The Study and Design of a Skewed-Associative Object Cache on the Number Theory Function

CHEN Bin

Department of Mathematics and information science, Weinan Teachers university, Weinan Shaanxi 714000

ccbb3344@163.com

Abstract: On the number theory function, the object table indirection implies at least two memory accesses for each data access in object systems, and an addition must be performed to generate the address of the memory location. A new skewed-associative cache architecture is proposed in this paper to reduce memory access delay in object-oriented processor. The hash function construction method and replacement policy for 2-way skewed-associative object cache are also presented. Special issues of object cache, such as filling and data purging are discussed. Simulation results show that 2-way skewed-associative object cache exhibits higher hit ratio than 2-way set associative cache with the same cache size and hardware complexity, and thus 2-way skewed- associative object cache is preferred when constructing object-oriented processors.

Key words: object-oriented; programming; skewed-associative cache; object-oriented processor

基于数论函数的一种偏斜相联对象 Cache 研究与设计

陈　斌

渭南师范学院　数学与信息科学系，陕西 渭南 714000

ccbb3344@163.com

摘　要: 基于数论函数,提出一种偏斜相联(skewed-associative)对象 Cache，解决了面向对象处理器中对象间接寻址引发的多次存储访问和访存延迟问题；给出了 2 路偏斜对象 Cache 的基本结构和散列函数的构造方法；提出适合于偏斜相联对象 Cache 的伪最近最少使用 Cache 替换算法，并讨论了对象 Cache 中的数据预取与更新策略，保证了对象数据的完整性及一致性。模拟结果表明，与组相联对象 Cache 相比，偏斜对象 Cache 具有较高的 Cache 命中率

关键词: 面向对象编程;偏斜相联 Cache;面向对象处理器

1 引言

面向对象程序的性能问题也越来越成为人们关注的焦点。在对象系统中，每个对象都有唯一的标识或者对象引用，从对象标识到对象物理存储位置的映射称为对象寻址，当前面向对象处理器中对象寻址方式主要分为两种：直接寻址方式和间接寻址方式[1][2][3]。为了解决间接寻址方式引发的多次存储访问问题，实现对象数据的快速访问，本文提出一种偏斜相联(skewed-associative)对象 Cache，解决了对象间接寻址引发的多次存储访问和地址转换问题，提高了对象 Cache 命中率。与相同容量的 2 路组相联 Cache 相比，2 路偏斜相联 Cache 具有更高的 Cache 命中率，2 路偏斜相联 Cache 可以直接集成在面向对象处理器中，提高存储访问的速度，具有一定的应用价值。

基金项目：陕西省教育厅基金项目（09JK426）（09JK432）（09JK430），基础数学省重点学科资助

FUNDCATION : Shaanxi Provincial Education (09JK426) (09JK432）（09JK430）and Key subjects of Shaanxi Provincial Fundcation

2 偏斜相联对象Cache

为了解决间接寻址方式引发的多次存储访问问题，实现对象数据的快速访问，Williams 提出了对象 Cache 的概念[1]。

所谓对象 Cache，就是将对象引用或唯一标识

978-1-935068-31-0

(OID)和偏移量（Offset）拼接构成一个虚拟地址，并将这个虚拟地址与 Cache 的某一存储位置相关联。与虚拟存储系统中的 Cache 不同，对象 Cache 中标签域存放的是 OID 和 Offset 的拼接，每次存储访问时，对象 Cache 首先根据 OID 和 Offset 生成 Cache 索引并比较 OID 与 Offset，如果相同则 Cache 命中，直接返回数据；否则查询对象表从存储器中读入对象数据[1][3]。由于 Cache 的命中率比较高，因此对象 Cache 的引入能够极大提高对象存储访问的速度，同时保留了间接寻址所带来的各种优点。然而与传统的 Cache 相比，对象 Cache 的命中率还是比较低，这是因为面向对象程序在运行时频繁地创建和销毁对象，对象数量众多；同时对象大小不一，极不规则，因此只有提高 Cache 容量和相联程度才能获得满意的 Cache 命中率，但是这样同时也急剧提高了 Cache 的构造成本。

Seznec 等人提出了偏斜相联 Cache 的概念，如图 1（A）所示，2 路组相联 Cache 由 2 个不同的存储体构成，访存地址 A 经过散列函数映射到不同存储体的同一行 $f(A)$；偏斜相联 Cache 的结构与组相联 Cache 稍有不同，即每个存储体使用不同的散列函数，访存地址 A 在第一个存储体里映射到第 $f_0(A)$ 行，而在第二个存储体中则映射到第 $f_1(A)$ 行。由于每个存储体使用不同的散列函数，不同的访存地址 A_0, A_1, A_2 可能映射到第一个存储体的同一行，但是在另一个存储体中则会映射到不同的三行。研究表明 2 路偏斜 Cache 的命中率与 4 路组相联 Cache 的命中率相当，且具有较低的 Cache 构造成本[4][5]。

为提高面向对象处理器中对象 Cache 的命中率，本文提出一种 2 路偏斜相联对象 Cache，基本结构如图 2 所示。该 Cache 结构与传统 2 路组相联 Cache 最大的不同就在于使用了两个散列函数。访问对象 Cache 的虚拟地址由 OID 和 Offset 拼接而成的，Offset 的低位部分用来实现 Cache 块内寻址，虚拟地址剩余的部分将传递给散列函数计算访问 Cache 的索引。散列函数的选择对于 Cache 的性能有着极大的影响。本文采用基于 XOR 的散列函数，主要是由于基于 XOR 的散列函数易于硬件实现，具有较低的计算延迟，且对于大多数的应用程序都表现出比较好的性能[6]。 与 Williams 提出了对象 Cache 不同，本文给出的对象 Cache 标签域只需要存储 OID 和 Offset 的高位部分，这样节省了构造对象 Cache 所需的存储资源，降低了 Cache 成本。

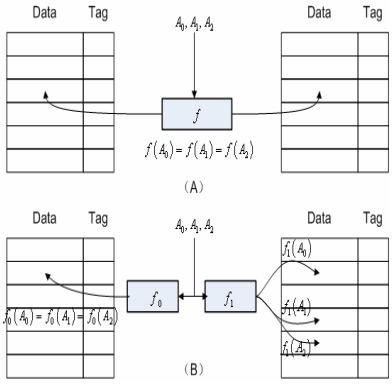

Figure 1 Set-associative Cache Cache associated with the skew

图 1 组相联 Cache 与偏斜相联 Cache

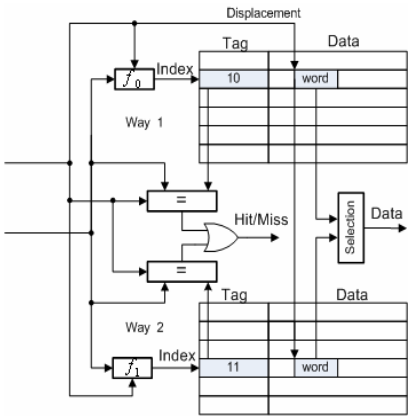

Figure 2. 2 Way Cache object associated deflection structure

图 2. 2 路偏斜相联对象 Cache 结构

3 偏斜Cache散列函数

如图 1 所示，对于 2 路组相联 Cache，当 3 个不同地址竞争同一组 Cache 时，总会有一个存储块被换出到主存中，但是在 2 路偏斜 Cache 中，由于每个存储体都对应一个散列函数，因此当存储体 i 出现冲突时，在另一个存储体 j 中出现 Cache 冲突的概率就非常小，因此只要选择合适的散列函数可以避免上述情况的出现。理想情况下，映射到第一个存储体某个位置的所有地址应均匀地散列在所有其他存储体的 Cache 行中，这样才能尽可能提高 Cache 命中率。

统计数据显示，对象平均大小非常小，Smalltalk 中对象的平均大小在 70 字节左右，Java 中对象平均大小为 30 字节左右，90%以上的对象

只需要 8-bit 对象偏移量就可以表示[1][7]，因此偏移量的高位部分对于大多数对象都没有使用，本研究中将 OID 的低 m-bit 与 Offset 除去用于块内寻址的 c-bit 部分后的低 m-bit 拼接作为散列函数的 2m-bit 输入向量。假定 Cache 每路有 2^m 个 Cache 行，每一行为 2^c 字节， 对象标识 OID 表示为：

$$OID = A_2^o 2^m + A_1^o \tag{1}$$

对象内偏移 Offset 可以表示为：

$$Offset = A_3^f 2^{c+m} + A_2^f 2^c + A_1^f \tag{2}$$

则 Cache 的索引地址 B_0 和 B_1 按如下方式计算：

$$B_0 = f_0(OID \mid Offset) = A_2^f \oplus A_1^o \tag{3}$$

$$B_1 = f_1(OID \mid Offset) = \sigma(A_2^f) \oplus A_1^o \tag{4}$$

其中|表示二进制串的拼接，$\oplus$ 表示位异或（XOR）操作，σ 表示m-bit的完美洗牌，即单向循环移位操作。公式（3）和（4）给出的散列函数能够满足上述偏斜相联Cache的要求，当OID相同Offset不同，或OID不同Offset相同时都能够给出不同索引值；且索引的每一位只需进行一次XOR操作就可以获得，散列函数计算延迟较低。此外与Williams提出了对象Cache不同，本文给出的对象Cache标签域只需要存储OID及offset的高位部分 A_3^f，这是因为由公式(3)和公式(4)可知，当 B_0 与OID给定时，A_2^f 可以由 B_0 与OID确定，因此只需要比较OID与 A_3^f 就可以判定Cache是否命中。假设OID为32-bit，offset为 16-bit，$m=12$，且Cache块为 16 字节大小，则Offset的低 4 位用于Cache块内寻址，高 12 位用于计算Cache索引，标签中仅需存储OID，这样节省了构造对象Cache所需的存储资源，降低了Cache成本。

4 Cache替换算法及一致性

Cache 冲突是不可能避免的，一旦发生必须选择 Cache 块换出到主存中。常用的 Cache 替换算法有随机法，FIFO，LRU 等。其中 LRU 算法既利用了历史上 Cache 中块地址流的调度情况，又正确反映了程序的局部性特点，因此在传统 Cache 中取得了较好的效果。但是 LRU 算法却不适合于偏斜相联 Cache，这是因为偏斜相联 Cache 的索引地址 B_0 和 B_1 随着输入地址的变化而不断变化，B_0 和 B_1 并没有固定的对应关系，[8]中基于统计的方法给出了适合于偏斜相联 Cache 的各种 Cache 替换算法，但是这些算法难以使用硬件实现。本文提出一种适合于偏斜相联 Cache 的伪 LRU 算法，即每个 Cache 行设置一个最近使用位（RU），以表示该 Cache 行最近是否使用过。RU 初始值为 0，当 Cache 行被访问时，其对应的 RU 位置 1；每隔一定的周期将所有的 RU 位全部清 0。当发生 Cache 冲突时，从 RU 位为 0 的 Cache 块中随机选择一个作为被替换的 Cache 块，若所有 Cache 块的 RU 位都为 1，则从 RU 位为 1 的 Cache 块中随机选择一个。

如前所述，大多数对象一般较小，一个 Cache 行可以装入整个对象。因此采用 write-back 的 Cache 更新算法能够保证对象数据的一致性。

然而与传统 Cache 不同，对象大小不一，极不规则，因此在预取或者写回对象数据时必须保证数据的完整性，确保其他对象数据不受影响。如图 3 所示，假设读入固定大小的块到 Cache 行中，由于对象 3 比较小，对象 4 的部分数据也被读入到了 Cache 行中，然而该 Cache 行的标签中存放的是对象 3 的 OID 与 Offset 的高位部分，因此若此时访问对象 4 的第一个字，仍然会发生 Cache 丢失。当该 Cache 行进行写回操作时，就会覆盖对象 4 的部分数据，有可能导致对对象 4 的修改丢失。

为了防止上述情况的出现，按照如下步骤完成 Cache 块装入和写回操作：

根据 OID 和 Offset 从对象表中查找对象的基地址 A_o，同时从对象表中取出对象的大小 S_o；

比较 Offset 和 S_o，若 $S_o < offset$，则发生越界错误；

假定 Cache 行的大小为 S_l，则装入块的基地址 $A_b = A_o + offset - (offset \bmod S_l)$；

计算实际需要装入的数据量 S_b，若 $A_o + S_o - A_b < S_l$，则 $S_b = A_o + S_o - A_b$，否则 $S_b = S_l$；

从地址 A_b 开始装入 S_b 字节数据。

Cache 块写回的过程与装入的过程相似，这里不再赘述。使用该方法一方面防止了对其他对象数据的修改，保证了数据的完整性和一致性；另一方面又可以减少 Cache 与主存之间的通信量。

978-1-935068-31-0 © 2010 SciRes.

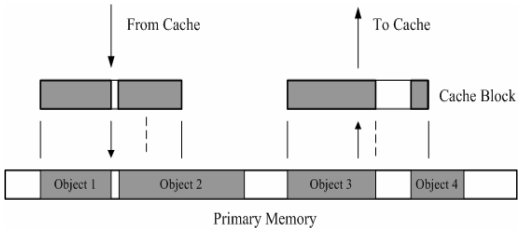

Figure 3. Division of the virtual address mode

图 3 虚拟地址划分方式

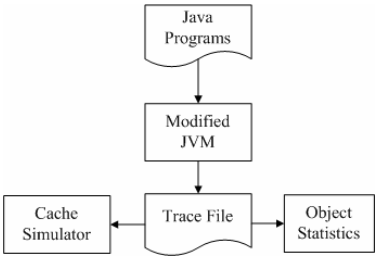

Figure 4. Object Cache Simulation

图 4 对象 Cache 模拟

5 偏斜对象Cache性能

为验证偏斜相连对象 Cache 的有效性，搭建了如图 4 所示的 Cache 软件模拟环境，首先在修改了的 JVM Hotspot2.0 上运行表 1 中的测试程序，JVM 被配置成在解释模式下运行，以捕获每个对象事件[9]。为获得运行各种应用程序时 Cache 的性能，测试程序包括游戏、编译工具、文字编辑器、浏览器及 Mp3 播放器等各种类型的应用实例。修改之后的 JVM 将对象创建和对象访问等事件记录到 Trace 文件中，所记录的各种事件是程序运行时对对象操作的完整再现。每个对象在 Trace 文件中都有一个唯一 ID，OID 按照对象的创建次序顺序分配，且与对象的物理存储位置无关。Cache 模拟器从 Trace 文件读入对象操作事件进行 Cache 性能测试。

影响 Cache 性能的因素比较多，包括 Cache 的容量，Cache 块大小，关联程度等。表 2 给出了 2 路偏斜相联对象 Cache 在不同配置下 Cache 命中率的测试结果。从表中可以看出对象 Cache 对 Cache 行的大小比较敏感，当 Cache 行只有 16 字节时命中率偏低，32 字节和 64 字节时命中率较高，相同容量不同大小 Cache 行之间的命中率相差比较大。XBrowser， Jlayer 和 Xerlin 当 Cache 行为 32 字节时 Cache 命中率最高，而其他程序则是 Cache 行为 64 字节时 Cache 命中率最高，究其原因在于 Xbrowser 和 Xerlin 对象平均大小只有 30 字节左右，而其他程序对象的平均大小界于 40 到 60 字节之间（该统计包括数组对象）。由于对象 Cache 的每一行只能通过 OID 和 offset 来索引，因此如果对象比较小，较大的 Cache 行就会造成更多的 Cache 存储空间浪费，导致 Cache 的命中率下降。Javacc 运行时同时存活的对象数目较少，只有 300 多，对象寿命相对较长，因此其 Cache 命中率最高。

Table 1. 2-way skewed Cache hit rate associated objects

表 1. 2 路偏斜相联对象 Cache 命中率

Benchmark	Cache Size (KB)	Cache Line Size(Bytes)			Benchmark	Cache Size（KB）	Cache Line Size（Bytes）		
		16	32	64			16	32	64
Address	16	89.37	95.97	97.76	Jtris	16	93.26	97.10	98.34
	32	89.86	96.34	98.31		32	93.63	97.52	98.87
	64	90.13	96.54	98.59		64	93.82	97.73	99.07
	128	90.27	96.67	98.73		128	93.89	97.93	99.16
Javacc	16	96.11	99.26	99.54	Xbro“”wser	16	89.51	95.02	92.48
	32	96.32	99.46	99.70		32	90.68	96.15	93.66
	64	96.39	99.51	99.89		64	90.68	96.15	94.83
	128	96.40	99.53	99.94		128	91.22	96.64	94.82
Jedit	16	90.78	96.38	96.86	Xerlin	16	87.34	94.65	91.51
	32	91.22	96.76	97.76		32	87.96	95.16	92.21
	64	91.54	97.01	98.28		64	87.95	95.15	92.83
	128	91.79	97.19	98.60		128	88.29	95.43	92.82

本文将 2 路偏斜对象 Cache(2SW)与直接映射(DM)、2 路组相联(2SA)、4 路组相联(4SA)对象 Cache 进行了比较，如图 5 到图 8 所示，分别给出了当 Cache 大小从 16KB 到 128KB 时的测试结果，其中 Cache 行设定为 32 字节，组相联 Cache 采用最近最少使用替换算法。从图中可以看出，与 2 路组相联对象 Cache 相比，2 路偏斜相联对象 Cache 的命中率较高，与 4 路组相联对象 Cache 的命中接近。但是由于 4 路组相联 Cache 的控制更加复杂，成本更高，因此 2 路偏斜相联对象 Cache 是面向对象处理器更合适的选择。

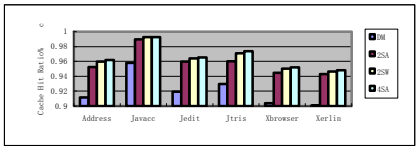

Figure 5 16KB set-associative Cache Cache associated with the skew comparison

图 5 16KB 组相联 Cache 与偏斜相联 Cache 比较

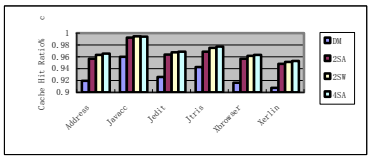

Figure 6　32KB set-associative Cache Cache associated with the skew comparison

图 6 32KB 组相联 Cache 与偏斜相联 Cache 比较

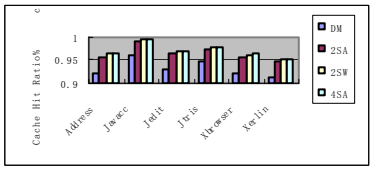

Figure 7　64KB set-associative Cache Cache associated with the skew comparison

图 7 64KB 组相联 Cache 与偏斜相联 Cache 比较

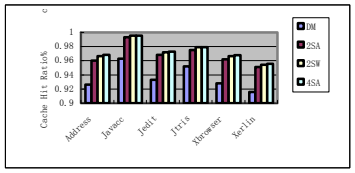

Figure 8 128KB set-associative Cache Cache associated with the skew comparison

图 8 128KB 组相联 Cache 与偏斜相联 Cache 比较

6 结束语

本文给出了 2 路偏斜相联对象 Cache 的基本结构，讨论了对象 Cache 散列函数，Cache 替换算法，Cache 一致性等问题。实验模拟结果表明， 2 路偏斜相联对象 Cache 在面向对象处理器中具有一定的应用价值和实用价值。

References

[1] Ifor W. Williams, Object-based memory architecture, [PHD thesis], Manchester: university of Manchester, 1989.

[2] N. Vijaykrishnan, N. Ranganathan, R. Gadekarla, Object-Oriented Architectural Support for a Java Processor, ECOOP'98, 1998,330-355,.

[3] Greg Wright, Matthew L. Seidl, Mario Wolczko, An object-aware memory architecture, Sun Technical report, 2005

[4] Seznec. A case for two-way skewed associative caches, in Proceedings of the 20th Annual Interna-. tional Symposium on Computer Architecture, 1993,169–178,

[5] Seznec. A new case for skewed-associativity. IRISA Technical. Report #1114, 1997.

[6] Hans Vandierendonck, Koen De Bosschere, XOR-Based Hash Functions, IEEE Transactions on computers, 2005, 54(7):800-812

[7] S. Dieckmann and U. Hölzle. A study of the allocation behavior of the SPECjvm98 Java benchmarks. In Proceedings of the European Conference on Object-Oriented Programming, June 1999.

[8] Pierre Michaud,IRISALNRIA. A statistical model of skewed-associativity. in Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, 2003, 204-213.

[9] L indho lm T, Yellin F., The Java Virtual Machine Specification, MA: Addison-Wesley, 1996

978-1-935068-31-0 © 2010 SciRes.