第 34 卷 第 18 期 计 算 机 工 程 2008 年 9 月
Vol.34 No.18 Computer Engineering September 2008
·博士论文· 文章编号：1000—3428(2008)18—0007—03 文献标识码：A 中图分类号：TP333

基于修正 LRU 的压缩 Cache 替换策略

田新华，欧国东，张民选
(国防科技大学计算机学院，长沙 410073)

摘 要：以优化压缩 cache 的替换策略为目标，提出一种优化的基于修正 LRU 的压缩 cache 替换策略 MLRU-C。MLRU-C 策略能利用压缩 cache 中额外的 tag 资源，形成影子 tag 机制来探测并修正 LRU 替换策略的错误替换决策，从而优化压缩 cache 替换策略的性能。实验结果表明，与传统 LRU 替换策略相比，MLRU-C 平均能降低 L2 压缩 cache 失效率 12.3%。

关键词：影子 tag 机制；压缩 cache；替换策略

Replacement Policy for Compressed Cache Based on Modified LRU

TIAN Xin-hua, OU Guo-dong, ZHANG Min-xuan
(School of Computer, National University of Defense Technology, Changsha 410073)

【Abstract】 This paper proposes an improved compressed cache replacement policy——Modified LRU Replacement Policy for Compressed Cache(MLRU-C) based on modified Least recently Used(LRU) policy for improving compressed cache replacement policy. MLRU-C can use the extra tag in compressed cache to construct shadow tag mechanism, which can detect, predict, and correct bad replacement decisions made by LRU policy so as to improve performance of compressed cache replacement policy. Experimental results show MLRU-C can decrease L2 compressed cache miss rate by 12.3% compared with conventional LRU replacement policy.

【Key words】 shadow tag mechanism; compressed cache; replacement policy

1 概述

压缩cache是一种利用数据压缩增加cache有效容量的技术，它通过增加cache的有效容量来降低cache的失效率，从而间接提高cache的性能。压缩cache技术的引入带来了 2 个问题：增加cache命中延迟[1]和使cache的替换行为复杂化[2]。目前，有关压缩cache技术的研究[1-4]主要关注于 第 1 个问题，而对第 2 个问题还缺少研究。为此，本文针对第 2 个问题，研究压缩cache替换策略的优化技术。

压缩cache中的 cache行由于各有不同的压缩性能，且在cache中占用不同的空间，导致压缩cache比传统cache具有更复杂的替换行为。但与传统cache相比，压缩cache也提供了额外的tag资源[3-4]可用于优化压缩cache替换策略。本文研究了一种优化压缩cache替换策略的方法，利用压缩cache中额外的tag资源形成影子tag机制来探测并纠正LRU替换策略的错误替换，形成基于LRU修正的压缩cache替换策略(Modified LRU Replacement Policy for Compressed Cache, MLRU-C)。模拟试验表明，MLRU-C替换策略相比传统的LRU替换策略能明显降低L2 压缩cache的失效率，从而提高它的访存性能。

2 压缩 cache 结构

本文的压缩cache层次结构如图 1 所示，L1 指令和数据cache为传统非压缩cache，分别保存非压缩的指令和数据；L2 cache是压缩cache，保存非压缩的指令、压缩或非压缩的数据，采用简单常见模式压缩(Simple Frequent Pattern Compression, S-FPC)编码对数据进行压缩。该压缩编码是Alaa提出的常见模式压缩(Frequent Pattern Compression, FPC)编码的简化版本[1]，它将每个 32 bit字按表 1 所示分成 4 种模式，对每个字按模式编码成前缀和值。这样，每个 64 Byte的cache行可被编码压缩成 16 个 2 bit的前缀与 16 个值，形成压缩cache行，再根据它被压缩后的总长分配若干个 4 Byte的cache子块。访问L2 cache时，压缩数据需先通过解压缩流水线进行解压缩，而非压缩的指令和数据则直接通过非压缩行旁路传输以减少延迟。当L1 数据cache中的数据写回时，需首先通过压缩流水线进行压缩，然后再写回到L2 cache。

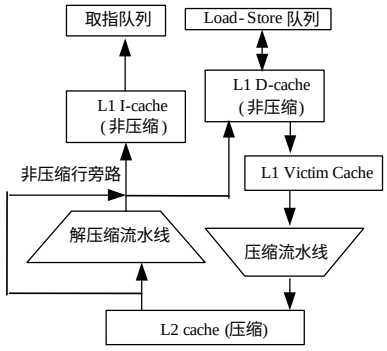

图 1 压缩 cache 层次结构

表 1 简单常见模式压缩编码

前缀	编码模式	数据长度/bit
00	0 值	0
01	8 bit 符号位扩展	8
10	16 bit 符号位扩展	16
11	不可压缩字	32

L2 压缩 cache 为 8 路组相联结构，如图 2 所示，其 tag

基金项目：国家自然科学基金资助项目(90207011)
作者简介：田新华(1971－)，男，博士研究生，主研方向：微电子与固体电子学，计算机系统结构；欧国东，博士研究生；张民选，教授、博士生导师
收稿日期：2007-10-20 **E-mail：** xhtian@nudt.edu.cn

阵列每组有 8 个 tag，但每组最多可保存并寻址 4 个 64 B 未压缩 cache 行或 8 个压缩 cache 行，与传统 cache 相比，每组多提供了 4 个 tag。每个 tag 除包括传统 cache 中的有效位、地址 tag 等状态位外，还包括 1 bit 压缩标志，表明该 cache 行是否被压缩。此外，每个 tag 还包括 cache 子块偏移和 size 字段，分别表示该 tag 对应压缩 cache 行的起始 cache 子块位置和占用的 cache 子块数(不包括前缀)。因为每组有 64 个 cache 子块，所以 cache 子块偏移字段有 6 bit；每个 cache 行最多占用 16 个 cache 子块，因此，size 字段有 4 bit。

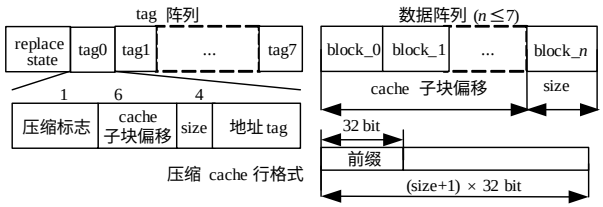

图 2　L2 压缩 cache 结构

3　MLRU-C 替换策略

3.1　LRU 替换策略的错误替换决策

目前 cache 大多采用 LRU 替换策略。该策略总是选择最久未被重用的 cache 行作为替换候选。由于程序引用流的模式事先未知，因此 LRU 替换策略选择的替换候选难免有盲目性，在遇到下述 2 种引用流模式时会出现错误的替换决策：

(1)若某 cache 行被引用而进入 cache 后，就不再被引用，或只在该组的 MRU 位置被引用，离开 MRU 位置后，就不再被引用直至它离开 cache，则称此 cache 行为死块(D 块)。死块导致 cache 污染加剧，替换策略要尽早将其替换，以修正此错误。

(2)若某 cache 行被替换出 cache 后又马上被引用，则称此 cache 行为活块(L 块)。替换策略应使活块比其他类型 cache 行更晚被替换，使之能被重用从而修正错误。

对于其他既非死块又非活块的 cache 行，不需要特殊对待，只需按照 LRU 替换策略原本的方式处理。对此类 cache 行不做任何标记，因此，又称为 U 块(Untag Block)。

3.2　影子 tag 机制

本文利用压缩 cache 中额外的 tag 资源形成影子 tag 机制来判定 LRU 替换策略可能出现的错误替换决策。影子 tag 机制与每个组关联，其结构如图 3 所示。

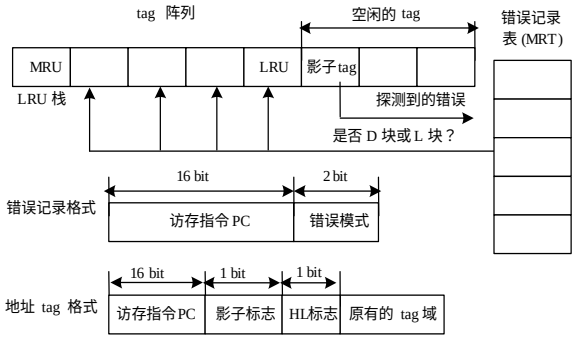

图 3　探测 LRU 替换错误的影子 tag 机制

由于压缩 cache 中每组都有 8 个 tag 与 4 个未压缩 cache 行或最多 8 个压缩 cache 行对应，因此当该组有一个或多个非压缩 cache 行时，该组将出现一个或多个空闲 tag，把其中一个作为影子 tag，记录前一次该组被替换 cache 行的地址以及对应访存指令地址 PC 的低 16 bit。

发生 cache 失效时，新进入的 cache 行优先使用该组当前的影子 tag；若没有影子 tag，则任选一空闲 tag 作为新 cache 行的 tag；若既无影子 tag，又无空闲 tag，说明该组已经保存了 8 个压缩 cache 行，此时应根据替换策略使用被替换 cache 行的 tag 作为新 cache 行的 tag。而被替换 cache 行在离开 cache 后，或者将自己的 tag 交给新 cache 行使用(这种情况出现在无空闲 tag 时)，或者将自己的 tag 继续保留在 tag 阵列中，成为一个新的影子 tag。当出现多个 cache 行同时被一个新 cache 行替换时，选择其中一个被替换 cache 行的 tag 作为影子 tag。这样替换策略将保证在任何时候，每组 tag 中最多只有一个影子 tag，当某组无空闲 tag 时，该组将没有影子 tag。

影子 tag 的内容与普通 tag 基本一致，区别仅在于影子 tag 对应的 cache 行并不在 cache 中，它实际是一个已离开 cache 的 cache 行在 tag 阵列中留下的影子；而普通 tag 在 cache 中则有对应的 cache 行。为区分一个 tag 是否为影子 tag，每个 tag 都有 1 bit 影子标志，用于表明当前时刻其是否为影子 tag。此外，tag 还有一个访存指令 PC 字段，如图 3 所示，用于保存访问此 cache 行的访存指令地址 PC 的低 16 bit。当发生 L2 cache 访问时，处理器把导致该次访问的访存指令地址 PC 的低 16 bit 传送给 L2 cache，并保存到所访问 cache 行 tag 的这个字段中，当探测到 LRU 替换错误时，此字段将被登记到错误记录表中。

3.3　基于 LRU 错误修正的压缩 cache 替换策略 MLRU-C

3.3.1　死块的判定

死块的判定方法如下：当新 cache 行进入 cache，首先被置于 MRU 位置。若该行离开 MRU 位置后一直不被程序访问，则逐步转移到 LRU 位置，直至被替换出 cache，成为一个影子 tag。接下来，若进入该组的新 cache 行未命中影子 tag，则说明影子 tag 原有的 cache 行是个死块，此影子 tag 中原有 cache 行的访存指令地址被登记到错误记录表，并标记为 D。

3.3.2　活块的判定

活块的判定方法如下：当某 cache 行被替换并成为影子 tag 后，有 cache 失效命中此影子 tag，且此 cache 行在成为影子 tag 后到被再次命中并重新调入 cache 之前的这段时间，该组存在其他未命中过的 cache 行，则影子 tag 中的 cache 行是一个活块，标记为 L；若该组其他 cache 行都命中过，则影子 tag 中的 cache 行是被正确替换的 cache 行，而不是活块。

为判断 cache 行成为影子 tag 后直到它因命中而重新进入 cache 之前的这段时间里，该组是否存在其他未曾命中的 cache 行，tag 中还设置了一个 HL(Hit Live Block Flag)位，如图 3 所示，表示该组在上次 cache 行替换操作后，此 tag 对应的 cache 行是否被命中过。每个 tag 的 HL 位在该组出现替换时复位，在此 tag 对应 cache 行命中时置位。当新 cache 行命中影子 tag 时，活块判定机制需检查 LRU 栈中是否存在一个 HL 位为 0 的 tag，若存在，则影子 tag 中的 cache 行是一活块，它将被记入错误记录表，并标志为 L，否则它是一个 U 块。

3.3.3　错误记录表

当某 cache 行或它的影子分别被新 cache 行替换或使用后，过去探测到的替换错误信息会丢失，因此需要一个存储结构来保存这些替换错误信息，这通过错误记录表(Mistake Record Table, MRT)来实现。如图 3 所示，MRT 的每个入口有 2 个字段，分别记录导致 LRU 发生错误替换的访存指令地址的低 16 bit 和错误的模式(2 bit)。MRT 为 Hash 表结构，入口数为 2 000，可根据访存指令 PC 进行索引，从而获知此访

存指令将导致的错误模式。

3.3.4 MLRU-C 替换策略

本文利用压缩 cache 的影子 tag 机制设计了基于 LRU 错误修正的压缩 cache 替换策略 MLRU-C。该策略能监测导致 LRU 替换策略产生错误替换的访存指令，并将其记录到 MRT 中，形成反馈环节来纠正随后可能出现的错误替换，减少 cache 失效率。该策略算法如下：

(1)新进入 cache 的 cache 行以及导致 cache 命中的 cache 行都将被置于 MRU 位置，对应访存指令地址的低 16 bit 将更新 tag 中访存指令地址 PC 域。

(2)当某组发生替换时，首先在该组中从 MRU 位置(不包括 MRU)向 LRU 位置依次查找是否存在死块。通过该组 cache 行对应 tag 中的访存指令地址 PC 对 MRT 进行索引，判断它是否为死块。若有多个死块，则先按照这些死块的尺寸大小选择替换候选，尺寸大的死块将优先被替换，从而可以释放更多空间；若按照尺寸大小无法选择替换候选，则按照死块在 LRU 栈中的位置，优先替换更靠近 MRU 位置的死块，以尽快替换出死块。

(3)若没有找到死块，则该组 cache 中的 cache 行是活块或 U 块。这时按照从 LRU 到 MRU 的顺序优先选择 U 块作为替换候选，因为 U 块将来被访问的概率低于活块。若该组 cache 行都是活块，则优先替换位于 LRU 位置的活块。

(4)当活块被替换，则对应此活块的 tag 或者成为影子 tag，或者被新进入的 cache 行使用(此时该组无空闲 tag 资源)。如果是第 2 种情形，则以此活块 tag 的指令 PC 为索引，将 MRT 中对应的入口删除。如果是第 1 种情形，则看该组接下来的引用流能否命中从 MRU 到 LRU 位置的 cache 行或影子 tag：如果命中，则 MRT 不发生变化；如果不能命中，则影子 tag 将被新进入 cache 的引用所使用，影子 tag 中原有 cache 行的访存指令地址 PC 在 MRT 对应入口的错误模式由原来的活块修改为死块。

(5)当死块被替换为影子 tag 后，有后续访问命中此影子 tag 且在此过程中该组所有有效 tag 的 HL 位至少有一个未置位，则影子 tag 中访存指令 PC 域对应 MRT 入口的错误模式由死块变为活块；若全部置位，则从 MRT 中删除此入口；其他情形，MRT 保持不变。

(6)当 U 块被替换，则依照前述死块、活块判定方法确定对 MRT 进行的操作。

4 模拟实验与结果

4.1 实验方法

为检验MLRU-C替换策略提升cache性能的效果，对cache行为仿真器DineroIV进行了修改和扩充，使它能够对本文描述的L2 压缩cache在LRU, OPT[5]以及MLRU-C替换策略下的cache行为进行仿真。DineroIV以存储引用序列和cache设置参数为输入，用于仿真和评估一个存储层次的性能。图 4 给出了整个仿真系统的软件结构和仿真流程，仿真流程具体描述如下：

(1)仿真过程生成的存储引用序列信息保存在 MySQL 数据库中，系统中各功能模块相互独立，只与 MySQL 数据库进行信息交互。在图 4 中，x 表示引用流中每个引用的序号；$Ref(x)$表示引用 x 所在 cache 行地址；$V(x)$表示引用 x 所在 cache 行压缩后占用 cache 子块数；$PC(x)$表示引用 x 对应访存指令地址的低 16 bit(只用于 MLRU-C 替换策略的仿真)；$ReuseID(x)$表示引用 x 访问的 cache 行被重用时的重用引用序号(只用于 OPT 替换策略的仿真)。

(2)生成程序的存储引用序列，并存入 MySQL 数据库。修改了 SimpleScalar 仿真工具集中的 Sim-Cache，使之能支持模拟上述压缩 cache 结构，并利用它模拟运行 Spec2000Int 中的 6 个测试程序(bzip2, gcc, mcf, parser, twolf, vpr)以及 SPEC2000FP 中的 4 个测试程序(art, equake, mesa, swim)。模拟运行首先跳过初始时的 10 亿条指令，然后将其后 300 万条 M 指令的 L2 cache 的访问引用流记录到 MySQL 数据库中。模拟的 cache 层次结构配置如下：L1 指令和数据 cache 是容量为 64 KB 的 LRU 4 路组相联传统 cache，cache 行大小为 64 B；L2 压缩 cache 的物理容量为 1 MB。

(3)利用 MySQL 数据库系统的数据管理功能计算引用序列的重用引用。

(4)利用存储引用序列信息数据库，在 DineroIV 上分别对 LRU、OPT 以及 MLRU-C 替换策略下压缩 cache 的行为进行模拟仿真，并统计 cache 失效率。

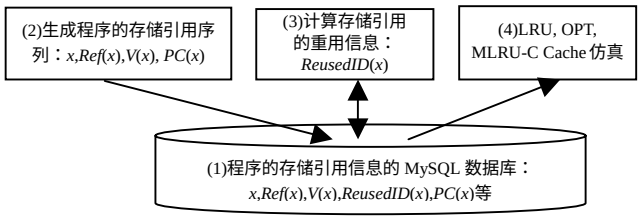

图 4 压缩 cache 替换策略仿真系统结构

4.2 实验结果

由于 OPT 替换策略代表了 cache 替换策略的性能上限，因此模拟 OPT 替换策略可以评估 MLRU-C 对压缩 cache 替换策略优化的程度。表 2 给出了这 10 个测试程序分别在 LRU, MLRU-C 以及 OPT 替换策略下的 L2 压缩 cache 的失效率，从中可以看到，MLRU-C 替换策略的失效率明显低于 LRU 替换策略，而 OPT 替换策略下的 L2 压缩 cache 的失效率又低于 MLRU-C 替换策略。

表 2 L2 压缩 cache 在替换策略下的失效率 (%)

	LRU	MLRU-C	OPT
bzip2	17.80	16.56	15.88
gcc	0.65	0.54	0.41
mcf	61.88	59.40	58.95
parser	17.59	14.78	12.76
twolf	7.22	4.93	2.82
vpr	5.84	4.18	2.91
art	30.20	25.16	22.37
equake	55.37	51.15	48.14
mesa	23.70	22.89	18.43
swim	59.90	46.23	38.15
avg	28.02	24.58	22.08

在上述 10 个测试程序中，twolf 的失效率减少得最为明显，它在 LRU, MLRU-C, OPT 三种替换策略下的失效率分别为 7.22%, 4.93%, 2.82%，该程序在 MLRU-C 替换策略下的 L2 压缩 cache 失效率比 LRU 替换策略降低了 31.7%，而此时 OPT 下的 L2 压缩 cache 失效率比 LRU 替换策略降低了 60.9%。所有 10 个测试程序在 LRU, MLRU-C 和 OPT 替换策略下的 L2 压缩 cache 失效率的平均值分别为 28.02%, 24.58% 和 22.08%，MLRU-C 对比 LRU 降低了 L2 压缩 cache 失效率 12.3%，而 OPT 替换策略比 LRU 降低了 L2 压缩 cache 失效率 21.2%。这说明 MLRU-C 替换策略能明显降低压缩 cache 的失效率，提高 cache 性能。 (下转第 16 页)