并发与计算:实践与经验
并发计算。:Pract。Exper。2016;00:1 - 35
在线发表于 Wiley InterScience(www . InterScience . Wiley . com)。DOI: 10.1002/cpe

高层建筑设计技术综述

斯巴塞 • 米塔尔*

海德拉巴印度理工学院

摘要

“转换后备缓冲器”(TLB)缓存虚拟到物理地址转换信息，用于从嵌入式设备到高端服务器的系统中。由于 TLB 的访问非常频繁，而 TLB 的失误代价极高，因此谨慎管理 TLB 对于提高处理器的性能和能效非常重要。在本文中，我们介绍了构建和管理顶级域名的技术概况。我们在几个维度上描述了这些技术，以突出它们的相似性和区别。我们相信，本文将对芯片设计师、计算机架构师和系统工程师有用。版权所有 2016 约翰 • 威利父子有限公司

收到。。。

关键词:复习；分类；TLB；超页面；预取；电源管理；虚拟缓存；工作量特征描述

†

1. 介绍

虚拟到物理地址转换是计算系统中的一个关键操作，因为它是在具有物理高速缓存的处理器中的每个内存操作上执行的，并且是在具有虚拟高速缓存的处理器中的每个高速缓存未命中上执行的。为了利用内存访问局部性，具有基于页面的虚拟内存的处理器使用 TLB 缓存最近的翻译。因此，TLB 管理对于提高处理器效率变得至关重要。TLB 未命中需要昂贵的页面遍历，例如，在 4 级页表设计中，TLB 未命中可能需要多达 4 次存储器访问[1].因此，应用程序可能会花费很大一部分(例如，高达 50% [2,3】)的时间在 TLB 小姐搬运。

为了最小化 TLB 失败率和延迟，需要仔细选择设计参数，例如 TLB 尺寸、关联性(直接映射、模拟退火†、*或 FA)和层级数量(一个或两个)，然而，已知这些呈现严格的权衡。其他设计选项加剧了这些挑战，例如，在多核处理器中使用单个基本页面、超页面或多页面大小、私有或共享 TLB、预取等。虽然使用虚拟缓存可以降低 TLB 对性能的影响[*4】，他们提出了自己的挑战。很明显，智能技术

*通信地址:印度 IIT 海得拉巴斯巴塞米塔尔，印度特兰加纳桑加雷迪 502285。电子邮件:sparsh0mittal@gmail.com.

本文中经常使用以下缩写词:“地址空间 ID”(ASID)、“数据 TLB”(DTLB)、“先进先出”(FIFO)、“全关联”(FA)、“输入/输出”(IO)、“指令集体系结构”(ISA)、“指令 TLB”(ITLB)、“java 虚拟机”(JVM)、“实时”(JIT)、“最近最少使用的”(LRU)、“内存管理单元”(MMU)、“页表条目”(PTE)、“页表 walker”(PTW)、“物理地址”(PA)、“物理页号”(PPN)、“程序计数器”本文中，除非特别提到“L1 缓存”、“L1 数据缓存”或“L1$”等。，L1 和 L2 分别指 L1 • TLB 和 L2 • TLB。

优化 TLB 的设计和管理。最近，已经提出了几种技术来解决这些问题。

贡献:本文介绍了设计和管理顶级域名的技术概况。数字 1 介绍论文的结构。我们首先讨论管理顶级域名的权衡和障碍，并对顶级域名的研究工作进行分类和概述(2).然后，我们讨论关于 TLB 架构探索和工作负载表征的研究(3).此外，我们提出了用于改善 TLB 覆盖和性能的技术，例如，超老化、预取等。（4).我们还讨论了节约 TLB 能源的技术(5)，包括虚拟高速缓存设计(6).此外，我们回顾了 TLB 对图形处理器和中央处理器-图形处理器系统的设计建议(7).最后，我们简要讨论了未来的挑战(§8).

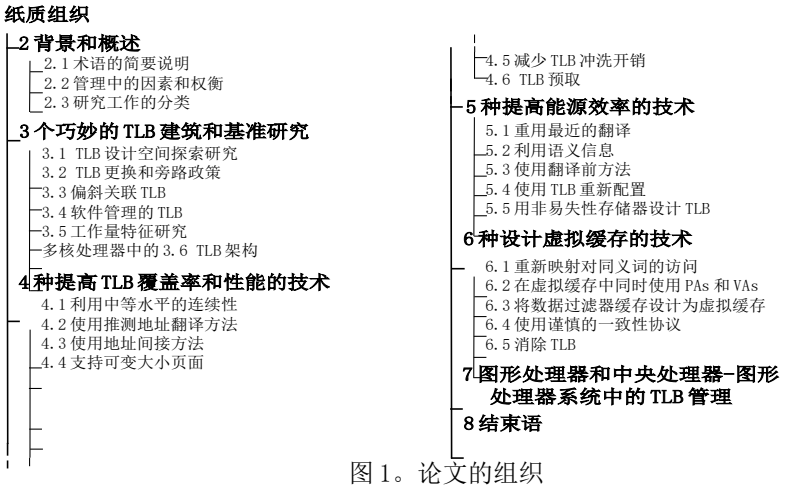


图 1。论文的组织

范围:TLB 管理与相关领域有重大重叠，如虚拟内存、页表管理、缓存管理；然而，为了简明起见，本文仅包括专注于 TLB 改进的作品。我们讨论了为中央处理器、图形处理器和中央处理器-图形处理器异构系统提出的技术。我们包括架构和系统级技术，而不是电路或应用级技术。我们专注于论文的关键见解，仅包括选定的量化结果，以显示改进的范围。我们相信这篇论文对研究人员和计算机架构师有价值。

2. 背景和概述

我们首先讨论在本文中有用的术语和概念(2.1).然后，我们讨论了设计和优化顶级域名所涉及的挑战(2.2).最后，我们沿着几个维度对研究工作进行分类，以提供该领域的概述(2.3).有关商用处理器中 TLB 配置的详细信息，请读者参考以前的工作[2,5–12].请注意，“地址转换缓存”和“目录后备表”是其他不常用的术语，用于指代 TLB [2].

2.1. 背景和术语

覆盖范围:覆盖范围(或范围或映射大小[13])显示了由其所有条目映射的内存总和。显然，对于具有大工作集的应用程序，覆盖率需要很高。例如，彭等[3]请注意，只有少数(例如 4 台)电脑造成大部分(例如> 70%)的未命中，可以通过提高 TLB 覆盖率来降低这些未命中。

超页面:超页面是指大小和对齐是系统页面大小(例如 4KB 或 8KB)两倍的幂的虚拟机页面，并映射到连续的物理页面。超级页面

对多个相邻的虚拟页面使用单个 TLB 条目，这提高了 TLB 覆盖率并减少了未命中。因此，超页面对于映射大对象是有用的，例如大数组、内核数据等。并且对于应对现代应用的不断增加的工作集特别有用。

TLB 击落:对事件，如页面交换，特权变更，上下文切换和页面重新映射等。，V2PA 映射更改。这使得翻译映射已经改变的 TLB 条目无效，这被称为 TLB 击落。在多核处理器中，失效消息需要发送到所有保存已更改条目的每个内核的 TLB，因此，中断会导致可伸缩性瓶颈[14].

硬件和软件管理的 TLB:由于 TLB 命中/未命中确定发生在关键路径上，因此它总是在硬件中实现。相比之下，TLB 缺失的处理可以用硬件或软件来实现。

在硬件管理的 TLB 中，在 TLB 未命中时，硬件状态机执行页表遍历，为给定的虚拟设备找到有效的 PTE。如果 PTE 存在，则将它插入 TLB，应用程序通过使用该 PTE 正常运行。如果没有找到有效的 PTE，硬件会引发页面错误异常，由操作系统处理。操作系统将请求的数据带入内存，设置一个 PTE 将请求的虚拟设备映射到正确的虚拟设备，然后恢复应用程序。因此，在硬件管理的 TLB，TLB 入口结构对软件是透明的，允许使用不同的处理器，同时仍然保持软件兼容性。硬件管理的 TLB 用于 x86 体系结构[15].

在软件管理的 TLB 中，在 TLB 未命中时，硬件引发“TLB 未命中”异常并跳转到陷阱处理程序。陷阱处理程序是操作系统中的一个代码，它通过启动页表遍历并在软件中执行转换来处理 TLB 缺失。页面故障(如果有)也由操作系统处理，类似于硬件管理的 TLB 的情况。具有软件管理的 TLB 的处理器在其 ISA 中有特定的指令，允许在任何 TLB 插槽中加载 pte。TLB 条目格式由国际海底管理局规定。TLB 的软件管理简化了硬件设计，并允许操作系统使用任何数据结构来实现页表，而不需要改变硬件。然而，软件管理也会招致更大的损失，例如由于管道冲洗。MIPS 和 SPARC 架构通常使用软件管理的顶级域名[15].

缓存寻址选项:对于 L1 缓存查找，索引和标记可以从 VA 或 PA 获得，因此有四种可能的寻址选项[4,9,16]，即物理/虚拟索引、物理/虚拟标记(PI/VI-PT/VT)，如图所示 2。请注意，PI-PT、VI- PT 和 PI-VT 设计要求对所有内存操作进行 TLB 访问。我们现在讨论这些寻址选项。

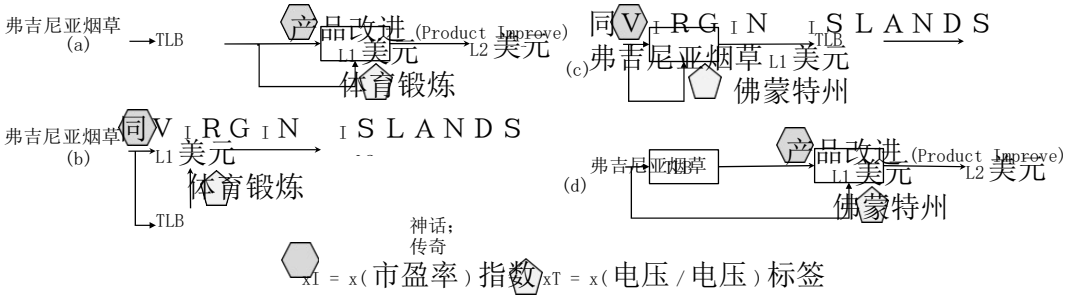


图 2。四种高速缓存寻址选项:(a) PI-PT，(b) VI-PT，(c) VI-VT 和(d) PI-VT。

(a) PI-PT:在 L1 缓存查找之前，索引和标签都是从 TLB 获得的，因此 TLB 位于关键路径。

(b) VI-PT: L1 缓存索引源自 VA，并行执行 TLB 查找。标签是从 TLB 提供的 PA 获得的，用于标签比较。与 PI- PT 设计相比，VI-PT 设计部分隐藏了 TLB 访问的延迟，因此被广泛使用。要使用虚拟地址执行索引，虚拟索引位应该与物理索引位相同，当从页面偏移量[17】，如图 3(b)。为此，缓存关联性应该大于或等于缓存大小除以页面大小。

例如，对于 32KB L1 缓存和 4KB 页面，关联性至少应为 8。然而，这增加了高速缓存能量消耗，但可能不会提供相应的未命中率降低。

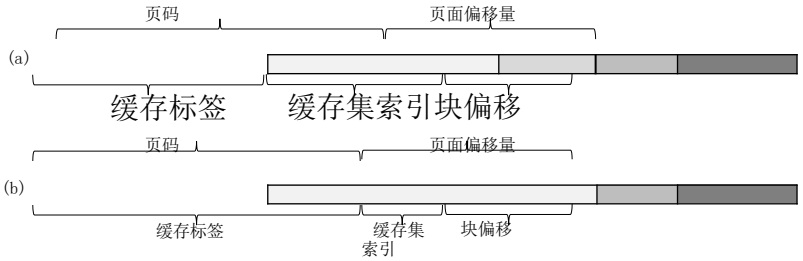


图 3。(a)根据物理地址中的位计算不同的值(一般情况)(b)完全根据页面偏移量计算高速缓存索引的特殊情况

(c) 虚拟磁带库:索引和标签都是从虚拟磁带库获得的。只有在 L1 缓存未命中时才访问 TLB(假设 L2 缓存是 PI-PT)。这种设计大致称为虚拟缓存。由于 L1 缓存命中率高，TLB 访问大大减少，尽管 TLB 访问现在位于 L1 缓存未命中的关键路径上。此外，需要对从虚拟 L1 缓存中逐出的块执行 V2PA 转换，以执行对物理 L2 缓存的写回。虚拟磁带库缓存的其他挑战将在第节中讨论 2.2。

(d) PI-VT:指数由 TLB 提供的 PA 计算得出。由于标记前需要索引，因此 TLB 在关键路径中，每次访问都需要 TLB 访问。显然，这种设计没有任何优势，因此很少使用[16].

同义词和同音异义词:映射到同一个 PA 的同一个进程中的不同 VAs 被称为同义词。当一个相同的虚拟变量存在于不同的进程中，每个进程映射到一个不同的虚拟变量时，它被称为同音异义词。

2.2. *TLB 管理中的因素和权衡*

架构参数的选择:对 TLB 访问的时间限制(例如，在 L1 缓存标签检查之前，但在虚拟仪器设计中计算虚拟仪器之后)对 TLB 设计提出了严格的要求。TLB 尺寸、关联性和层数等参数是众所周知的设计权衡。例如，使用 L0 TLB 可以减少访问等待时间，然而，L0 TLB 的小尺寸的要求导致高失败率，这需要挤压和重新发布许多指令[18].类似地，在多核处理器中，私有的每个内核的 TLB 减少了访问延迟，并且比共享的 TLB 更能扩展到大量内核。然而，共享 TLB 提供了容量优势，这可以减少 TLB 失误。虽然这些 TLB 设计权衡中的一些与高速缓存相似，但是它们不同的功能和优化目标导致了重要的差异
[19 因此，需要对 TLB 进行单独研究，以解决这些问题。

使用超级页面的挑战:页面大小的选择对 TLB 的有效性至关重要。虽然大页面减少了 TLB 未命中，但当仅使用大页面的一小部分时，它们会导致物理内存的浪费。此外，由于操作系统在页面粒度上跟踪内存使用情况，即使一个字节的变化也需要在修改内存映射文件时将整个页面写回辅助存储，这将导致巨大的输入输出流量。此外，大页面不允许使用页面颜色[20]，这是一种广泛使用的支持操作系统控制缓存的方法。此外，较大的页面需要在操作系统操作和内存管理方面进行不小的更改[11,21].相比之下，小页面支持细粒度内存管理和高能效操作。

使用可变页面大小的挑战:为了适应不同应用程序的内存访问模式，可能允许多种页面大小。例如，x86-64 支持 4KB、2MB 和 1GB(其中 1GB 大小未在所有系统中启用)页面大小[5].同样，Sparc、Power 和 Itanium 处理器允许多种页面大小。然而，利用这些页面大小会导致

更高的复杂性，并且当页面大小大于可用内存的相邻区域时，会导致碎片。此外，由于在TLB访问期间地址的页面大小是未知的，所以不能确定SA TLB中的集合索引。为了解决这个问题，需要使用FA TLB或每页一个TLB的大小，这会产生额外的开销并导致TLB空间的次优使用。此外，允许多种页面大小的TLB的未命中开销高于允许单个页面大小的TLB[11].

最小化TLB脱靶量开销:考虑到TLB脱靶量的巨大开销，需要智能策略来最小化这种开销。例如，一些处理器(如Sparc)使用软件管理的直接映射虚拟内存缓存，称为“转换存储缓冲区”(TSB) [22].TSB条目保存给定虚拟专用网络的翻译(PPN)。在ITLB或DTLB的航班上，TSB正在寻找丢失翻译的虚拟专用网络。命中导致从TSB到TLB的条目加载，而未命中则需要搜索哈希表。同样，改进PTW设计可以减少遗漏开销[1,23–27].然而，这些设计选择带来了它们自己的挑战和权衡。

最小化TLB刷新开销:在具有整合工作负载的多任务系统和虚拟化平台中，进程/任务之间的上下文切换需要TLB刷新。为了避免这种情况，虚拟机和/或进程特定的标签可以与每个TLB条目一起使用，这允许跨上下文切换保留它们。然而，这种标记TLB的方法使TLB成为一个共享资源，这导致了争论。

提高能效:在某些处理器中，TLB功耗可能接近芯片功耗的16 %[8,16].此外，由于TLB经常被访问，它也是一个热点。例如，Kadayif等人[28]显示ITLB的功率密度可以是7.8毫瓦/平方毫米，而DL1和IL1的功率密度分别是0.67和0.98毫瓦/平方毫米。因此，TLB电力管理至关重要。

虚拟高速缓存中的挑战:在虚拟高速缓存中，同义词可能存储在不同虚拟地址下的不同高速缓存集中，对一个块的修改会使其他块过时，而使同义词无效会破坏高速缓存的局部性。同样，同音异义词可能会导致访问错误的数据。此外，页面权限需要与每个块一起存储，并且在页面权限发生变化时，需要更新该页面中所有块的权限信息。类似地，在页面映射改变时，块的VA也需要改变。虚拟高速缓存的其他挑战包括保持与传统处理器架构和操作系统的兼容性[17]并确保高速缓存一致性[29].由于这些挑战，虚拟缓存很少在商业系统中使用。GPU中的TLB设计:由于其面向吞吐量的体系结构，GPU中的内存访问强度远高于CPU，并且由于GPU的锁步执行模型，单次未命中会导致整个warp的停滞[24].由于这些原因，TLB管理在GPU中更加重要。保持与中央处理器内存管理单元兼容性的要求提出了进一步的限制论图形处理器内存管理单元设计[26].

2.3. *研究工作的分类*

桌子I根据计算单元、评估平台和优化指标对作品进行分类，而表II根据作品的主要思想和方法对作品进行分类。因为真实系统和模拟器提供了互补的见解[30】，它们都是评估顶级域名不可或缺的工具。因此，表I还总结了不同作品所使用的评价平台。

我们现在对表进行评论I和II并总结了其他要点以提供进一步的见解。

1. 可以通过降低访问次数来降低TLB动态能耗，例如，基于存储器访问的空间和时间局部性，可以记录和重用先前的翻译(表II).为了增强这些技术的有效性，已经提出了基于编译器的方案，寻求减少页面转换并增强局部性[16,48,49,56].此外，在重用翻译的作品中(表II)，一些作品在同一个周期和连续的周期中执行重用[54,68]，而其他作品仅在连续的周期中执行重用。

表一.基于处理单元、评估平台和优化目标的分类

种类	参考
处理部件	
图形处理器和中央处理器-图形处理器系统	[242631
中央处理器（central processing units 的缩写）	几乎所有其他人
评估平台	
真实系统	[12243233
模拟器和真实系统	[3151721233439
模拟器	几乎所有其他人
优化目标	
表演	几乎全部
活力	[568101618262829314060

表二。基于关键思想和管理方法的分类

种类	参考
管理方法	
超页或多页尺寸	[235711122021343638406165
软件管理的 TLB	[35616667
软件缓存	[32
用非易失性设计 TLB 记忆	[31
重用最后一个翻译	[164849545657596869
编译器的使用	[162848495658596770
使用记忆区域或语义信息	[16424758
使用私有/共享页面信息-象征式互动	[129397174
投机的使用	[618586375
TLB 重组	[54145
TLB 分割	[76
预取	[324323739616770737778
解决多容器中的问题	[1152933397374
解决冲洗/冲洗问题	[124333955707679
TLB 一致性问题	[102980
虚拟缓存	[81017295053556080
消灭 TLB	[31062
绕过 TLB	[1871

2. 可以通过减少咨询的方式来减少 TLB 动态能量[42]，位已预充电[58]或访问的条目[43]并通过降低 TLB 关联性。同样，虚拟高速缓存可用于减少 TLB 访问。

3. 可以通过使用重新配置来减少 TLB 泄漏能量[5,41,45]并使用非易失性(即低泄漏)存储器来设计 TLB [31].

4. 通过增加 TLB 覆盖范围(例如，通过使用超页面或可变页面大小)，通过使用预取、软件缓存、TLB 分区和减少刷新开销(表 II).

5. 一些技术使用标签(例如，ASID)来移除同音异义词和/或减少刷新开销[1,17,50,51,76,79].

6. 由于堆栈/堆/全局静态数据和私有/共享数据表现出不同的特征，利用这种语义信息和页面分类信息(分别)允许设计有效的管理技术(表 II).
7. 因为 ITLB 的失败率通常比 DTLB 低得多[13,18,39]，一些研究人员只关注 DTLB [3,6,45,47,58,61,70,78].其他一些作品只关注 ITLB [49,59].
8. 一些关于 TLB 管理的著作也对高速缓存进行了修改[47,71]或在缓存中使用类似的优化[41,42].
9. 一些技术通过使用额外的存储来工作[1,18,51,68,75]，如布隆过滤器[51].
10. 一些作品对 TLB 失误进行分类，以获得见解[15,67].
11. 一些作品建议对 Java 应用程序进行 TLB 增强[3,44,71].

3. TLB 建筑和基准研究

一些作品探索了 TLB 架构，并研究了页面大小的影响(表 II)，TLB 关联性[11,21,61,64,79]，替换政策[13,31,37,45,61,71,79]，TLB 的水平等级[3,35,44,61]和端口数量[47,68].我们首先总结探索 TLB 设计空间的作品(3.1 和 3.2)然后查看偏斜关联(3.3)和软件管理的 TLB(3.4).我们进一步调查了表征基准套件的 TLB 行为的工作(3.5)，例如 SPEC [13,36,61]，PARSEC [15,39]和 HPCC [36].最后，我们回顾一下多核处理器中的 TLB 设计问题(3.6).

3.1. TLB 设计空间探索研究

陈等[13]使用 SPEC89 基准评估不同的 TLB 架构。微 TLB 是一个 FA TLB，最多 8 个条目，与指令缓存并行访问。它试图利用指令访问的高局部性，在未命中时，它从共享的 TLB 加载。他们发现，由于不同应用的不同 WSS，他们对微 TLB 条目的要求是不同的，例如，对于具有许多嵌套循环的应用，失败率非常高。在将页面大小从 4KB 增加到 16KB 时，只有几个(例如 7 个)条目足以容纳整个工作集。至于替代政策，从 LRU 到先进先出再到随机的失误率增加，但先进先出政策的表现接近 LRU。至于 ITLB，他们发现由于指令位置的原因，大多数应用程序对 ITLB 的要求很低。出于同样的原因，数据引用支配着共享的 TLB，因此，数据 TLB 和共享的 TLB 的行为是相似的。对于共享 TLB，性能取决于 TLB 的覆盖范围，因为它决定了 TLB 可以覆盖多少应用程序工作集。他们还将单独的指令/数据 TLB 与共享的 TLB 进行了比较，发现由于指令和数据引用之间的竞争，共享的 TLB 可能显示出更高的未命中率。

彭等[3]评估几种提高嵌入式系统上 Java 应用程序的 TLB 性能的技术。至于超级页面，他们用 32 入口 TLB 评估 64KB 和 1MB 页面大小。他们的应用程序的工作集大小最多为 16MB，因此，使用 1MB 页面可以完全消除 TLB 遗漏。相比之下，对于某些应用程序(工作集为 305KB)，使用 64KB 页面可以提供与使用 1MB 页面同样多的改进，而在其他应用程序中，由于它们的工作集大于 TLB 覆盖范围(32 64KB)，因此只能提供很小的好处。他们进一步评估了具有 4KB 页面大小的 2 级 TLB 设计，并通过实验不同的大小/关联性，发现具有 4 路 16 入口 level 和 8 路 256 入口 level 的设计是最好的。此外，2 级 TLB 设计比 1 级 TLB 设计提供更高的性能。超级老化需要大量的操作系统更改和很少的硬件更改，对于 2 级 TLB 来说情况正好相反。此外，与超级老化不同，2 级 TLB 设计的使用仅导致覆盖范围的小幅增加。

至于预取缓冲区中的预取，他们观察到预取的准确性和覆盖范围对缓冲区的大小相对不敏感，因此 4 条目缓冲区就足够了。此外，使用预取比将 TLB 大小增加 4 个条目提供更高的性能。他们还考虑直接预取到 TLB(即，不使用单独的预取缓冲器)，并发现它对大多数应用程序都很有效，尽管它对一个预取精度和覆盖范围较差的应用程序会导致 TLB 污染。他们进一步评估了不使用 TLB 的无 TLB 设计，从而完全消除了 TLB 惩罚，使这种设计对嵌入式系统特别有吸引力。这种设计适用于托管运行时环境，该环境管理通常位于同一地址空间中的对象空间。通过比较所有的方法，他们发现超老化和无 TLB 提供了最高的性能。两级 TLB 的性能优于预取，预取的性能优于单级 TLB。此外，就增加硬件开销而言，可以对不同的技术进行排序，如图所示 4。

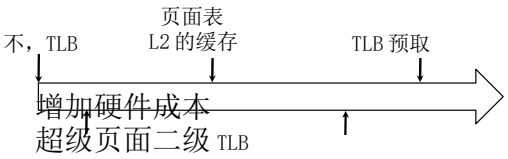


图 4。不同技术的相对硬件开销[3]

Choi 等人[43]建议将 TLB 组织为两个部分:双向过滤器 TLB 和主双向 TLB，分别有 8 个和 32 个条目。在 TLB 访问中，首先只访问过滤器 TLB，如果未能访问，则在下一个周期中访问主 TLB。在主 TLB 命中时，TLB 访问请求被服务，条目被移动到过滤器 TLB，从过滤器 TLB 驱逐的条目被移动到主 TLB。当过滤器和主 TLB 都失败时，新条目被加载到过滤器 TLB 中，并且其被驱逐的条目被移动到主 TLB。与分层 TLB 设计不同，他们的设计不强制包含，这增加了总容量。此外，与 TLB 足协相比，他们的技术通过每次访问查阅更少的条目来减少动态能量。

Juan 等人[8]比较 FA、SA 和直接映射 DTLBs 在相同失败率下的功耗。他们发现，对于小型和大型 TLB 尺寸，FA 和 SA DTLBs 的功耗分别最低，因为 SA DTLB 的尺寸需要更大，以实现与 FA DTLB 相同的失败率。他们还提出了电路级修改，以降低 DTLB 的功耗。他们还评估了虚拟磁带库指令缓存，以将对 ITLB 的访问减少到接近零。

3.2. *TLB 替代和绕过政策*

童等[71]请注意，在托管语言(如 Java)中，垃圾收集器仅在垃圾收集时回收分配的内存。因此，一些页面变得“无用”，即它们的大部分对象都是死的。这种页面是大多数 TLB 缺失的原因，因为它们的块中只有少数被引用，并且只被引用了几次，因此它们具有较差的空间和时间局部性。他们首先提出了一种基于采样的机制来检测无用的页面。他们的机制只对整个 TLB 使用一个“标记计数器”，而不是对每个 TLB 条目使用一个计数器。第一个传入的 TLB 条目保留计数器，其 VA 标签存储在计数器的标签中。对于页面的每次访问，计数器都会递增。因此，计数器仅在访问特定页面(由其标记指定)时递增，因此术语“标记计数器”也是如此。如果计数器超过阈值，页面将被标记为常规，如果 TLB 条目被逐出，页面将被标记为无效，并且该信息存储在 L2 TLB。在这两种情况下，计数器都会被释放。它们表明静态选择的阈值(例如 31)提供了相当高的精度。

进一步提出了两种利用无用页面信息的方案。首先，他们修改替换策略，以便在驱逐时，首先驱逐无用的页面，否则，使用基线替换策略(LRU)。第二种方案将虚拟地址-物理地址 DL1 高速缓存的一种方式扩展为虚拟地址。在 L1 TLB 未命中到无效页面时，条目从 L2 TLB 获取，并与行一起插入 DL1 高速缓存，而不是在 L1 TLB。未来对这一行的引用显示了在 L1 TLB 的失误，而是命中缓存。如果访问了页面的另一行，则该行将被读入

DL1 缓存的扩展方式遵循相同的过程。这个计划防止无用的页面进入和痛打 TLB。此外，它将无用页面的块限制在高速缓存的一个路中，从而减轻了与可以使用任何高速缓存路的常规页面块的竞争。第二种方案提供了比第一种方案更高的性能改进，而第一种方案又优于传统的和抗冲击的高速缓存替换策略。

3.3. *偏斜联想 TLB*

Seznec [64]提出了一种在“倾斜关联缓存”之后设计的“倾斜关联 TLB”81].倾斜关联高速缓存寻求减少冲突未命中，以允许使用低关联高速缓存，该低关联高速缓存也具有低访问时间[82].它旨在通过使用不同的哈希函数为每个缓存路建立索引，将冲突的块分散到多个集合中。给定的块地址与一组固定的块冲突，但是这些块以其他方式与其他地址冲突，这进一步分散了冲突。通过仔细选择散列函数，可以大大减少冲突，因此，倾斜关联高速缓存通常比集合关联高速缓存具有更高的利用率和更低的未命中率。然而，倾斜关联缓存打破了集合的概念，因此，它不能使用依赖于集合排序的替换策略。

偏斜联想 TLB 设计[64]遵循以下属性。首先，它至少需要 2n+1 路，其中 2n 是大于 K 的二的一次幂，即支持的页面大小数量。例如，对于 K = 4，需要 8 向 TLB。这是为了确保足够多的路径，以便任何 TLB 路径都可以映射整个虚拟空间。此外，TLB 的参赛作品数量是
允许 TLB 映射虚拟内存的大的连续区域的最大和最小页面大小的比率。此外，使用不同的索引函数对 TLB 的不同方式进行索引。每一个虚拟地址都以各种方式映射到一个唯一的位置，如果一个虚拟地址映射到 TLB 的某个特定方式，那么它的页面大小就是已知的。此外，对于任何支持的页面大小，仅使用页面大小的应用程序可以使用整个 TLB 空间，这允许同时支持多种页面大小。他们表明，偏斜关联 TLB 有效地支持多种页面大小，然而，它的有效关联性(即，具有指定页面大小的任何虚拟页面的位置数量)更小
而不是 TLB 的结合性(例如，当 K = 4 时，8 路中有 2 路)。

3.4. *软件管理的 TLB*

Nagle 等人[35]请注意顶级域名的软件管理(参见第节 2.1)可能会导致巨额罚款和 TLB 管理的复杂性。不同类型的 TLB 失误的相对频率在不同的操作系统上可能有所不同，例如 Ultrix 和 Mach。对于相同的应用程序二进制文件，不同操作系统的总 TLB 未命中计数和 TLB 未命中惩罚可能有很大差异。此外，通过降低内核 TLB 失误惩罚，可以显著减少总 TLB 服务时间。他们还研究了影响 TLB 表现的 TLB 规模和关联性等因素，例如，TLB 的规模在 32 至 128 个条目之间每增加一倍，TLB 的服务时间就会减少近一半。

3.5. *工作量特征研究*

McCurdy 等人[36]研究 SPEC-FP(SPEC 2000 基准测试套件的浮点部分)和 HPCC 基准测试的 TLB 行为[83]和一些真正的科学应用。基于与实现无关的实验，他们发现，对于 4KB 和 2MB 的页面大小，这些基准高估和低估了(分别)良好应用程序性能所需的 TLB 条目数量，因此，这些基准无法准确表示应用程序的 TLB 概况。他们在 x86 AMD 皓龙系统上进一步验证了这些结论。该系统使用两个不同的 TLB，这两个 TLB 具有不同的大小和关联性来处理不同的页面大小，因此，页面大小对该系统的性能有显著影响。他们观察到，基于基准测试结果对页面大小的错误选择可能会导致应用程序性能损失高达 50%。它们还表明，基于与实现无关的结果，可以预测应用程序在其他 x86 系统中的 TLB 行为(具有不同的 TLB 和页面大小值)。

Kandiraju 等人[61]研究 SPEC2000 基准的 DTLB 行为。他们注意到几个 SPEC 基准有很高的 TLB 失败率。优化，如增加 TLB 大小或关联性和超老化有利于有限和独特的应用程序集。他们进一步比较了相同总尺寸的 1 级 TLB 设计和 2 级包容性 TLB 设计。由于包容性降低了有效大小，2 级 TLB 显示出更高的未命中率，但是，对于 1 级 TLB 未命中率低于阈值的应用程序，由于访问时间减少，它提高了性能。大多数应用程序没有清楚地显示 TLB 失误的不同阶段，只有少数程序计数器对大多数 TLB 失误负责。

为了判断 TLB 软件管理的范围，他们将性能与 OPT(最优)和 LRU 替换策略进行了比较。对于某些应用，OPT 替换策略提供了比 LRU 低得多的失败率，这表明编译器/软件优化可能有助于将失败率降低到纯硬件管理策略所能达到的水平之外(LRU)。例如，如果可以静态分析应用程序的引用行为，编译器可以在应用程序二进制文件中放置提示，以优化替换决策。他们还指出，即使使用 OPT，未命中率也很大，因此，容忍未命中延迟的技术(如预取)非常重要。他们评估了三种将数据放入预取缓冲区的预取方案。他们观察到预取减少了大多数应用程序中的未命中，因为它们显示了一些访问的规律性。

Bhattacharjee 等人[15]研究 PARSEC 基准的 TLB(ITLB 和 DTLB)行为。他们使用真实处理器进行真实评估，并在其上执行“原生”和“simlarge”数据集。由于在真实系统上获取地址转换信息具有挑战性，他们还使用模拟器并在其上使用“simlarge”数据集。它们记录了同一转换导致多核缺失的频率、操作系统的贡献以及并行化属性的影响。他们观察到一些 PARSEC 基准测试显示了大量的 TLB 失误，这严重影响了他们的性能。此外，来自不同岩心的 TLB 缺失显示出显著的相关性。由于不同的线程处理相似的数据/指令，几乎所有(例如 95%)的未命中都已经被其他线程显示。它们定义了“内核间共享”(ICS)未命中，这样，如果内核上的 TLB 未命中是由于对翻译条目的访问而发生的，该翻译条目具有与过去 M 条指令中剩余内核之一上的先前未命中所请求的翻译相同的 VA 页面、PA 页面、页面大小、保护值和进程标识，则该未命中就是 ICS。随着 M 的增加，分类为 ICS 的未命中的比例增加。

他们还注意到，一个核心的 TLB 缺失通常与另一个核心的缺失有一个可预测的步幅距离。因此，如果一个未命中的虚拟页面与先前匹配未命中(相同的页面大小和进程标识)之间的差异为 S，则该未命中是具有跨步 S 的“内核间可预测跨步”(ICPS)未命中。为了消除冗余和步长可预测的未命中，可以通过共享 TLB 和内核间预取等方案来利用内核之间的 TLB 未命中模式(参见 3.6).

3.6. 多核处理器中的 TLB 架构

对于多核处理器，一些研究人员评估了私有的每核 TLB [1,29,39,74]共享条目存储在单独的缓冲区中[1,39]或者在 TLB 本身[74].其他研究人员评估共享 TLB [29,39].

Lustig 等人[39]请注意，对于 SPEC 和 PARSEC 基准，ITLB 失败率比 DTLB 失败率小几个数量级。此外，对 DTLB 来说，即使是 L2 的命中也会招致管理费用，并导致 TLB 总点球的很大一部分。他们提出了两个减少 TLB 失误的方案。他们的第一个方案旨在减少 ICS 失误(3.5).每次 TLB 未命中时，当前内核(称为“领导者”)都会填充其自己的 TLB，并将此转换放入其他(称为“追随者”)内核的每内核预取缓冲器中。在命中预取缓冲器中的条目时，它被移动到 TLB。这种方案的一个限制是，它盲目地将一个条目推送到所有的从动核，即使该条目可能只被几个核共享。为了避免这种无用的预取，对于每个核，它们使用对应于剩余核的置信度估计计数器。前导核心的计数器在命中或逐出时递增/递减，而不使用(分别)跟随核心中的预取条目。然后，只有当领导者的计数器对应时，条目才会被推送给追随者

对该跟随器的访问超过了阈值。这种置信度估计方案显著提高了性能。

他们的第二个计划试图减少 ICPS 失误(3.5).例如，如果核心 0 在第 6、7、9、10 页及以后的页面上未命中，核心 1 在第 10、11、13、14 页上未命中，则步长为 4。因此，尽管不同虚拟页面上的内核未命中，但两个内核(1，2，1)上地址之间的差异是相同的。基于此，第二种方案将虚拟页面中重复出现的内核间步距记录在内核间共享的表中，以供其他内核使用。该表是在预测和预取所需翻译的 TLB 未命中上查找的。在上面的例子中，在观察对核心 1 上的页面 10 和 11 的访问时，可以发出对页面 13 和 14 的预取，这避免了对它们的未命中。

他们进一步建议共享最后一级 TLB (SL-TLB)。为了避免严格纳入的开销，SL-TLB 旨在“大部分纳入”L1 顶级域名。因此，在未命中时，条目被放置在 L1 和 SL-TLB，然而，替换决定由每个 TLB 独立做出。在实践中，这种方法提供了接近完美的(例如，97%)包含，尽管在击落，它需要检查 L1 和 SL-TLB。由于 SL-TLB 在内核间共享，因此它比私有顶级域名占用更多的通信时间。为了进一步提高 TLB 命中率，他们在 SL-TLB 中使用了跨步预取[30].他们表明，他们的技术显著减少了 TLB 失误。

Srikantaiah 等人[74]请注意，随着 TLB 条目数量的增加，一些应用程序显示未命中显著减少，而其他应用程序显示的减少可以忽略不计。这些申请分别被称为“借款人”和“捐赠人”。此外，当同一多线程应用程序的不同线程共享许多页面时，相同页面的重复翻译会浪费 TLB 容量。这些观察激发了容量共享，因此，它们提出了一种“协同 TLB”设计，具有每个核心的私有顶级域名，并允许来自一个 TLB 的受害者条目存储在另一个 TLB，以模仿分布式共享的 TLB。从借款人的 TLB 驱逐的条目被插入到捐赠者的 TLB，从捐赠者的 TLB 驱逐的条目被丢弃。通过指定一些参赛作品作为借款人或捐赠人，可以确定 TLB 的性质，其余参赛作品遵循获奖方案。与私人 TLB 设计相比，这种设计减少了 TLB 失误，提高了性能。然而，该方案的一个限制是，它将一些本地 TLB 命中改变为远程 TLB 命中，这导致了大得多的延迟，因此，对于一些应用，性能的提高与未命中的减少不相称。为了解决这个问题，他们提出了在多个顶级域名中复制条目和/或将条目从一个 TLB 迁移到另一个的策略。对于这两种方法，它们都提供了“精确”和“启发式”方法，其中精确方法维护大量的访问历史，而启发式方法只跟踪本地和远程命中。它们表明，复制策略不会影响多程序工作负载，但会提高多线程工作负载的性能，因为复制只有在共享的情况下才有好处。迁移策略提升了多程序和多线程工作负载。此外，启发式方法的性能接近精确方法。最后，他们的协同 TLB 设计智能地使用复制和迁移，比使用其中任何一种都提供更高的性能。

李等[1]使用页面分类(私有/共享)信息来改进多核处理器中的 TLB 管理。他们用“部分共享缓冲区”(PSB)扩展每个私有 TLB，该缓冲区存储不同内核共享的翻译，并记录进程标识。PSB 不存储私人翻译，这避免了他们的远程位置和 PSB 的污染。共享翻译分布在所有内核的 PSB 中。他们通过记录访问虚拟页面的内核，将虚拟页面分为私有页面和共享页面。在 L2 TLB 小姐，如果一个页面/分类是共享的，页面表和家庭 PSB 被并行搜索。在 PSB 命中时，条目被复制到请求者核心的本地 TLB。否则，page table 提供的条目会插入到 TLB 本地和 PSB(如果共享)。TLB 击落传统上停止所有内核，然而，他们的技术避免了停止不存储特定翻译的内核，因此，他们的技术实现了单个无效的击落，这带来了低得多的开销。与传统设计类似，他们的私有 TLB 设计也需要在上下文切换或线程迁移时进行 TLB 刷新，但是 PSB 不需要刷新，因为它使用进程标识作为标签。在重新激活进程时，可以使用 PSB 剩余的共享条目，这减少了上下文切换开销。与共享的 TLB 相比，他们的技术随着内核数量的增加而扩展，与私有的 TLB 相比，他们的技术通过避免共享的重复来消除 TLB 容量缺失和 TLB 共享缺失

条目。他们表明，他们的技术通过降低翻译延迟和失败率来提高性能。

4. 提高 TLB 覆盖率和性能的技术

在本节中，我们将讨论提高 TLB 覆盖率的技术(4.1-4.4)，减少 TLB 冲洗开销(4.5)和 TLB 预取(4.6)。

4.1. 利用中等水平的连续性

Talluri 等人[2]提出了两种用于提高 TLB 性能的子块 TLB 体系结构，这两种体系结构需要比超页面更小的操作系统复杂性。“完整子块 TLB”(CS-TLB)使用单个标签和一个(64KB)超页面大小的区域，但是对于每个(4KB)基本页面映射有单独的 PPN、有效位和属性(参见图 5 其缺点是增加了 TLB 的面积。在“部分子块 TLB”(PS-TLB)中，PPN 和属性在基本页面映射之间共享，因此，PS-TLB 条目比 CS-TLB 条目小得多。只有当基本页面映射到相邻的物理页面并具有相同的属性时，才允许在 PS-TLB 中的基本页面映射之间共享 TLB 条目(参见图 5(c))。不满足这些条件的页面的翻译存储在不同的 TLB 条目中。他们提出了一种分配物理内存的算法，称为“页面保留”，通过尽最大努力分配对齐的内存来增加这种共享的可能性。该算法尽最大努力(但不保证)将相邻的基本虚拟页面映射到相邻的对齐的基本物理页面。总的来说，CS-TLB 允许将任意物理页面映射到超页面，而 PS-TLB 在构建超页面时提出了更强的限制。CS-TLB 不需要更改操作系统，因此适合在操作系统更改不合适时获得高性能。PS-TLB 只需要很小的操作系统更改，在使用页面保留方面，它的表现优于 superpage TLB。

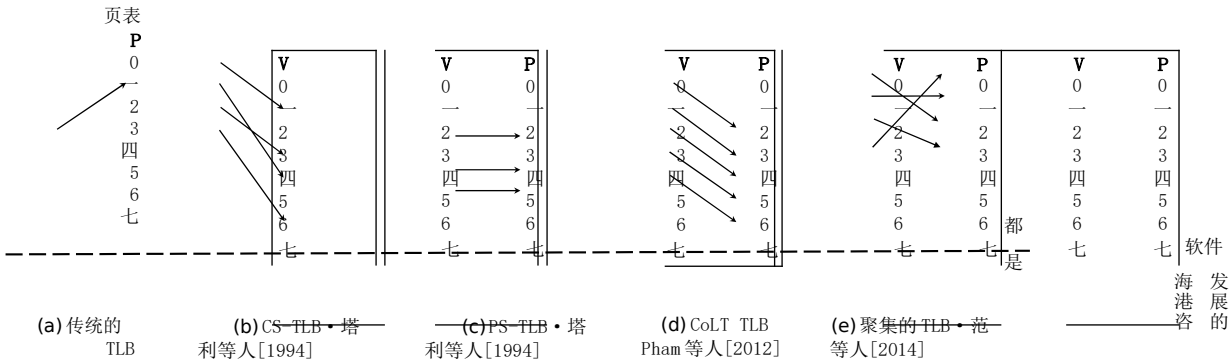


图 5。说明(a)常规 TLB (b) CS-TLB [2TLB 聚苯乙烯[2] (d)“联合大范围”(CoLT) TLB [21]和(e)集群 TLB [37]. 对于 CS-TLB 和 PS-TLB，子块因子为 4，对于群集 TLB，群集因子为 4。

Pham 等人[21]注意，虽然超页减少了 TLB 未命中，但它们也需要生成和管理连续的物理页，例如，对于 4KB 的基本页大小，需要 512 个相邻页来生成 2MB 的超页。此外，操作系统内存管理机制自然会导致中等邻接度(例如，几十页)，该邻接度小于生成超页面所需的数百页邻接度，但这是在没有诸如生成超页面的复杂算法和膨胀的输入/输出流量等开销的情况下实现的。这种内存管理机制的例子包括“伙伴分配器”、“内存压缩守护程序”和 Linux“透明的巨大页面支持”[84]. 他们提出了一种技术，在 TLB 合并多个连续的 V2PA 翻译，以增加其覆盖范围，如图所示 5(d)。这仅在 TLB 未命中时执行，以避免对查找延迟产生任何影响。虽然合并的条目不能在关闭的时间窗口中使用，但它们不会影响命中率。他们假设一个两级 TLB

SA L1 TLB 和 L2 TLB 的 4KB 页面大小。另一个小的 TLB 足总缓存超级页面，并与 L1 TLB 平行咨询。L2 • TLB 包括 L1 • TLB，但不包括超级大国 TLB。

他们提出了三种不同的技术。在第一种变体中，合并仅在南非 L1 和 L2 顶级域名中进行。为了合并 J 个条目，用于计算 TLB 集(索引)的位被左移 log2 J，如图所示 6。为了增加合并条目的数量，可以进一步移动索引位，
然而，这也增加了冲突遗漏，因为更多的相邻条目现在映射到同一个集合。此外，增加 J 会增加搜索相邻翻译的开销。在第二个变体中，合并仅在 TLB 足协进行。在所有 TLB 未命中时，缓存块最多提供八次转换。如果可以合并，条目将加载到 TLB 足协，否则将加载到 L1 和 L2 顶级联赛。插入 TLB 足协后，将检查该条目和现有条目之间进一步合并的机会。在第三个变体中，L1/L2 顶级联赛和 TLB 足总被合并。在所有 TLB 未命中时，检查高速缓存行中存在的邻接量。如果小于阈值，则将其插入 L1 和 L2 顶级域名。但是，如果它大于 L1/L2 顶级联赛允许的毗连性，那么它将被插入 TLB 足协。由于 FA TLB 的大小很小，一些有用的合并条目可能会被驱逐，因此，他们的技术将该合并条目的一部分提取到 L2 TLB。他们的技术增加了 TLB 覆盖率，并通过减少 TLB 失误提高了性能。

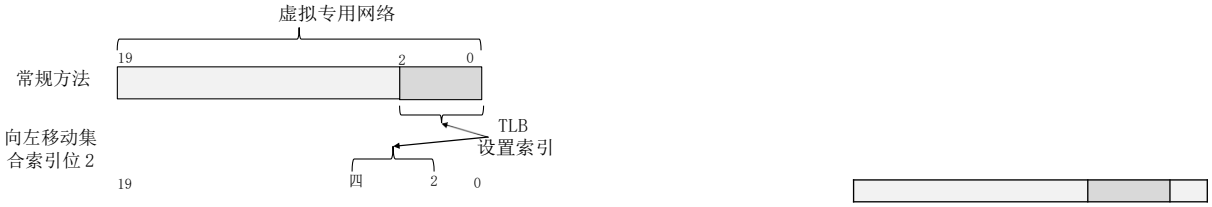


图 6。在传统的集合索引方法下，连续的虚拟专用网被映射到连续的 TLB 集合。
在将索引位左移 log2 J 位时，可以将 J 个虚拟专用网映射到同一个集合(该图中 J=4)。

Pham 等人[37]请注意，除了简单的邻近性之外，更多的翻译显示了群集的空间局部性，其中相邻虚拟页面和相应物理页面的群集是相似的。他们提出了一种利用“聚集空间局部性”的 TLB 设计，也可以利用“连续空间局部性”[21]当它存在时。他们假设“聚类因子”为 K
(例如，K = 4)。然后，如果一个虚拟页面集群指向一个物理页面集群，这样在忽略较低的 log2 K 位时，集群的所有虚拟专用网都具有相同的位，集群的所有 PPN 都具有相同的位，那么 pte 可以只存储在一个“集群 TLB”(cTLB)条目中，如图所示
5(e)。除了 cTLB，他们还使用“传统 TLB”来存储没有任何空间位置的翻译。由于每个 cTLB 条目可能存储不同数量的有效翻译，因此在 cTLB 中使用传统的替换策略(如 LRU)是无效的。因此，它们说明了 cTLB 条目的新近性和有用性，其中有用性显示了有效和被引用的子条目的数量。在驱逐时，这些引用的翻译被安装到传统的 TLB，其余的翻译被驱逐。这允许不合并那些被发现不太有用的翻译。他们的技术不需要操作系统支持，即使在连续空间局部性(被[21])变得稀缺。他们的技术通过减少 TLB 失误和增加 TLB 覆盖率来提高性能。

罗默等人[38]提供动态创建超页面的策略，以减少 TLB 遗漏，同时限制超页面提升的开销。超页面需要映射存储器的物理上连续的部分，然而，潜在超页面的组成页面可能不是物理上连续的，因此，需要页面复制，这是超页面提升的开销。他们提出了一个主要策略，它的近似变体(都是在线策略)和一个离线策略。他们的主要促销政策将每一次 TLB 错过都归因于一个潜在的超级页面，如果这个页面已经构建的话，它将会减轻这种错过。当归因于超页的未命中超过阈值时(基于提升开销和 TLB 未命中之间的权衡计算)，超页被创建。潜在的超页有两种方式可以避免错过页 p。首先，超页可以包括页 p 和 TLB 的现有页 q，因此

q 的现有 TLB 条目也包括 p。其次，超页提供的 TLB 容量增加可以避免驱逐先前的 p 翻译。在两者中，第一种方法在确定最佳超页候选方面更有效，并且也更容易跟踪。因此，他们还评估了仅跟踪第一个原因的促销政策，该政策在消除 TLB 失误方面的有效性接近于跟踪两个原因的主要政策。他们进一步评估了一个离线策略，该策略假设知道完整的 TLB 错过流，并在此基础上，以最高的促销“收益对成本”比率迭代地促销超级页面，直到不存在这样的超级页面。与这种无法在实践中实施的离线策略相比，他们的主要策略实现了相当的性能。此外，与使用 4KB 页面相比，它们的策略减少了执行时间。

4.2. *使用推测地址转换方法*

在“基于预留的物理内存分配”方法中[2,12]，最初小(4KB)页用于所有内存。当 2MB 虚拟机区域的所有 4KB 页面都被使用时，该区域将被提升为大页面。为了实现这一点，将小页面放入保留区，以便将 4KB 页面放在物理内存的 2MB 区域内，类似于它们在虚拟机的 2MB 区域内的位置。因此，在“大页面保留”(LPR)中，连续的虚拟页面也是连续的物理页面。Barr 等人[63]提出了一种技术，该技术利用由这种存储器分配器产生的邻接性和对齐性来预测来自相邻页的 PA 的地址转换。他们使用“SpecTLB”，这是一个缓冲区，在 TLB 小姐被咨询，以检查显示 TLB 小姐的虚拟页面是否可能是 LPR 的一部分。如果是这样，丢失页面的 PA 可以使用该页面在 LPR 内的位置在 LPR 的开始和结束 PA 之间进行插值。

由于 spectralb 预测的映射可能不正确，因此必须针对对页表的访问确认 spectralb 中的命中。SpecTLB 提供的转换允许推测性执行，同时并行执行页表访问。如果 SpecTLB 生成的转换被验证，则推测执行被提交，否则，执行从第一个错误的预测重新开始。因此，SpecTLB 试图通过从关键路径中移除页表访问来减少 TLB 未命中惩罚，尽管它没有减少对页表的访问。此外，当使用大页面可能不切实际时，例如在虚拟化中，它寻求提供接近大页面的性能。它们表明 SpecTLB 的预测精度很高(例如 99%)。此外，与 TLB 预取相比，他们的技术更适合具有随机访问模式的程序，并且他们的技术执行的推测非常有用。

Pham 等人[34]请注意，较大的页面会导致内存管理开销，例如在过度使用的处理器中缩小虚拟机之间的内存重复数据消除范围，以及妨碍虚拟机迁移的便利性等。因此，虚拟化模块通常会将客户操作系统的大页面分成小的物理页面，以失去其 TLB 优势为代价来实现系统级收益。他们提出了一种技术来恢复因页面拆分而失去的地址转换优势。他们指出，页面拆分基本上不会改变 PA 或 VA 空间，因为大多数组成的小页面不会被重新定位。因此，大多数这些小页面在 PA 和 VA 空间中都保持了它们原来的对齐和邻接。他们的技术跟踪这种对齐的、连续的但分开的部分，并通过基于存储在 TLB 的关于单个推测性大页面翻译的信息进行插值来推测小页面。例如，在图中 7，其中一个大页面由四个小页面组成，访客大页面 GVP4-7(GVP =访客虚拟页面)是分裂的，尽管如此，SPPs(系统物理页面)可以使用插值方法来推测。这些推测在页表遍历中得到了验证，页表遍历现在发生在关键路径之外。因此，他们的技术允许拆分大页面进行细粒度的内存管理，同时保持 TLB 性能接近原始大页面。他们的技术与 SpecTLB 方法有相似之处 63]但与 SpecTLB 方法不同，它们不需要特殊的单位进行投机。此外，他们还提出了使用 PTE 预取来减轻不正确推测的性能影响的策略，这降低了访问页表来验证某些推测的需要。

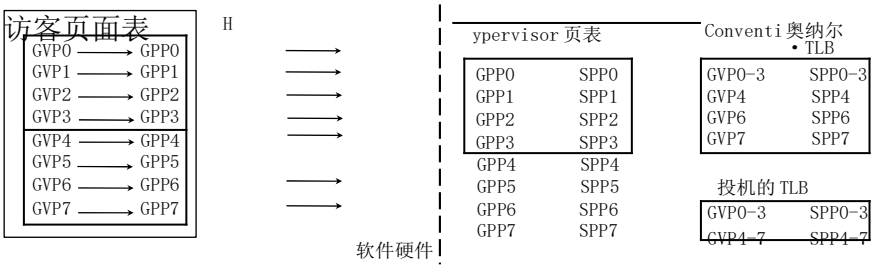


图 7。投机性 TLB 设计的例证[34] (GVP =来宾虚拟页面，GPP/SPP =来宾/系统物理页面)。这里客大页 GVP4-7 是拆分的，但是 spp 可以插值。为了映射页表，传统的 TLB 需要四个条目，而推测的 TLB 只需要两个条目(包括 GVP4-7 的推测条目)。

4.3. *使用地址间接方法*

Swanson 等人[7]请注意，处理器可以处理的 PAs 范围通常会超过实际可用的内存容量。例如，使用 32 个 PA 位和 1GB DRAM，生成的 PA 只有四分之一有效。对于这种情况，他们建议使用这个未使用的 PA 范围的一小部分(称为“影子”页面)来再次虚拟化物理内存。应用 VAs 和 PAs 之间的这种间接级别允许从非相邻的“真实”页面创建任意大小的相邻超页面。访问时，存储控制器将影子 PAs 转换为真实 PAs，并将该映射存储在“存储控制器 TLB”(MTLB)中，如图所示 8。处理器和内存管理软件使用影子地址，它们作为虚拟地址的映射存储在 TLB，作为高速缓存块的物理标签存储在高速缓存中。他们的技术(被称为影子内存设计技术或 SMDT)允许使用处理器中已经存在的超级内存能力。它们表明，SMDT 允许实现两倍大的 TLB 的性能。但是，SMDT 不能用于没有未使用的物理内存的系统，它可能不支持精确的例外。

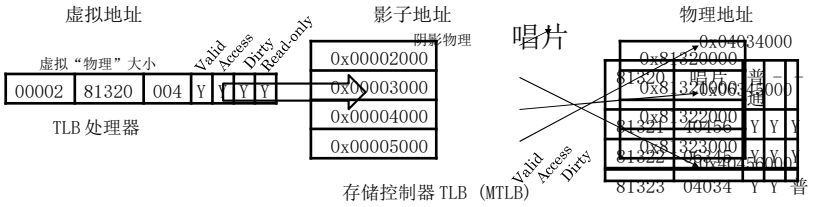


图 8。阴影记忆设计技术图解[7]

Park 等人[65]请注意 SMDT [7]允许从非连续的物理页面创建超页面，但是，它在 TLB 保持部分粗略映射。此外，TLB 计划[2]在 TLB 存储完整的地图信息，但超级页面创建的便利性有限。为了将这两者结合在一起，帕克等人[65]提出一个设计，使用来自 SMDT 的 MTLB 和来自 PS-TLB 方案的 TLB 条目格式。这种格式允许识别处理器内部的无效映射，这避免了指令备份硬件的需要，不像在 SMDT，无效映射被识别得晚，因此，许多指令需要被备份(即，被挤压和重新启动)。此外，在 SMDT，影子和真实 PA 之间的映射变化(例如，在分页时)需要刷新与映射相关的所有缓存块，而它们的设计只需要重置 TLB 中的相关有效位。在 SMDT，影子到真实 PA 的条目可能在 MTLB 找不到，从而导致错过。然而，在他们的设计中，这样的失误自 TLB 以来就没有发生过

保持每个基本页面的有效位。此外，每个基本页面的参考信息不由 MTLB 管理，而是存储在中央处理器中，以提高页面替换策略的效率。与使用单个基本页面大小相比，他们的技术显著减少了 TLB 失误，接近于 SMDT 消除的失误。他们技术的局限性在于其最大超页大小受限于 PS-TLB 所允许的大小。

4.4. 支持可变大小页面

Papadopoulou 等人[6]研究 TLB 密集型横向扩展应用程序和商业应用程序的 TLB 行为，并观察到一些应用程序经常使用超级页面。此外，应用程序更喜欢最大的超级页面，并且中间页面大小的使用很少。在此基础上，他们提出了一个预测内存访问是否到达超页的方案。对于大小预测，他们设计了两个预测器，分别使用指令地址和基址寄存器值来访问预测器表。由于在 SA TLB 中支持多种页面大小具有挑战性(2.2)并且预测确切的页面大小会导致较大的延迟，他们建议仅区分 8KB 页面和超页面(64KB、512KB、4MB)，因为通常只使用 8KB 和 4MB 大小。他们的 TLB 使用最大的标签大小为 8KB 的页面，因此，可以支持任何页面大小。任何页面大小都可以使用任何设置。所有超页的索引位都是相同的，并且由于不经常使用 64KB 和 512KB 的页面，它们不会造成 TLB 压力。

在一般情况下，预测是正确的，TLB 访问是一个打击。在 TLB 未命中的情况下，需要第二次 TLB 访问，并进行相反的页面大小预测，以解决预测错误。如果 TLB 访问仍然导致未命中，则需要访问页表。总体而言，他们提出了一种 SA TLB 设计，该设计同时存储不同页面大小的翻译，并寻求实现 FA TLB 的失败率和 SA TLB 的访问延迟/能量。他们还讨论了显示页面大小使用大变化的应用程序的精确页面大小预测。此外，他们评估“偏斜关联 TLB”设计[64]并使用突出的页面大小预测方法来增强它，以提高它的有效关联性。他们表明，只有一个 32B 超页预测器，他们的技术允许 4 路 256 入口 TLB 比 128 路 FA TLB 捕获更多的内存引用。他们的方法的一个限制是，当最小和最大页面大小的差异很大时，TLB 压力会大大增加，例如，当 2MB 和 1GB 页面共享相同的索引时，512 个连续的 2MB 页面竞争相同的集合。对于这种情况，可以在页面大小组之间进行预测。

Talluri 等人[11]评估使用大页面大小(32KB)和两个页面大小(4KB 和 32KB)来提高 TLB 性能。对于两个页面大小，他们使用以下页面大小分配方案。对于 32KB 的内存区域，如果在最后的 K 个引用中至少访问了一半的 4KB 大小的块，则该区域将被提升到大页面。他们观察到，随着页面大小在 4KB 和 32KB 之间每增加一倍，WSS(间隔中引用的不同页面)的增加幅度从 1%到 30%不等。页面大小增加不到 2 倍的原因是空间局部性的存在。例如，在循环中线性迭代内存区域的应用程序已经在工作集中覆盖了地址空间，因此页面大小的增加对 WSS 没有太大影响。对于地址空间稀疏的应用程序，WSS 出现了大幅增长。他们还注意到，使用两种页面大小时，WSS 的增加量小于使用 8KB、16KB 或 32KB 页面。因此，使用两种页面大小会比单个甚至 8KB 的大页面产生更少的内存需求。使用 32KB 页面时，TLB 的消费物价指数贡献显著降低。在使用两种页面大小时，CPITLB 的减少范围从接近零到仅使用 32KB 页面实现的减少。他们还讨论了在 SA TLB 中提供可变页面支持的不同方法，例如，用小页面大小或大页面大小的页码或精确页面大小来索引 TLB。

4.5. 减少 TLB 冲洗开销

Venkatasubramanian 等人[79]请注意，通过虚拟化进行的工作负载整合增加了不同虚拟空间的数量以及它们之间的上下文切换频率。由于上下文切换需要刷新 TLB，因此虚拟化可以将 TLB 刷新次数增加 10 次，并将 DTLB 和 ITLB 的未命中率分别增加 5 次和 70 次。此外，使用每个虚拟机标签标记每个 TLB 条目可以避免虚拟机之间上下文切换时的 TLB 刷新，但不能避免同一虚拟机的两个进程之间的上下文切换。到

为了避免刷新这两种类型的上下文切换，他们建议使用特定于进程的标签，这些标签是从“页面目录基寄存器”(x86 中的 CR3 寄存器)中生成的。为了减少标签中的位数，可以使用位屏蔽或完美散列，使得最终标签在各进程中仍然是唯一的。他们的技术可用于虚拟化和非虚拟化平台，并可有效降低 TLB 刷新和未命中率。他们还研究了 TLB 规模、关联性和替换政策以及 TMT 规模对改进的影响。

Venkatasubramanian 等人[76]还要注意，在整合工作负载的情况下，虚拟机的 TLB 配额取决于虚拟机运行的时间、虚拟机中执行的工作负载的 WSS 以及共享 TLB 的其他虚拟机中的工作负载。使用他们标记的 TLB 方法，他们建议通过根据当前份额选择受害者，在 TLB 路级别控制虚拟机的 TLB 配额。这种 TLB 分区类似于基于路的高速缓存分区。利用这一点，可以实现不同虚拟机或进程之间的性能隔离和优先级强制。

Chukhman 等人[70]请注意，在多任务系统中，TLB 中的任务间干扰会损害性能和可预测性。为了减少这种开销，可以在上下文切换和进程状态期间保存 TLB 条目(特别是虚拟专用网)。此外，通过仅保存正在运行或将在不久的将来使用的条目(称为“扩展的实时集合”或 ELS)，可以进一步降低存储开销。他们的技术决定了上下文切换时的 ELS。编译器提取关于程序的静态信息，例如寄存器使用和内存访问模式。此外，操作系统收集运行时信息，例如发生上下文切换的代码位置和指针值。基于编译器信息，他们的技术找到了指针值，这些值对于确定活动虚拟专用网络的任务很有用。利用这一点以及访问模式和阵列大小，可以估计出未来可能访问的虚拟专用网络。基于此，在抢占一个任务后，下一个任务的 ELS 在 TLB 被预取。要预取的条目数量是基于 TLB 配置、数组维数、循环长度和算法特性来确定的。此外，最早需要的条目被赋予第一优先级。他们表明，他们的技术减少了由于任务间干扰导致的 TLB 失误，这提高了 TLB 的可预测性，并导致对“最坏情况执行时间”的更严格估计。

Villavieja 等人[33]提出一种技术来减轻由页面重映射(而不是上下文切换)引起的 TLB 击落带来的性能损失。基于对真实系统的研究，他们指出，随着内核数量的增加，开销和击落频率迅速增加，甚至超线性增加。击落几乎会停止所有内核，尽管其中许多内核可能不会存储已更改的映射[14].此外，由于内存重映射，内存或文件 IO 操作的应用程序中经常会发生枪战。此外，在 16 核和 128 核机器中，多达 4%和 25%的周期可能用于击落。他们建议添加一个二级 TLB，实现为双向 4096 条目缓存，作为“字典目录”(滴滴出行)。它跟踪整个系统的 L1 顶级域名中每个条目的位置，因为一个翻译可能存在于多个 L1 顶级域名中。滴滴出行拦截了 L1 顶级域名中的所有插入和驱逐，不会导致延迟损失，因此，它是最新的。为了执行远程 TLB 失效，它使用“等待 TLB 失效缓冲区”(PTIB)。在 TLB 击落，无效请求首先被发送到滴滴出行，然后，滴滴出行转发无效请求到仅存储改变的映射的那些核心的 PTIB。这避免了向不存储改变的条目的核发送无效，并且还避免了中断核的需要及其相关成本，例如流水线刷新、上下文保存和存储等。他们表明，他们的技术将击落的延迟开销降低了一个数量级。

4.6. *TLB 预取*

一些预取技术将数据放在单独的预取缓冲器中[3,32,39,61,77,78】，鉴于
其他人带来了 TLB 本身的数据[3,24,37,67,70].我们现在回顾这些技术。

Bala 等人[32]建议使用预取和软件缓存来减少内核 TLB 未命中的数量和惩罚。他们指出，在“进程间通信”(INPC)之后，导致 TLB 未命中的大部分页表访问是可以预测的。因此，在 INPC 期间在内核中预取它们可以消除大量未命中。对于三级页表

层次结构，他们预取 L1K [32]映射 INPC 数据结构的 pte、映射消息缓冲区及其相关 L3 PTEs 的 L3 PTEs 和映射过程代码段和堆栈的 L3 PTEs。预取的 TLB 条目存储在不同的“预取 TLB”中。他们还提出了一个“TLB 软件缓存”(STLB)，存储从硬件 TLB 替换的 L1K、L2 和 L3 条目。最初，STLB 代码从通用陷阱处理程序分支出来并检查 STLB。STLB 命中导致条目插入硬件 TLB，因此，避免了通用陷阱处理程序的损失。STLB 未命中导致代码分支回通用陷阱处理程序。由于页表的分层组织，一个级别的 TLB 未命中通过探测下一个级别来处理，这可能导致级联的 TLB 未命中。STLB 通过提供 TLB 条目的平面组织和避免对页表的进一步访问来缓解这种级联未命中。他们表明，预取消除了近一半的内核 TLB 未命中，缓存通过为 TLB 未命中提供有效的陷阱路径来降低 TLB 未命中成本。

Saulsbury 等人[77]提出了一种 TLB 预取技术，该技术基于这样的思想运行，即过去在紧密时间邻域中访问的页面将来也将在紧密时间邻域中访问。他们的技术使用具有前一个和下一个指针的双链表来构建 LRU 堆栈。在从 TLB 驱逐一个条目时，它被放在堆栈的顶部，并且它的下一个指针被存储为先前被驱逐的条目(其先前的指针被存储为该条目)。在未命中时加载条目时，它们的预取技术会将该条目的上一个和下一个指针所指向的条目存储在预取缓冲区中。他们表明，他们的技术可以准确预测超过一半的 TLB 失误，并通过预取来提高性能。他们的技术的局限性在于，它将预测信息(例如指针)存储在页表中，这需要对页表进行修改。此外，他们的技术可能不适用于不使用 LRU 替代政策的顶级域名。

Kandiraju 等人[78]评估几种预取技术，例如，“任意步长预取”、“马尔可夫预取”和“基于最近时间的预取”[77]在顶级域名的情况下。他们还提出了一种距离预取技术，根据连续地址之间的步距预测要预取的地址。例如，如果访问的地址是 10、11、13、14、16 和 17，那么“1”的步距之后是“2”的步距，因此，两个表的条目足以进行预测。所有技术都将预取放入预取缓冲区。对于具有常规分级访问的应用程序，所有技术都具有良好的准确性。对于数据集较大的情况，由于需要保持较大的历史，马尔可夫预取在预测表较小的情况下性能较差。RP 对于具有可预测的历史重复的应用程序表现良好。任意跨步预取对于许多使用马尔可夫预取和快速原型的应用程序来说表现良好，但是对于没有跨步访问的应用程序来说，它表现不佳。总的来说，DP 提供了最高的预测精度，并且适用于不同的页面大小和 TLB 配置。当条纹图案存在时，DP 寻求利用条纹图案，并且在没有条纹图案的情况下变得越来越基于历史。因此，它的表现接近于更好的基于历史的技术。

5. 提高能源效率的技术

在本节中，我们将讨论节省 TLB 动态能量的技术(5.1-5.3)和泄漏能量(5.4-5.5)。下一节将讨论使用虚拟高速缓存设计来降低 TLB 动态能量(6)。

5.1. 重用最近的翻译

Kadayif 等人[59]请注意，由于指令流的高度局部性，通过存储和重用对当前页面的转换，可以显著减少对 ITLB 的访问，因为只有在访问另一个页面时才需要访问 ITLB。他们提出了四种策略来实现这一点，它们使用硬件和/或软件控制。在第一种策略中，硬件检查由个人电脑生成的虚拟仪器，并将其与当前存储的翻译的虚拟专用网部分进行比较。在匹配的情况下，避免进入 ITLB。这种策略的局限性在于需要对每次指令提取进行检查。在第二个策略中，对 ITLB 的访问由编译器明确控制。执行可以转换

在两种情况下的指令页之间:在分支指令上和在页边界上。他们假设对于分支指令的目标，总是执行 ITLB 访问。此外，对于页面边界情况，编译器以目标作为下一条指令(即，下一页上的第一条指令)来插入分支指令。这将第二种情况简化为第一种情况。

第二种策略的一个限制是假设分支目标在不同的页面上过于保守。为了避免这种情况，在第三种策略中，编译器静态地分析代码，并发现目标是否在同一页上，当目标作为“立即操作数”或“PC 相对操作数”提供时，就会出现这种情况。然而，第三种策略仍然是保守的，因为它不能使用运行时信息。第四种策略使用运行时信息来查找分支目标是否指向不同的页面，以及分支是否被采用。这是通过修改分支预测器实现的。他们表明，所有四种策略都显著降低了 TLB 能量。在这四种策略中，第二种策略消耗的能量最高。第一个和第三个策略的性能相当，而第四个策略是最好的，它非常接近一个只有在需要时才访问 ITLB 的最佳方案。

Ballapuram 等人[54]评估具有多端口 TLB(例如，4 端口和 4 端口)的多问题处理器，并观察到在同一周期和连续周期内发生的大部分 d TLB 访问是针对同一页面或甚至同一高速缓存块的。他们提出了两种技术来利用这种行为。对于 N 宽处理器，第一种技术使用 N 个(即“N 选 2”)比较器来检测同一周期中的相同虚拟专用网络。基于此，只有唯一的虚拟专用网被发送到 DTLB。例如，图 9 和 9(b)显示分别到达 TLB 四个港口的虚拟仪器和相应的虚拟专用网络。通过周期内压缩，可以比较同一周期中的虚拟专用网络，以避免相同虚拟专用网络的 TLB 访问。在图中 9(c)可以避免周期 J 中的两个虚拟专用网络和周期 J+1 中的一个虚拟专用网络的访问。这种“周期内压缩”技术可以减少 TLB 的端口数量和访问延迟。

他们的第二种技术被称为“循环间压缩”，锁存最近使用的 TLB 条目及其虚拟专用网络。在下一个周期，读取锁存器以获得转换。对于多端口 TLB，每个端口使用一个锁存器。在图中 9(d)在周期 J+1 中可以避免对第一和第二端口的访问，因为它们与在周期 J 中观察到的相同 47] (5.2).他们表明他们的技术显著降低了 TLB 能量。他们进一步结合了这两种技术，并将其应用于 ITLB 和 DTLB，并观察到这种方法提供了更大的节能。

0x8395B	0x8395B	0x8395B	0xEEFFD
–	–	0x87241	–

图 9。(一)四个顶级域名端口的虚拟地址示例，以及(二)它们对应的虚拟专用网络。(c)在周期 J 和 J+1 (d)中应用“周期内压缩”后的虚拟专用网络在同一端口连续两个周期应用“周期间压缩”后的虚拟专用网络[54].因此，这两种技术都利用局部性来减少 TLB 访问。

重新排序”。对于嵌套循环的最内部循环，如果两个数组具有相同的大小和相同的引用函数(即访问模式)，它们会在两个数组之间执行“数组交织”，如图所示 10。然后，为了提高指令调度的有效性，如果至少存在一条指令，从不同的迭代中连续调度它的两个实例不会导致指令间页转换，则它们执行“循环展开”。在这两者之后，一起执行“指令调度”和“操作数重新排序”，其中前者聚集访问相同页面的指令，而后者通过重新排序操作数来改变存储器访问模式。他们证明了使用这两种方法最小化页面转换的问题是 NP 完全的，因此，他们提出了一种贪婪的启发式方法。这种启发式方法记录最后一条预定指令。然后，在所有准备调度的指令中，选择一对指令(如果可能，否则是单个指令)，该对指令不会导致它们之间以及与最后调度的指令之间的页面转换。

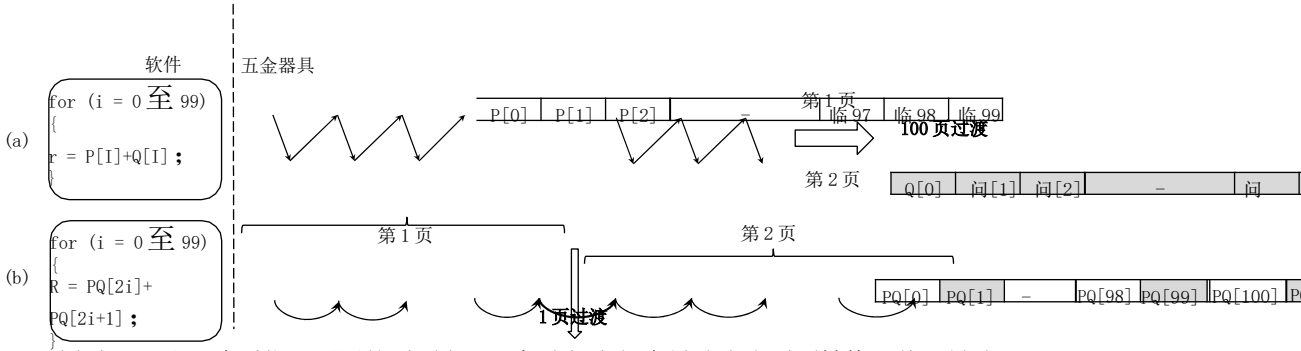


图 10。(a)由于 P 和 Q 阵列位于不同的页面上，一起访问它们会导致许多页面转换，从而导致 TLB 效率低下。(b)使用阵列交错，可以显著减少页面转换。[48]

对于 ITLB 优化，他们注意到当(1)程序的循环块超过页面限制，并且(2)被调用方和调用方函数在不同的页面上时，连续的指令可能来自不同的页面。他们提出了一种代码放置策略，重新分配程序的功能块，以最小化页面切换。例如，在单个页面上使用函数位置限制，以确保将函数放置在同一页面中。循环和函数是按照循环计数和对它们的调用次数的递减顺序来考虑的，这使得这是一种贪婪的方法。他们表明他们的技术在减少页面转换和 TLB 能量方面是有效的。Kadayif 等人[16]提出编译器技术来智能地将一些翻译存储在“翻译寄存器”(TRs)中，这些寄存器可以最大限度地重复使用，以避免访问 DTLB。编译器指示哪些 TRs 应该用于不同的数据访问，并且只有当保证翻译存储在其中时才部署它们，否则访问 DTLB。为此，编译器在加载/存储指令本身中插入关于访问 DTLB 或 TR 的信息，运行时可以看到这些信息。编译器识别四种类型的数据访问。第一种类型是全局标量变量，它是静态可分析的。如果多个变量在同一个页面上，它们的翻译将被加载到 TR 中。第二种类型是全局数组变量，通常在循环中寻址。编译器应用循环变换，例如，对于基于步幅的阵列扫描，单个循环可以被分成两个循环，使得在虚拟页面上执行外部循环迭代，其中翻译被存储在 TR 中，并且该 TR 被用在内部循环中，其中数据访问发生在内部循环中一页纸。

第三类是动态分配数据的堆访问(例如，通过 malloc())，通常使用指针访问。通过强制分配的内存在一个页面内(假设分配的大小小于一个页面)，并跟踪内存被访问的周期，可以在此期间加载和使用 TR。第四类是显示高度局部性的堆栈访问，特别是在 C 和 Fortran 语言中，数组是通过指针/引用传递的。对他们来说，一个 TR 就足够了。它们表明，对于基于数组和指针的

应用程序，他们的技术通过避免 DTLB 访问提供了巨大的节能，尽管他们的技术增加了执行的指令数量。

5.2. *利用语义信息*

根据编程语言的惯例，虚拟地址空间通常被划分为多个不重叠的语义区域，如堆栈、堆等。，表现出不同的特点。一些技术利用这些信息来重新构建和优化 TLB。

Lee 等人[47]建议在不同的流中分离堆栈、堆和全局静态访问，并将它们导向专用的 TLB 和缓存组件。他们把一个 DTLB 分解成一个“堆栈 TLB”，一个“全球静态 TLB”和一个“堆 TLB”。堆 TLB 也被用作堆栈和全局静态 TLB 的二级 TLB。每个 DTLB 通道都通向相应的 TLB。与访问 FA TLB 相比，这种方法通过使用小得多的堆栈和全局静态 TLB 来服务大多数访问，从而避免冲突未命中并减少动态能量。此外，与单片 TLB 相比，这些小型 TLB 可以很容易地多端口化，并且面积/延迟开销更小。类似地，它们将 DL1 缓存分解为一个小的“堆栈缓存”、一个小的“全局静态缓存”和一个大的缓存，用于服务剩余的访问(参见图 11)。他们的技术在 DTLB 和 DL1 缓存中节省了大量能源。

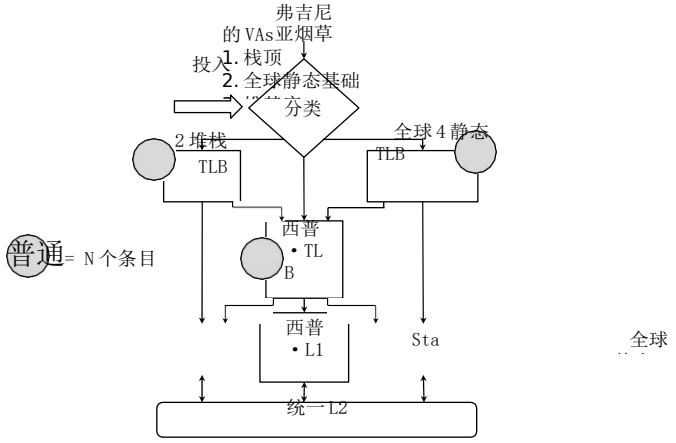


图 11。TLB 和缓存的语义感知分区[47]。通过将堆栈、堆和全局静态访问定向到专用的 TLB 和高速缓存组件，与对所有流使用单个 TLB 的传统设计相比，可以更有效地利用它们的不同特性。这些 TLBs 缓存的大小是根据每个流中存在的局部性来确定的。

Ballapuram 等人[44]请注意，在 Java 应用程序中，对 ITLB 的大多数访问都来自于获取和执行本机代码的堆。然而，这些访问与 JVM 的代码访问冲突，并增加了 ITLB 未命中。他们建议将 ITLB 划分为一个用于 Java 代码访问的 ITLB 和一个用于 Java 应用程序堆访问的 ITLB。第一个 ITLB 是小的(2 个条目)，因为 Java 代码是静态的，而第二个 ITLB 是大的(32 个条目)，因为 Java 应用程序代码是动态的，包括在运行时创建和释放对象，因此从相同或连续的页面分配新对象的机会很低。他们还注意到，Java 应用程序通常创建短暂的对象，这些对象表现出良好的局部性，并占总内存访问的很大一部分。在此基础上，他们进一步提出将 Java 应用程序代码的 ITLB 划分为两个级别，其中较小的第一级 ITLB 支持 Java 应用程序中的动态对象代码访问，较大的第二级用作备份。

此外，在 Java 应用程序中，全局静态数据区域中的读取可以忽略不计，大多数数据内存访问都是由于对堆的读取和写入。在此基础上，他们提议将 DTLB 分为“读 DTLB”和“写 DTLB”。加载地址和存储地址分别用于读取 DTLB 和写入 DTLB，这避免了争用。在执行 Java 应用程序期间，JVM 引用读 DTLB，而用于读取堆数据的应用程序数据引用和用于本机代码写访问的写 DTLB 由动态或 JIT 编译器访问。他们的技术拯救了 TLB

能量对性能的影响可以忽略不计，并且通过结合所有技术(参见图 12)，节能可以进一步提高。

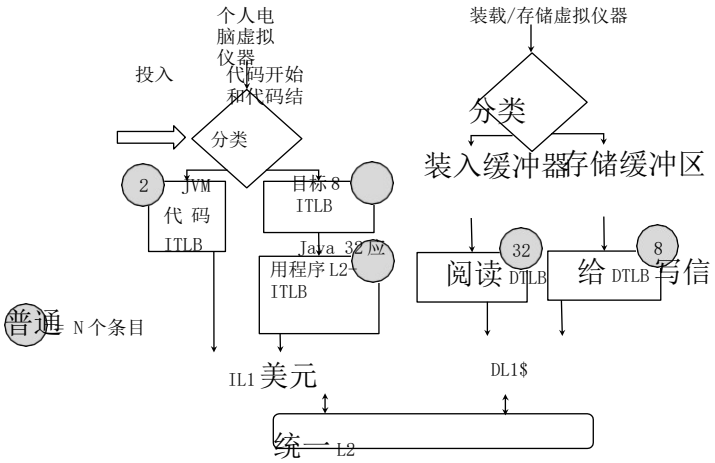


图 12。面向 Java 应用的语义感知 TLB 设计[44]

Ballapuram 等人[58]请注意，在许多应用程序中，访问堆栈、全局数据和堆的频率会降低。由于堆访问的随机性，预测堆访问具有挑战性。相比之下，堆栈访问是高度可预测的，应用程序访问的全局数据的大小在编译时是固定的。因此，堆栈和全局数据的地址转换只需要很少的位，而不是完整的虚拟专用网。基于这些，他们将 L1 TLB 重新组织成“堆栈 TLB”和“全局静态 TLB”，而 L2 TLB 则用于所有数据访问。他们进一步提出了“堆栈 TLB”的推测翻译方案和全局静态 TLB 的确定性翻译方案。这两种技术都通过减少每次 TLB 访问中预充电的位数来节省能量。对于任何 VPN 转换，第一种方案推测性地预充电与上次转换中预充电的位数相同的位数。然后，它检查翻译的正确性。如果发现转换不正确(这种情况很少发生)，它将被挤压，预充电的位数将增加，并在下一个周期获得正确的转换。第二种方案的工作原理是，根据应用程序的全局数据足迹，计算寻址全局数据所需的最小位数。由此，需要预充电和比较的低位比特的数量由加载器确定性地计算，因此，该方案不会导致性能开销。在应用程序执行期间，全局数据访问中剩余的高阶位保持不变。他们的方案节省了大量能源，性能损失可以忽略不计。

方等[42]请注意，在并行访问模式下，所选 L1 缓存集的所有 K 路的标签和数据都是并行访问的。因为这些访问中至少有 K 1 个是无用的，所以这种方法会导致能量浪费。他们提出了一种使用特权级和存储区的技术
减少 TLB 和 L1 高速缓存能量的信息。他们注意到“用户模式”指令提取不能命中存储“内核模式”代码的缓存块。因此，它们将这种特权级信息与 IL1 和 ITLB 的每个块一起存储，并且在用户模式指令提取时，只参考存储用户模式代码的方式。类似地，对堆数据的访问不能命中存储堆栈数据的块，反之亦然。通过将该信息存储在每个 DL1 高速缓存块中，避免了对那些保证会导致未命中的路的访问。他们的技术在不影响访问延迟或命中率的情况下节省了缓存和 TLB 动态能量。

5.3. *使用翻译前方法*

薛等[18]提出一种通过页码预测和预翻译来减少 TLB 访问的技术。它们采用 PT L1 缓存，避免了同义词问题。他们指出，在基位移寻址模式下，基地址(BA)是最终 VPN 的一个很好的预测器，原因可能有三个:(1)大部分静态指令使用 BAs，BaS 精确地映射到一个 VPN，用于整个程序的执行。对于页码不同的情况

页面与 BA 相比，前面的页码可以作为预测。(2)这种 BA 的动态部分甚至高于静态部分，并且它们可以被容易地预测(3)如果 BA 到 VPN 的映射在大的执行持续时间内保持不变，即使许多 BA 在整个执行期间具有一对多的映射，也可以进行精确的预测。他们观察到，对于 PARSEC 和 SPEC 基准，原因(2)和(3)主要有助于页码的高度可预测性。此外，他们的方法实现了高预测精度(ITLB 为 99%，DTLB 为 52%-75%之间，不同的国际审计准则)。

基于此，他们使用 BA 来访问预测器表，该预测器表包括预测页面的标签和 TLB 条目。使用这种支持，他们的技术预测计算出的虚拟地址的页面。正确的预测消除了 DTLB 访问的需要，而不正确的预测不会招致任何损失，因为计算出的虚拟地址用于访问 DTLB 进行翻译(参见图 13)。如果在表中找不到 BA，则不会进行预测。在这种情况下，以及在预测错误的情况下，用预测的 VA 和翻译来更新表格。至于不使用 BA 的取指令，它们利用页面级局部性，并基于前一个 PC 预测下一个 PC 页面。对于指令占用空间小的工作负载，该方案运行良好。通过加强包容性，这张桌子与 L1 • TLB 保持一致。他们的技术在不影响性能的情况下显著降低了功耗，并提供了比使用 LO TLB 更高的性能/能量改善。

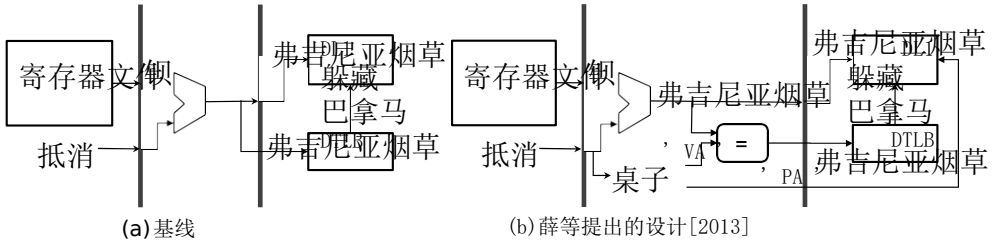


图 13。(a)基底位移寻址模式下的基线设计(b)薛等提出的设计[18]用于避免 DTLB 访问

5.4. *使用 TLB 重构*

一些处理器使用两级 TLB 设计，每个页面大小有不同的 L1 顶级内存，例如 4KB、2MB 和 1GB，如图所示 14。Karakostas 等人[5]请注意，在这些处理器中，大的动态能量被 L1 • TLB 消耗(由于对多个 TLB 的访问)和小页面大小(4KB)的页面行走消耗。此外，所有 L1 顶级域名对点击率的贡献并不相同，特别是在使用增加 TLB 覆盖范围的方案时，因此，访问所有 L1 顶级域名可能不会提高性能。在此基础上，他们提出了一种 TLB 路重构技术，通过记录对 L1 TLB 路的点击来发现每条路的效用。然后，如果发现实际失败率与路数较少的失败率之间的差异低于阈值，则他们的技术以 2 的幂粒度禁用路。如果两个时间间隔之间的失败率差异大于阈值，则启用所有方法。他们进一步增加了 L1 范围的 TLB，并将其 TLB 重新配置方案应用于“冗余内存映射”方案[27]。他们表明，他们的技术节省了地址转换的动态能量，TLB 小姐惩罚略有增加。

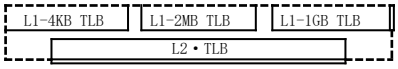


图 14。一个两级 TLB 组织，使用不同的 L1 顶级域名支持多种页面大小。

Balasubramonian 等人[41]展示了可重新配置的 TLB 和缓存组织，它们被设计为逻辑上的两级和物理上的一级非包含层次结构。两个级别之间的划分可以动态执行，以控制延迟和容量。例如，对于 2MB 的总缓存空间，L1 大小的范围可以在 256KB 到 2MB 之间(剩余空间用于 L2)，并且

关联性可以在 1 到 4 之间。同样，512 入口的 FA TLB 可以设计成 64 个入口的倍数，总共有八种可能的尺寸。他们的管理技术寻求根据每个应用程序的命中/未命中延迟容限，在每个阶段的未命中率和命中延迟之间实现正确的权衡。在分支频率和高速缓存未命中计数发生变化时，会检测到相位变化，此时，每种高速缓存配置都会评估 100K 个周期，并使用提供最佳性能的配置，直到检测到相位变化。除了缓存重新配置，它们还执行 TLB 重新配置。如果 TLB 未命中处理程序周期超过 1M 周期间隔的总执行时间的阈值分数(例如，3%)，则 L1 TLB 大小增加。此外，如果间隔中引用的 TLB 条目的数量少于一半，则 L1 TLB 的大小会减小。他们表明，他们的技术在提高性能和能效方面是有效的，不同的缓存/TLB 大小对于不同的应用/阶段是最佳的。

Delaluz 等人[45]介绍了两种动态改变 DTLB 大小的技术。在第一种技术中，在每个间隔(例如，1M TLB 访问)中，DTLB 大小最初被设置为最低(例如，原始大小的 1/64)。在固定周期(例如，100K TLB 访问)之后，如果未命中率高于前一间隔中的未命中率，则 DTLB 大小加倍，如果未命中率较低，则大小减半。他们的第二种技术基于这样的观察，即很大一部分 TLB 条目是死的(即，在驱逐之前不太可能看到访问)。他们对每个条目使用计数器，这些计数器在命中和未命中时进行适当更新。在固定的(例如，32 个)未命中间隔内未被访问的条目被认为是死的，并且是驱逐的候选。如果没有发现无效条目，则使用 LRU 替换策略选择候选条目。因此，通过优先驱逐死条目，他们的技术旨在将活条目在 TLB 保留更长时间。他们的技术在不影响失败率的情况下节省了大量能源，第二种技术比第一种技术更有效。

5.5. *用非易失性存储器设计 TLB*

刘等[31]请注意，GPU 工作负载的高内存访问强度以及对 TLB 的罕见写入使 STT-RAM 成为设计 GPU TLBs 的合适候选，因为 STT-RAM 的密度高于 SRAM，但写入延迟/能量也更高[85,86].此外，由于对 TLB 的大多数读取都指向少数条目，因此他们提出了一种方案，该方案试图通过选择性地权衡容量来减少热块的读取延迟。该方案使用“差分检测”策略，该策略减少了读取延迟，但使用两个单元来存储一个值。为了找到热门数据，他们注意到数据项的页面大小与其访问模式有关，因此，大页面的页面显示出比小页面更高的访问频率。基于此，在将 PTE 加载到 TLB 时，大于 4KB 页面大小的 PTE 以 DS 模式存储，否则以正常模式存储。由于这种启发式方法可能无法捕获几个热的小页面，因此它们使用计数器来记录访问频率，当计数器超过阈值时，通过使用未使用的条目将其转换到 ds 模式。至于从 DS 模式降级到正常模式，当 DS 模式中的 TLB 条目准备退出时，它首先被转换到正常模式。只有正常模式下的条目会被直接逐出 TLB。与基于静态随机存取存储器的 TLB 和简单的基于 STT-RAM 的 TLB 相比，他们的技术提高了性能和能效。

6. 虚拟高速缓存设计技术

不同的作品使用不同的方法来解决虚拟缓存中的同义词问题。一些作品将针对不同虚拟设备的访问重新映射到单个虚拟设备[50,55】，这就叫领先[50]或主要[55]并且是基于第一次触摸(6.1).其他作品提出了混合虚拟高速缓存设计[17,51]，其中 PA 用于具有同义词的地址，VA 用于非同义词地址(6.2).处理同义词的其他方法包括维护两级缓存之间的指针，以便在替换时使同义词无效[52,60] (6.3)，使用智能一致性协议[29] (6.4)，使用在共享数据的应用程序之间共享的单个全局 VA 空间(6.5)并使用虚拟和物理标签[17,80].我们现在回顾一些虚拟缓存设计。

6.1. 重新映射对同义词的访问

Yoon 等人[50]提出了一种虚拟 L1 高速缓存设计，其执行同义词的动态映射，即，对于用一个 VA(比如 Vi)高速缓存的数据块，对具有不同地址(Vj)的相同数据块进行的同义访问被重新映射以使用 Vi。它们将物理页 P 的“活动同义词”定义为在给定时间间隔 T 内与 P 同义的虚拟页，其中 T 表示来自页 P 的一个或多个块存储在高速缓存中的间隔。同义页面组称为“等价页面集”(EPS)。他们注意到，对于典型大小的 L1 缓存(例如，32-64KB)，具有活动同义词的页面数量很少，并且这种活动同义词的数量也很少。基于此，只需要一个小表来记住[Vj，Vi]重映射。此外，在程序执行的不同时间，来自相同 EPS 的不同 VAs 是前导 VAs，因此，需要动态跟踪重映射信息。

此外，对于大多数应用程序来说，从具有活动同义词的页面访问缓存块，尤其是使用不同虚拟地址缓存的页面，只占总访问的一小部分。因此，重映射表将很少被访问。基于此，他们使用 VA 的散列位来使用另一个小存储结构预测表中同义词的存在，并避免在没有同义词存在时访问表。在 L1 小姐，TLB 访问与虚拟仪器和 PA(皮)获得。Pi 用于访问另一个名为“活动同义词检测”的表
表”。在不匹配的情况下，创建一个[Pi，Vi]条目，Vi 用作前导 VA。在匹配时，记录 Vk，如果 Vi = Vk，则检测到活动同义词，并在重映射表中创建一个条目。总的来说，由于使用唯一的前导 VA 来访问高速缓存中的页面，
他们的技术简化了内存层次结构的管理。他们表明，他们的技术显著降低了 TLB 能量，并提供了与理想但不可行的虚拟缓存相同的延迟优势。

邱等[55]提出了一种通过跟踪同义词来支持虚拟缓存的技术。对于具有同义词的页面，一个虚拟变量称为主虚拟变量，其余虚拟变量称为次虚拟变量。对于没有同义词的页面，VA 本身就是主 VA。对于虚拟缓存，主 VA 作为每个页面的唯一标识符，而不是 PA。因此，主虚拟阵列的使用解决了单核和多核处理器的缓存层次结构中的一致性问题。为了将二级虚拟仪器的翻译存储到主虚拟仪器中，它们对每个内核使用“同义词后备缓冲区”(SLB)。在他们的设计中，L1 缓存使用 VA 作为索引，使用主 VA 作为标记。对于 L2 缓存，索引和标记都是从主 VAs 派生的。L2 缓存存储反向指针来检测 L1 缓存中的同义词。如果在 SLB 未命中，将使用 VA 检查 L1 缓存标签，该 VA 可以是主要的，也可以是次要的。L1 缓存命中确认主虚拟设备，但如果未命中，数据块可能在不同的集中。因此，L2 缓存被访问，使用反向指针的命中确认该地址是主虚拟地址，并导致 L1 缓存未命中。否则，L2 缓存或内存将返回数据。最终，在 TLB 检测到同义词，确定主词还是次词。对于次词，次/主译名存储在 SLB。由于同义词很少，SLB 非常小，因此可以顺序访问或与 L1 缓存并行访问。SLB 的覆盖率比 TLB 高得多，因为每个 SLB 条目代表一个完整的部分，包括几页，因此，SLB 的覆盖率随着内核数量、应用程序占用空间和物理内存大小而扩展。SLB 不记录 PAs，因此不受 V2PA 绘图变化的影响。SLB 只是在消除同义词时才被刷新，这比 V2PA 映射中的变化要少得多。此外，SLB 支持的缓存比 TLB 支持的缓存具有更高的命中率。它们表明，8-16 个条目的小 SLB 可以充分解决同义词问题，并且产生的性能开销可以忽略不计。

6.2. 在虚拟缓存中同时使用 PAs 和 VAs

Park 等人[51]提出一种混合虚拟缓存设计，其中 VAs 和 PAs 分别用于非同义词和同义词页面。因此，它们一致地使用 VA 或 PA 来唯一标识所有缓存级别中的每个物理内存块，这避免了一致性问题。对于翻译来说，每个 VA 都进入一个“同义词过滤器”，由于同义词很少出现，所以它是使用布隆过滤器设计的。过滤器正确地确定地址是否是同义词，尽管它可能错误地将非同义词确定为同义词。对于非同义词，VA(和 ASID)用于所有缓存级别，当需要引用内存时，使用 ASID+VA 引用 TLB

以获得 PA(称为“延迟翻译”)。对于被过滤器预测为同义词的地址，访问 TLB，但如果结果为假阳性，则不执行翻译，仅使用 ASID+VA(参见图 15)。当虚拟页面重新映射或页面状态从私有(非同义词)变为共享(同义词)时，页面的缓存块将从缓存层次结构中清除，尽管此类事件很少发生。他们表明，他们的技术大大降低了 TLB 功率。

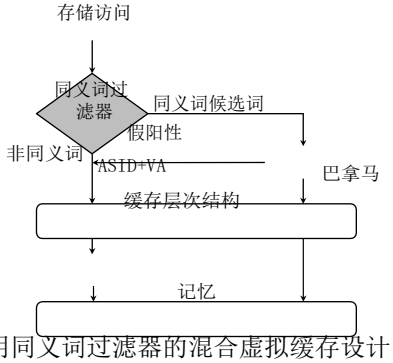


图 15。使用同义词过滤器的混合虚拟缓存设计[51]

Basu 等人[17]请注意，在大多数应用程序中，只有极少数页面具有同义词，并且它们被访问的频率相对较低。此外，大多数同义词页只看到读访问，因此不会产生读/写不一致。此外，由于页面映射或保护/许可的改变而导致的 TLB 失效很少发生，并且它们通常涉及进程的所有页面，因此可以很容易地用高速缓存刷新来处理。在此基础上，他们提出使用混合虚拟缓存技术，用 VAs 缓存一些数据，用 PAs 缓存其他数据。在他们的技术中，操作系统根据虚拟分配程序定义的标志(例如，虚拟机私有、虚拟机共享)和应用程序的提示，确定具有读/写同义词或映射/权限频繁变化的地址。这种地址使用 PAs 进行缓存。对于剩余地址，使用虚拟缓存；只有在 L1 缓存未命中时才访问 TLB，并且只需要查找一半的 L1 缓存路。由于逐出脏的 L1 高速缓存块需要写回物理 L2 高速缓存，因此它们会在每个虚拟标记的块中添加一个物理标记，以平滑地处理这种逐出。此外，在这些物理标签中顺序搜索可能保存一个块的所有集合减轻了当 L1 高速缓存接收一致性消息时 PA 到 VA 转换的需求，因为 L2 高速缓存使用 PA 来处理一致性。当需要时，他们的技术默认为物理缓存，因此提供了与传统操作系统/体系结构的兼容性。与 VI-PT 缓存相比，他们的技术显著减少了 TLB 访问，还节省了 L1 动态能量。

6.3. *将数据筛选器缓存设计为虚拟缓存*

Bardizbanyan 等人[52]介绍在 DL1 缓存之前访问的“数据过滤器缓存”(DFC)的设计。DFC 使用虚拟标签，这避免了 DTLB 访问的需要，并且允许在流水线的早期访问 DFC，例如，与存储器地址生成并行。因此，离散傅立叶变换未命中不会导致损失，而离散傅立叶变换命中避免了通常伴随 DL1 命中的负载危险。DL1 缓存包含 DFC，每个 DFC 块存储存储该块的 DL1 缓存路以及页面保护信息。在替换 DFC 块时，将其 DL1 高速缓存路和索引(从 DFC 标签的最低有效位获得，因为 DL1 高速缓存是虚拟索引的)与其他 DFC 块的路和索引进行比较，如果存在同义词 DFC 块，则使其无效。这解决了同义词问题，并且由于 DFC 的小尺寸而导致可忽略的开销。DFC 使用直写策略，这提供了更简单实现的优势，并且由于存储不如加载频繁，因此只会导致很小的开销。在上下文切换中，DFC 无效，这将处理同音异义问题。他们还讨论了为加载操作推测性地访问 DFC 的策略，以及为填充操作使用关键字优先填充方案。他们的技术降低了 DL1 缓存和 DTLB 的能耗，也提高了性能。

6.4. 使用谨慎的一致性协议

Kaxiras 等人[29]请注意，确保多核处理器中虚拟 L1 缓存的一致性具有挑战性，因为由于同义词的存在，发送到虚拟缓存的一致性请求需要 PA 到 VA 地址转换。为了解决这个问题，他们建议使用一种不发送任何请求流量(即降级、转发或无效)到虚拟 L1 缓存的协议。因此，他们使用无目录协议，该协议使用同步和“无数据竞争”语义实现“自失效”和“自降级”。他们还考虑了三种 TLB 布局方案，如图所示 16:(a)“核心-TLB-L1 $-网络-有限责任公司”(b)“核心-L1 $-TLB-网络-有限责任公司”和(c)“核心-L1 $-网络-TLB-有限责任公司”。选项(a)和(b)分别使用 PT 和 VT L1 缓存，并且两个选项都使用每个内核的 TLB。他们的技术允许将 TLB 放在 L1 和网络之后，以便在逻辑上共享它，并在访问 LLC 之前访问它(图 16(c))。这避免了需要保持一致的每核 TLB 的开销。此外，共享 TLB 通过避免复制和简化共享/私有分类来提供更高的有效容量。他们的技术简化了连贯性，同时仍然支持同义词。与 PT L1 缓存相比，他们的 VT L1 缓存(选项(c))减少了 TLB 访问，共享 TLB 的使用有助于减少 TLB 未命中。此外，与使用“修改的、排他的、共享的、无效的”(MESI)目录协议(需要 PA 到 VA 的转换)相比，他们的技术提供了能量和性能优势。

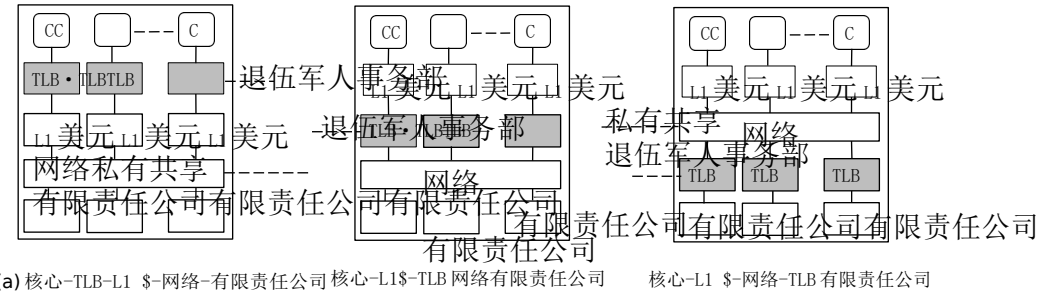


图 16。TLB 配售的不同选择[29]

6.5. 消灭 TLB

Wood 等人[10]建议消除 TLB，转而使用 VT 数据缓存来存储 pte。为了避免同义词问题，他们规定所有进程共享一个全局 VA 空间，共享数据的进程必须使用该空间。在他们的设计中，pte 与数据和指令存储在同一个缓存中，并使用 VAs 进行访问。为了翻译一个虚拟仪器，需要提取它的 PTE 来确定 PA。为此，计算 PTE 的全局 VA。通过强制所有页表在 VA 空间中是连续的，可以通过访问以 VPN 为索引的页表来获得 PTE。对于多核处理器，他们的方法避免了 TLB 一致性问题，因为页表只缓存在核心的数据缓存中，而不是 TLB。因此，缓存的 pte 会按照缓存一致性策略的指导进行一致的更新。由于转换仅在缓存未命中时执行，因此在缓存中存储 pte 对其正常行为和整体性能的影响很小。他们表明，他们的方法的性能主要取决于缓存未命中率(以及缓存大小)，对于大容量缓存，他们的方法提供了良好的性能。

Jacob 等人[62]将软件管理的地址转换作为一种完全避免使用 TLB 的方法。它们使用虚拟磁带库缓存层次结构，有限责任公司未命中调用操作系统缓存未命中处理程序，该处理程序翻译地址并获取相应的物理数据。他们指出，对于大型有限责任公司，失败率很小，因此不经常需要翻译。此外，由于 LLC 未命中由内存管理器处理，因此操作系统可以灵活地实现任何 V2PA 映射、替换策略、保护方案或软件定义的页面大小。因此，他们的方法允许完全在软件中定义超页面、共享内存和细粒度保护，这比基于硬件的实现提供了更高的灵活性。他们表明，对于大型有限责任公司和合适的块大小(≥ 64B)，他们的方法表现接近基于硬件的

使用 TLB 的方法。他们的技术性能对缓存大小和块大小非常敏感，而基于硬件的方法对 TLB 配置非常敏感。

7. 图形处理器和中央处理器-图形处理器系统中的 TLB 管理

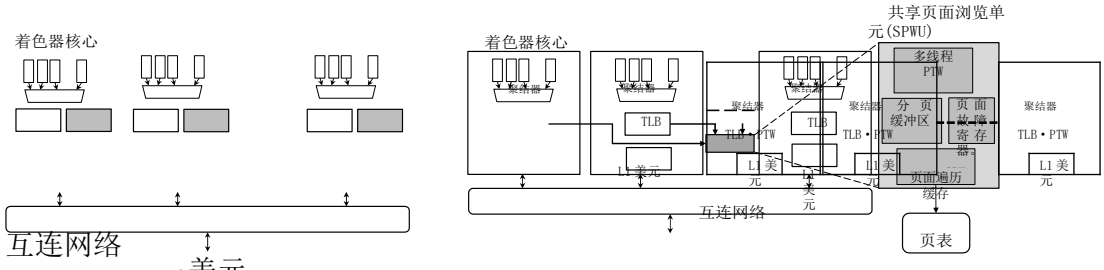
虽然传统的中央处理器-图形处理器异构系统为中央处理器和图形处理器使用单独的虚拟地址空间和虚拟地址空间，但最近的系统正朝着完全统一的地址空间发展，因为它们通过使数据对象可从中央处理器和图形处理器全局访问来促进编程并减少数据传输开销。然而，中央处理器和图形处理器体系结构的根本差异，以及统一地址空间的要求，需要对图形处理器上的地址转换进行详细的研究。我们现在回顾一下 TLB 对图形处理器和中央处理器-图形处理器系统的设计方案。

Vesely 等人[24]分析商用集成异构处理器上跨 CPU 和 GPU 的共享虚拟机。他们观察到图形处理器上的 TLB 缺失比中央处理器上的延迟高出近 25 倍。这是因为 TLB 未命中请求和响应通过 PCIe 传输到 IOMMU(“外围组件互连高速”)，而 IOMMU 的 PTW 无法访问可能存储最新 pte 的 CPU 缓存。此外，未命中需要重新安排扭曲(或波前)。为了隐藏这种延迟，在服务 TLB 未命中时提高并发性的技术至关重要。他们还注意到，IOMMU 可能允许多达 16 个同时的页表访问，并且在此之外，由于排队效应，页表访问的延迟迅速增加。它们进一步运行 4KB 和 2MB 页面大小的应用程序。2MB 页面大小的使用显著减少了 TLB 失误，但是，它仅在某些应用程序中提高了性能。此外，与缓存和内存延迟相比，内存访问模式的差异对地址转换开销的影响更大。此外，TLB 预取的有效性取决于应用程序内存访问模式的局部性。GPU 上的 TLB 击落延迟与 CPU 上的相当，并且使单个 GPU TLB 条目无效的延迟与刷新整个 GPU TLB 的延迟几乎相同。

Pichai 等人[25]请注意，与没有 TLB 的设计相比，设计类似于 CPU TLB 的 GPU TLB 会导致显著的性能损失。这种图形处理器 TLB 设计是:128 入口，3 端口，每核心 TLB 是阻塞的(即，在 TLB 小姐，另一个扭曲被切换-在继续执行，如果它没有内存指令)。此外，TLB 和 PTW 与缓存并行访问。性能损失的原因是 GPU 的“循环弯曲调度”交错了不同线程的内存访问，这增加了缓存和 TLB 未命中。此外，由于锁定步骤的执行，一个线程中的 TLB 未命中也会使其他经纱停滞。此外，来自不同经纱的 TLB 脱靶量通常发生在紧密接近的地方，这些经纱由 PTWs 进行序列化。他们建议对 TLB 和 PTW 进行简单的扩建，以减轻这种性能损失。首先，将端口数量从 3 增加到 4 恢复了大部分损失，因为在 TLB 将请求合并到单个访问中的同一个 PTE 之前，合并单元需要的转换少于 32 次。

第二，他们提出了两种非阻塞 TLB 设计:(1)只要它们导致 TLB 命中，即使有内存指令，切入曲速也会继续。TLB 小姐，曲速被切断了。(2)在 TLB 命中的线程访问缓存，而不会在 TLB 未命中的同一经线中等待另一个线程。这提高了缓存命中率，因为该扭曲的数据可能会在被已接入扭曲的数据逐出之前出现在缓存中。设计(2)与设计(1)一起应用。他们指出，这两种设计显著提高了性能。他们还提出了 PTW 调度策略，通过结合使用上述策略，性能接近理想的(但不可行的)32 端口、512 入口 TLB，并且没有访问延迟开销。他们提出的设计如图所示 17(a)。他们还证明了将 TLB 感知集成到曲速调度和动态曲速构建方案中可以将包含 TLB 的开销控制在可接受的范围内。

Power 等人[26]提出一种与(x86-64) CPU 页表兼容的 GPU MMU 架构。他们的发现基于从执行罗迪尼亚基准中获得的见解。他们的基线设计是一个“理想的内存管理单元”，每个内核有无限大小的 TLB，页面遍历周期为 1 个周期。他们的第一个设计使用了每个核心的私有内存管理单元，这消耗了大量的功率，也位于关键路径上。他们的第二个设计将 TLB 移动到降低 TLB 的合并单元之后



(a) 采用内核级 TLB 和 PTWs 的设计[25]

(b) 使用每个内核的 TLB、共享的高线程 PTW 和共享的页面遍历缓存(reg。= register) [26]

图 17。为图形处理器提出的内存管理单元设计。Pichai 等人[25]建议使用每个内核的 PTWs，而 Power 等人[26]提出一种多线程共享 PTW 设计，带有页面遍历缓存。

极大地访问(例如，85%)。他们指出，合并单元大大减少了内存访问，因此，他们的第二个设计在暂存内存访问和合并单元之后使用了每个内核的专用 L1 TLB。因此，只有全局内存引用才能访问 MMU。但是，他们注意到，由于 GPU 是高度带宽密集型的，平均每个内核中随时都可能有 60 到 1000 个页表遍历处于活动状态，因此，传统的阻塞 PTW 设计会导致很大的排队延迟。因此，第二种设计的性能很差。

因此，他们的第三个设计使用多线程 PTW，所有图形处理器内核共享 32 个线程。在每个核心的 L1 TLB 中的 TLB 未命中时，在共享的 PTW 中执行页表访问。平均而言，第三种设计的性能与第二种设计相似，提供了理想 MMU 近 30%的性能。第三种设计性能差的原因是来自一个内核的请求排在其他内核之前。此外，图形处理器 TLB 失败率远高于中央处理器 TLB。为了减少 TLB 未命中延迟，他们的第四个设计在共享 PTW 中添加了一个页面遍历缓存，从而减少了页面表遍历的延迟和内核的停滞时间。因此，由于避免了排队延迟，该设计实现了与理想 MMU 设计几乎相同的性能。第四种设计是他们推荐的设计，如图所示 17(b)。他们还讨论了页面错误的处理，TLB 冲洗和击落。对于几个应用程序，内核之间共享 TLB 条目的程度较低，因此，共享的 L2 TLB 不会提高性能，添加页面遍历缓存比共享的 L2 TLB 更有效。类似地，TLB 的预取器在性能上只提供了很小的提高。他们还观察到，使用大页面(例如 2MB)会降低 TLB 失败率，但是，由于需要保持与 CPU 页表的兼容性，仅使用大页面可能无法解决性能问题。

8. 结束语

在本文中，我们介绍了设计和优化顶级域名的技术概况。我们介绍了 TLB 的术语和背景，并强调了 TLB 管理层的权衡。我们回顾了关于 TLB 的工作负载特性研究、减少 TLB 延迟、未命中率和能耗的技术以及减少 TLB 访问的虚拟缓存设计。我们强调了研究工作的关键属性，以强调其相似和不同的特点。我们现在简要总结一些未来的研究挑战。

从表中可以明显看出 I，大多数现有的 TLB 管理技术已被提议用于中央处理器。目前还不清楚这些技术是否有效，甚至是否适用于图形处理器和中央处理器-图形处理器系统[87].探索 GPU 和异构系统的新兴设计空间，同时最大限度地利用 TLB 管理的传统智慧，对于未来的研究至关重要。

考虑到评估张力腿平台完整设计空间所需的大量模拟时间，许多工作仅测量 TLB 脱靶量，而不测量性能[8,9,67,79]，而其他人则通过分析模型评估绩效[11,13,40,77].然而，这种方法可能不准确

模拟 TLB 管理技术的绩效影响。展望未来，开发高速、多功能的仿真平台对于推动未来 TLB 的研究至关重要。

最近的趋势，如增加系统核心数量和应用程序工作集大小，虚拟化等。使 TLB 的管理对未来的系统更加重要。在我们看来，要应对这一挑战，需要在所有抽象层次上采取协同方法。在电路级，非易失性存储器和创新的阵列组织可以降低面积/能量开销。在架构级别，诸如 TLB 重配置、超老化、预取等技术。可以结合在一起，把它们中最好的结合在一起。在编译器级别，以 TLB 友好的方式重组应用程序的内存访问模式可以增强 TLB 管理的架构技术的有效性。最后，应用程序固有的容错能力可用于积极的优化，例如推测转换和在接近阈值的电压下操作 TLB 以节省能量。

参考

[1] 李，梅尔赫姆，琼斯。PS-TLB:利用页面分类信息为未来的 CMP 提供快速、可扩展和高效的翻译。2013 年 ACM 架构和代码优化事务(TACO)；9(4):28.

[2] 在操作系统支持较少的情况下超越超级页面的 TLB 性能。ASPLOS 1994 年；。

[3] 提高嵌入式系统上 java 应用的 TLB 性能的硬件/软件方法的综合研究。存储系统性能和正确性研讨会，2006 年；102 – 111.

[4] 虚拟地址缓存。第 1 部分:单处理器中的问题和解决方案。
IEEE 微 1997；17(5):64 – 71.

[5] 卡拉科斯塔斯五世、甘地 J、希尔医学博士、麦金利 KS、奈米罗夫斯基 M、斯威夫特 MM、Unsal OS 等..节能地址转换。高性能计算机架构国际研讨会(HPCA)，2016 年；631 – 643.

[6] 基于预测的超级页面友好型 TLB 设计。2015 IEEE 第 21 届高性能计算机架构国际研讨会(HPCA)，IEEE，2015；210 – 222.

[7] 利用阴影记忆支持的超页增加 TLB 覆盖范围。ISCA，1998 年；204 – 213.

[8] 胡安 • T、朗 • T、纳瓦罗 • JJ。降低 TLB 电力需求。ISLPED，1997 年；196 – 201.

[9] TLB 的低成本高速地址转换机制。ISCA 1990 年；:355 – 363.

[10] 伍德 • 达、艾格斯 • SJ、吉布森 • G、希尔医学博士、彭德尔顿 • JM。一种缓存内地址转换机制。计算机体系结构新闻，第 14 卷，1986 年；358 – 365.

[11] 帕特森大学希尔医学博士孔 S。支持两种页面大小的权衡。ISCA，1992 年；415 – 424.

[12] 实用，透明的操作系统支持超级页面。2002 年操作系统评论；36(国际单位制):89 – 104。

[13] 陈 JB，博格 A，Jouppi NP。基于仿真的 TLB 性能研究。ISCA 1992 年；:114 – 123.

[14] 统一指令/翻译/数据一致性:一个协议来统治它们。高性能计算机架构国际研讨会(HPCA)，2010 年；1 – 12.

[15] 描述芯片多处理器上新兴并行工作负载的 TLB 行为。PACT，2009 年；29 – 40.

[16] 通过编译器引导的地址生成降低数据 TLB 能力。集成电路和系统的计算机辅助设计 2007 年 IEEE 交易；26(2):312 – 324.

[17] 巴苏阿，希尔医学博士，斯威夫特医学博士。利用机会虚拟缓存降低内存参考能量。ACM SIGARCH 计算机体系结构新闻，第 40 卷，2012 年；297 – 308.

[18] 通过页码预测和推测性预翻译减少 TLB 计算机辅助制造搜索。2013 年国际低功率电子与设计研讨会；341 – 346.

[19] Bhattacharjee A 等人..用于芯片多处理器的共享末级 TLB。HPCA，2011 年。

[20] 陈 l，王 y，崔 z，黄 y，鲍 y，陈 m .分散超页:一个弥合超页与页面着色差距的案例。2013 年国际计算机设计会议(ICCD)；177 – 184.

[21] Pham B，Vaidyanathan V，Jaleel A，Bhattacharjee A. CoLT:联合大范围张力腿平台。
2012 年国际微体系结构研讨会；258 – 269.

[22] 翻译存储缓冲器。 https://blogs.oracle.com/elowe/entry/translation storage buffers 2005. - -

[23] 大范围内存管理单元缓存。2013 年国际微体系结构研讨会；383 – 394.

[24] 异构系统共享虚拟内存架构的观察和机遇。系统和软件性能分析国际研讨会，2016 年；161 – 171.

[25] 图形处理器地址转换的架构支持:为具有统一地址空间的中央处理器/图形处理器设计内存管理单元。ASPLOS，2014 年；743 – 758.

[26] 动力 J，希尔医学博士，伍德 DA。支持 100 个 GPU 通道的 x86-64 地址转换。高性能计算机架构国际研讨会(HPCA)，2014 年；568 – 578.

[27] Karakostas V，Gandhi J，Ayar F，Cristal A，Hill MD，McKinley KS，Nemirovsky M，Swift MM，U nsal O .用于快速访问大内存的冗余内存映射。计算机体系结构国际研讨会(ISCA)，2015 年；66 – 78.

[28] 运用软硬件技术优化教学能源。2005 年美国计算机学会电子系统设计自动化会议；10(2):229 – 257.

[29] 高效虚拟高速缓存一致性的新视角。ISCA，2013 年；535 – 546.

[30] 米塔尔对处理器高速缓存的最新预取技术的调查。2016 年美国计算机学会计算调查；。

[31] 一个基于 STT-RAM 的可动态配置的 GPU 架构翻译后备缓冲器。2014 年亚洲和南太平洋设计自动化会议；355 – 360.

[32] 巴拉克、卡舒克、威赫尔 • 韦。翻译后备缓冲区的软件预取和缓存。1994 年 USENIX 协会第一届 USENIX 操作系统设计和实现会议录；18.

[33] 维拉维耶亚 C、卡拉科斯塔斯 V、维拉诺瓦 L、埃西翁 Y、拉米雷斯 A、门德尔松 A、纳瓦罗 N、克里斯塔尔 A、Unsal OS。滴滴出行:使用共享 tlb 目录减轻 tlb 击落对性能的影响。并行架构和编译技术国际会议，2011 年；340 – 349.

[34] 大页面和轻量级内存
虚拟化环境中的管理:您能做到两者兼顾吗？2015 年国际微体系结构研讨会；1 – 12.

[35] 软件管理 TLB 的设计权衡。计算机体系结构新闻，第 21 卷，1993 年；27 – 38.

[36] 调查高端科学应用在商品微处理器上的 TLB 行为。国际摄影测量与空间科学学会，2008 年；95 – 104.

[37] 范博，巴特查吉，埃克特，罗赫。通过利用页面翻译中的聚类来增加 TLB 覆盖范围。高性能计算机架构国际研讨会(HPCA)，2014 年；558 – 567.

[38] 罗默 TH，奥赫里奇 WH，卡林 AR，贝尔沙德 BN。使用在线超级页面推广减少 TLB 和内存开销。计算机体系结构新闻，第 23 卷，1995 年；176 – 187.

[39] 芯片多处理器的改进:核间协作预取器和共享末级 TLB。2013 年 ACM 架构和代码优化事务(TACO)；10(1):2.

[40] 李 • JH，李俊，郑世文，金世德。高性能低功耗的银行促销 TLB。2001 年国际计算机设计会议；118 – 123.

[41] 动态可调的内存层次结构。IEEE 计算机事务 2003；52(10):1243 – 1258.

[42] 方 z，赵 l，姜 x，卢思勒，伊尔尔，李特，李瑟。通过利用内存区域和特权级别语义来降低缓存和 TLB 功耗。系统架构杂志 2013；59(6):279 – 295.

[43] 嵌入式系统的低功耗 TLB 结构。IEEE 计算机体系结构快报 2002；:3.

[44] 巴拉普拉姆 CS，李 HHS。提高 JVM 上 Java 应用的 TLB 能量。SAMOS，2008 年；218 – 223.

[45] 通过动态调整大小来减少 dTLB 能量。ICCD，2003 年；358 – 363.

[46] 常。超低功耗 TLB 设计。日期:2006 年；1122 – 1127.

[47] 李 HHS，巴拉普拉姆政务司司长。使用语义感知多边分区的高能效数据 TLB 和数据缓存。2003 年国际低功率电子和设计研讨会；306 – 311.

[48] TLB 功率降低的代码转换。2010 年国际并行编程杂志；38(3-4):254 – 276.

[49] 嵌入式系统中指令 TLB 功率降低的基于边界的过程放置。嵌入式系统软件和编译器国际研讨会，2010 年；2.

[50] 尹赫，苏希 GS。重访虚拟 L1 缓存:使用动态同义词重映射的实用设计。2016 年高性能计算机体系结构国际研讨会；212 – 224.

[51] 高效的同义词过滤和混合虚拟缓存的可扩展延迟翻译。ISCA 2016；。

[52] 设计一个实用的数据过滤器缓存来提高能效和性能。2013 年 ACM 架构和代码优化事务(TACO)；10(4):54.

[53] 夏霞，陈 CW，刘。面向虚拟缓存的高能效同义词数据检测和一致性。微处理器和微系统 2016；40:27 – 44.

[54] 巴拉普拉姆 CS，李 HHS，Prvulovic M .数据 TLB 能源减少的同义地址压缩。ISLPED，2005 年；357 – 362.

[55] 同义词后备缓冲区:虚拟缓存中同义词问题的解决方案。IEEE 计算机事务 2008；57(12):1585 – 1599.

[56] 减少数据 TLB 能量的编译器指导的代码重构。硬件/软件代码设计和系统综合国际会议，2004 年；98 – 103.

[57] Haigh JR，Wilkerson MW，Miller JB，Beatty TS，Strazdus SJ，Clark LT .一款低功耗 2.5 GHz 90nm 一级缓存和内存管理单元。IEEE 固态电路杂志 2005；40(5):1190 – 1199.

[58] 巴拉普拉姆 C，普塔斯瓦米 K，罗赫 GH，李 HHS。基于熵的低功耗数据 TLB 设计。2006 年国际嵌入式系统编译器、体系结构和综合会议；304 – 311.

[59] 直接生成物理地址以节省指令 TLB 能量。微体系结构国际研讨会，2002 年；185 – 196.

[60] 王，贝尔，列维 HM。两级虚拟-真实缓存层次结构的组织和性能。ISCA 1989 年；:140 – 148.

[61] 描述 SPEC CPU2000 基准的 d-TLB 特性。SIGMETRICS，2002 年；129 – 139.

[62] 没有 TLB 的单处理器虚拟内存。IEEE 计算机事务 2001；50(5):482 – 499.

[63] 推测地址翻译的机制。
计算机体系结构国际研讨会(ISCA)，2011 年；307 – 317.

[64] 偏斜关联 TLB 上多种页面大小的并发支持。IEEE 计算机事务 2004；53(7):924 – 927.

[65] 通过影子内存和部分子块 TLB 提高超页面利用率。2000 年国际超级计算会议；187 – 195.

[66] 雅各布 BL，穆奇 TN。看一看几个内存管理单元、TLB 填充机制和页表组织。《操作系统评论》，第 32 卷，1998 年；295 – 306.

[67] 朴槿惠，安 • GS。一种用于软件管理的 TLB 的软件控制预取机制。*微处理和微程序设计 1995；41(2):121 – 136.*

[68] 奥斯汀 TM，搜狐 GS。多问题处理器的高带宽地址转换。ISCA，1996 年；158 – 167.

[69] 斯特莱克 WD。VAX-11/780:DEC PDP-11 系列的虚拟地址扩展。AFIPS，1978 年。

[70] 嵌入式多任务系统中减少干扰的上下文感知 TLB 预加载。2010 年大湖超大规模集成电路研讨会；401 – 404.

[71] 管理运行时语言的耐贫瘠页面 TLB。*2014 年国际计算机设计会议(ICCD)；270 – 277.*

[72] 芯片多处理器中基于 TLB 的私有页面高效检测。IEEE 并行和分布式系统事务 2016；27(3):748 – 761.

[73] 用于芯片多处理器的核间合作 TLB。ASPLOS 2010;:359 – 370.

[74] 芯片多处理器中高性能地址转换的协同 TLB。2010 年国际微体系结构研讨会；313 – 324.

[75] Chiueh Tc，Katz RH。消除物理地址缓存的地址转换瓶颈。*ACM SIGPLAN 通知，第 27 卷，1992 年；137 – 148.*

[76] 标记翻译后备缓冲器的性能:一个模拟驱动的分析。吉祥物，IEEE，2011；139 – 149.

[77] 基于新近的 TLB 预加载。ISCA 2000 年；。

[78] 《为 TLB 预取而努力:一项应用驱动的研究》。2002 年国际计算机体系结构研讨会；195 – 206.

[79] 《虚拟化平台的 TLB 标签管理框架》。2012 年国际并行编程杂志；40(3):353 – 380.

[80] 多处理器虚拟地址高速缓存的一致性。ASPLOS，1987 年；72 – 81.

[81] 双向偏斜关联缓存的一个例子。计算机体系结构新闻，第 21 卷，1993 年；169 – 178.

[82] 米塔尔，波伦巴，维特尔，谢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，2014 年。

[83] 高性能计算挑战基准。http://icl.cs.utk.edu/hpcc/ 2016.

[84] 透明的巨大支持。科索沃核查团论坛，第 9 卷，2010 年。

[85] 米塔尔 S，维特尔 JS。将非易失性存储器用于存储和主存系统的软件技术综述。IEEE 并行和分布式系统交易(TPDS)2016；27(5):1537 - 1550.

[86] 非易失性存储器系统在超大规模高性能计算中的机遇。科学与工程计算(CiSE)2015；17(2):73 - 82.

[87] 中央处理器-图形处理器异构计算技术综述。2015 年美国计算机学会计算调查；47(4):69:1 - 69:35.