第 35 卷 第 4 期 计 算 机 工 程 2009 年 2 月
Vol.35 No.4 Computer Engineering February 2009
·软件技术与数据库· 文章编号：1000—3428(2009)04—0058—03 文献标识码：A 中图分类号：TP311

片上多处理器中基于步长和指针的预取

肖俊华[1,2]，冯子军[1,2]，章隆兵[1]
(1. 中国科学院计算技术研究所，北京 100080；2. 中国科学院研究生院，北京 100039)

摘 要：在对大量程序访存行为进行分析的基础上，提出基于步长和指针的预取方法。能捕获规整的数据访问模式和指针访问模式。在 L2 cache 和内存之间采用全局历史缓存实现该预取方法。全系统模拟结果表明，该预取方法对商业应用测试程序的性能平均提高 14%，对科学计算测试程序的性能平均提高 34.5%。

关键词：片上多处理器；步长预取；指针预取

Stride and Pointer Based Prefetching in Chip Multiprocessor

XIAO Jun-hua[1,2], FENG Zi-jun[1,2], ZHANG Long-bing[1]
(1. Institue of Computing Technology, Chinese Academy of Sciences, Beijing 100080;
2. Graduate University of Chinese Academy of Sciences, Beijing 100039)

【Abstract】 Based on the analysis of benchmark program's memory access behavior, this paper proposes stride and pointer based prefetching scheme which can capture regular data access pattern and pointer access pattern, and implements a prefetcher between L2 cache and memory using a global history buffer. Full system simulation shows that this prefetching scheme can improve the commercial benchmark's performance by 14%, and improve the scientific benchmark's performance by 34.5%.

【Key words】 chip multiprocessor; stride prefetching; pointer prefetching

1 概述

随着大规模集成电路和深亚微米技术的发展，处理器和内存间性能的差距越来越大，访存时延成为提高处理器性能的主要障碍。预取技术能有效容忍长访存时延，预取就是在数据使用之前将它取到 cache 中，用来减少处理器的停滞时间。目前，在单处理器设计中已有很多针对预取的研究，简单的基于硬件的预取技术[1]识别和捕获单位步长的数据访问模式，较复杂的基于硬件的预取技术[2]捕获常量步长序列，文献[3]提出利用访问预测表来监视数据访问模式并动态地发送预取，基于相关的预取技术[4]保存之前的 cache 失效地址并通过后续失效与历史的关联来触发预取，文献[5]提出基于依赖的预取用于捕获指针访问模式。

在片上多处理器中，多个处理器核竞争共享的 cache 以及有限的片外带宽使得预取成为一种新的挑战。大量的商用多核处理器，如 IBM 的 Power4/Power5、SUN 的 Niagara、Intel 的 Core 等，大都采用简单的单位步长的预取器来捕获规整的数据访问模式，也有少量采用稍微复杂一点的预取器，如 Fujitsu 的 SPARC64 支持非单位步长的预取。

对大量的程序访存行为分析之后，发现可将 cache 中失效的访问模式分为 4 类：下一行访问，非单位步长访问，指针访问，非规整访问。下一行访问和非单位步长访问属于空间访问的范畴，可利用步长预取来捕获。指针访问出现在链接数据结构(LDS)中，LDS 包括链表、树和图。指针访问难以被 cache 完全缓存，并且难以用传统预测技术来捕获，一般使用基于依赖的预取来捕获。少数的非规整访问可以通过基于相关的预取来捕获。

本文提出基于步长和指针的预取方法，结合步长预取和基于依赖的预取，能同时捕获规整空间访存模式和 LDS 中的指针访存模式，在实现时同时在预取请求队列和失效请求队列之间采用优先级的仲裁方式，用来降低预取操作对于片外访存带宽的负面影响。

2 基于步长和指针的预取

现代商用处理器通常采用基于单位步长的流缓存预取，这种预取不能捕获指针的访存模式。目前，LDS 被大量程序应用，并且随着 C++和 Java 等使用链接对象图和函数表语言的流行，LDS 变得更加重要。但是，由于 LDS 动态生成的特性，使它难以被 cache 完全缓存，有必要捕获指针访问，因此本文提出一种能同时捕获步长访问模式和指针访问模式的预取方法。

2.1 步长预取

基于步长的预取针对的是一系列地址有等距离步长的数据访问，在本文中称这种访问序列为步长序列。整个预取包括 2 方面：(1)动态识别步长序列中的步长；(2)尽早发送预取请求，使在处理器实际访问它时已将其预取至 cache 中。步长序列常出现在循环中，假定一定步长的访问来自相同的 load 指令，跟踪这条 load 指令序列的访问地址，步长即可被检测出来。最简单的步长预取方法是使用访问预测表(RPT)，RPT 中的每一项都包含失效指令 PC 域 *I*、load 地址域 *D*、步长域 *S* 和状态域 *status*。当一条 load 指令在 cache 中失效时，相应的指令地址 *I* 和数据地址 *D*1，被插入到 RPT 中，同时状态设为 no_prefetch。随后，如果相同 PC 的 load 指令再一

作者简介：肖俊华(1975－)，男，博士研究生，主研方向：高性能微处理器设计，计算机体系结构；冯子军，博士研究生；章隆兵，副研究员、博士

收稿日期：2008-07-01 **E-mail：**xiao_jh@ict.ac.cn

次发生失效，数据地址为 *D*2，则其在 RPT 中命中，依据 *S*=*D*2-*D*1 计算步长，同时 *D*2 和 *S* 被插入到 RPT 中，状态变为 prefetch。这时，将 *D*2+*S*, *D*2+2×*S*,⋯,*D*2+*d*×*S* 一系列地址预取过来，其中，*d* 是预取的深度。采用 2 种状态会带来没有必要的预取，为提高精度可采用 3 状态，即同一条 load 指令发生 3 次失效并且数据地址形成步长序列时将相应的地址序列预取过来。

2.2 指针预取

基于依赖的预取动态识别 LDS 所访问的 load 指令，收集 load 指令之间的依赖关系，预测程序将会继续沿着相同的轨迹遍历，预取引擎根据上文构建的依赖关系描述，沿着预测路径进行预取。具体结构如下：处理器维护最近的 load 指令队列，称为潜在生产者窗口(PPW)，PPW 中的每一项包括 ADDRVAL 域(load 的值)和 PR 域(生产者)。依赖关系在 load 指令提交时创建，PPW 中生产者的 load 指令和消费这些地址的 load 指令间的关联关系存放在关联表(CT)中，CT 中的每一项包括 PR 域、CN 域和 TMPL 域，表示在一条产生某个地址的 load 指令(PR 域)和后续使用这个地址的 load 指令(CN 域)之间的依赖关系，TMPL 域包括 opcode 和 offset，用于计算预取地址。每条 load 指令完成后检查 CT 表，查找这条指令是否是 CT 中潜在的消费者，如果匹配，使用 load 的值和预取产生规则(load 的地址加上在 CT 中偏移值 *offset*)形成一个预取地址，并且发送到预取请求队列。另外，为了增加预取的深度使得完成的预取能触发其他的预取，在预取 Buffer 中维护请求消费者队列，当取回一个预取块后，将 list 中的每个消费者假设成为新的生产者重新探测 CT 表，产生后续预取请求。

2.3 基于步长和指针的预取

为实现同时捕获步长序列和指针序列的预取机制，结合步长预取和指针预取如图 1 所示。包括步长预取器和指针预取器，这 2 个预取器分别有各自的访问模式表，用在 L2 cache 中失效 load 指令的 PC 值同时访问步长预取器和指针预取器，如果该值命中其中一个访问模式表，根据相应的算法得到预取的地址流，如果同时命中，根据指针预取算法得到预取的地址流。

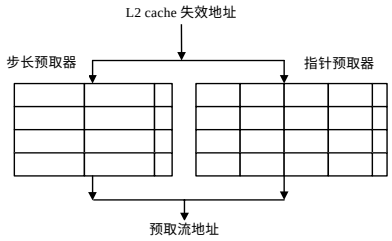

图 1 步长预取和指针预取的原型结构

由于上述结构含有 2 个访问模式表，可能出现冗余的访问模式信息以及存在大量老化的历史数据，为解决这一问题，本文借鉴分支预测器中的全局历史表的思想，将 2 个访问模式表合并。采用全局历史缓存[6]实现基于步长和指针的预取，如图 2 所示。包括索引表(Index Table)和全局历史缓存(Global History Buffer, GHB)。使用在 L2 cache 中失效 load 指令的 PC 值访问索引表，索引表中每一项包含 *tag* 域和 *link* 域，*tag* 域用于 PC 值的匹配，*link* 域是指向 GHB 的指针。GHB 是一个 *n* 项的先进先出表，保存最近的 *n* 个 L2 cache 失效地址，使用环形的缓存实现。GHB 中有 *miss_address* 域、*next* 域以及 *identify* 域。*miss_address* 域保存全局的失效地址，*next* 域保存链接指针，*identify* 域中的值识别是步长预取还是指针预取。链接指针将 GHB 中的项串成地址链表，每个地址链表是 1 个步长的地址列表或 1 个指针生产者与消费者的关联列表。

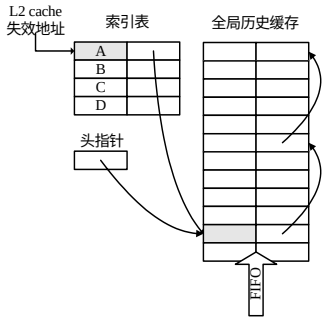

图 2 用全局历史缓存实现预取的过程

当 load 指令在 L2 cache 中失效时，将失效地址以先进先出的形式存入到 GHB 中，然后更新索引表中该 load 指令的 *link* 域，GHB 将该失效地址创建至对应 PC 值的地址列表中，形成新的地址列表，计算地址列表中相邻项的失效地址间的差值得到步长，如果步长值相等，表示该列表是固定步长的地址列表，随后，在 L2 cache 中失效的 load 指令命中步长序列，则发送步长预取请求；load 指令在提交的时候将该指令的消费者和索引表中所有 load 指令的生产者进行比较，如果匹配，则在 GHB 中形成新的生产者和消费者相关联的地址列表，随后，在 L2 cache 中失效的 load 指令命中关联列表，则发送指针预取请求。

预取方法是否实用取决于 2 点要求：(1)避免预取请求对于正常失效请求的影响；(2)避免预取的数据对于 L2 cache 的污染。针对第(1)个要求本文采用了基于优先级的命令请求仲裁方法，发送的预取地址首先在 L2 cache 不忙的时候访问 L2 cache，如果该地址不在 L2 cache 中，预取请求发送到预取请求队列，在失效请求队列没有准备好时发送预取请求，也就是说在预取请求队列和失效请求队列之间有优先级的关系，预取请求的优先级总是低于正常失效请求的优先级，用于减少预取请求对于正常失效请求的影响。针对第(2)个要求本文的解决途径是将预取过来的数据存放到 L2 cache 中 LRU 的最低级别中，预取回来的数据，直接送入到 L2 cache 中，并设定预取的 cache 行为 LRU 算法中的最低级别，这样对于 L2 cache 的污染降到最低，减少对 L2 cache 的污染。

3 实验方法

本文采用 Simics 全系统模拟器，并使用 GEMS 工具集进行扩充。实验中 CMP 结构包含 8 个单线程的处理器核、共享的 L2 cache、用于处理器核和 L2 cache 互连的交叉开关。每个处理器核的结构和 MIPS R10000 类似，包括：4 发射，动态调度，频率为 2 GHz，7 个功能部件(3 个定点、2 个浮点和 2 个 LD/ST)，gshare 分支预测器，误预测的开销为 8 拍，重排序队列的大小为 128 项，私有的 L1 指令 cache 和 L1 数据 cache，各为 32 KB，2 路组相联，访问时延为 1 拍。L2 cache 在处理器核间共享，4 MB，4 路组相联，访问时延为 10 拍。每个内存通道提供 3.2 GB/s 的带宽，访问内存的时延是 200 拍。

本文选取的测试程序包括商业应用测试程序(apache, SPECjbb, oltp, zeus)、科学计算测试程序(来自 SPLASH-2 的 barnes, ocean 以及来自 SPEComp 的 apsi, fma3d)。

4 实验分析

图 3 给出了基于步长和指针的预取对于各种测试程序的性能加速比。

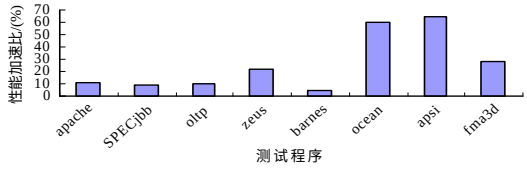

图 3 预取对性能的加速

可见，采用这种预取方法，性能得到了显著提升，特别是对于规整的科学计算测试程序，其性能提高从 10%~55%不等，平均提高了 34.5%，商业应用测试程序的性能提高从 6%~15%不等，平均提高了 14%。与商业应用测试程序相比，预取对于科学计算测试程序的性能提高更加显著的原因在于科学计算存在大量的循环和数组等规整的行为，而商业应用测试程序的访存行为不规整，传统的预取方法很难到达很好的效果。

图 4 给出了基于步长和指针的预取对各种测试程序的预取覆盖率，商业应用的覆盖率平均为 40%，科学计算的覆盖率平均为 57%。图 5 给出了基于步长和指针的预取对各种测试程序的预取精度。商业应用的精度平均为 48%，科学计算的精度平均为 66.8%，预取的精度较高，可见在失效请求队列和预取请求队列之间采用优先级的策略之后，预取对于访存的带宽影响很小，表明这一策略是有效的。

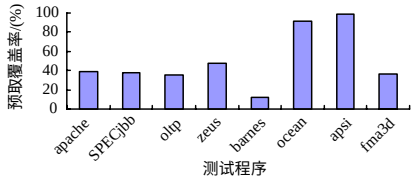

图 4 预取的覆盖率

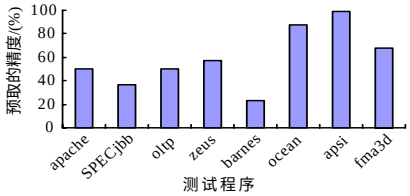

图 5 预取的精度

5 结束语

本文提出基于步长和指针的预取方法，将该方法用于片上多处理器中从内存到 L2 cache 的预取。全系统的模拟结果表明该预取方法能有效提高商业应用测试程序和科学计算测试程序的性能，特别是对于访存行为较为规整的科学计算测试程序的性能有较大提高，对于访存行为较为不规整的商业应用测试程序的性能也有一定提高。由于在失效请求队列和预取请求队列中采用优先级机制，因此预取对于访存带宽的影响并不大。

参考文献

[1] Palacharla S, Kessler R E. Evaluating Stream Buffers as a Secondary Cache Replacement[C]//Proc. of ISCA'94. Chicago, USA: [s. n.], 1994.

[2] Fu J W C, Patel J H, Janssens B L. Stride Directed Prefetching in Scalar Processors[C]//Proc. of the 25th Interanational Symposium on Microarchitecture. Portland, Oregon, USA: [s. n.], 1992.

[3] Chen Tienfu, Baer J L. Effective Hardware-based Data Prefetching for High-performance Processors[J]. IEEE Trans. on Computers, 1995, 44(5): 609-623.

[4] Joseph D, Grunwald D. Prefetching Using Markov Predictors[J]. IEEE Transactions on Computers, 1999, 48(2): 121-133.

[5] Roth A, Moshovos A, Sohi G S. Dependence Based Prefetching for Linked Data Structures[C]//Proceedings of ASPLOS'98. San Jose, California, USA: [s. n.], 1998.

[6] Nesbit K, Smith J. Data Cache Prefetching Using a Global History Buffer[C]//Proc. of HPCA'04. Madrid, Spain: [s. n.], 2004.

(上接第 57 页)

表 6 第 2 组 UUT 测试序列

次序	1	2	3	4	5	6
M1	T10	T11	T5	T7	T9	T8
M2	T7	T10	T6	T1	T8	T3
M3	T9	T2	T4	T11	T7	T1
次序	7	8	9	10	11	时间/s
M1	T1	T3	T4	T2	T6	791
M2	T2	T9	T11	T4	T5	791
M3	T5	T6	T8	T10	T3	791

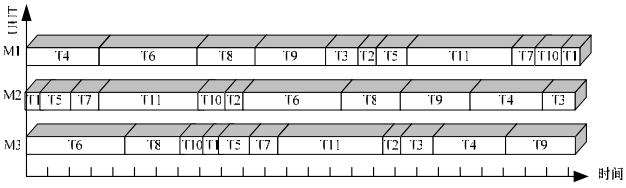

图 2 并行度为 100%的测试序列

对比 2 组 UUT 资源增加前后测试序列可并行度变化，可知随着资源增加，测试序列可并行度提高。

观察表 5、表 6 还可以发现，当 A, B, C, D 资源分别为 2, 3, 2, 3 时，2 组 UUT 均可以实现 3 件并行测试时间等于一件顺序测试时间，真实、直观地体现了并行测试系统节省测试资源、提高测试效率的优点。

6 结束语

本文主要探讨了并行测试中相同资源约束下不同测试序列提高效率存在差异的内在原因；针对前述现象提出了可并行度的概念，用于刻画不同测试序列的固有属性。

通过实例说明了并行测试序列可并行度是测试序列的固有属性：在相同资源约束下，第 2 组序列比第 1 组可并行度高；随着测试资源增加，测试序列可并行度可以达到 100%，反映了并行测试资源利用率高的鲜明特点。

参考文献

[1] 马 敏, 陈光禹, 陈东义. 基于 Petri 网和模拟退火遗传算法的并行测试研究[J]. 仪器仪表学报, 2007, 28(2): 331-336.

[2] 谢 政. 网络算法与复杂性理论[M]. 长沙: 国防科技大学出版社, 2003: 256-284.

[3] 张铁柱. 遗传算法在系统可靠性优化中的应用研究[D]. 哈尔滨: 哈尔滨理工大学, 1999.

[4] 李立军, 郭恒亮, 苏建忠. 遗传算法用于雨水管网的优化设计[J]. 黑龙江水利科技, 2003, 33(5): 1-3.

[5] 李 巍, 杨锁昌, 侯 雷, 等. 并行自动测试系统的任务调度[J]. 微计算机信息, 2006, 22(8): 79-81.