 Alaa R. Alameldeen。2018 年。Compresso:实用主内存压缩。2018 年第 51 届 IEEE/ACM 国际微架构研讨会 (MICRO)。546–558。doi :10.1109/MICRO.2018.00051 [19] David L. Dill. 1996. Mur 验证系统。载于《计算机辅助验证》， Rajeev Alur 和 Thomas A. Henzinger 编。Springer Berlin Heidelberg,柏林,海德堡,390–393 页。

[20] Ronald G. Dreslinski、Gregory K. Chen、Trevor Mudge、David Blaauw、Dennis Sylvester 和 Krisztian Flautner。2008 年。可重构节能近阈值缓存架构。2008 年第 41 届 IEEE/ACM 国际微架构研讨会。第 459–470 页。doi :10.1109/MICRO.2008.4771813 [21] Albin Eldstål-Ahrens、Angelos Arelakis 和 Ioannis Sourdis。2023 年。FlatPack:压缩内存的灵活压缩。《并行架构与编译技术国际会议论文集》（伊利诺伊州芝加哥）（PACT 22）。美国计算机协会,纽约,美国,96-108。doi :10.1145/3559009.3569653

[22] Ricardo Fernández-Pascual、Alberto Ros 和 Manuel E. Acacio。2017 年。《保持沉默还是保持活力:论缓存一致性多核中干净数据驱逐的影响》。J. Supercomput. 73, 10 (2017 年 10 月), 4428–4443。

[23] Krisztián Flautner、Nam Sung Kim、Steve Martin、David Blaauw 和 Trevor Mudge。2002 年。《休眠缓存:降低泄漏功耗的简单技术》。载于第 29 届计算机体系结构国际研讨会论文集（阿拉斯加州安克雷奇）(ISCA 02)。IEEE 计算机学会,美国,第 148-157 页。

[24] Amin Ghasemazar、Prashant Nair 和 Mieszko Lis。2020 年。同义词库:通过动态聚类实现高效缓存压缩。刊于第二十五届编程语言和操作系统架构支持国际会议（瑞士洛桑）(ASPLOS 20)论文集。美国计算机协会,纽约,第 527–540 页。doi :10.1145/3373376.3378518 [25] Gorka Irazoqui、Thomas Eisenbarth 和 Berk Sunar。2016 年。跨处理器缓存攻击。载于第 11 届 ACM 亚洲计算机与通信安全会议（中国西安）(ASIA CCS 16)论文集。美国计算机协会,纽约,纽约州,美国,第 353-364 页。doi :10.1145/2897845.2897867 [26] Aamer Jaleel、Eric Borch、Malini Bhandaru、Simon C. Steely Jr. 和 Joel Emer。 2010年。利用包容性缓存实现非包容性缓存性能:时间局部感知 (TLA) 缓存管理策略。国际微架构研讨会。

[27] S. Kaxiras、Zhigang Hu 和 M. Martonosi。2001 年。缓存衰减:利用代际行为降低缓存泄漏功耗。载于《第 28 届计算机体系结构国际研讨会论文集》,240–251。doi :10.1109/ISCA。 2001.937453

[28] Samira M. Khan、Alaa R. Alameldeen、Chris Wilkerson、Jaydeep Kulkarni 和Daniel A. Jiménez。2013 年。《利用混合单元缓存架构提升多核性能》。2013 年 IEEE 第 19 届高性能计算机架构 (HPCA) 国际研讨会。第 119–130 页。doi :10.1109/HPCA.2013.6522312 [29] Jinkwon Kim、Mincheol Kang、Jeongkyu Hong 和 Soontae Kim。2022 年。利用块间熵增强具有多样化数据的块的可压缩性。2022年IEEE高性能计算机体系结构（HPCA)国际研讨会。1100–1114。doi :10.1109/HPCA53966.2022.00084 [30] Jungrae Kim、Michael Sullivan、Esha Choukse 和 Mattan Erez。2016 年。位平面压缩:在众核架构中实现更佳压缩的数据转换。2016 年 ACM/IEEE 第 43 届计算机体系结构国际研讨会（ISCA）。329–340。doi :10.1109/ISCA.2016.37

[31] Sowoong Kim、Myeonggyun Han 和 Woongki Baek。2022 年。DPrime+DAbort:基于 Intel TSX 的非包含缓存层次结构中的高精度、无计时器目录侧通道攻击。2022 年 IEEE高性能计算机架构 (HPCA) 国际研讨会。67–81。doi :10.1109/ HPCA53966.2022.00014

[32] Soontae Kim、Jongmin Lee、Jesung Kim 和 Seokin Hong。2011 年。残差缓存:基于压缩和部分命中的低能耗低面积二级缓存架构。载于第 44 届 IEEE/ACM 国际微架构研讨会（巴西阿雷格里港）(MICRO-44)论文集。美国计算机协会,纽约,第 420-429 页。doi :10.1145/2155620.2155670 [33] J. Kin、Munish Gupta 和 WH Mangione-Smith。1997 年。过滤器缓存:一种节能的内存结构。载于《第 30 届国际微架构研讨会论文集》 ,第 184–193 页。doi :10.1109/MICRO.1997.645809 [34]李伟航、Andrés Goens、Nicolai Oswald、Vijay Nagarajan 和 Daniel J. Sorin。2024. 确定不同一致性协议所需的最小虚拟网络数量。2024年ACM/IEEE第51届计算机体系结构国际研讨会（ISCA）。

[35] Y. Li 和 M. Gao。2023 年。《重子:基于压缩和子块技术的高效混合内存管理》。2023 年 IEEE 高性能计算机体系结构（HPCA)国际研讨会论文集。IEEE 计算机学会,美国加利福尼亚州洛斯阿拉米托斯,第 137-151 页。doi :10.1109/HPCA56546.2023.10071115 [36] Jason Lowe-Power、Abdul Mutaal Ahmad、Ayaz Akram、Mohammad Alian、Rico Amslinger、Matteo Andreozzi、Adrià Armejach、Nils Asmussen、Brad Beckmann、 Srikant Bharadwaj 等。 2020 年。gem5 模拟器:版本 20.0+。 arXiv 预印本arXiv:2007.03152 (2020)。

[37] Milo MK Martin、Mark D. Hill 和 Daniel J. Sorin。2012 年。片上缓存一致性为何将持续发展。ACM 通讯 55,7（2012 年 7 月）,78–89。doi :10.1145/2209249.2209269

[38] Hassan Mujtaba. 2020. AMD Ryzen 5000 Zen 3 “Vermeer” 裸机图,首发高清芯片特写图及细节图。https : //wccftech.com/amd-ryzen-5000-zen-3-vermeer-undressed-high-res-die-shots-close-ups-pictured-detailed/

[39] Ashley O. Munch、Nevine Nassif、Carleton L. Molnar、Jason Crop、Rich Gammack、Chinmay P. Joshi、Goran Zelic、Kambiz Munshi、Min Huang、Charles R. Morganti、 Sireesha Kandula 和 Arijit Biswas。 2024 年。2.3 Emerald Rapids:第五代英特尔® 至强® 可扩展处理器。 2024 年 IEEE 国际固态电路会议 (ISSCC),卷。 67. 40–42。 doi:10.1109/ISSCC49657.2024.10454434 [40] Vijay Nagarajan、Daniel J. Sorin、Mark D. Hill 和 David A. Wood。2020 年。《内存一致性和缓存一致性入门》第二版。Springer Cham。

[41] Tri M. Nguyen 和 David Wentzlaff。2015 年。MORC:面向众核的压缩缓存。载于第 48 届国际微架构研讨会（夏威夷威基基）（MICRO-48)论文集。美国计算机协会,纽约,纽约州,美国,第 76-88 页。doi :10.1145/2830772.2830828

[42] Biswabandan Panda 和 André Seznec。2016 年。字典共享:一种高效的压缩缓存压缩方案。2016 年第 49 届 IEEE/ACM 国际微架构研讨会（MICRO）。1-12。doi :10.1109/MICRO。

2016.7783704

[43] Gagandeep Panwar、Muhammad Laghari、Esha Choukse 和迅健。 2024 年。
DyLeCT:实现类似大页面的硬件压缩内存翻译性能。2024 年 ACM/IEEE 第 51 届计算机体系结构国际研讨会（ISCA）。

[44] Sungbo Park、Ingab Kang、Yaebin Moon、Jung Ho Ahn 和 G. Edward Suh。
2021. BCD 重复数据删除:使用部分缓存行重复数据删除实现有效的内存压缩。发表于第 26 届 ACM编程语言和操作系统架构支持国际会议（美国虚拟会议）（ASPLOS 21)论文集。美国计算机协会,纽约,纽约州,第 52-64 页。doi :10.1145/3445814.3446722

[45]根纳迪·佩希缅科 (Gennady Pekhimenko)、维韦克·塞沙德里 (Vivek Seshadri)、奥努尔·穆特鲁 (Onur Mutlu)、菲利普·B·吉本斯 (Phillip B. Gibbons)、迈克尔·A.
Kozuch 和 Todd C. Mowry。2012 年。Base-Delta-immediate 压缩:片上缓存的实用数据压缩。刊于第 21 届并行架构与编译技术国际会议论文集（美国明尼苏达州明尼阿波利斯）（PACT 12）。美国计算机协会,纽约州纽约市,第 377–388 页。doi :10.1145/2370816.2370870

[46] Joshua San Miguel、Jorge Albericio、Natalie Enright Jerger 和 Aamer Jaleel。
2016. 用于空间值近似的 Bunker Cache。2016 年第 49 届IEEE/ACM 国际微架构研讨会（MICRO）。1-12。doi :10。

1109/MICRO.2016.7783746

[47] Joshua San Miguel、Jorge Albericio、Andreas Moshovos 和 Natalie Enright Jerger。2015 年。Doppelgänger:用于近似计算的缓存。2015 年第 48 届IEEE/ACM 国际微架构研讨会（MICRO）。第 50–61 页。doi :10.1145/2830772.2830790

[48] Yingying Tian、Samira M. Khan 和 Daniel A. Jiménez。2013 年。基于时间的多级关联包容性缓存替换。ACM 架构代码汇刊
Optim. 10, 4, 第33条 (2013年12月), 24页。doi :10.1145/2541228.2555290 [49] Yingying Tian、Samira M. Khan、Daniel A. Jiménez 和 Gabriel H. Loh。2014 年。末级缓存重复数据删除。载于第 28 届 ACM 国际超级计算会议（德国慕尼黑)ICS 14) 论文集。美国计算机协会,纽约,美国,第 53-62 页。doi :10.1145/2597652.2597655 [50] Po-An Tsai 和 Daniel Sanchez。2019 年。“压缩对象而非缓存行:基于对象的压缩内存层次结构”。国际编程语言和操作系统架构支持会议。

[51] Xiaowei Wang、Charles Augustine、Eriko Nurvitadhi、Ravi Iyer、Li Zhao 和Reetuparna Das。2021 年。具有高效 SRAM 内数据比较的缓存压缩。2021 年 IEEE 网络、架构和存储 (NAS) 国际会议。1–8。doi :10.1109/NAS51552.2021.9605440

[52] PT Wolkotte、GJM Smit、N. Kavaldjiev、JE Becker 和 J. Becker。2005 年。片上网络和总线的能量模型。2005 年国际片上系统研讨会论文集,第 82–85 页。doi :10.1109/ISSOC.2005.1595650

[53] Mengjia Yan、Read Sprabery、Bhargava Gopireddy、Christopher Fletcher、Roy Campbell 和 Josep Torrellas。2019 年。攻击目录而非缓存:非包容性世界中的侧信道攻击。2019 年 IEEE 安全与隐私研讨会(SP)。第 888–904 页。doi :10.1109/SP.2019.00004

[54]杨俊、张友涛和 Rajiv Gupta。2000 年。数据缓存中的频繁值压缩。载于第 33 届 ACM/IEEE 国际微架构研讨会论文集（美国加利福尼亚州蒙特雷）（MICRO 33）。美国计算机协会,纽约,第 258-265 页。doi :10.1145/360128.360154 [55] Li Zhao、Ravi Iyer、Srihari Makineni、Don Newell 和 Liqun Cheng。2010 年。NCID:一种用于灵活高效缓存层次结构的非包含缓存、包含目录架构。载于《第七届 ACM 国际计算前沿会议论文集》。

[56]赵子睿、亚当·莫里森、克里斯托弗·W·弗莱彻和何塞普·托雷拉斯。
2024. 末级缓存侧通道攻击在现代公有云中切实可行。发表于第29届ACM编程语言和操作系统架构支持国际会议论文集第2卷（美国加州拉霍亚）（ASPLOS 24）。美国计算机协会,纽约,第582-600页。doi :10.1145/3620665.3640403