多核处理器中缓存划分技术综述

橡树岭国家实验室

随着片上内核数量和应用程序内存需求的增加，明智地管理缓存资源不仅具有吸引力，而且势在必行。缓存分区，即根据应用程序的内存需求在应用程序之间划分缓存空间，是一种很有前途的方法，可以提供共享缓存的容量优势和私有缓存的性能隔离。然而，天真地划分缓存可能会导致性能损失、不公平和缺乏服务质量保证。很明显，要实现缓存分区的全部潜力，需要智能技术。在本文中，我们介绍了在多核处理器中划分共享缓存的技术。我们根据重要特征对技术进行分类，并提供缓存分区领域的鸟瞰图。

类别和主题描述:a . 1[普通文献]:介绍性和综述；计算机系统组织:系统架构

通用术语:设计、算法、性能

附加关键词和短语:回顾、分类、多核处理器、共享缓存、分区、公平性、服务质量

自动呼叫管理参考格式:
斯巴塞米塔尔。2017.多核处理器中高速缓存划分技术综述。ACM Comput。生存。50，2，第27条(2017年5月)，39页。
DOI:http://dx.doi.org/10.1145/3062394

1. 介绍

处理器设计的最新趋势使得缓存资源的有效管理比以往任何时候都更加重要。随着应用程序核心数量和内存需求的增加，谨慎的缓存体系结构和管理策略对于应对面积和功耗预算停滞不前的挑战变得至关重要。研究人员最近探索了私有和共享缓存设计，这些设计具有明显的优势和劣势。虽然私有缓存避免了干扰，但它们不能解决应用程序之间和应用程序内部的缓存需求变化，并且由于容量有限，它们不能有效降低未命中率。相比之下，共享缓存可以提供更高的总容量来降低未命中率；但是，使用传统的管理策略，它们可能会由于应用程序之间的干扰而表现出性能损失、不公平和服务质量(QoS)不足[Yun和Valsan2015; Herdrich等人2016].

作者在ORNL和IIT海得拉巴工作时都写了这篇文章。
作者地址:S. Mittal，E-621，印度技术学院(IIT)计算机科学与工程系，海德拉巴，Sangareddy，Telangana 502285，印度；电子邮件:sparsh0mittal@gmail.com.

DOI:http://dx.doi.org/10.1145/3062394

缓存分区[1]有望通过提供私有缓存的性能隔离和共享缓存的容量优势来解决这一难题。众所周知，不仅不同的应用程序，甚至多线程应用程序的不同线程都可能表现出完全不同的缓存需求和性能敏感度[Muralidhara 等人。2010]. 除了正在运行的应用程序之外，内核之间的性能差异也可能由于处理器设计本身而产生，例如 NUCA 设计导致的缓存延迟差异和光伏导致的内核频率差异[Kozhikkottu 等人。2014]. CP 可以有效地补偿内核之间的这些性能差异。除了吞吐量之外，CP 还可以针对公平性和 QoS 目标进行优化，这在整合服务器和云计算等共享计算平台中尤为重要。通过避免干扰，CP 可以为每个应用程序提供更高的有效缓存容量，从而实现更大规模的性能(例如 2X[Jaliel 等人。2008] 或 1.5X [Nikas 等人。2008]) 缓存。CP 可以减少片外未命中和带宽争用，这甚至可能有利于那些缓存配额减少的应用[Jin 等人。2009; 潘与派 2013]. 此外，通过减少执行时间并允许对未使用的缓存进行电源门控，CP 可以提高能效。很明显，CP 是一种通用且强大的管理方法，适用于广泛的使用场景。

虽然有前途，但 CP 不是万能的。随着内核数量的增加，可能的分区数量呈指数级增长，这使得简单的方案(例如强力搜索)变得无效。事实上，找到具有最小总体高速缓存未命中率的分区(即最优分区)的问题是 NP 难的[Yu 和 Petrov2010;斯通等人。1992] 然而，最优分区可能并不公平[Brock 等人。2015]. 简单的处理器性能测试可能会导致大量的分析和重新配置开销，并且实现处理器性能测试所需的硬件支持(例如，真 LRU)可能过于昂贵，或者在大多数处理器上不可用。很明显，智能设计方法和参数选择对于充分发挥阴极保护的潜力至关重要。最近提出的几项国家方案建议寻求应对这些挑战。

贡献:在本文中，我们介绍了在多核处理器中划分共享缓存的技术。数字 1 介绍了文章的概述。我们首先介绍 CP 的背景，并讨论其中涉及的权衡(第节 2).我们还讨论了 CPT 在真实处理器中的使用。然后，我们从几个角度对研究工作进行分类，以提供见解(第 3).然后，我们从粒度的角度来回顾 CPT(第节 4)，用于分区的策略(部分 5)，以及他们寻求优化的目标(第节 6).此外，我们总结了在不同环境中使用 CPTs 的作品(第 7)以及将清洁生产与其他管理方法相结合(第 8).最后，我们展望了未来的挑战(第 9).

范围:为了做一个全面而有重点的介绍，我们将本文的范围限制如下。而高速缓存可以在不同特性的块之间划分(例如，读/写密集型[Khan 等人。2014]，频繁/不频繁的写回等。)，在本文中，我们只包括在多核处理器中的不同应用程序或线程之间划分共享缓存的技术。我们回顾多核处理器中的 CP，而不是中央处理器-图形处理器系统[李和金 2012; Mekkat 等人

[1] 本文中经常使用以下首字母缩略词:“辅助标签目录”(ATD)、带宽(BW)、“双峰插入策略”(BIP)、“高速缓存分配标记”(CAT)、高速缓存分区(CP)技术(CPT)、“每指令周期”(CPI)、“动态插入策略”(DIP)、“动态电压/频率缩放”(DVFS)、公平加速(FS)、插入点(IP)、“每周期指令”(IPC)、“最后一级高速缓存”(LLC)、“最近最少使用的”(LRU)、“最低有效位”(LSB)、“存储器级并行性”(MLP)、“未命中状态保持寄存器”(MSHR)、“未命中状态保持寄存器” “相变存储器”(PCM)、过程变量(PV)、“比例积分微分”(PID)、保护距离(PD)、替换策略(RP)、重用距离(RD)、比例因子(SF)、虚拟存储器(VM)监视器(VMM)、加权加速比(WS)。 WC 也指缓存关联性。Following acronyms are used frequently in this article: “auxiliary tag directory” (ATD), bandwidth (BW),

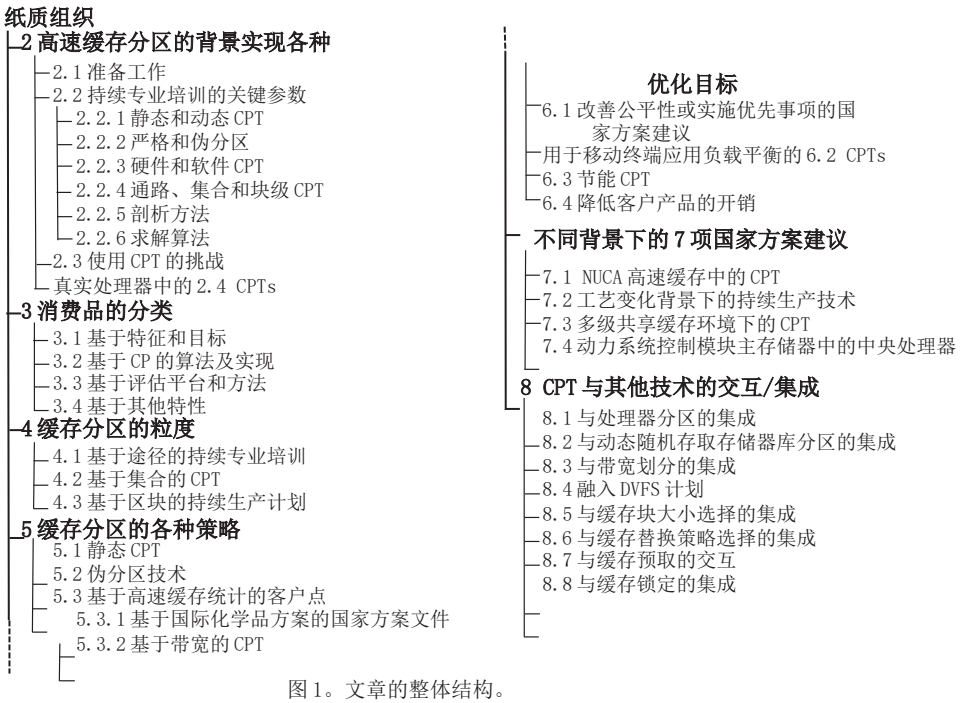

图 1。文章的整体结构。

2013]. 由于不同的项目使用不同的评估方法，我们专注于它们的定性见解，并且只包括选定的数字结果。本文有望对缓存设计和管理领域的研究人员和实践者有所帮助。

2. 高速缓存分区的背景

2.1. 预赛

我们现在讨论一些在整篇文章中都有用的概念/术语。

配额分配和实施策略:CPT 由用于确定每个应用程序配额的“配额分配”策略和实际实施这些配额的“配额实施”策略组成。例如，分配策略可以根据两个应用程序的缓存敏感度，将它们的配额分别确定为缓存的 75%和 25%。现在，对于具有 64 种颜色和 16 条路的缓存，基于颜色的策略可以通过分别提供 48 种和 16 种颜色来实施这些配额，而基于路的策略可以通过分别向这些应用程序提供 12 条和 4 条路来实施这些配额。

受害者选择、插入和推广策略:一个 RP 可以分解为以下三个策略。当需要在高速缓存中插入新的块时，“牺牲者选择策略”决定要驱逐的块。“插入策略”决定插入新块的优先位置。对于看到缓存命中的块，“提升策略”决定其在优先级链中的位置如何更新。例如，对于 LRU RP，最近最少的块被逐出，新的块被插入最近的位置，并且看到命中的块被提升到最近的位置。

栈属性:RPs 服从栈属性[Mattson 等人。1970]，如果高速缓存有多个 WC 路(对于相同的集合计数)，命中 WC 路高速缓存的访问也将命中。

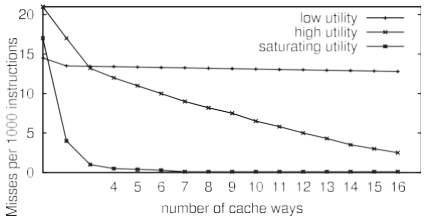

图 2。不同效用曲线的图示(缓存配额下未命中的变化)。

集合抽样方案:集合关联高速缓存的几个关键特征(例如，未命中率)可以通过只对其几个集合进行抽样来估计，这指的是集合抽样。它允许降低分析开销。

重用距离:RD 指对一个缓存块的两次访问之间不同的访问次数，例如在访问流 M F G K E J H N E K J F G H 中，G 的 RD 为 6 [Mittal2016b]. 可以对整个高速缓存或每个集合(盘和排)计算不同的访问 2013; Duong 等人 2012], 分别称为全局或集合特定的 RDs。

基于缓存行为的应用程序分类:基于它们的缓存行为，应用程序可以分为多个类别。没有或确实受益于缓存的应用程序分别被称为缓存“不敏感”和“友好”。“流应用程序”具有非常大的工作集，并且由于缓存重用不足，它们显示出与任何 RP 的冲突。“抖动应用程序”的工作集大于缓存容量，因此，它们抖动 LRU 管理的缓存，尽管它们可能会受益于使用抗抖动 RPs 的缓存。类似地，基于通过增加缓存配额实现的未命中率或执行时间的降低率，应用程序也可以被描述为具有低、高或饱和效用，如图所示 2.

2.2. 氯化聚乙烯的关键参数

CPT 可以根据关键参数进行表征。我们现在讨论其中的一些。

2.2.1. 静态和动态 CPT。静态和动态 CPT 分别在离线和运行时做出配额分配决定(参见表 I).

2.2.2. 严格分区和伪分区。根据为应用程序确定的缓存配额是否严格执行，CPT 可以分别分为严格(硬)分区和伪(软)分区(参见表 I).

2.2.3. 硬件和软件 CPT。CPTs 可以在硬件或软件中实现(参见表 I).

2.2.4. 路、集和块级 CPT。根据分配缓存的粒度，CPT 可以分为路级、集级(或颜色)和块级(参见表 II). 在物理地址中，集合索引和物理页码之间的重叠位称为页面颜色。基于集合的 CPT 通过控制这些位来改变分配给核心的颜色(以及集合)的数量来工作[Gui 等人。2014; 林等。2009]. 基于路、集和块的 CPT 提供越来越细粒度的分配，例如，块大小为 64B、系统页面大小为 4KB 的 16 路 4MB 高速缓存有 16 路、64 种颜色和 65536 个块。还要注意，任何 CPT 都需要为每个缓存块提供缓存映射，以避免绕过它，因为绕过会使缓存一致性变得复杂，并且需要特殊的管理技术[Mittal2016b]. 粗粒度分配可能不会强制执行精确的分区大小，而细粒度分配可能会仅对大量内核有益。

表一.基于静态/动态、严格/伪和硬件/软件的CPT的属性/挑战

静态	(+)对于小的核数，静态CPT可能有助于评估所有可能的分区，并找到从CP获得收益的上限。 ()随着内核数量的增加，应用程序的可能组合呈指数级增长，这使得使用静态CPT变得不可行。静态CPT不能适应缓存行为的时间变化。
动态的	(+)只有动态CPT适用于大核数。 ()动态CPT会导致运行时开销，如果应用程序行为随时间变化是一致的，那么它们可能是不必要的。
严格的	(+)严格的分区对于保证服务质量和公平性非常重要。 ()这可能会导致缓存利用率低下，尤其是当分配粒度较大时(例如，基于路的CPT)。例如，如果核心不发送对集合的访问，则路 分配给该集合中的核心的数据仍未使用。此外，一个内核的死块不能被其他内核驱逐，即使这些内核可以通过这样做获得额外的命中。
假的	(+)伪分区可能具有更简单的实现，并且可能允许核心窃取其他核心的配额以提高性能。 ()但是，由于这个原因，应用程序的瞬时配额可能与目标相差很大(例如，相差八分之二[Halwe等人。2013]). 2011].
基于硬件	(+)硬件管理对于减少分析和重新配置开销非常重要。硬件CPT可以在细粒度上使用，例如，几十万个周期。 ()增加其运行所需的硬件支持可能会带来挑战。
基于软件的	(+)软件控制对于考虑其他处理器组件、管理方案和系统级目标(如优化公平性)非常重要(相对于缓存级目标(如最小化未命中率))。 ()软件CPT(例如，涉及页面着色的CPT)通常会导致更高的开销，因此只能以粗略的粒度调用。

表二。基于路/集/块的并行处理技术的特性和挑战

CPT	属性/挑战
基于方式	(+)基于路的CP提供了相对简单的实现、免刷新的重新配置以及获取路级剖析信息的便利性。由于这些原因，以及大多数现有作品针对少量内核的事实，基于路的CP 受到了比基于集合或基于块的CP更多的关注(参见表四)。()基于路的CPT只有在WC ≥ 2N (N =核心数)时才有意义，因为每个核心至少需要分配一条路。因此，它需要高速缓存 关联性，导致高访问延迟/功率开销[Mittal等人。2014b]. 2014].
基于集合	(+)它提供了比基于路径的CP更高的粒度，并且易于软件控制。()它需要对操作系统功能进行重大更改，并且可能会使虚拟内存管理变得复杂。因为基于集合的CPT中的重新配置改变了许多的集合索引 块，这些块需要刷新或迁移到新的集合索引。为了减少这种开销，可以降低重新配置频率[Lin等人。2009; 2009],2009] 2014a; 米塔尔和张2013; 2009] 2009].
基于块的	(+)它提供了最高的粒度，并且随着内核数量的增加，它将变得越来越重要。 ()获取基于块的(细粒度)分配的配置信息具有挑战性，因此，一些基于块的CPT通过线性插值路级监视器的未命中率曲线来获取该信息[Sanchez和Kozyrakis2011], 完全准确。此外，可能需要更改RP和附加位来标识每个块的所有者核心。

表三。不同分析和解决方案方法的特性/挑战

从收集分析数据...	
实际缓存	(+)这种方法不会产生额外单元的存储和复杂性开销，因此，它特别适合在真实系统上执行的研究。 ()可能需要逐一评估每种可能配置的性能/失败率[Lin 等人。2008; 2013], 时间开销。或者它可能需要专门分配几个缓存集来试验不同的策略[Jaliel 等人。2008]
独立剖析单元	(+)使用单独的每核监控单元可以提供关于每个程序的单独行为的更准确的估计[谢和 Loh2009; 2006], 而不干扰主缓存[Mittal 等人。2014a; 2013].2008],
解决方法	
启发式方法	(+)它们通常更简单。 ()它们需要手动参数调整，可能无法保证优化目标。
数学/分析模型	(+)它们基于严格的理论基础，因此可以随着内核数量的增加提供强大的保证和更好的可扩展性。 ()它们可能需要通过使用训练数据集或代表性工作负载进行离线模拟来学习参数，并持续调整参数以 让模型保持最新。此外，它们的硬件实现可能会产生大量开销。

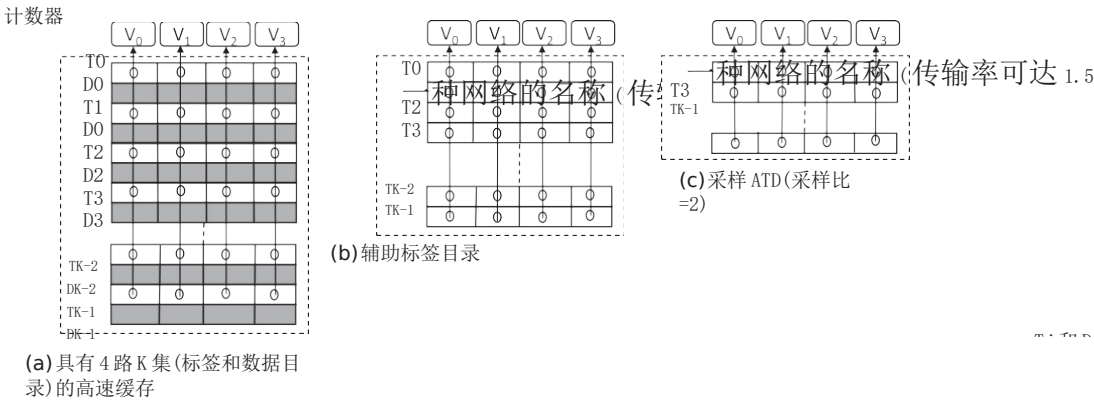

图 3。从(a)实际缓存和(b，c)单独的分析单元(ATD)中收集核心的分析数据。(假设在每一组中，从 MRU 到 LRU 的路是有序的。计数器 V0 和 V3 分别从所有集合的 MRU 和 LRU 块更新，以此类推。)

2.2.5. 剖析方法。用于指导 CP 算法的分析数据可以从实际缓存本身或单独的监控单元中收集(参见表 III 和图 3).

2.2.6. 求解算法。虽然大多数 CPT 基于利用对问题的洞察力开发的试探法工作，但是一些 CPT 使用众所周知的求解算法或数学模型(参见表 III). 这些算法/方法的例子包括动态规划、梯度下降算法、反馈控制理论等(参见表 V).

2.3. 使用 CPT 的挑战

我们现在总结使用 CPT 的权衡和障碍。

不切实际的假设:大多数基于路径的 CPT 假设真 LRU 替换策略，尽管商用处理器仅使用 LRU 的近似值[Kedzierski 等人。2010]. 这是因为真 LRU(每个集合的 WC log2(WC)位)的存储开销很大，而对于大型关联缓存，真 LRU 和伪 LRU 的性能差别很小。然而，这些伪 LRU 火箭和其他最近的火箭(例如，BIP[库雷希等人。2007]) 不要遵循堆栈属性，因此，基于某种方式的 CPT 可能不适用于这些 RPs。此外，一些 CPT 的建模框架是在假设完全关联缓存的情况下开发的[Petoumenos 等人。2006] 或者偏斜关联缓存[桑切斯和科济拉基斯 2011] 因此，当应用于集合关联高速缓存时，这些模型可能不完全准确。

提高性能的挑战:许多 CPT 以识别的方式处理不同的未命中，尽管由于瞬时 MLP 和 NUCA 效应，不同未命中的延迟可能不同(第节 5.3.1). 此外，其他因素，如带宽拥塞或负载不平衡，可能会造成性能瓶颈，并否定竞争优势。由于这些原因，CPT 带来的失败率降低可能不会转化为相应的性能提升。为了避免这个问题，一些 CPT 考虑了每次未命中的延迟影响；然而，这些模型可能并不直接、完全准确和可移植。

范围有限:CP 只对有限责任公司敏感的应用程序有用，这些应用程序会对共享缓存造成破坏性干扰。因此，CP 变得没有必要，甚至对小足迹有害[库克等人。2013] 或局部友好的[詹等。2014; 常和苏希 2007] 应用和大规模缓存，例如，堆叠 DRAM 缓存[Mittal 和 Vetter 2016]。

需要谨慎的设计选择:CPT 需要优化多个通常相互冲突的目标，如吞吐量、公平性、服务质量、能效、负载平衡等。这需要仔细选择 CPT 设计参数，例如配额分配和实施策略、分区间隔[Moreto 等人。2009],等等。

2.4. 实处理器中的并行处理技术

CPT 已经集成到几个产品系统中。例如，一些英特尔处理器为基于通路的 CP 提供支持[英特尔公司 2016]. 类似地，使用页面着色的基于软件的共享高速缓存分区方法[Lin 等人。2008] 已经集成到 Linux 操作系统中[OSU-CSE 新闻 2010].

英特尔至强处理器 E5-2600 v3 家族为实现共享缓存服务质量提供支持[Herdrich 等人。2016]. 它提供了一种“缓存监控技术”来跟踪各种程序的共享缓存使用情况，还提供了一种“缓存分配技术”来根据操作系统/VMM 策略分配缓存配额，例如，减少争用、隔离抖动程序和避免缓存不足。缓存分配技术可以与任何底层配额实施方案配合使用，例如基于路/块/集的方案。AMD 皓龙处理器采用 6MB 16 路共享 L3 缓存[康威等。2010].为了避免高速缓存污染，L3 控制器检测高速缓存访问强度高但命中率低的应用程序。这些应用程序的行没有被提升到 MRU 位置，而是被设置到 LRU 位置或 LRU 堆栈的中间位置[康威等人。2010]. L3 缓存也被分区以存储内存控制器(1MB)的正常数据(5MB)和缓存目录(探测过滤器)。与保留单独的片内探测滤波器相比，这种设计节省了芯片空间，因为探测滤波器仅在多处理器中启用，而在单处理器设计中禁用。此外，与将目录保存在动态随机存取存储器中相比，将其保存在静态随机存取存储器中可以实现快速读/写访问，并减少内存延迟和带宽。L3 缓存(5MB)和目录(1MB)的配额是根据最小化之间的权衡来选择的 L3 缓存未命中和最小化降级[康威等人。2010].

3. CPTS 的分类

3.1. 基于特征和目标

桌子 IV 根据作品的属性和优化目标对作品进行分类。显然，动态 CPT 和严格 CPT 比它们的对应部分使用得更频繁，因为它们能够动态适应工作负载行为，并分别为实施系统级策略实施严格的配额。来自表格 IV，同样明显的是，CP 已经被用于各种各样的优化。

3.2. 基于循环前缀的算法及实现

桌子 V 重点介绍了不同作品使用的解决方案算法/方法，以及它们获取分析信息的方法(参见表 III 作为背景)。诸如集合采样、部分标记和基于布隆过滤器的监控单元等方案以较小的不准确性为代价来降低剖析开销。表中还突出显示了使用这些方案的作品 V。

3.3. 基于评价平台和方法

桌子 VI 总结了每部作品所使用的评价平台。模拟器允许灵活地试验可能无法在实际硬件上实现的不同策略/参数。相比之下，真正的处理器能够实现快速和真实的评估，因此允许使用大的执行长度[Lin 等人。2008],这对于评估大型缓存尤其重要。一些作品使用分析模型[斯通等人。1992；布洛克等人。2015；Petoumenos 等人 2006；Planas 等人 2007；哦等等。2011；刘等。2010；桑切斯和科济拉基斯 2011；徐等。2006] 为实际评估开发/指导他们的技术和模拟器。分析模型有助于评估独立于特定处理器架构或工作负载的极限增益(例如，降低失败率的最佳 CP)。但是，这些模型可能会孤立地研究缓存，因此，可能不会模拟缓存优化对系统性能的影响及其与其他组件的交互。显然，这三个平台/方法都是不可或缺的，并且提供了互补的见解。

要获得 CPT 可伸缩性的粗略估计，请参见表 VI 还会根据评估核心的最高数量对其进行分类。大多数作品将它们的技术与非托管共享缓存和其他相关技术进行比较。有些作品还提供了与静态分区(私有)缓存和分区全关联缓存的比较，它们在表中突出显示 V 爱达荷（Idaho 的缩写）

3.4. 基于其他功能

为了获得更多的见解，我们现在根据附加特征对作品进行分类。

(1) 一些作品根据应用/线程的缓存行为对它们进行分类，以指导 CPTs 或创建代表性工作负载[周等人。2016；常和苏希 2007；Kaseridis 等人 2014；Moreto 等人。2008；Jaleel 等人 2008；Nikas 等人 2008；金等。2009；Moreto 等人。2009；赫雷罗等人。2010；刘等。2014；库克等人。2013；林等。2008，2009；叶等。2014；Sundararajan 等人 2012；Planas 等人 2007；迪布达尔和斯滕斯特伦 2007；刘等。2010]. 有些作品对失误进行分类[雷迪和彼得罗夫 2010；Sundararajan 等人 2012；Nikas 等人 2008].

(2) 当已经为某个时间间隔指定了缓存配额时，一些 CPT 可能会故意增加一个子时间间隔的配额，并减少另一个子时间间隔的配额，以便在整个时间间隔内平均实施指定的配额[Pan 和 Pai2013；常和苏希 2007]. 相比之下，大多数其他 CPT 会在整个时间间隔内强制执行确切的指定配额。

表四。基于核心竞争力特征和目标的分类

种类	参考
静态或动态 CPT	
静态 CPT	[Stone 等人。1992; 2010; 2014; 2009; 2010; 2015; 2007; 2011; 2015]
静态和动态 CPT	[Rafique 等人 2006; 2004; 2013; 2008; 2014;2011b; 2004]
动态 CPT	几乎所有其他人
伪分区或严格分区	
伪分区	[雷迪和彼得罗夫 2010; 2008; 2012; 2009; 2013; 2014; 2006; 2012]
伪分区和严格分区	[Rafique 等人 2006]
严格划分	几乎所有其他人
分配粒度	
路级	[库雷希和帕蒂 2006; 2016; 2014; 2012;2010; 2009; 2004; 2015;2015; 2009b; 2014;2010; 2009a; 2012; 2008; 谢与陆 2009; 2013; 2011; 2008;2013; 2010; 2005; 2015; 2013; 2011b; 2014; 2007; 2011; 2004; 2011; 2000; 2007; 2010; 2008; 2016;2007; 2006; 2010; 2004;2011a; 2016; 2016]
设置/颜色级别	[米塔尔等人。2014a; 2008; 2013; 2009;2009; 2014; 2010; 2009; 2010; 2014; 2008; 2014; 2007; 2009; 2015]
块级	[桑切斯和科济拉基斯 2011; 2007; 2014; 2012; 2014; 2006; 2012]
优化目标	
表演	几乎全部
公平	[Kim 等人 2004; 2014; 2009; 2006; 2008; 2007; 2010; 2015; 2012; 2015; 2009a, 2009b; 2012; 2008; 2009; 2013; 2008; 叶和赖曼 2005; 2009; 2014; 2015;2014; 2009; 2010; 2011; 2008; 2006; 2006; 2016]
服务质量或优先级	[卡斯托尔和桑切斯 2014; 2009; 2013; 2007; 2009a; 2014; 2012; 2013; 2014; 2014; 2007;2006; 2011; 2016; 2006; 2004;2011a; 2016]
静态能 动态能量	通过对未使用的缓存进行电源门控[Sundararajan 等人。2012; 2013;2014a; 2011; 2010; 2012]2012; 2010; Varadarajan 等人 2006], 2010; 2013; 2010; 2012], 2013; 2016]
功率封顶	[王等。2012]

表五.基于CPTs算法和实现的分类

解决方法/启发式	
回归和曲线拟合	[Srikantiah等人。2009a; 2011b; 2010;2014; 2012; 2011a]
机器学习	[Bitirgen等人2008; 2014]
动态规划	[Kozhikkottu等人。2014; 2008; 2015]
反馈控制理论	[Srikantiah等人。2009b; 2012]
梯度下降算法	[hasenlaugh等人2012; 2016]
基于市场的方法	[王和马丁内斯2015]
图形着色	[Jung等人。2010]
从获得的分析信息	
独立的分析单元	[库雷希和帕蒂2006; 2014a; 2010; 2013; 2012; 2008; 2009;2010; 2015; 2005;2009; 2015; 2015; Kaseridis等人2014; 2014; 2009; 以及其他人2014; 2011; 2010; 2011]
实际缓存本身	几乎所有其他人
降低分析开销	
设定取样	[库雷希和帕蒂2006; 2014a; 2014; 2010; 2013; 2012; 2012; 2008; 2009; 2008; 2009; 2015; 2009; 2015; 2015; 2014; 2014; 2009; 2010; 2008;2011]
部分标签	[Kaseridis等人。2009, 2014; 2008]
布隆过滤器	[Nikas等人2008]

(3) 在一些作品中，所有共享的高速缓存(例如，L2和L3)都是分区的。2011b, 2011a]，而在其他工作中，只对单个末级共享缓存(可能是L2或L3)进行分区。
(4) 在某些处理器中，大多数分区是私有的，剩余的缓存在内核之间共享[赫雷罗等人。2010; 布洛克等人。2015; 刘等。2014; 桑切斯和科济拉基斯2011; 库克等人。2013; 林等。2009; 谢与陆2010; 迪布达尔和斯滕斯特伦2007;结算等人。2006].虽然这种方法排除了对整个缓存进行分区，但它可以同时提供私有缓存和共享缓存的好处。相比之下，在大多数其他核心处理器中，所有分区都是分配给核心的私有分区。
(5) 基于堆栈属性，单个WC路ATD可以为所有路1到WC的缓存提供命中/未命中信息[Qureshi和Patt2006]. 因此，利用ATD的CPT可以在每个重新分区事件中通过不止一种方式来改变核心的路数。相比之下，有些技术收集的分析信息比目前的方法多或少[Nikas等人。2008;赫雷罗等人。2010] 因此，每次这些技术只能通过一种方式改变配额。
(6) 有些技术要求单个曲线的脱靶量曲线是凸的[Stone等人。1992]，而其他技术没有这样的要求[Brock等人。2015; 王与马丁内斯2015; 库雷希和帕蒂2006].
(7) 一些CPT在替换时强制执行分区决定(例如，王和陈[2014]，桑切斯和科济拉基斯[2011]，和Rafique等人[2006])，而其他人通过高速缓存分配来实施它们(例如，Lin等人[2008] 和米塔尔等人。[2014a]).

表六。基于不同作品评价平台和方法的分类

种类	参考
评估平台	
真实系统	[库克等人。2013; 2008; 2007; 2014; 2009; 2009; 2014; 2016; 2015;2016]
模拟器	大多数其他人
评估的核心数量	
2个内核	[结算等人。2006; 2013; 2008; 2007; 2011; 2013; 2009; 2008; 以及其他人2013; 2009; 2007; 2004; 2006; 2004; 2007; 2016; 2011; 2004]
4 ≤芯≤ 8	[Suh等人。2001; 2015; 2014a; 2011; 2006; 2007; 2014; 2010; 2009; 2014, 2010; Nikas等人2008; 2010; 2009b; 2014; 2010; 2012; 2008;2013; 2010; 2008; 2014;2009; 2005; 2009; 2014;2015; 2015; 2014; 2011b;2012; 2014; 2009; 2012; 2010; 2010; 2008; 2006;2007; 2006]
16 ≤芯≤ 32	[桑切斯和科济拉基斯2011; 2008; 2014;2015; 2012; 2009a; 2012; 2010; 2016; 2016]
64 ≤核心≤ 256	[Kaseridis等人。2010; 2015; 2011]
与...比较...	
静态分区缓存	[库雷希和帕蒂2006; 2015; 2015; 2008; 2010; 2009a; 2014a;2012; 2013; 2014; 2011; 2009; 2005; 2010; 以及其他人2010; 2013; 2011b; 2012;2014; 2007; 2009; 2011;2008; 2008; 2006; 2010; 2011a]
分区全关联缓存	[王和陈2014]

4. 高速缓存分区的粒度

在本节中，我们将根据高速缓存分配的粒度来讨论CPT。

4.1. 基于方式的客户产品组合

Suh等人[2004] 提出一种基于路的CPT，它通过使用每路命中计数器来估计应用程序的缓存效用。根据效用值，计算边际收益，然后，迭代地，一种方法从边际收益最小的应用转移到边际收益最大的应用。最后，将新分区与前一个分区进行比较，选择更好的分区。为了强制实施缓存配额，在替换时，如果应用程序的实际配额大于或小于其目标配额，则可以收回由相同或其他(分别)应用程序拥有的数据块。

库雷希和帕蒂[2006] 提出一种CPT，该CPT基于应用程序的缓存效用(即，随着分配的增加，未命中率降低)而不是缓存需求(即，间隔中引用的唯一块的数量)来为应用程序分配缓存配额。为了获得效用值，他们对每个核心(应用程序)使用ATD，并且基于集合采样思想，只监控很少的集合。ATD只存储标签，不存储数据，它记录每次命中的LRU位置。由于LRU遵循堆栈属性，ATD允许估计缓存

为应用程序分配不同路数的情况下的命中率。根据命中率/未命中率信息，获得每个高速缓存通道的效用，然后它们的 CP 算法周期性地确定最大化总效用的高速缓存配额，这与最小化未命中总数相同。然而，随着内核(应用程序)数量的增加，可能的分区数量呈指数级增长。他们提出了一种贪婪的 CP 算法，该算法迭代地为应用程序分配一条具有最高效用的路径。如果所有应用的效用曲线都是凸的，则该算法是最优的[Stone 等人。1992]. 对于非凸曲线应用的一般情况，他们提出了“前瞻算法”在每次迭代中，该算法计算所有应用程序的“最大边际效用”和最小边际效用出现的次数。具有最大 MMU 值的应用程序被分配了实现 MMU 所需的方式数。分配完所有路后，迭代算法停止。如果所有应用程序都显示凸效用曲线，则前瞻算法简化为上面讨论的贪婪算法。他们表明，与共享缓存和等分区缓存相比，他们的 CPT 提高了性能和公平性。

Planas 等人[2007] 提出了两个度量应用程序缓存敏感度的标准。第一个指标(wK%)显示了应用程序通过所有方式实现其 IPC 的 K%(例如，90%)所需的缓存路数。例如，在 16 路 LLC 中，w90%大于或小于 8 的应用可分别归类为高或低效用。第二个度量(wLRU)估计当一个应用程序与另一个应用程序一起执行时，LRU 会给该应用程序提供多少种方式。当单独执行时，通过与每个应用程序的高速缓存访问成比例地缩放高速缓存关联性来估计该度量。基于这些指标提供的见解，LRU 和 CPT 寻求最大限度地减少失误的表现被称为最小失误(例如，库雷希和帕蒂[2006])，可以在双核系统中预测。例如，假设情况 1，其中两个应用程序的 w90%之和小于关联性。现在，如果两个应用程序的 w90% < wLRU(案例 1A)，那么 LRU 将获得与明米斯相似的性能。如果一个应用程序的 w90% < wLRU，而另一个应用程序的 w90% > wLRU(案例 1B)，则 LRU 会损害第二个应用程序的性能。在这种情况下，最小化可以提高性能。在情况 2 中，两个应用程序的 w90%之和超过了关联性，对于这种情况，LRU 和明米斯适用于不同的应用程序。他们的方法的局限性在于，它只适用于真 LRU，对于两个以上的内核变得不可行。

Bitirgen 等人[2008] 提出一种机器学习方法，以协同人工使用多个共享资源(特别是 L2 缓存配额、功率预算和片外带宽)，从而实现更高级别的性能目标。他们的技术周期性地在应用程序之间重新分配资源，以响应程序行为的动态变化。他们使用“人工神经网络”的集合来学习程序性能的近似模型，该模型是分配的资源及其最近行为的函数。表征当前 L2 缓存状态和最近行为的属性包括 L1D 缓存中过去 20K 和 1.5M 指令的读/写未命中和命中，以及分配给程序的脏缓存路的比例。这些属性对最近和很久以前的程序行为进行建模，并估计应用程序可能生成的写回次数。通过对所有人工神经网络的估计进行平均，获得最终的性能估计。这种方法允许给人工神经网络估计分配置信水平，也提高了准确性。他们还讨论了训练人工神经网络和提高其准确性的策略，例如，避免过度拟合和减少估计误差。

为了优化系统级性能指标，需要搜索资源分配的不同可能组合。由于穷举搜索是不可行的，他们使用了基于随机爬山的搜索启发式方法。搜索算法的重点是

全局分配空间的特定区域，其中每个区域将资源位置优化限制在一个程序子集内，并系统地将公平配额分配给其他程序。他们通过在分配空间上均匀随机地分配资源来选择用于训练模型的样本，这允许选择代表性样本。此外，随着新样本的每次添加，现有样本将被丢弃，以使样本保持较长的持续时间，并且仍然对最近的样本给予更高的优先级。最后，选择并实现期望优化期望度量的资源分配配置。他们试验了一种基于方式的 CPT，一种基于每核 DVFS 的技术来管理芯片功率分配，以及一种在应用程序之间分配带宽的策略。他们表明，与单个资源的孤立管理或多个资源的不协调管理相比，他们基于人工神经网络的协同管理方法提供了更高的吞吐量和公平性。

Rafique 等人[2006] 介绍一种技术，该技术允许操作系统基于系统级目标指定缓存配额，配额由硬件强制实施。操作系统根据缓存路指定配额，并且在替换时强制实施配额。如果线程的实际份额变得大于或小于其目标配额，则由相同或其他(分别)线程拥有的块将被替换。实施配额的粒度可以是设置级的，也可以是整体缓存级的。缓存级配额方案的优势在于，如果线程在其他缓存部分占用的空间较少，它会在某些部分为线程分配更大的缓存空间。相比之下，集级配额方案在集级强制实施配额，而不管剩余缓存中线程的缓存利用率如何。因此，与缓存级方案相比，设置级方案还提供了更强的实施配额的保证。

这两种方案的一个限制是，由于它们使用了严格的分区，如果线程的缓存要求降低，分配给它的块可能不会被回收。为了避免这个问题，他们采用了一种在每组中使用一个计数器的机制。每当 RP 在集合中选择的替换候选由于其所有者的份额小于目标配额而被保留时，该集合的计数器递增。当计数器达到阈值时，替换候选项将被逐出，即使它会导致配额违规。至于确定配额的操作系统级策略，他们研究静态分配配额，动态决定配额以提高公平性(例如，错过率均衡[Kim 等人。2004]) 或者实现性能差异化。他们表明，他们的方法避免了频繁操作系统干预的开销，同时仍然为操作系统提供了实现各种缓存管理策略的灵活性。

4.2. 基于集合的 CPT

林等。[2008] 提出基于页面着色的动态和静态 CPT，以优化双核系统的性能和服务质量。它们的动态性能 CPT 对当前分区运行一个时间间隔，对相邻分区运行一个时间间隔(即增加/减少每个内核的配额)，并选择总未命中次数最少的分区。他们针对服务质量的动态 CPT 寻求确保第一个应用程序的性能降级不超过阈值(与相同分区的同类工作负载的基线执行相比)，并且第二个应用程序的性能达到最大。该 CPT 定期将第一个应用程序的 IPC 与基线 IPC 进行比较，如果 IPC 比基线 IPC 低一个阈值，则增加第一个应用程序的缓存配额，如果已经达到最大值，则停止第二个应用程序。如果 IPC 高于基线，则恢复第二个应用程序(如果已停止)或增加其缓存配额。为了实施 CP，他们将每个空闲内存页面列表分成多个具有相同颜色空闲页面的列表。在页面出错时，以循环方式从这些分配了颜色的空闲列表中搜索页面。缓存分配发生变化时，重新着色页面的内容为

仅在被访问时移动。他们的静态 CPT 通过评估所有可能的候选找到最佳分区。他们的实验证实了他们的 CPT 的有效性。

林等。[2009] 提出一种基于着色的 CPT，它使用低成本的硬件结构来减少页面重定位开销。为了将操作系统页面映射到缓存颜色并记住映射，他们使用“区域映射表”，将内存页面分组到多个内存区域。因此，重新映射是在存储器区域的粒度上执行的，因为在表中为每一页保留一个条目是不可行的。区域映射表允许将页面映射到任何缓存颜色。他们使用采样的 ATD 来获取分析信息，该信息用于根据访问频率、不同存储区域的 L2 缓存访问的空间位置以及多个存储区域共享一种缓存颜色时缓存未命中的增加，将应用程序分为不同的类别。基于这种分类，决定高速缓存配额，例如，显示流行为的几个区域被分配给高速缓存颜色，其工作集适合高速缓存的区域被给予大量颜色，而那些显示弱高速缓存敏感度的区域被给予少量颜色。分配给应用程序的颜色数量取决于其内存区域的特性。这些配额通过重新配置区域映射表来实施。与共享和静态分区缓存相比，它们的 CPT 提高了吞吐量和公平性。

张等[2009] 提出了一种基于页面着色的 CPT，旨在减少重新分区的开销。他们的 CPT 使用未命中比率曲线来根据颜色数量确定每个应用程序的缓存配额。由于页面着色进一步限制了内存空间分配和重新着色，导致了大量开销，因此它们的 CPT 最多重新着色固定数量(Z)的热页面。z 计算为每次重新着色的成本与重新着色事件之间的持续时间之比。通过扫描页表来识别每个进程的热点(最常访问的)页，并通过利用访问模式的空间局部性来降低其开销。他们表明，由于只对选定的热点页面重新着色，他们的技术降低了基于颜色的 CP 的开销。

Tam 等人[2007]提出一种基于软件的静态 CPT，它使用页面着色来分配缓存配额。对于每个应用程序，它们都会生成 L2 失败率曲线和“施工中报废失速率曲线”，这些曲线显示了由于不同 L2 大小的内存延迟而导致的失速周期。由于脱靶率的变化可能与性能没有直接关系，他们注意到失速率曲线比脱靶率曲线更适合于指导阴极保护。停顿率曲线说明了诸如 L2 失败率、指令退出停顿对 L2 失败率的敏感性、存储器总线争用和较低层次存储器(例如 L3 和主存储器)的可变延迟等因素。此外，失速速率曲线可以从真实系统上可用的硬件计数器获得。它们的 CPT 提高了共享缓存的性能，对于确定可以共同调度的应用对非常有用。

金等。[2009] 提出一种基于静态颜色的 CPT，用于在不同虚拟机(可能运行不同操作系统)中执行的应用程序之间提供性能隔离，这与其他工作不同(例如，Lin 等人[2008])，其中 CP 实现了单个操作系统的应用程序之间的性能隔离。他们在 VMM 通过使用页面着色方法执行 CP，这样来自不同虚拟机的数据被映射到不同的颜色，从而映射到不同的集合。他们在 Xen 虚拟机管理程序中实现他们的 CPT，这对来宾操作系统是透明的。他们表明，他们的 CPT 通过混合使用缓存敏感和缓存污染应用程序来提高工作负载的性能。

4.3. 基于块的并行处理技术

桑切斯和科济拉基斯[2011] 为具有良好散列和高关联性的缓存提供一个 CPT，例如，zcache [Sanchez 和 Kozyrakis2010]，这可以被建模

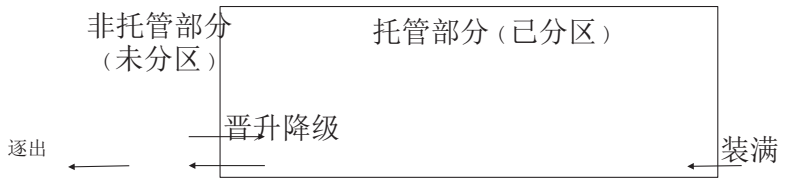

图 4。在 Sanchez 和 Kozyrakis 的技术中，将缓存划分为托管/非托管部分，并在各部分之间流动[2011].

使用独立于应用的分析模型。这些高速缓存提供了很高的可能性，通过简单地修改 RP 就可以避免从大部分块中逐出。在 CPT 中，缓存由所有块共享，分区大小在替换时强制执行。由于从另一个分区替换一个块会导致干扰(即使该块是死的)，并且从同一个分区替换不会随着分区的数量而扩展，因此它们的 CPT 只要求每个分区的逐出/插入速率平均匹配。但是，这仍然会导致干扰，为了避免干扰，它们的 CPT 会将大部分(例如 85%)而不是全部缓存分区。他们的 CPT 将缓存分为托管和非托管部分，并且只对托管部分进行分区。通过从非托管部分而不是其他分区获取空间，分区可能会稍微超出其配额，因此，无论分区数量多少，它们的 CPT 都会保持每个分区的关联性。在驱逐时，无人值守部分中的块被优先驱逐。通过使用合适大小的非托管部分，可以几乎完全避免从托管部分驱逐。块最初被插入托管部分，被降级到非托管部分(只需要更改标记，不需要实际移动)，并在命中时被逐出或再次提升。这如图所示 4.

他们的 CPT 试图将分区的实际配额与目标(通过分配方案找到，例如“前瞻算法”[Qureshi 和 Patt2006]) 通过匹配降级率和插入率。在任何驱逐中，驱逐优先级大于分区特定阈值的所有候选都被降级。阈值取决于所有分区的插入速率和大小，并通过让分区超过其配额，然后根据它们超过配额的程度调整阈值来确定。此外，他们使用基于时间日志的 LRU，其中每个分区都有一个计数器来记录其时间日志，该计数器在 P 次访问后递增(分区大小的 1/16)。每个传入的块都获得其分区的当前时间日志。使用这些，它们避免了跟踪驱逐优先级的需要，只要分区超过其配额，就将所有候选对象降级到固定时间日志之下。固定点根据每个分区看到和逐出的候选数量进行调整。他们表明，与基于路径的 CPT 和伪 CPT 相比，他们的技术提供了更高的性能，并且可以更好地扩展到大量内核(例如 32 个)。他们的 CPT 的一个限制是，它无法对整个缓存进行分区，这可能会导致某些工作负载的性能下降。王和陈[2014] 提出了一种基于替换的 CPT，用于即使在有大量分区的情况下也能保持高关联度。他们将“无用”定义为块对应用程序性能的无用性。不同的 RPs 对无效值进行不同的排序，例如，OPT(最优)、“最少使用”和 LRU RPs 分别按下一次访问时间、访问频率和上次访问时间对块进行排序。它们表明，基于替换的 CPT 在提高关联性和实现分区(缓存配额)目标之间有权衡，因为前者需要驱逐最无用的设备候选，而后者需要优先替换过大分区的候选。他们讨论了一种“分区优先”的政策，这种政策更倾向于实施配额，而不是提高关联性。此策略首先选择超过其

目标大小最大，然后从该分区的候选块中替换无用性最高的块。但是，由于关联性降低，此策略无法扩展到大量分区。

他们提出的CPT通过缩放其块的无用性来控制分区大小。它为每个分区分配一个“比例因子”，在每次驱逐时，它用比例因子来衡量一个分区的替换候选的无用性。然后，它驱逐最大规模无效的候选人。因此，从整个高速缓存的角度来判断分区的块的无用性，并且通过适配SF，可以控制分区的大小。使用分析建模，他们表明在某些假设下，他们的CPT保持分区大小在统计上接近他们的目标大小，并且分区的关联性不依赖于分区的总数。他们进一步提出了他们的CPT的实际实现，使用基于时间日志的LRU[桑切斯和科济拉基斯2011]. 块的时间日志和当前时间日志之间的差异越大，块的无用性就越高。他们还使用基于反馈的方法来调整SF值。对于分区，在固定的插入或逐出之后，如果实际大小大于目标大小，并且分区可能会增长(即插入数量超过逐出数量)，则其SF会增加。他们的CPT显著提高了性能，实际实现获得了接近分析模型预期的性能。

5. 缓存分区的各种策略

在本节中，我们将回顾几种CP策略，如静态CP、伪分区、基于缓存统计的CP等。

5.1. 静态CPT

斯通等人。[1992] 提出了一种优化高速缓存分区以最小化整体高速缓存未命中率的方法。在假设每个应用的未命中比率曲线是凸的情况下，它们表明最佳配额是应用的未命中比率的导数变得相等的点。他们的算法使用贪婪分配方法，将下一个块分配给失败率导数最大的应用程序，直到所有的块都被分配。

对于拥有两个以上内核的处理器，Brock等人。[2015] 提出理论背景和一种“分区共享”分配技术，其中一些内核可以共享高速缓存，而另一些内核可以有自己的私有分区。它们定义了“自然分区”，使得每个应用程序在其自然分区中的未命中比率与共享缓存中的未命中比率相同，因此，自然分区缓存的性能与共享缓存相当。该公式允许将PS问题简化为简单的分区，因此，分区的最优解至少与PS的最优解一样好。换句话说，尽管存在大量可能的分区共享组合，但分区通常是最好的选择。他们进一步提出了一种寻找最优划分的动态规划算法。该算法逐应用查找分区。当添加新的应用程序Ai时，它会获得ci数量的缓存，这将使其未命中和未命中的总和最小化，第一个i 1应用程序的最佳分区的缓存大小为C ci，其中C是总缓存大小。不同于斯通等人的技术1992]，除了吞吐量之外，他们的算法还可以优化公平性和服务质量。他们的最佳PS算法的性能明显优于共享缓存和相等分区。

5.2. 伪分区技术

伪分区技术通过控制插入和提升——来工作[谢和陆2009; Halwe等人。2013; Jaleel等人2008; Hasenplaugh等人2012;Kaseridis等人2014] 或者使用基于衰减/重用距离的管理来控制

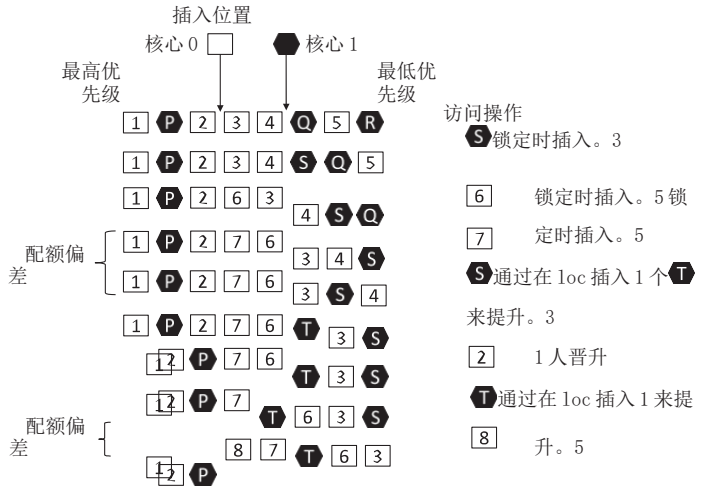

图 5。谢和 Loh 的伪划分方案的工作示例[2009].

缓存中块的生存期[Petoumenos 等人。2006; Duong 等人 2012]. 我们现在讨论其中的一些。

谢与陆[2009] 提出一种伪分区技术，通过控制缓存插入和提升策略来工作。他们首先使用实用程序监视器(Qureshi 和 Patt)找到每个核心(比如 Qi)的缓存配额目标 2006]. 然后，来自核心 I 的新块被插入到优先位置 Qi，因此，核心的配额决定其插入位置。缓存命中以概率 P(典型值为 3/4、1 等)将块提升一个优先级。)并且因此，优先级与概率保持相同 1 ． p ．“受害者选择政策”与 LRU 的政策相同，即驱逐最不优先的街区。具有较小缓存配额的内核块被插入到较低的位置，因此，它们在升级时会遇到更多竞争，并且可能会更快被逐出。对于配额较大的内核，情况正好相反。为了避免流应用程序的缓存抖动，他们首先通过查看未命中或未命中率是否超过特定阈值来检测它们。对于此类应用程序，以优先级 S 进行插入，其中 S 显示流应用程序的总数。因此，不管它的目标配额如何，每个流应用程序都只有一条路可走。此外，此类内核的升级概率降低，这确保只有显示出显著重用的块才能升级到更高优先级的位置。它们的持续生产时间如图所示 5.他们表明，通过处理不同的内存访问模式，他们的技术提供了比仅针对自适应插入或容量划分的 CPT 更高的吞吐量和公平性。他们基于分配的路的数量来决定插入位置的方法的一个限制是，许多分区可能具有低插入位置，这导致在近 LRU 位置的严重争用和在近 MRU 位置的难以逐出的块。

Jaleel 等人[2008] 请注意，在共享缓存中使用 DIP 比 LRU 管理的缓存提供了更好的性能，但它无法考虑单个线程的行为。请注意，DIP 政策的工作原理是从 LRU 政策和 BIP 政策中找到最佳政策，其中 BIP 政策以小概率在 MRU 位置插入区块，以大概率在 LRU 位置插入区块[Qureshi 等人。2007]. 他们提出了一个线程感知的 DIP，该 DIP 使用试探法来减少具有大量内核的处理器中的集对监视器的开销。他们表明，他们的技术优于线程不知道的 DIP 和基于方式的 CPT。

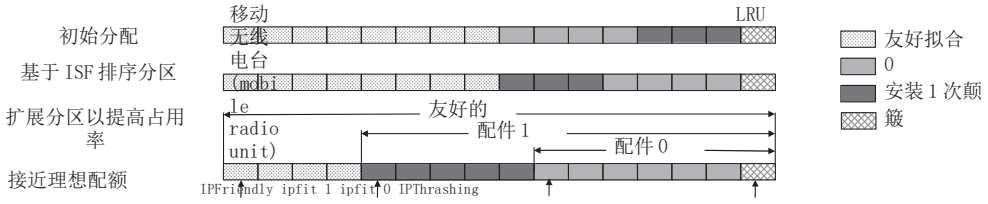

图 6。Kaseridis 等人伪划分方案的工作。[2014] 对于包含 4 个程序的 16 路高速缓存。

Halwe 等人。[2013] 请注意，在伪分区技术中，一个内核可能会窃取其他内核的路，而在严格分区技术中，分配给某些内核的路可能仍未使用。他们扩展了谢和陆的技术[2009]并且规定具有 W 路目标分配的核可以向/从其他核捐赠/窃取最多 W/2 路。为了实施该规则，他们调整牺牲者选择策略，使得如果一个核心的实际配额小于 W/W/2，则在高速缓存未命中时，从可以捐赠块的其他核心替换一个块，但是如果条件不成立，则替换来自同一核心的块。

Kaseridis 等人[2014] 提出一种基于程序的缓存友好性和 MLP 行为来执行缓存软分区的技术。它们将程序分为三种类型:(1)缓存适配(它们的工作集适合缓存，但它们受益于缓存)；(2)缓存友好(它们的工作集大于缓存大小，并且受益于缓存)；以及(3)抖动/流传输(具有非常大的工作集大小或没有高速缓存重用的那些)。使用每个内核的采样 ATD，他们发现每个程序的未命中率随其缓存分配的变化。根据该信息，颠簸程序是指单个和所有高速缓存通道的分配之间的未命中率差小于阈值的程序，适合程序是指在某个高速缓存配额下未命中率低于阈值的程序，其余程序是友好程序。此外，程序的 MLP 数被视为 MSHR 中的条目数，因为它显示了从内核到内存层次结构的任何时候未完成的长延迟内存请求的数量。他们的 CPT 分两步走。第一步，估计每个程序的理想缓存配额。他们首先计算一个程序的“效用率”，定义为每个高速缓存路和 MLP 的累积命中分数的比率。然后，“边际效用”被定义为分配 k 个额外路的效用率的变化除以 k。使用这个度量，高速缓存路被迭代地分配给具有最高边际效用的程序。在第二步中，通过修改插入和提升策略来执行伪分区，目标是每个内核的平均缓存占用率保持接近第一步中预测的理想容量。请注意，“插入点”指的是 LRU 堆栈中插入一行的位置。他们的 CPT 对不同的程序使用不同的 IPs，使得痛击程序在 LRU 栈中只获得最后一个途径，友好程序比拟合程序获得更高的 IPs。此外，在一次点击中，一个块被提升到它的 IP，因此，它不能前进超过那个。同一类别节目的 IPs 是根据它们的“干扰敏感因子”来决定的，在接近 LRU 的位置显示大量点击的节目被认为对高速缓存争用更敏感。灵敏度越高的程序获得的 IPs 越高，反之亦然。他们的 CPT 还监控程序的平均缓存占用率，如果它明显低于理想配额，程序的 IP 将在下一个时期增加，以缩小理想和实际占用率之间的差距。他们的技术如图所示 6. 他们表明，他们的 CPT 提高了吞吐量和公平性。

Petoumenos 等人[2006] 介绍一种通过使用基于衰减的管理来控制程序缓存配额的技术。他们的技术是由统计模型指导的

重用距离，定义为两次连续访问一个地址之间的事件数。为了衡量研发，他们根据缓存替换来量化时间，称为“缓存分配节拍”(CAT)。CAT 允许将时间(事件)与缓存空间相关联。该模型考虑了程序特征(例如，其研发概况)和缓存特征(例如，容量、快速原型)来估计缓存命中和未命中。他们将“时空”定义为缓存中某个块的生存期(以 CAT 为单位)与该块所占空间的乘积。命中和未命中对应的时空分别是有用空间和无用空间。有用空间与无用空间的比率，称为“有用比率”，显示了程序利用其缓存配额的能力以及任何 CPT 的改进潜力。他们将“衰减间隔”定义为在 CAT 中测量的时间段，以便在衰减间隔内未被访问的块成为替换的候选块，而不管其 LRU 状态如何。基于这些信息，程序的缓存配额可以通过基于它们的缓存效用和优先级调整它们的 DIs 来控制。因此，与低优先级和低局部性的程序块相比，具有更高优先级和时间局部性的程序块可以在高速缓存中停留更长的时间。他们的技术用不同的 DIs 估计未命中，然后选择使总未命中最小化和总有用时空最大化的 DIs。为了实现服务质量，可以选择 DIs 来确保为每个程序分配一定的缓存配额。他们表明，他们基于衰减的方法在管理缓存和提高有用率方面是有效的。此外，他们的技术比基于方式的 CPT 更能减少失误。

Duong 等人[2012] 请注意，为了避免缓存污染，块只能保留在缓存中，直到发生预期的重用。这个重用距离被称为“保护距离”，它在最大重用和及时驱逐之间取得了平衡。在插入或提升时，块的重用距离被设置为 PD。在对集合的每次访问中，集合中所有块的局部放电值减少 1，具有 0 值的块成为替换候选。他们注意到，在多核处理器中，增加应用程序的局部放电会通过减缓其块的替换来增加其缓存配额。基于这一认识，他们提出了一种 CPT，为每个应用程序找到 PDs，从而使整体缓存命中率最大化。他们基于启发式的 CPT 基于三个见解工作:(1)单核命中率大的应用程序对多核命中率的贡献也高，(2)应用程序的多核 PD 预计接近其单核命中率的峰值之一，以及(3)每个应用程序只需要检查几个(例如 3 个)重要的峰值。利用这些见解，他们的 CPT 根据命中率对应用程序进行排序，并将命中率最高的应用程序添加到列表中。然后，以迭代的方式，考虑命中率次高的应用程序，并结合列表中现有应用程序的峰值评估其每个峰值(从单核命中率中找到)。使用搜索算法，找到多核命中率最大化的组合峰值。他们的 CPT 提高了吞吐量和公平性，并且可以随着内核数量的增加而扩展。

5.3. 基于缓存统计的并行处理技术

虽然大多数 CPT 由未命中或未命中率指导，但一些 CPT 主要致力于未命中的估计延迟影响，而其他 CPT 则考虑了 CP 的带宽和写回影响。这些作品在表中突出显示 VII. 我们现在讨论其中的一些作品(也参见第节 7.4).

5.3.1. 以国际化学品方案为基础的消费价格指数。Subramanian 等人[2015] 提出一个“应用程序减速模型”(ASM)来估计由于共享高速缓存和主内存干扰而导致的应用程序减速。减速定义为与其他应用程序一起运行时的执行时间与在系统上单独运行时的执行时间之比。他们指出，准确估计单个请求的干扰效应具有挑战性，因为内存系统中同时服务多个请求。因此，ASM 起作用了

表七。指导持续专业培训的参数

根据未命中估计的IPC或内存延迟	[Moreto等人。2008; 2010; 2012; 2005; 2010; 2008; 2014; 2007; 2009b; 2014a; 2013; 2015; 2014; 2011;2011; 2009]
带宽	[Kaseridis等人。2010; 2010; 2005]
写回和未命中	[周等。2012]
失败率	几乎所有其他人

基于整体(而非个别)请求服务行为。基于应用程序性能与其访问共享缓存的速率之间的相关性，ASM将速度减慢估计为Calone/Cshared。共享可以很容易地测量，为了测量卡洛恩，他们定期执行两个步骤。首先，为了消除内存带宽争用的影响，它们在短时间内给予内存控制器上的应用程序请求最高优先级。这也有助于在没有主存储器干扰的情况下找到高速缓存未命中损失。在没有来自优先应用程序的请求的情况下，另一个应用程序的请求可能会被调度，这可能会延迟优先应用程序的请求，因此，ASM也会考虑并减去这种排队延迟的影响。其次，为了估计共享高速缓存干扰的影响，他们发现在服务争用未命中(即那些将在单独执行中命中的未命中)时消耗了额外的时间。这是根据争用未命中的数量(从采样的ATD获得)和服务缓存未命中(在第一步中计算)和命中所花费的平均时间来估计的。为了估计Calone，这些额外的周期从存储器控制器给予应用程序最高优先级的时间中减去。他们表明ASM在估计减速方面相当准确。

在ASM的基础上，他们进一步提出了CPT，旨在最大限度地减少减速。他们的CPT首先以不同的方式估计每个应用程序的速度。如果Ck显示的是一个有k路的应用程序的缓存访问速率，那么有k路的应用程序的速度下降就是Calone/Ck。Ck是根据一个时期内共享高速缓存中命中和未命中的次数以及用k路服务这些访问所需的估计周期数来计算的。在以不同的方式估计减速后，他们计算边际减速效用为(slowdown k slowdown)/k。然后，使用“前瞻算法”对缓存进行分区[Qureshi和Patt2006] 除了它使用“边际减速效用”而不是像库雷希和派特那样使用“边际错过效用”[2006]. 它们表明，与基于共享缓存和未命中率最小化的CPT相比，它们的CPT提供了更高的性能和公平性。

Moreto等人。[2008] 请注意，缓存未命中的性能影响取决于应用程序的L2未命中的MLP。符合ROB的群集L2未命中可以同时得到服务，因此，它们产生的未命中延迟几乎与孤立的L2未命中相同，如图所示7. 因此，他们的CPT为孤立的未命中分配了比聚集的未命中更高的“MLP惩罚”。使用Mattson的堆栈算法[Mattson等。1970],他们为不同数量的高速缓存路找到高速缓存未命中。然后，他们分别计算缓存未命中转换为命中以及缓存未命中转换为命中对缓存大小增加或减少的性能影响。如果一个核心的高速缓存路的数量增加，那么一些未命中将变成命中。为了找到他们的罚分，他们在相应的条目中存储了堆栈距离(Di)和此失误的所有者-核心。在每个循环中，他们为Di的每个可能值找到堆栈距离不小于Di的访问次数。因此，如果同时服务J个未命中，则每个未命中的成本被指定为1/J。类似地，为了找到将变成未命中的命中的惩罚，他们找到堆栈距离不小于Di(包括

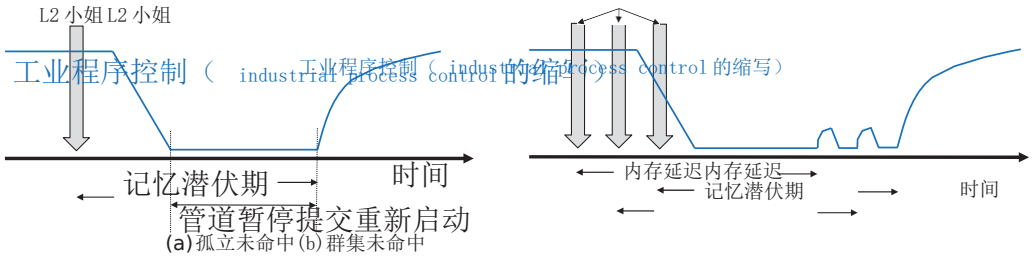

图 7。群集未命中导致的未命中惩罚几乎与孤立未命中相同[Moreto 等人。2008].

未命中和命中)。基于该信息，估计未命中簇的长度。L2 指令未命中会延迟提取，因此它们具有固定的未命中延迟和 MLP 惩罚。基于上述思想，他们评估不同的分区，并找到一个具有最小总“MLP 惩罚”的分区，并在下一个时期使用它。他们还研究了他们技术的一个变体，其中 MLP 成本与运行应用程序的 IPC 相乘，从而赋予更高 IPC 的应用程序更大的权重，以公平为代价优化吞吐量。他们表明，与 LRU 管理的缓存和不知道 MLP 的 CPT 相比，他们的技术提供了更高的吞吐量。

Moreto 等人。[2009] 提出一种 CPT，允许优化吞吐量、公平性或每应用服务质量指标。他们使用采样的 ATD 来估计不同数量的分配路径的 L2 未命中，并从应用程序性能的 CPI 堆栈模型中获得相应的 IPC 值。基于此，他们的 CPT 为线程分配缓存路，以满足其服务质量要求。剩余的路在线程之间分布，以满足整体需求(例如，公平性或吞吐量)。因此，他们的技术可以将服务质量目标转化为缓存配额要求。它们的 CPT 在优化所需目标(公平性或吞吐量)方面是有效的，并且在大多数情况下，还可以允许以最大 IPC 的所需百分比运行应用程序(在单独运行整个缓存时实现)，而不考虑协同运行的应用程序。

5.3.2. 基于带宽的 CPT。于和彼得罗夫[2010] 提出一种 CPT，其重点在于降低片外带宽要求，而不是主要减少未命中，因为前者对性能有直接影响。他们指出，具有大量未命中的应用程序可能不会消耗最大的片外带宽(单位时间内传输的数据量)。如果内存访问聚集在一起，未命中次数较少的应用程序可能需要更大的带宽。基于此，可以减少带宽较低的应用程序的缓存配额。他们使用通过离线启发式算法找到的静态集合级 CP，该算法的输入是不同缓存大小的应用带宽。该算法迭代评估不同的分区，以找到整体带宽需求最小的分区。在每一步中，算法都会搜索在增加缓存配额时带宽需求减少最多的任务。为该任务确认了较大的缓存配额。通过以这种方式迭代，在剩余的缓存空间中搜索剩余任务的配额。他们表明，他们的技术在减少 BW 需求方面是有效的。

6. 实现各种优化目标的 CPTS

在本节中，我们将根据优化目标来讨论 CPT。

6.1. 用于提高公平性或实施优先级的 CPT

金等人。[2004] 根据缓存未命中和未命中率，定义五个度量标准来衡量协同调度应用程序的公平性。他们表明，其中两个指标与执行时公平性密切相关。他们进一步提出了静态和动态 CPT，使用这些指标来优化公平性。静态 CPT 需要一个分析运行，在此期间，假设 LRU RP，估计不同路数的缓存未命中。在此基础上，估计不同可能分区下的未命中总数，并选择一个分区，该分区为所选公平度量提供最佳值。动态 CPT 反复执行两个步骤:“配额分配”和“调整”在配额分配步骤中，为所有应用程序评估所选的公平性度量。基于该度量，选择两个显示分区的最小不公平和最不公平影响的应用，并且如果它们之间的不公平差异超过阈值，则固定缓存空间从显示较小不公平的应用转移到显示较大不公平的应用。然后，在排除这两个应用程序之后，对剩余的应用程序重复该过程。在调整步骤中，如果接收增加的配额的应用的未命中率的减少大于或小于阈值，则提交或恢复配额分配步骤中做出的决定。他们表明，最大化公平性通常会提高性能，而优化吞吐量可能不会优化公平性，因为吞吐量的提高可能来自不公平地增加某些应用程序的配额。

Srikantaiah 等人[2009b] 使用反馈控制理论来划分高速缓存，以提高公平加速并提供服务差异化。假设为每个应用程序提供了性能目标(例如，IPC)，则它们的“分区控制器”会确定新目标以最大化缓存利用率。每个应用程序还有一个“应用程序控制器”，它独立地确定跟踪这些新目标所需的缓存份额(就缓存路而言)。如果由这些控制器确定的缓存配额的总和超过缓存大小，则调用协商模块，该模块在针对指定目标之一的请求配额之间进行协商:对于服务区分，它首先减少低优先级应用程序的配额，而对于公平加速改进，它以公平的方式减少应用程序的配额(即，与其当前配额成比例)。类似地，如果总缓存配额小于缓存大小，他们的技术会增加缓存配额，以使用与上述相反的方法来最大化缓存利用率。因此，他们的技术确定了可行的缓存配额，以产生输出性能目标。在下一个时期，应用程序和分区控制器将再次使用这些输出目标。每个应用控制器被设计成一个 PID 控制器。基于上一个纪元及其配额所实现的性能，它找到了下一个纪元满足性能目标所需的配额。然而，PID 控制器没有考虑系统历史和特定领域的知识，因此，它可能会在高速缓存分配中引入振荡。为了解决这个问题，他们使用一个模型来估计性能(IPC)作为缓存配额(路数)的函数。控制器提供的见解有助于通过跟踪控制决策的历史来避免振荡。它们表明，它们的公平加速和服务差异化机制在优化各自目标方面非常有效。

王等[2012]提出了一种 CP 和容量扩展技术，用于在功率受限的多核中限制 LLC 的最大功率，并在不同程序之间实现公平或不同的缓存访问延迟。基于反馈控制理论，他们采用了两级协同控制器设计。首先，“有限责任公司功率控制器”以粗粒度(例如，10M 周期)运行，并且它试图通过控制可操作的有限责任公司组的数量来限制给定预算的最大有限责任公司功率(例如，最大功率的 80%)。其余的银行都被断电了。其次，la- tency 控制器以更精细的时间粒度(例如，1M 周期)运行，并且它控制每对相邻内核上两个程序之间的延迟比。为了实现

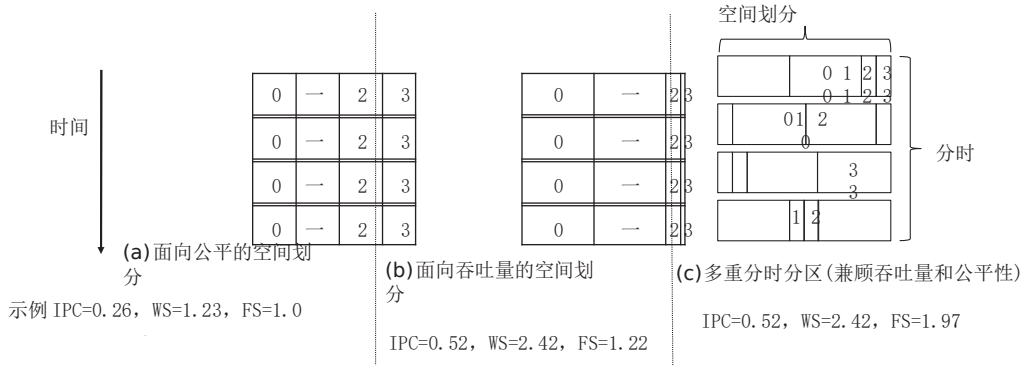

图 8。空间分区((a)和(b))和多重分时分区(c) [Chang 和 Sohi2007]计划。4 核工作负载(art-art-art-art)的示例结果强调了方案(c)可以同时提高公平性和吞吐量。

公平或差异化，它寻求为所有程序实现相同的缓存访问延迟，或为更高优先级的程序(分别)实现更短的延迟，即使总缓存容量在变化。根据控制理论，它计算分配给每个程序的存储体的相对数量，然后根据可用存储体的总数计算实际存储体的数量。这两种控制器都设计成“比例积分”控制器，它们的参数是通过用几个有代表性的程序进行实验，然后用曲线拟合来确定的。它们的设计为精确控制和系统稳定性提供了理论保证。他们的实验表明，他们的技术可以非常精确地控制 LLC 功耗和程序的缓存延迟，并且还允许在性能和功耗之间进行权衡。还要注意，Srikantaiah 等人。[2009b] 和王等人[2012] 分别使用 PID 和 PI 控制器。

常和苏希[2007] 提出了一种改进吞吐量和公平性同时提供服务质量的 CPT。他们指出，不同的优化目标之间存在权衡，因此优化吞吐量会损害公平性，反之亦然(参见图 8(a) 和(b))。因此，他们的 CPT 不是使用单个不公平或低吞吐量的分区，而是在不同的时间间隔内将不同大小的分区分配给不同的应用程序。因此，应用程序的配额在不同的时期会扩大和缩小，如图所示 8(c). 由于颠簸应用程序(工作集大于相等分区的应用程序，从增加的缓存容量中受益匪浅)的吞吐量已经很低，因此在收缩期减少其配额不会显著降低其性能，但在扩展期增加其配额会大大提高其性能，从而补偿收缩期的减速。可以以平等的方式向不同的应用程序提供扩展机会，以实现公平，并以不同的方式实现优先级。此外，通过分配配额，可以长期保证服务质量，与等分区情况相比，配额平均限制了应用程序的运行速度。他们进一步指出，当应用程序不显示破坏性干扰时，基于 LRU 的缓存共享可能会超出 CP。因此，他们建议将 CPT 与基于的缓存管理策略相结合[常和苏希 2006]. 执行周期被划分为由 CPT 或共享技术控制的周期，这取决于有多少应用程序从中受益。他们表明，这种技术可以在有干扰和无干扰的情况下提高应用程序的性能。

库克等人[2013] 呈现双核系统的静态和动态 CPT，其中一个内核优先于另一个内核。他们的国家方案建议旨在优先考虑

核心不会受到伤害，而未优先化的核心可以取得最大的进步。他们的静态 CPT 评估两个应用程序的路分配的所有组合，以及优先级核心性能下降最少的组合，找到一个非优先级核心性能最高的组合。与共享缓存和相等分区相比，这种 CPT 降低了优先级核心的性能下降。它们还提供了一个动态 CPT，当应用程序阶段发生变化时，也就是说，当固定时间段内 LLC 未命中率的变化超过阈值时，该 CPT 就会执行。CPT 从给除了一条路之外的所有路分配优先级核心开始，并开始减少其路分配，直到“每千指令未命中”的变化超过阈值。剩余的方法给了未优先化的应用程序。与静态 CPT 相比，动态 CPT 实现了优先核的并发性能，同时显著提高了非优先核的性能。

6.2. 面向机器翻译应用负载平衡的并行处理技术

潘与白[2013] 请注意，MT 应用程序中线程的集合特定 RD (SSRD)直方图在特定 RDs 处有拐点，在两者之间有平坦区域。因此，平坦区域不会显著降低漏检率。在一些情况下，拐点出现在 RDs 上，因此在线程之间平均划分缓存并不能提供最高的性能，即使线程显示出类似的 IPC 和数据重用。他们的 CPT 通过临时使用不相等的分区来提高基于线程对称性和 SSRD 曲线形状的缓存利用率。如果一个线程当前正在远离一个拐点的地方运行，那么它的缓存配额将被窃取，并被给予一个“优先”线程，以将其带到下一个拐点。如果其他线程的配额位于曲线的平坦部分，则优先线程的总体优势大于其他线程的损失。此外，对于某些缓存大小，增加优先线程的配额比从研发配置文件中推断的要多，可以提高其他线程的性能，即使它们的配额减少了。发生这种情况有两个原因。首先，由于执行期间配额的持续变化，线程通常会将数据留在其他线程的分区中。第二，对有利线程的过度分配降低了其驱逐，因此，有利线程分区中不利线程的剩余数据也降低了驱逐。因此，对不利线程的大量访问在有利线程的分区中命中，这抵消了它们自己的分区中命中的减少。由于偏爱单个线程会导致不公平，他们的 CPT 基于应用程序的对称特性，以“循环”方式选择偏爱的线程。因此，通过暂时取消所有线程的缓存配额，它们的 CPT 提高了缓存利用率和吞吐量，同时确保了负载平衡。

Muralidhara 等人[2010] 给出了两个在机器翻译应用中提升最慢线程的 CPT。第一个 CPT 记录所有线程的 CPI，然后对高速缓存路进行分区，以便将大量路交给 CPI 较高的线程，反之亦然。这如图所示 9. 由于这个 CPT 没有考虑线程的缓存敏感度，所以他们的第二个 CPT 构建了一个模型，说明 CPI 如何随着缓存配额而变化。为此，首先在两个时间间隔内使用 CPT。根据这些间隔中记录的不同高速缓存路的 CPI 值，通过使用“三次样条插值”的曲线拟合为每个线程建立性能模型使用这些模型，CPT 重复地从最快的线程向最慢的线程进行单向传输，直到其他线程变得最慢(从性能模型中找到)。此时，缓存分配通过一步恢复，并且接受这种分区。第二个 CPT 优于第一个 CPT、共享缓存和等分区缓存。

6.3. 节约能源的持续生产技术

CPT 通过减少片外未命中和允许提前完成来节省动态能量。CPT 可以确定所有应用程序的总缓存需求，以及这是什么时候

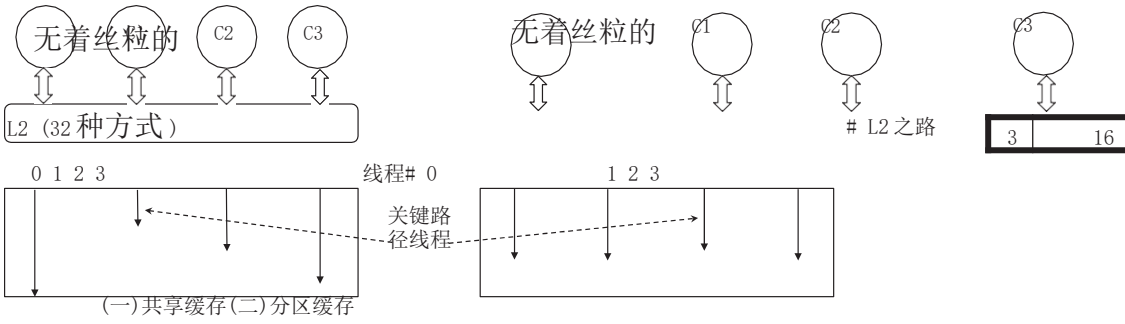

图9。在机器翻译应用中使用CP加速关键路径线程。2010].

小于总容量的剩余缓存可以进行电源门控，因此，CPT也有助于节省泄漏能量。我们现在讨论为节能而提出的CPT。

米塔尔等人。[2014a] 提出一种CPT，它使用动态剖析来估计不同LLC规模下运行的应用程序的能耗。假设LLC中的集合数量为X，它们的剖析单元估计不同集合数量的高速缓存的未命中率，即。，X，X/2，X/4，X/8等等，而库雷希和派特的剖析单元[2006]估计只有X个集合的高速缓存的不同路数的未命中率。根据这些未命中率估计，通过使用线性插值来估计其他集合计数的未命中率。利用这一点，可以估计不同高速缓存分区配置的能量值，使得具有小效用的应用的配额降低或仅增加少量。因此，在某些配置中，可能不会将所有颜色分配给内核。基于这些，为下一个间隔选择具有最小能量的高速缓存分区配置，并且对未使用的高速缓存颜色进行电源门控以节省能量。他们的技术提供了比无螺纹泄漏节能技术更高的节能和性能。

Sundararajan等人[2012]提出一种改进性能和节约成本的CPT能源有限责任公司。他们使用“前瞻算法”[库雷希和帕蒂2006]以确定高速缓存配额，除了仅当未命中率的降低超过阈值时才向核心分配路。因此，某些方法可能不会分配给任何内核，它们可以通过电源门控来节省泄漏能量。此外，他们的CPT实施路对齐，这样在所有的集合中，一个核心的数据块只以它自己的方式存储(参见图10).通过使用这些信息，在从内核访问高速缓存时，只需要访问分配给它的路，这节省了动态能量。他们的CPT带来了性能和能效的大幅提升。

Kotera等人[2011]提出了一种基于路的CPT，它通过对高速缓存路进行功率门控来节省功率。它们将应用程序的局部性计算为对LRU块和MRU块的访问次数之比。该比值越小，局部性越高(参见图11) 因此，在MRU位置具有最多命中的应用程序需要很少的方法来仍然实现高命中率。通过比较两个应用程序的该比率，可以决定增加或减少分配给它们的路数。此外，通过将该比率与较高和较低的阈值进行比较，做出关于功率门控的决定。通过调整这些阈值，他们的技术可以有利于性能或能量优化。他们技术的一个限制是它只适用于2核系统。

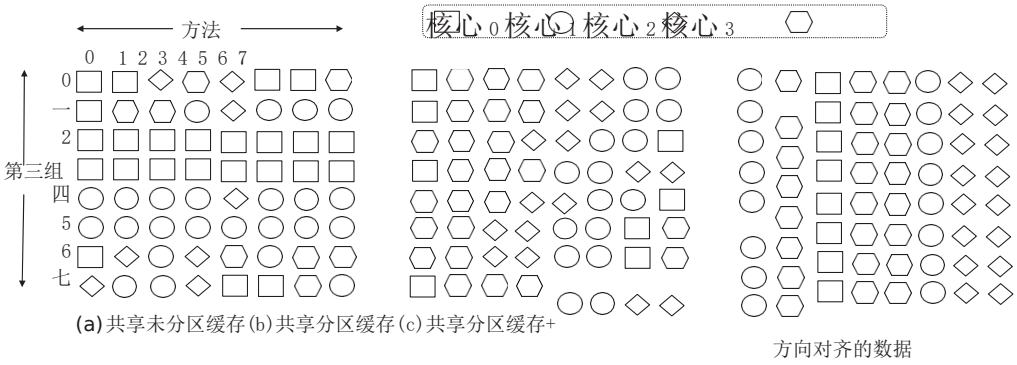

图 10。(a)未分区高速缓存，(b)分区高速缓存，以及(c)具有路对齐的分区高速缓存中的数据布局[Sundararajan 等人。2012].

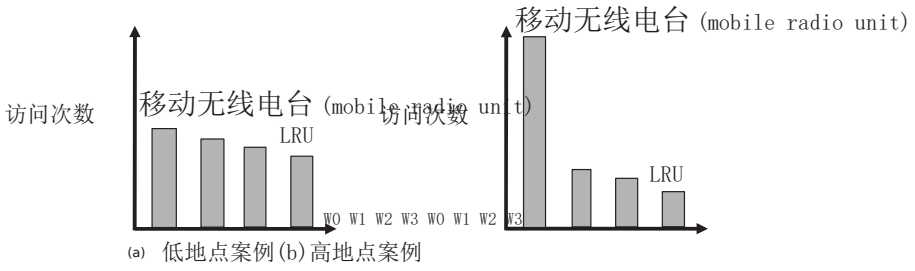

图 11。基于对不同方式的访问来确定位置[Kotera 等人。2011].

6.4. 降低客户产品的开销

谢与陆[2010] 请注意，CPT 通常会产生大量的分析硬件开销和算法执行的延迟开销。然而，共享高速缓存中只有一个或几个抖动应用程序造成干扰，因此，用简单的技术限制这些应用程序可以提供类似于复杂 CPT 的性能优势。在他们提出的技术中，显示在一个时间段内大的高速缓存访问以及大的单独未命中率或单独未命中次数的应用被识别为抖动。单独未命中和单独未命中率由 ATD 估计，根据各自的阈值决定“大”或“小”。然后，他们的技术为每个颠簸应用程序分配固定且少量(例如 2)的路。剩余的应用程序共享维护方式，因此不会对这些方式执行 CP。他们的技术只需要识别颠簸应用程序，从而避免了常规 CPT 的复杂性。通过允许更好地利用共享缓存，他们的技术提供了比使用严格分区的复杂 CPT 更高的性能。为了进一步降低其技术的开销，他们移除了 ATD，并简单地使用每个应用程序的绝对未命中数来识别抖动应用程序，这仍然提供了合理的性能。他们的技术的局限性在于，它不能保证服务质量/公平性，并且可能不适用于具有不同缓存要求的应用程序的一般情况。

Kedzierski 等人[2010] 提出一种用伪 LRU RPs 收集剖析信息的方法，例如用于指导基于路径的 CPT 的“最近未使用”(NRU)和“二叉树”。他们的工作通过放宽真 LRU 的要求，减少了基于路径的 CPT 的开销。读者可以参考 Kedzierski 等人的文章[2010]为了

表八。为各种环境提议的持续专业培训

NUCA 缓存	[常和苏希 2007; 2010; 2009; 2009; 2010; 2005; 2011; 2012; 2007]
受光伏影响的缓存	[Kozhikkottu 等人。2014]
多个共享缓存	[Kandemir 等人。2011a, 2011b]
非易失性主存储器	[周等。2012, 2016]

这些 RPs 的工作。对于 NRU，他们计算 U，即使用的位数设置为 1 的块数。因此，如果一个被访问的块的已用位为 1，这意味着堆栈距离在 1 和 U 之间，并且考虑到与真 LRU 相比的过度估计，他们用因子 x (1)对其进行缩放。如果一个被访问的块的使用位为 0，那么它意味着在 U 1 和 WC 之间的堆栈距离，因此，它们假设堆栈距离为 W。根据这些估计值，它们生成一个堆栈距离直方图，用于指导 CPT。此外，在 CP 之后，NRU 被用于核心所拥有的高速缓存块，例如，如果核心的所有块的已用位被设置，则除了刚被访问的位之外，所有这些位都被重置。他们还提出了基于二叉树的 LRU 分析逻辑。与使用真 LRU 的处理器相比，使用伪 LRU 的处理器性能损失很小，性能损失随着线程数量的增加而增加(例如，8 线程为 7%)。

Nikas 等人[2008] 提出了一种基于“布隆过滤器”方法来发现应用程序的缓存感知度，从而指导客户端应用程序。对于每个核心，他们在每个集合中添加一个 BF。当一个标签被逐出时，它的 M 个最低有效位(最低有效位)被用来索引一个 BF 条目，该条目被改变为 1。在高速缓存未命中时，使用标签的 M 个最低有效位来查询相应的 BF 条目，如果这存储 1，则称为“远未命中”，这表明向内核分配一个额外的路可能(由于 BF 中的误报)已经将该未命中转换为命中。远未命中的数量被缩放以减少误报的影响。他们还监控每个核心对 LRU 位置的命中，这显示了在从核心带走一个方向时转换为未命中的命中数量。在此基础上，在每次迭代中，他们的启发式 CP 算法会比较任何内核的最小命中收益和最大命中损失。如果后者更大，则“远未命中”计数器对应的内核缓存配额增加一个，而“LRU 计数器”对应的内核配额减少一个。当没有最大增益低于最小损耗或考虑了所有内核时，迭代停止。他们表明，他们的 CPT 在提高绩效方面是有效的。

7. 不同语境中的 CPTS

桌子 VIII 总结了为特定环境呈现 CPT 的作品。我们现在来回顾一下。

7.1. NUCA 高速缓存环境下的并行处理技术

Jung 等人[2010]为三维堆叠的 NUCA 缓存提供一个 CPT。他们指出，在 NUCA 缓存中，内存访问时间取决于未命中次数和命中延迟(取决于内核和缓存体之间的距离)。他们将最小化平均内存访问时间的问题建模为图着色问题，其中将高速缓存存储体分配给核心相当于用与核心相同的颜色对存储体着色。由于解决这个问题对于大量内核变得不可行，他们提出了一种启发式方法。首先，最靠近核心的存储体被分配给这些核心。然后，对于每个未分配的存储体，在所有相邻的已分配的存储体中，搜索该存储体(Bʌ)，使得将未分配的存储体分配给与该已分配的存储体相同的核心(Bʌ)导致最高的性能改善。重复这一过程，直到所有银行都已分配完毕。它们表明，它们的 CPT 提供了比同等分区的缓存和延迟变化不敏感的 CPT 更大的性能。

赫雷罗等人[2010] 提出了一个面向运行具有共享数据的机器翻译应用程序的拼接芯片多处理器的CPT。处理器有多个独立的L2高速缓存，逻辑上分为私有部分和共享部分。从本地L1逐出的块存储在私有区域，从相邻缓存溢出的块存储在共享区域。如果应用程序有类似的缓存需求，这允许创建大型私有缓存；如果额外的容量仅对少数应用程序有益，则允许创建大型共享缓存。共享部分存储唯一的块，但是共享数据可以根据请求在私有部分复制。因此，私有/共享部分的相对大小控制复制量。他们提出了一种节点本地CPT方案，用于控制私有和共享部分的大小，而不需要集中式组件或繁重的节点间通信。它们分别使用一个计数器，该计数器在命中私有或共享部分的LRU时递增或递减。在一个时期结束时，如果计数器低于/高于低/高阈值，则它们的CPT分别增加一个共享/私有通道。它们还使用溢出块分配器，试图将更多被驱逐的块发送到共享部分更大的L2缓存。他们的CPT提高了性能和能效，并且通过允许分布式重新分区，还提供了更好的可扩展性。

7.2. 过程变化背景下的持续生产技术

Kozhikkottu等人[2014]请注意，芯片内光伏[米塔尔2016a]会导致多核CPU中的内核频率不相等(例如，4核CPU中的2.5、2.4、2.2和1.8 GHz)，限制更快内核的频率会降低性能。他们提出了一种光伏感知的并行处理技术，允许每个内核以其最高频率运行，并通过并行处理解决由此产生的性能差异。他们试验了带有同步障碍的机器翻译应用程序。他们的CPT分两步走。在第一步中，他们通过改变缓存配额和忽略在障碍处停止所花费的时间来表征所有线程的性能。第二步，进行基于路径的CP，以最大化最慢线程的性能。他们将空间CP问题建模为一个动态规划问题，并通过在重新配置间隔中临时扰动配额来进一步细化解决方案，以找到一个分区，使任何线程的最低吞吐量最大化。它们表明，它们的CPT提供了比共享和均分缓存更高的性能。

7.3. 多级共享缓存环境下的并行处理技术

Kandemir等人[2011b] 提出一种同时划分多个共享高速缓存(例如L2和L3)的技术，用于提升线程显示不同高速缓存需求的MT应用中最慢的线程。他们注意到，由于L2/L3的未命中率显示出相关性，并且其未命中的惩罚是不同的，因此未命中或未命中率不适合用于指导CP。因此，他们使用“平均内存访问时间”(AMAT)来指导CP，其计算方法是内存指令消耗的周期数除以内存指令数。在他们的技术中，操作系统为每个程序绑定一个辅助线程。基于当前和以前时期分配给线程的L2路和L3路的性能，辅助线程为每个线程构建一个性能模型，显示不同数量的L2路和L3路的AMAT。这是通过使用回归方法拟合表面模型来完成的。基于该模型，可以找到最慢和最快的线程(根据AMAT)，如果它们的性能相差超过阈值，缓存路将从最快的线程传输到最慢的线程。他们的CPT优于共享缓存、相等CP、最佳静态CP、仅L2分区或仅L3分区以及基于独立性能模型的L2和L3不协调分区。

表九。持续专业培训与其他方法的整合/互动

种类	参考
处理器分区	[Srikantiah 等人。2009a; 2016]
动态随机存取存储器存储体分区	[刘等。2014]
动态随机存取存储器-带宽划分	[Lo 等人。2016; 2011; 2010; 2008;2016]
DVFS	[王和马丁内斯 2015; 2016; 2008]
缓存块大小选择	[古普塔和周 2015]
替换策略选择	[詹等。2014]
预取	[hasenlaugh 等人 2012; 2008; 2015]
缓存锁定	[苏亨德拉和米特拉 2008]
网络流量管理	[Lo 等人。2016]

7.4. 相变存储器主存储器环境下的相变存储器

对于带 PCM 主存储器的处理器，周等。[2012] 提出一种 CPT，寻求最大限度地减少未命中和写回，因为 PCM 具有高写能量/延迟和低写耐久性[Mittal 等人。2015, 2014b]. 他们注意到写访问可能会导致写回。此外，如果脏块保留在缓存中，直到对同一块的下一次写入，写回也不会发生。它们将写操作 Wi 到块 B 的“写回避免距离”(WAD)定义为在 Wi 和下一次写操作 B 之间对 B 的所有访问的最高堆栈距离。因此，如果高速缓存关联性小于其 WAD，写操作将导致写回。此外，类似于 LRU 的堆栈属性，不会导致在 M 路高速缓存中写回的写也不会导致在多于 M 路的高速缓存中写回。基于这些，他们扩展了实用监视器[库雷希和帕蒂 2006] 为了跟踪脏块和 WAD，并使用它们，他们估计不同数量的缓存路的写回。由此，他们的 CPT 找到了一个分区，用于最大化可避免的写回次数和命中次数的加权和。通过改变权重，他们的 CPT 可以优化不同的目标，例如吞吐量、PCM 能量等。由于不同应用的最佳权重值不同，他们建议根据写队列拥塞和写队列占用率定期调整这些权重。通过减少写回，与共享缓存和不知道写回的 CPT 相比，它们的 CPT 在吞吐量、能效和 PCM 寿命方面带来了巨大的改进。

8. CPT 与其他技术的交互/集成

一些研究人员将持续专业技术与其他管理方法相结合，或者研究它们之间的相互作用，以利用它们之间的协同作用，并使更广泛的应用受益。桌子 IX 总结了这些工作，我们现在讨论其中的一些。

8.1. 与处理器分区的集成

Srikantaiah 等人[2009a] 请注意，当应用程序之间的“线程级并行性”(TLP)程度差异很大时，在它们之间平均分配处理器可能不会带来最佳性能。此外，应用程序对计算和内存资源的要求各不相同，因此，它们建议以集成方式执行“处理器分区”(PP)和 CP(参见图 12). 因为 PP 和 CP 的可能组合的数量非常大，所以他们使用两种见解来修剪搜索空间。首先，与竞争对手相比，竞争对手对公平加速有更大的影响，基于此，他们使用迭代方法，竞争对手先完成竞争对手，然后是竞争对手。这也允许将任何 CP 方案与 PP 方案一起使用。第二，公平加速和 QoS 度量之间有很强的相关性(即，应用程序性能的下降不应超过阈值，例如 5%)，基于此，在 QoS 度量上被认为不可接受的分区可以被移除，而不会丢失好的候选

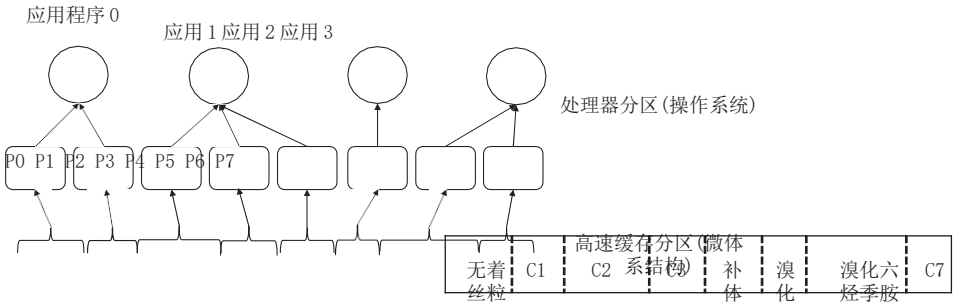

图 12。具有 4 个应用程序、8 个处理器和共享 LLC 的协同 CP 和 PP 的示例[srikantiah 等人。2009a].

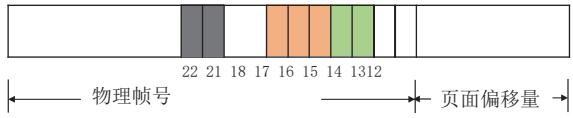

图 13。在具有 8GB 内存和 64 个存储体的处理器上，仅存储体(21-22)、仅缓存(16-18)和重叠(14-15)位[Liu 等人。2014].

公平加速度量。他们评估了两种情况下的应用程序 IPCs:处理器的相等分区和通过在相等分区中交替增加和减少处理器份额而形成的“交替分区”。这些值被馈送到回归预测器，该预测器使用“最小二乘误差”方法来找到任何处理器份额的 IPC。在此基础上，找到了基于公平加速度量的最优进程，而基于服务质量度量的剪枝是不可接受的。类似地，对于 CP，他们首先记录相等和交替的高速缓存分区的 IPC，然后使用回归预测器找到任何高速缓存共享下的 IPC。利用该信息，在不违反服务质量约束的情况下，找到最优合作伙伴。与 CP (10M 周期)相比，PP 以更粗的粒度(65M 周期)完成，因此，可以为同一个 PP 实施多个缓存分区。他们表明，他们的技术在公平加速方面比操作系统实施的隐式分区和相等分区表现得更好。此外，集成 PP 和 CP 比单独使用它们中的任何一个都更好。

8.2. 与动态随机存取存储器库分区的集成

刘等。[2014] 提出了一种“垂直分区”方法，将有限责任公司和动态随机存取存储器库进行分区，以满足各种应用的高速缓存/内存需求。他们注意到，在物理地址中，用于计算有限责任公司集合索引和用于计算动态随机存取存储器组的比特很少(例如，两个)是常见的。这如图所示 13. 这些公共位允许对高速缓存和存储体进行分区，并且通过使用这些位以及来自存储体或仅高速缓存位的位之一，可以获得具有不同分区粒度的 VP 技术。

它们表明，对于多种工作负载，VP 比仅缓存/存储体分区提供更高的性能。此外，对于具有高数据共享的 MT 应用程序，随机交错页面分配的性能优于任何分区。他们在大量工作负载上评估这些技术，并使用数据挖掘方法来确定分区和合并规则。例如，一个分区规则是，具有 LLC 抖动和其他类型应用程序组合的工作负载应该仅使用 VP 控制公共位，或者使用 VP 控制具有缓存位的公共位。类似地，具有高和中等缓存密度应用程序但没有的工作负载

抖动应用程序应该使用仅存储体分区。机器翻译应用程序使用随机页面分配。为了充分利用分区和共享，它们形成了合并规则，允许合并某些应用程序类型的分区。例如，高和中等缓存强度应用程序的分区是共享的，而缓存调整和抖动应用程序的分区是共享的。因此，灵活的划分和合并允许考虑应用程序的任意开始和终止。他们在Linux内核中实现了他们的VP方法，并表明它比未修改的Linux以及仅缓存和仅银行分区提供了更高的性能。

8.3. 与带宽划分的集成

刘等。[2010]研究CP和BW分配的相互作用。他们制定了一个BW分区的分析模型，该模型考虑了许多L2未命中、指令和内存级别的并行性、排队延迟、来自多个内核的争用等等。从这个模型中，他们观察到带宽划分是否能够提高性能取决于应用之间的未命中频率的差异，并且随着带宽的减少，性能提高的范围增加。因此，通过减少应用程序未命中频率的差异和减少总缓存未命中，CP可以降低带宽分区对性能的影响，从而缓解带宽压力。然而，如果CP增加了未命中频率的差异，那么它会增加带宽划分对性能的影响。因此，例如，对于高速缓存不敏感的应用，CP不能提高性能，但是通过改变未命中频率的差异，CP增强了带宽划分在提高性能方面的有效性。他们的微体系结构模拟实验证实了从分析模型中获得的见解。

8.4. 与DVFS计划相结合

王与马丁内斯[2015]将应用程序之间的共享资源(芯片功率预算和LLC容量)划分问题建模为动态分布式市场，其中在一个核心上运行的每个应用程序都是一个代理，资源价格根据“需求”和“供应”而变化最初，每个代理都有购买资源的预算，并且它开发了一个性能模型作为分配资源的函数。全局仲裁器确定所有资源的初始价格，每个代理对资源进行竞价，以最大化其效用(性能)。基于所有这些出价，仲裁器分别提高和降低高需求和低需求资源的价格。代理人在新的价格下再次出价，这种迭代过程在市场收敛时停止，也就是说，当迭代中的价格变化非常小时(或者已经执行了阈值次数的迭代)，代理人的出价变化不会提高其自身的效用。此时，执行资源分配。因此，与其他只考虑资源边际效用的技术相比，他们的技术也考虑了资源的竞争程度。因此，如果资源“P”定价高(即，竞争激烈)，那么代理将开始对更便宜的资源(即，竞争较少的资源)进行竞价，即使“P”可以为代理提供更高的“边际效用”。所有代理都以分散的方式工作，他们技术中唯一集中的功能是定价方案，这相对简单。此外，为了优化吞吐量，可以将更大的预算分配给具有更大边际效用的代理，并且为了优化公平性，可以为所有代理分配相等的预算。他们通过将执行时间划分为计算和内存阶段，使用采样的ATD估计不同方式下的未命中，并为每个未命中计数找到内存阶段的长度来计算缓存效用值。他们还利用DVFS发现频率变化的功率效用值。他们表明他们的技术在提高产量方面是有效的

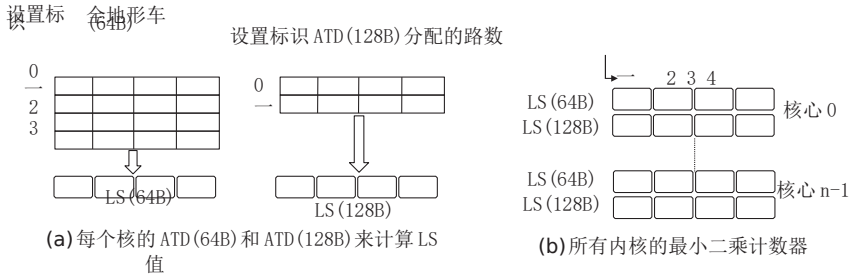

图 14。不同区块大小的每核心 ATD(省略 256B 和 512B 大小)[Gupta 和 Zhou2015].

公平性，并通过使用分散方法扩展到大量内核(例如 256 个)。

8.5. 与高速缓存块大小选择的集成

古普塔和周[2015] 请注意，对于几个内存密集型应用程序，使用大块大小会显著降低容量需求，因此，工作集大小(或容量需求)在很大程度上取决于块大小。因此，与具有较小块大小的大得多的高速缓存相比，具有较大块大小的较小高速缓存可以提供更高的性能。因此，对这种具有强局部性的应用程序使用大块大小允许它们将容量捐赠给其他应用程序，这可以提高具有良好时间局部性的应用程序的性能。基于这些见解，古普塔和周[2015] 提出一种动态改变应用程序的缓存块大小和缓存配额的技术，以利用空间和时间局部性来提高性能。对于时间和空间局部性的统一测量，他们计算“局部性得分”(LS)，对于最近访问的地址，它测量相邻地址在不久的将来被访问的概率。LS 取决于邻域的大小(即缓存块大小)和近期窗口(即重用距离)。对于每个内核，他们使用 ATDs 来计算每个块大小(64B、128B、256B 和 512B)的 LS 值，如图所示 14.由于最小二乘指示命中率，为了提高吞吐量，他们的 CP 算法寻求在总缓存容量的约束下最大化所有正在运行的应用程序的最小二乘值的加权和。任何应用程序的权重都是其 LLC 访问速率。对于每个应用程序，他们的算法为每个可能的配额分配确定合适的块大小。只有当由它产生的额外内存流量被最小二乘的显著改进证明是合理的时，才选择更高的块大小。然后，“前瞻算法”[库雷希和派特 2006]在每次迭代中，一种方法被分配给一个核心，该核心显示每单位容量的局部性增加最大。使用这种方法，他们的技术可以找到配额分配和相应的块大小。与纯 CP 技术相比，它们的技术通过协同执行 CP 和块大小选择来提供更高的性能。

8.6. 与缓存替换策略选择集成

詹等。[2014] 请注意 CPT 和抗鞭打 RPs(例如，BIP 库雷希等人。[2007],线程感知 DIP[Jaliel 等人。2008])具有互补的性质。RPs 基于应用程序的“局部性”在时间上共享 LLC，而 CPT 基于应用程序的“实用性”在空间上划分 LLC 因此，旨在避免抖动的 RPs 适用于局部性较差的应用程序，CPT 适用于效用值差异较大的应用程序。詹等。[2014] 建议结合快速原型选择执行持续生产测试，以优化局部性和实用性。他们使用两个监测单元以不同的方式寻找命中数:一个是 LRU 的 ATD 样本，另一个是 BIP 的。由于 BIP 不服从堆栈属性，

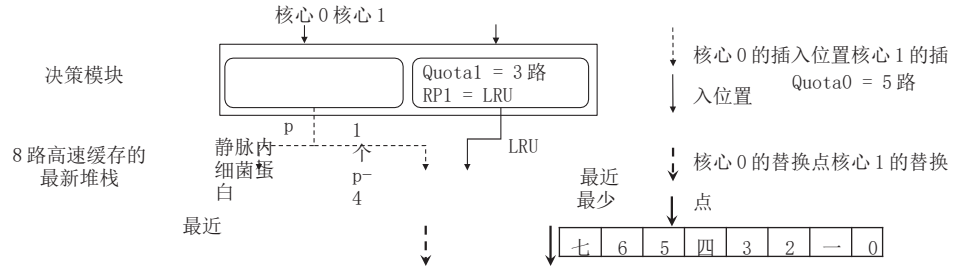

图 15。确定最佳 RP 和 CP[詹等。2014].

其监控单元根据其特性进行了不同的设计[詹等人。2014].本单元使用对数 2(WC)抽样的关联度为 2，4，...，WC /2 和 WC，使得关联性 w 的 ATD 显示在 BIP 下的 w 方向命中。通过对上述值进行线性插值，推导出其他路计数下的 BIP 命中率。基于从这些监控单元获得的效用和位置信息，他们的技术确定了最佳 CP 和最佳 RP，如图所示 15. 对于 CP，他们使用“前瞻算法”[Qureshi 和 Patt2006]，除了 MMU 是根据从 LRU 和 BIP 命中曲线的较高部分生成的命中曲线计算的。为核心选择的 RP 在其缓存部分中实现，至于提升策略，块看到命中由单个位置提升[谢和 Loh2009]. 由于利用了最佳路由协议提供的最大效用，与其他路由协议和路由协议相比，他们的技术提供了更高的吞吐量和公平性。

8.7. 与缓存预取的交互

Hasenplaugh 等人[2012] 使用梯度下降算法在不同线程和预取器之间划分缓存。他们的技术为每个线程计算一个阈值，该阈值决定了每个线程以高于正常的替换概率插入的内存访问的比例。为此，预取器也被建模为线程。每个线程被分配一组缓存集，称为“梯度区域”在其中一半的集合中，他们的技术试验了给线程分配比当前更高和更低的缓存配额。如果其中一半的性能明显优于另一半，那么线程的阈值将被调整，这将随后改变线程的缓存配额。用于确定较好的一半的评估度量可以是未命中率、IPC(线程的瞬时吞吐量)或用于通用优化的加权 IPC，并且通过选择加权 IPC 度量中的权重，可以实现服务质量目标。利用这种爬山方法，他们的技术试图通过做出局部决策来实现全局最优分割。他们的 CPT 还可以根据预取器带来的块的有用性来适应预取器的攻击性。他们的技术提高了吞吐量，同时实现了服务质量目标。

8.8. 与缓存锁定的集成

苏亨德拉和米特拉[2008] 评估多核系统共享 L2 缓存中基于集合的 CP 与缓存锁定的交互，并研究它们对应用程序 WCET(最坏情况下的执行时间)的影响。他们比较了锁定/分区的以下组合:(a)“静态/无分区”(b)“静态/核心级”和(c)“动态/核心级”(参见图 16).“在核心级 CP 中，在该核心上执行的每个任务都可以占用分配给该核心的全部配额。他们发现静态/核心级方案优于静态/无分区方案，因为在前一种方案中只有活动任务占用缓存配额，而在静态/无分区方案中即使空闲任务也使用缓存空间。然而，在每次抢占时，静态/核心级方案都需要缓存

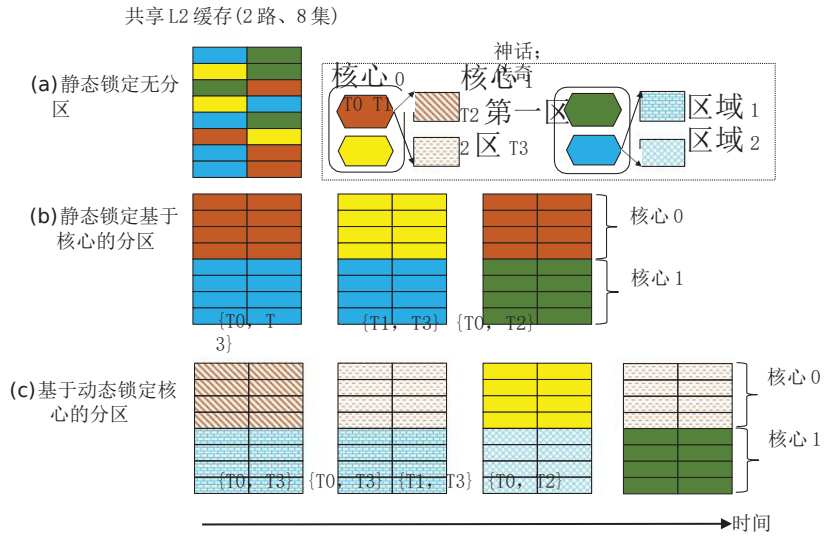

图 16。各种 CP 和锁定方法及其集成[苏亨德拉和米特拉 2008]. T0 到 T3 是四个任务。T0 和 T3 被分成两个区域，每个区域用于动态锁定。

分区重载和锁定，这是它的缺点。至于动态/核心级和静态/核心级方案之间的比较，前者对于具有许多区域的任务是有利的，而后者更容易实现。他们的研究为开发实时系统中的缓存管理技术提供了见解。

9. 未来的方向和结论

为了应对日益多样化的工作负载、使用模式和优化目标，系统设计人员正在转向异构处理器，例如 CPU- GPU 异构系统[米塔尔和维特尔 2015]. 由于微体系结构和运行在中央处理器和图形处理器上的工作负载之间的根本差异，在它们之间划分共享缓存带来了与多核中央处理器不同的挑战和机遇。显然，将现有的处理器扩展到 CPU-GPU 处理器并为其设计新的处理器将是研究人员感兴趣的下一步，尤其是当这些处理器逐渐成为主流计算系统时。

现代处理器采用多种缓存管理方法，例如，电源门控、数据压缩、缓存预取、缓存旁路等。CP 在现代处理器中的应用将取决于它与这些方法的协同集成。展望未来，明确需要详细研究 CP 与现有缓存管理方法的交互。

一些研究人员观察到，从模拟器和真实系统中获得的见解可能不同[库克等人。2013;林等。2008]. 它们的执行窗口/长度的差异、与其他因素的交互等等可能会导致截然不同的结论，例如，关于应用程序的缓存敏感性。有鉴于此，采用更严格的评估标准将促进可重复的研究，从而得出普遍适用的结论。

本文概述了多核系统中的缓存分区技术。为了强调不同技术的共同和独特特征，我们将它们分为许多类别，并回顾了它们的主要思想。希望本文的贡献不仅仅是综合了高速缓存分区研究的最新进展，还能激发这一领域的更多努力

分区在管理下一代计算系统缓存中的有效性。

参考

马努·阿瓦斯蒂，克希提伊·苏丹，拉吉耶夫·巴拉苏布拉莫尼安和异星战场。2009.动态硬件辅助软件控制的页面布局，用于管理大型缓存内的容量分配和共享。《高性能计算机体系结构国际研讨会论文集》(2009年，HPCA)。250–261.

内森·贝克曼和丹尼尔·桑切兹。2016.建模 LRU 以外的缓存性能。《高性能计算机体系结构国际研讨会论文集》(HPCA'16)。

拉马赞·比蒂尔根、恩金·伊佩克和何塞·马丁内斯。2008.芯片多处理器中多个交互资源的协调管理:一种机器学习方法。国际微体系结构研讨会论文集。318–329.

雅各布·布洛克、·叶、陈丁、·李、和·罗。2015.最佳缓存分区共享。在国际并行处理会议记录(ICPP'15)中。749– 758.

J.常和苏世杰。2007.芯片多处理器的协同缓存分区。在国际超级计算会议记录中。242–252.

张继川和古林达尔·苏希。2006.芯片多处理器的协同缓存。《计算机体系结构国际研讨会论文集》(06年，ISCA)。264–276.

德里克·乔、普拉巴特·贾恩、拉里·鲁道夫和斯里尼瓦斯·德瓦达斯。2000.使用软件控制缓存的嵌入式系统专用内存管理。设计自动化会议记录。416–419.

帕特·康威、内森·卡利亚纳松达拉姆、格雷格·唐利、凯文·莱帕克和比尔·休斯。2010.AMD 皓龙处理器的缓存层次和内存子系统。IEEE 微。30, 2 (2010), 16–29.

亨利·库克、米克尔·莫雷托、莎拉·伯德、坎赫·道、大卫·帕特森和克斯特·阿桑诺维奇。2013.对缓存分区进行硬件评估，以提高利用率和能效，同时保持响应能力。《计算机体系结构国际研讨会论文集》(ISCA'13)。308–319.

崔泽涵，陈立成，云冈宝，陈明宇。2014.一种基于交换的缓存集索引方案，利用超页和页面着色优化。设计自动化会议记录。1–6.

南多昂、达利·赵、泰苏·金、罗萨里奥·卡姆马罗塔、马特奥·瓦莱罗和亚历山大·维登鲍姆。2012.使用动态重用距离改进缓存管理策略。国际微体系结构研讨会论文集。389–400.

哈康·迪布达尔和佩尔·斯滕斯特伦。2007.一种适用于芯片多处理器的自适应共享/私有 NUCA 高速缓存分区方案。《高性能计算机体系结构国际研讨会论文集》(07 年，HPCA)。2–12.

绍拉布·古普塔和惠阳周。2015.空间位置感知缓存分区，实现有效的缓存共享。在国际并行处理会议记录(ICPP'15)中。150– 159.

普拉泰克·哈尔韦、希尔申杜·达斯和赫曼格·卡普尔。2013.使用受控缓存分区实现更好的缓存利用率。在 2013 年 IEEE 第 11 届可靠、自主和安全计算国际会议记录(DASC'13)中。179–186.

威廉·哈森堡、普里帕尔·阿胡佳、阿默·贾利尔、小西蒙·斯蒂尔和乔尔·埃默。2012.基于梯度的缓存分区算法。ACM Trans。建筑师。代码 Optim。(TACO) 8，4 (2012)，44。

安德鲁·赫尔德里奇、埃德温·维尔普兰克、普里亚·奥蒂、拉梅什·伊利卡尔、克里斯·吉诺斯、罗纳克·辛哈尔和拉维·伊耶。2016.缓存服务质量:在英特尔至强处理器 E5-2600 v3 产品家族中，从概念到现实。《高性能计算机体系结构国际研讨会论文集》(HPCA'16)。657–668.

恩里克·赫雷罗、何塞·冈萨雷斯和雷蒙·卡纳尔。2010.弹性协同高速缓存:一种用于芯片多处理器的自主动态自适应存储器体系。《计算机体系结构国际研讨会论文集》(ISCA 10 年版)。419–428.

丽莎·徐，史蒂文·莱因哈特，拉维尚卡尔·伊尔和斯里哈里·马基内尼。2006.CMPs 上的共产主义、功利主义和资本主义缓存政策:缓存作为共享资源。在并行体系结构和编译技术国际会议记录(PACT'06)中。13–22.

拉维·伊尔。2004.CQoS:在 CMP 平台的共享缓存中启用 QoS 的框架。在国际超级计算会议记录中。257–266.

拉维·伊尔、李钊、郭飞、拉梅什·伊利卡尔、斯里哈里·马基内尼、唐·纽厄尔、扬·索利欣、丽莎·许和史蒂夫·莱因哈特。2007.化学机械抛光平台中缓存/内存的服务质量策略和体系结构。表演。评价。版本 35，1(2007)，25–36。

阿默·贾雷尔、威廉·哈森堡、莫因丁·库雷希、朱利安·塞伯特、小西蒙·斯蒂尔和乔尔·埃默。2008.用于管理共享缓存的自适应插入策略。国际并行体系结构和编译技术会议论文集(PACT'08)。208–219.

金欣欣，陈浩刚，，，，，罗，。2009.虚拟化环境中的简单缓存分区方法。在 IEEE 应用程序并行和分布式处理国际研讨会会议录(ISPA'09)中。519– 524.

钟彬娴，金贤淑和郑敏景。2010.3D 堆叠多处理器系统中基于延迟感知实用程序的 NUCA 缓存分区。在超大规模集成电路片上系统会议(超大规模集成电路-系统芯片' 10)的会议记录中。125–130.

穆罕默德·坎德米尔、拉姆亚·普拉巴卡尔、穆斯塔法·卡拉科伊和·张。2011a。多程序工作负载的多层缓存分区。欧洲并行处理会议录。130–141.

Mahmut Kandemir，Taylan Yemliha 和 Emre Kultursay。2011b。一种基于助手线程的多线程应用动态缓存分区方案。设计自动化会议记录。954–959.

迪米特里斯·卡塞利迪斯、穆罕默德·费萨尔·伊克巴尔和利齐·库里安·约翰。2014.高性能多核系统共享末级缓存的缓存友好感知管理。IEEE Trans。电脑。63，4 (2014)，874–887.

迪米特里斯·卡塞利迪斯、j·斯图切利、陈坚和利齐·约翰。2010.一种用于大型化学机械抛光系统的使用非侵入式资源分析器的带宽感知内存子系统资源管理。《高性能计算机体系结构国际研讨会论文集》(HPCA'10)。1–11.

迪米特里斯·卡塞利迪斯、杰弗里·斯图切利和利齐·约翰。2009.面向多核架构的银行感知动态缓存分区。在国际并行处理会议记录(09 年 ICPP)中。18–25.

哈沙德·卡斯托尔和丹尼尔·桑切兹。2014.Ubik:针对延迟关键型工作负载的高效缓存共享和严格的服务质量。在美国计算机学会编程语言和操作系统架构支持国际会议记录中。729–742.

卡米尔·凯奇尔斯基、米克尔·莫雷托、弗朗西斯科·卡佐拉和马特奥·瓦莱罗。2010.使高速缓存分区算法适应伪 LRU 替换策略。在 2010 年 IEEE 并行与分布式处理国际研讨会(IPDPS'10)的会议记录中。1–12.

周欣宇·汗、阿拉·阿拉梅尔登、克里斯·威尔克森、奥努尔·穆特卢和丹尼尔·姬广亮·内兹。2014.通过利用读写差异提高缓存性能。《高性能计算机体系结构国际研讨会论文集》(HPCA'14)。

金圣贝姆、德鲁巴·钱德拉和颜索林。2004.芯片多处理器体系结构中的公平缓存共享和分区。在并行体系结构和编译技术国际会议记录(PACT'04)中。111–122.

I. Kotera、K. Abe、R. Egawa、H. Takizawa 和 H. Kobayashi。2011.CMPs 的功耗感知动态缓存分区。跨。HiPEAC (2011)，135–153。

Vivek Kozhikkottu、Abhisek Pan、Vijay Pai、Sujit Dey 和 Anand Raghunathan。2014.多线程程序的变异感知缓存分区。设计自动化会议记录。1–6.

李现代、赵桑杰和布鲁斯·查尔德斯。2011.云缓存:扩展和收缩私有缓存。《高性能计算机体系结构国际研讨会论文集》(HPCA'11)。219–230.

李在奎和金惠顺。2012.TAP:一种用于中央处理器-图形处理器异质结构的 TLP 感知高速缓存管理策略。《高性能计算机体系结构国际研讨会论文集》(HPCA'12)。1–12.

J.林，陆青，丁，张，张，萨达亚潘。2008.深入了解多核缓存分区:弥合模拟系统和真实系统之间的差距。《高性能计算机体系结构国际研讨会论文集》(08 年，HPCA)。367–378.

J.林，陆青，丁，张，张，萨达亚潘。2009.借助轻量级硬件支持实现软件多核缓存管理。在超级计算会议记录中。

邢林和拉吉耶夫·巴拉苏布拉莫尼。2011.完善基于实用程序的缓存分区的实用程序度量。与第 38 届计算机体系结构国际研讨会(ISCA-38) (2011 年)同时举行的第九届复制、解构和拆密年度研讨会。

方柳，江，颜索林。2010.了解芯片多处理器中的片外内存带宽划分如何影响系统性能。《高性能计算机体系结构国际研讨会论文集》(HPCA'10)。1－12.

、李勇、崔泽涵、云冈宝、陈明宇、武。2014.内存管理垂直化:通过多策略处理多样性。《计算机体系结构国际研讨会论文集》(ISCA'14)。169－180.

卢大伟、程立群、拉玛·戈文达拉朱、帕萨萨拉西·阮冈纳赞和克里斯特斯·科济拉基斯。2016.利用 heracles 大规模提高资源效率。ACM Trans。电脑。系统。(TOCS) 34 票、2 票(2016 年)6 票。

理查德·马特森、简·格策、唐纳德·斯卢茨和欧文·特拉杰。1970.存储层次的评估技术。《IBM 系统杂志》9，2 (1970)，78－117。

英特尔公司。2016.英特尔 64 和 IA-32 架构开发人员手册:3B 卷，系统编程指南，第 2 部分。检索自 http://goo.gl/sw24WL.

OSU-CSE 新闻。2010.英特尔把 OSU-CSE 放在里面。检索自 http://web.cse.ohio-state.edu/news/news118.shtml.

梵持·梅克特、阿努普·霍利、潘钟耀和安东尼娅·翟。2013.管理异构多核处理器中的共享末级缓存。在并行结构和编译技术国际会议记录(PACT'13)中。225－234.

斯巴塞米塔尔。2016a。管理过程变化的体系结构技术综述。电脑。调查 48，4 (2016)，54:1－54:29.

斯巴塞米塔尔。2016b。缓存旁路技术综述。低功耗选举。应用 6，2 (2016)，5:1－5:30。

斯巴塞米塔尔、延安的曹和。2014a。MASTER:一种使用动态缓存重新配置的多核缓存节能技术。IEEE Trans。VLSI Syst。22，8 (2014)，1653－1665.

斯巴塞·米塔尔、马修·波伦巴、杰弗里·维特尔和袁谢。2014b。用命运工具探索三维非易失性存储器和电子数据存储器的设计空间。技术报告 ORNL/TM-2014/636。美国橡树岭国家实验室。

斯巴塞米塔尔和杰弗里维特尔。2015.CPU-GPU 异构计算技术综述。电脑。*调查 47，4 (2015)，69:1－69:35。*

斯巴塞米塔尔和杰弗里维特尔。2016.动态随机存取存储器架构技术综述。IEEE Trans。*平行。发行版。系统。(TPDS) 27，6 (2016)，1852－1863。*

斯巴塞·米塔尔、杰弗里·维特尔和李东。2015.管理嵌入式动态随机存取存储器和非易失性片上高速缓存的体系结构方法综述。IEEE Trans。并行发行版。系统。(TPDS) 26，6 (2015)，1524－1537。

斯巴塞米塔尔和张钊。2013.管理器:面向服务质量系统的多核共享缓存节能技术。技术报告。爱荷华州立大学。

米克尔·莫雷托、弗朗西斯科·卡佐拉、亚历克斯·拉米雷斯、里索斯·萨科拉里欧和马特奥·瓦莱罗。2009.FlexDCP:一个面向 CMP 架构的服务质量框架。ACM SIGOPS Operat。系统。版本 43，2 (2009)，86－96。

米克尔·莫雷托、弗朗西斯科·卡佐拉、亚历克斯·拉米雷斯和马特奥·瓦莱罗。2008.MLP 感知动态缓存分区。在高性能嵌入式架构和编译器中。337－352.

Sai Prashanth Muralidhara、Mahmut Kandemir 和 Padma Raghavan。2010.应用程序内缓存部分。在 2010 年 IEEE 并行与分布式处理国际研讨会(IPDPS'10)的会议记录中。1－12.

康斯坦丁诺斯·尼卡斯、马修·霍斯内尔和吉姆·加赛德。2008.一种适用于多核架构的自适应布隆过滤器缓存划分方案。《嵌入式计算机系统国际会议论文集:体系结构、建模和仿真》(SAMOS'08)。25－32.

吴泰哲，李绮妍和赵三野。2011.末级缓存和带宽共享共同管理的分析性能模型。在 2011 年 IEEE 第 19 届计算机和电信系统建模、分析和仿真年度国际研讨会(MAS- COTS'11)的会议记录中。150－158.

潘和维贾伊·派。2013.平衡数据并行程序的不平衡缓存分区。在年度 IEEE/ACM 微体系结构国际研讨会(MICRO'13)的会议记录中。297－309.

帕夫洛斯·佩图梅诺斯、圣乔治·克拉米达斯、哈坎·泽弗、斯特凡诺斯·卡希拉斯和埃里克·黑格斯滕。2006.Stat- Share:通过衰减管理缓存共享的统计模型。《建模、基准测试和模拟研讨会论文集》(2006 年)。

米克尔·莫雷托·普拉纳斯、弗朗西斯科·卡佐拉、亚历克斯·拉米雷斯和马特奥·瓦莱罗。2007.解释动态缓存分区加速。IEEE Comput。阿奇。列特。6，1 (2007)，1－4.

美国计算机学会计算调查，第 50 卷，第 2 期，第 27 条，出版日期:2017 年 5 月。

库雷希，贾勒尔，帕特，斯蒂尔和埃默。2007.高性能缓存的自适应插入策略。《计算机体系结构国际研讨会论文集》(2007)，第381-391页。

库雷希和耶鲁帕蒂。2006.基于实用程序的缓存分区:一种低开销、高性能、运行时的共享缓存分区机制。在第39届年度IEEE/ACM微体系结构国际研讨会会议录中。423－432.

瑙曼·拉菲克、元泰克·林和米楚娜·托特索迪。2006.对操作系统驱动的CMP缓存管理的架构支持。并行体系结构和编译技术国际会议论文集。2－12.

R.雷迪和彼得罗夫。2010.高速缓存分区支持高能效、无干扰的嵌入式多任务处理。ACM Trans。嵌入。电脑。系统。(TECS) 9，3 (2010)，16。

丹尼尔·桑切兹和克里斯特斯·科济拉基斯。2010.零缓存:解耦方式和关联性。国际微体系结构研讨会论文集。187－198.

D.桑切斯和C. Kozyrakis。2011.优势:可扩展且高效的细粒度缓存分区。计算机体系结构国际研讨会论文集。57－68.

亚历克斯·结算，丹·康纳斯，恩里克·吉伯特，安东尼奥·冈萨雷斯。2006.多线程处理器的动态可重构高速缓存。j .嵌入。电脑。2，2 (2006)，221－233.

Shekhar Srikantaiah、Reetuparna Das、Asit K. Mishra、Chita R. Das和Mahmut Kandemir。2009a。芯片多处理器中集成处理器-高速缓存分区的一个例子。在高性能计算网络、存储和分析会议记录(SC'09)中。6.

谢哈尔·斯里坎塔亚、马赫穆特·坎德米尔和钱旺。2009b。夏普控制:芯片多处理器中受控的共享高速缓存管理。在第42届IEEE/ACM国际微体系结构研讨会(MICRO-42'09)的会议记录中。517－528.

H.斯通、图雷克和沃尔夫。1992.高速缓冲存储器的最佳分区。IEEE Trans。电脑。41，9 (1992)，1054－1068.

拉瓦尼亚·苏布拉马尼安、维韦克·塞沙德里、阿纳·戈什、周欣宇·汗和乌努尔·穆特卢。2015.应用程序减速模型:量化和控制应用程序间干扰对共享缓存和主内存的影响。在一年一度的IEEE/ACM国际微建筑研讨会(MICRO'15)的会议记录中。62－75.

G.爱德华·苏、斯里尼瓦斯·德瓦达斯和拉里·鲁道夫。2001.分析缓存模型及其在缓存分区中的应用。在国际超级计算会议记录中。1－12.

G.苏、鲁道夫和德瓦达斯。2004.共享高速缓冲存储器的动态分区。超级计算。28，1 (2004)，7－26.

余薇薇·苏亨德拉和图利卡·米特拉。2008.探索多核上可预测共享缓存的锁定和分区。设计自动化会议记录。300－303.

卡尔提克·孙达拉扬、瓦西利奥斯·波波多斯、蒂莫西·琼斯、奈杰尔·托法姆和比约恩·弗兰克。2012.合作分区:高性能CMPs的高能效缓存分区。《高性能计算机体系结构国际研讨会论文集》(2012年)，第1-12页。

大卫·塔姆、雷扎·阿齐米、利维奥·苏亚雷斯和迈克尔·斯图姆。2007.在软件中管理多核系统上的共享L2缓存。《操作系统和计算机体系结构之间的相互作用研讨会论文集》。26－33.

Keshavan Varadarajan、S. K. Nandy、Vishal Sharda、Amrutur Bharadwaj、Ravi Iyer、Srihari Makineni和Donald Newell。2006.分子缓存:一种动态创建特定应用异构缓存区域的缓存结构。在年度IEEE/ACM微体系结构国际研讨会会议录(MICRO'06)中。433－442.

王瑞生、陈立忠。2014.无用扩展:高关联性缓存分区。在年度IEEE/ACM微体系结构国际研讨会(MICRO'14)的进程中。356－367.

、凯妈、王。2012.功率受限芯片多处理器中用于应用公平性或差异化的缓存延迟控制。 IEEE Trans。电脑。61，10 (2012)，1371－1385.

王晓东和何塞·马丁内斯。2015.交换:多核架构中基于市场的可扩展动态多资源分配方法。《高性能计算机体系结构国际研讨会论文集》(HPCA'15)。113－125.

Y.谢和陆。2009.PIPP:多核共享缓存的提升/插入伪分区。在…里
计算机体系结构新闻，第37卷。ACM，174－183。

谢和陆。2010.通过包含颠簸工作负载的可扩展共享缓存管理。高性能嵌入式体系结构和编译器国际会议论文集。262－276.

叶英、理查德·韦斯特、程卓群和叶莉。2014.Coloris:一个使用页面着色的动态缓存分区系统。国际并行体系结构和编译会议录。381–392.

托马斯·叶和格伦·赖曼。2005.快速和公平:数据流服务质量。《嵌入式系统编译器、架构和综合国际会议论文集》(CASES'05)。237–248.

余和彼得·彼得罗夫。2010.多核平台通过缓存分区实现片外内存带宽最小化。设计自动化会议记录。132–137.

希澈·云和普拉塔普·库马尔·瓦尔桑。2015.评估 COTS 多核平台上缓存分区的隔离效果。《第 11 届嵌入式实时应用操作系统平台年度研讨会论文集》(OSPERT'15)。45.

詹东源，洪江，和沙瑞德·c·赛斯。2014.CLU:在共享末级缓存的线程感知容量管理中，协同优化局部性和实用性。IEEE Trans。电脑。63，7 (2014)，1656–1667.

小张，桑迪亚矮人和沈凯。2009.走向实用的基于页面着色的多核缓存管理。欧洲计算机系统会议录。89–102.

周淼、杜钰、布鲁斯·查尔德斯、拉米·梅尔赫姆和丹尼尔·莫斯。2012.相变主存系统中末级缓存的写回感知分区和替换。ACM Trans。阿奇。代码 Optim。(TACO) 8，4 (2012)，53。

周淼、杜钰、布鲁斯·查尔德斯、丹尼尔·莫斯和拉米·梅尔赫姆。2016.多核系统中内存层次结构的对称不可知协调管理。ACM Trans。阿奇。代码 Optim。(TACO) 12，4 (2016)，61。

2016 年 7 月收到；2017 年 1 月修订；2017 年 3 月接受