doi:10.3772/j.issn.1002-0470.2019.01.004

链表结构反馈预取机制①

张乾龙②*** 侯 锐*** 杨思博**** 张立新*

(*中国科学院计算技术研究所 北京 100190)

(**中国科学院大学 北京 100049)

(***中国科学院信息工程研究所 北京 100093)

(****北京大学软件与微电子学院 北京 100871)

摘 要 详细分析了已有针对链表结构(LDS)的预取方法,并分析了预取深度对预取性能的影响,同时分析了链表结构中单个生产者访存指令对应多个消费者访存指令的情况,并指出了现有链表结构预取器的不足。提出了针对链表结构的反馈预取机制,在原来预取器的基础上把预取命令查询处理器 Cache 的结果反馈给预取引擎,预取引擎根据反馈结果决定进一步预取操作。如果预取命令查询 Cache 发现已经命中,则反馈查询结果给预取器,预取器再针对同一个生产者指令产生预取命令。反馈预取机制可以和其他链式结构预取机制协同工作。实验结果表明,相比于无反馈的预取机制,针对链表结构的反馈预取机制当预取深度为 1 时,每周期执行指令数(IPC)平均提升 8.14%,L1-D Cache 缺失率平均降低 11.18%,新增硬件开销几乎可以忽略。

关键词 链表结构(LDS),指针追逐,数据预取,反馈预取,硬件预取

0 引 言

19 世纪 80 年代以来,摩尔定律驱动处理器性能以每年 60% 的速度提升,但是内存访问速度却没有跟上处理器性能提升的步伐,这两者之间的性能剪刀差导致了内存墙的出现,并且随着处理器性能的提升,内存墙问题越发严重。随着摩尔定律趋近终结,访存性能的提升越来越成为影响处理器性能提升的关键因素。

链表结构(linked data structure,LDS)普遍存在于大量程序和应用中,该结构的特点是:访存模式是指针追逐模式,消费者访存指令的访存基址寄存器,即生产者访存指令的目的寄存器,因此消费者访存指令必须等待生产者访存指令提交并取得其执行结果后才能发射执行。在提高链表结构访存性能方面,学术界提出了一些预取机制来降低访存的延迟[1-12]。此类预取方法通过记录链表结构访存指令的历史信息,当一条访存指令提交的同时发出预取命令,如此可以把预取操作的执行和普通指令的执行并行进行。Roth 等[3]提出的预取方法就属于此类方法,并取得了良好效果。有些情况下,因为普通指令早于预取命令执行结束,导致处理器流水线需要等待预取命令的执行结果而停顿。Roth 和 Sohi[4]基于此继续优化了预取方法,通过软件或者硬件方式更早地发出预取命令,保证普通指令执行结束前完成预取命令的执行,因此相比之前的预取方法达到了更好的效果。但有些情况下预取操作会因为数据已经在 Cache 中而早于普通指令结束执行,此时处理器流水线会因为等待普通访存指令的完成

① 国家自然科学基金优秀青年科学基金(61522212)和中科院前沿科学重点研究(QYZDB-SSW-JSC010)资助项目。

② 男,1988 年生,博士生;研究方向:计算机系统结构;联系人,E-mail:zhangqianlong@ict.ac.cn

(收稿日期:2018-04-27)

而造成停顿,这种情况是目前的预取机制[1-12]没有解决的。

为解决该问题,本文提出了硬件自动实现的链表结构访存反馈预取机制,通过将预取命令查询 Cache 的结果反馈给预取引擎,预取引擎据此做出进一步判断,如果查询结果是预取操作命中 Cache,则继续针对当前生产者访存指令进行预取;否则对当前生产者访存指令的预取操作已完成,继续针对后续访存指令进行预取。通过该反馈机制,可以进一步提升针对链表结构的预取器性能,进而提升系统访存性能。

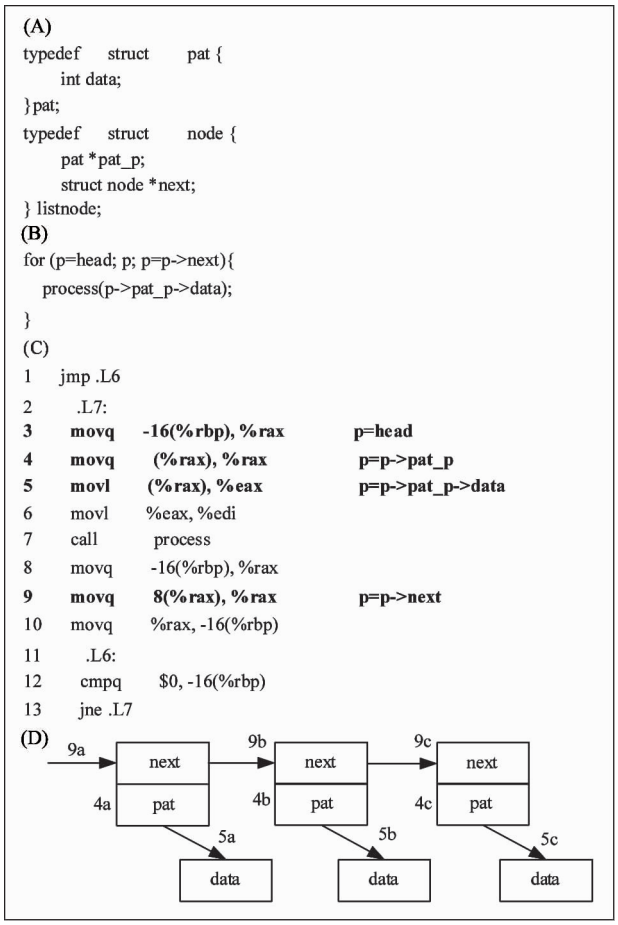
1 背 景

链表结构在程序中非常普遍,其生成的访存指令叫做链式访存指令,由于其访存地址不规律,访存指令之间互相依赖,因此也称此类指令为指针追逐访存指令。链式访存指令的程序局部性较差,Cache 缺失率较高,从而导致程序整体性能较差。本节首先介绍链表结构的基本概念,然后介绍相关现有预取算法。

1.1 链表结构

图 1 展示了常见的链表结构的例子(基于 C 语言),本文以 X86 体系结构为例。图 1(A)是链表节点数据结构,常见的链表节点中除 next 指针外,还会有其他数据成员,此处以 pat 类型指针为例。图 1(B)是示例程序中的部分源码,其作用是遍历链表并把节点中数据 pat _ p- > data 作为参数传给函数 process 进行处理。图 1(C)是源码(B)编译后生成的 X86 汇编代码,程序执行到指令 1 后跳转到 L6 标签处继续执行,指令 12、13 从栈中取出指针 p 的值并与零值对比,如果不为零则跳转到 L7 处执行,如果是零则结束执行。指令 3、4、5 取得指针 p,并从指令 p 处继续取得指针 pat _ p,进而取得数据 data。指令 6、7 跳转到函数 process 处运行。指令 8、9、10 获取 p- > next 对指针 p 进行遍历。

从图 1(C)可以看出,指令 4、9 的基址寄存器都是 rax,并且其值相同。从图 1(D)中可以看出,指令 9a 是生产者访存指令,指令 4a、9b 是消费者访存指



(A)源码中链表结构体定义,(B)源码中循环体,
(C)源码对应的 X86 汇编代码,(D)链表结构在内存中的布局

图 1 链表结构的定义

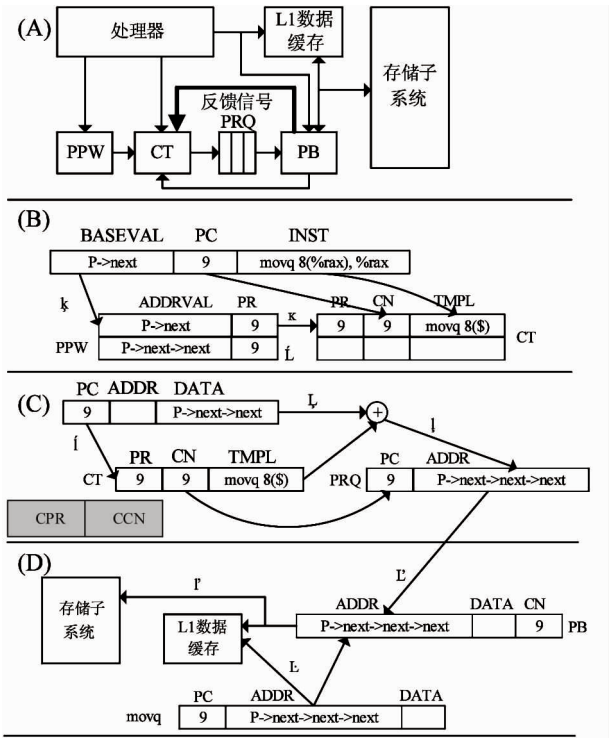
令。程序中的链表结构大量存在类似指令组合,并且往往一个生产者指令对应多个消费者指令。现有针对链表结构的预取方法[3,4],在执行指令 9a 的同时发出针对其消费者指令的预取命令,预取从程序计数器(programming counter, PC)角度距离 9a 指令最近的消费者,这里即 9b(PC 距离差为 0)。理想情况下,针对指令 9b 的预取命令和指令 4a 的正常执行会从时间上重叠,因此不会因为预取操作导致系统整体访存时间的增加。但是 Roth 等[3,4]的预取方法没有考虑到,虽然指令 4a 和指令 9b 访问的地址只相差偏移量 8,但是有可能这两条指令访问的地址不在同一个 Cache 块中。此时有可能指令 9b 发生 Cache 命中,指令 4a 发生 Cache 缺失,按照 Roth 等[3,4]的预取方法此时针对指令 9a 的预取已经结束,因此流水线会因为等待指令 4a 执行结束而停顿。反之,如果针对指令 9b 的访存地址进行预取的

命令查询 L1-D Cache 发现命中后,继续预取 9a 的另一个消费者 4a,则在一定程度上会降低指令 4a 的执行时间。本文通过反馈预取机制,对此类情况进行了反馈预取,在 Roth 等[3,4]基础上进一步提高了链表结构的预取性能。

1.2 现有链表结构硬件预取方法

现有链表结构的硬件自动预取方法主要由两部分组成:链表结构自动识别模块和预取引擎。因目前多篇面向链表结构的硬件预取论文及本文中所用硬件自动识别和预取的机制都是基于 Roth 等[3]发表的文章,本文以此为例结合图 1 介绍其运行原理。为节省篇幅,图 2 把 Roth 等[3]文章中的硬件逻辑和本文对原有硬件的修改集成进行展示,图中黑体部分为本文所添加的模块,包括图 2(A)中的"反馈信号"和(C)中 CPR/CCN,其作用将在 2.2 节进行详细介绍。图 2(A)是 Roth 等[3]文章中预取模块整体的结构图,其中潜在生产者窗口(potential producer window,PPW)用于存储已经提交的访存指令;指令相关表(correlation table, CT)用于存储生产者-消费者指令对和消费者的指令模板 TMPL,指令模板用于计算针对预取地址的偏移。CT 中对于同一个生产者可以有多个消费者与其对应,PPW 和 CT 都按照 LRU 方式进行排列。PRQ 是预取命令队列,用于缓存预取命令并等待 Cache 端口空闲后发送到预取缓冲区(prefetch buffer, PB)中,PB 按照 FIFO 的方式组织,用于查询 Cache 并存储预取回的数据,以防预取回数据直接放入 Cache 导致 Cache 污染。

图 2(B)是建立和更新 PPW 和 CT 的过程,其中 BASEVAL 是访存指令基址寄存器值,PC 是程序计数器,INST 是提交的访存指令,ADDRVAL 是访存指令目的寄存器的值,PR 是生产者指令的程序计数器,CN 是消费者指令的程序计数器,TMPL 是指令模板中的偏移部分,用于计算预取地址。PPW 和 CT 具体的运行过程如下。①把当前提交指令(以 9b 为例)作为消费者指令,查询 PPW 取得对应的生产者指令。如果当前提交指令的基地址寄存器 BASEVAL(rax)与 PPW 中某个已经提交指令(9a)的目的寄存器值(rax)相等,说明找到对应的生产者指令。然后②把这对PR生产者(9a)、CN消费者



(A)预取模块组织结构图,(B)指令9更新PPW和CT过程,(C)已提交访存指令查询CT并把预取命令插入PRQ队列中,(D)预取命令发送到预取缓冲区:查询Cache,把命令发往存储子系统

图 2 链表结构反馈预取系统

(9b)指令对插入 CT 表中,同时把消费者指令的指令模板 TMPL(movq 8(MYM))插入到 CT 中。③把当前已提交指令作为生产者指令插入 PPW 中。

图 2(C)是查询 CT 表,取得预取地址的过程。④把当前提交指令作为生产者指令,查询 CT 寻找是否有消费者指令与其对应,如果命中 CT(9a/9b 指令对)。⑤则把当前提交指令的目的寄存器(rax)值作为消费者指令的基址寄存器,加上或减去 CT 中 TMPL 部分的偏移值(0x8)得出真正的消费者访存指令的访存地址(rax +0x8)。⑥发送到 PRQ 中。

图 2(D)是 PB 查询 L1-D Cache 并向存储子系统发出预取命令的过程。⑦当 L1-D Cache 端口空闲时,PRQ 把预取命令(rax + 0x8)放入 PB 中。⑧PB 查询 L1-D Cache,如果发生缺失则把预取命令继续发往下一级存储系统,否则针对当前生产者指令的预取结束。⑨预取命令完成后,数据会存入 PB 中,后续内存访问指令同时查询 L1D 和 PB,如果命中 PB,处理器不需要等待即可拿到数据。

2 链表结构反馈预取机制

程序中链式访存指令的访存地址依赖关系一般是一对一或者一对多的关系,即一个生产者指令对应一个或多个消费者指令。根据 Roth 等[3,4]实验结果,针对一个生产者指令,从 CT 中挑选一个(假设预取深度为一)按照 PC 计算距离当前访存指令最近的消费者指令进行预取即可达到理想效果,预取深度大于 1 时效果也不会因此进一步大幅提升,本文实验同样得出这个结论,实验结果见图 3(A)、(B)。对链表结构而言,针对相同生产者指令,其所有消费者指令被执行到的可能性是相等的,即消费者指令的执行优先级相同;只是消费者指令被执行到的时间有先后顺序,即消费者指令的时间优先级不同。例如,图 1 中指令 9a 作为生产者,指令 9b 和指令 4a 作为消费者,如果执行 9a 时,发现对于 9b 的预取已经命中了 Cache,可以继续针对指令 4a 进行预取,并且该预取有效的可能性和针对指令 9b 的预取可能性相同。但是,按照 Roth 等[3,4]的预取算法,预取 9b 的命令命中 L1-D Cache 时预取即结束,这会导致对于指令 4a 预取机会的浪费。本文经过实验发现,从预取深度 1 到预取深度 4 发出的预取命令中,约 76% 的预取操作会命中 L1-D Cache,如图 4 所示,因此反馈预取可以基于 Roth 等[3]的预取方法进一步提高预取性能。此外,对于其他访存操作,反馈预取不起作用是因为其消费者指令的执行优先级不同。例如最常见的基于步幅(Stride)的预取器,假如当前访存地址是 Addr0,根据历史访存记录计算出下一个最可能的访存地址 Addr1(Addr0 + Stride)后,如果该地址命中缓存,通过反馈预取获取下一个预取地址是 Addr2(Addr0 + 2 · Stride),此时 Addr2 的执行优先级小于 Addr1,因为 Addr0 后,Addr2 被访问的可能性要小于 Addr0,并且在 Addr0 和 Addr2 之间,可能因为访存模式改变,Addr2 永远不会被访问到造成无效预取污染 Cache。因此,被预取地址之间执行优先级相同是能否使用反馈预取的决定因素。

为了进一步提高链表结构的预取性能,本文提出反馈预取机制针对链式访存指令中,对单个生产者对应多个消费者的链表结构进行反馈预取,其有效性依赖于:(1)单个生产者对应多个消费者的链表结构访存指令在所有链表结构中的占比;(2)单生产者指令对应多消费者指令的情况发生 Cache 缺失的个数占所有链表结构发生 Cache 缺失的比例。本节针对以上两种情况,基于 X86 体系结构详细分析 Olden 指针密集型的标准测试集。Olden 标准测试程序主要包含中小规模的科学计算(bh、em3d)、过程模拟(health、power)、图(mst、tsp、perimeter、voronoi)、排序等(bisort、treeadd)。Roth 等[3]详细分

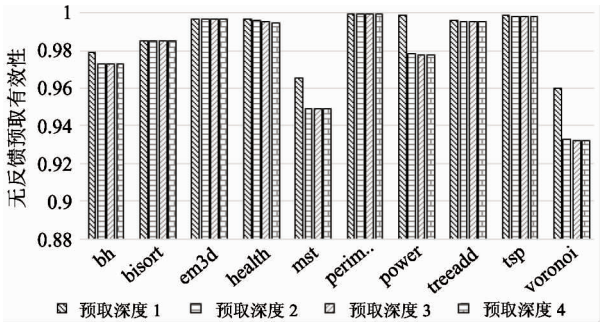


图 3(A) 无反馈预取的预取有效性平均约 98.2%

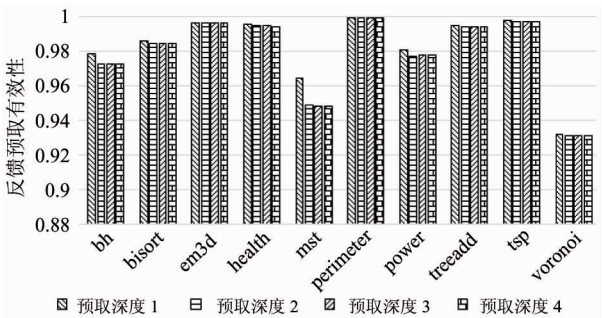


图 3(B) 反馈预取的预取有效性平均约 98.1%

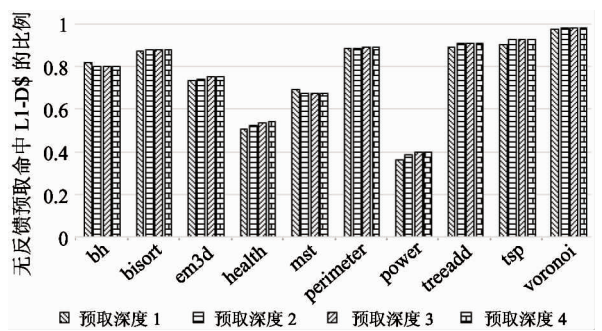


图 4 无反馈预取情况下,命中 L1-D Cache 的预取命令占所有预取命令的比例平均约 76%

析了 Olden 标准测试程序,对其中访存指令进行分类,并且统计其 Cache 性能表现。本文使用 X86 Pin 工具获取程序执行 Trace 并使用 Sniper 模拟器重新分析了该标准测试集,取得的数据如表 1 所示,与 Roth 等[3,4]在访存读操作比例、链式访存占所有访存比例、L1-D Cache 缺失率等方面有些差别,这是由于实验基于不同的体系结构及使用不同版本编译器导致。表 1 所统计数据只针对 L1-D Cache,Cache 大小为 64kB,采用 4 路组相连结构,每个 Cache 块为 64 B 大小,采用 LRU 替换算法。

通过表 1 中数据可以发现,单生产者对应多消费者的情况在链表结构访存指令中比较常见,平均占比从 13.9% 到 87.6% 不等。并且单生产者多消费者的情况造成的 Cache 缺失数,占所有链表结构导致的 Cache 缺失的比例从 1.6% 到 42.4% 不等(本文把命中 L1-D Cache 缺失状态保持寄存器(miss status holding register,MSHR)的情况也统计为命中 L1-D)。

综上数据显示,如果通过反馈预取对单生产者多消费者指令进行反馈预取,很有可能会进一步降低 L1-D Cache 缺失率,提高系统性能。

表 1　Olden 标准测试程序集分析(L1-D Cache 大小为 64 kB,Cache 块大小 64 B,4 路组相连)

程序名	数据集规模	数据结构	访存读操作比例	链式访存占所有访存比例	L1-D Cache 缺失率	多消费者链表结构比例	多消费者访存 Cache 缺失率
bh	4k body	八叉树	41.1%	10.4%	0.34%	22.7%	9.6%
bisort	250000 nums	二叉树	20.8%	43.8%	3.29%	68.3%	26.0%
em3d	2000n;10d;	链表	26.8%	27.8%	4.51%	13.9%	3.6%
health	5 level, 500iters	四叉树 链表	35.1%	79.9%	17.20%	49.3%	42.4%
mst	1024 node	链表数组	23.0%	45.9%	7.32%	32.7%	3.6%
perimeter	4000 · 4000 image	四叉树	16.5%	32.5%	1.87%	57.5%	1.9%
power	10000 nodes	多叉树 链表	10.5%	4.6%	0.51%	16.8%	1.6%
treeadd	256k nodes	二叉树	34.3%	51.8%	4.43%	87.6%	78.8%
tsp	100000 city	二叉树 链表	28.7%	68.7%	1.70%	22.7%	11.6%
voronoi	60000 points	二叉树	22.6%	55.6%	0.35%	28.6%	40.9%

2.1　反馈预取可行性分析

反馈预取的可行性依赖于两个方面。第一,从反馈预取的发出到被预取命令真正执行之间要有一定时间间隔。第二,反馈预取不会导致过多无效预取,污染 Cache。

对于第一点,Roth 等[3,4]详细分析了多个互相依赖指令之间的距离,对于 Olden 标准测试程序集中大部分程序,生产者指令和消费者指令之间相隔 128 条指令之多,该间隔足以完成多次反馈预取操作。同时,反馈预取的硬件设计中,会有机制保证如果发现将要进行反馈预取的指令已经被执行,则取消当前反馈预取操作,详细介绍参见 2.2 节。

对于第二点,本文通过实验发现,预取深度为 1 时反馈预取对比无反馈预取情况下,预取操作个数增加约为 4.27%,见图 5,预取深度为 1 时,反馈预取对比无反馈的预取有效性(即预取的数据被命中的个数占总的预取到 Cache 的预取数比例)为 99.55%,见图 3(A)、(B),从中也可以发现对于预取深度大于 1 的情况,反馈预取个数基本不再增加,这是因为对于一个生产者指令对应的消费者指令个数有限,不会随着反馈预取深度增加继续增加。由上述数据可知,采用反馈预取机制会增加预取操作的个数,并且预取操作的有效性很高,因此反馈预取可以进一步提升预取性能。

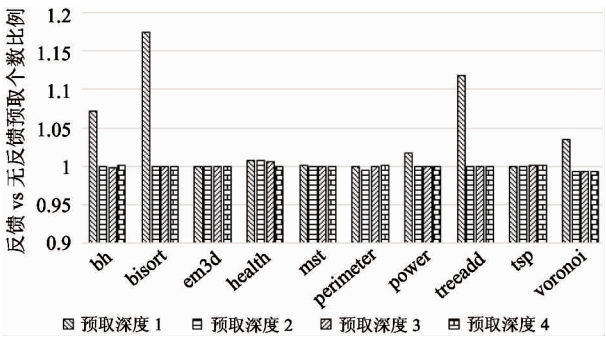


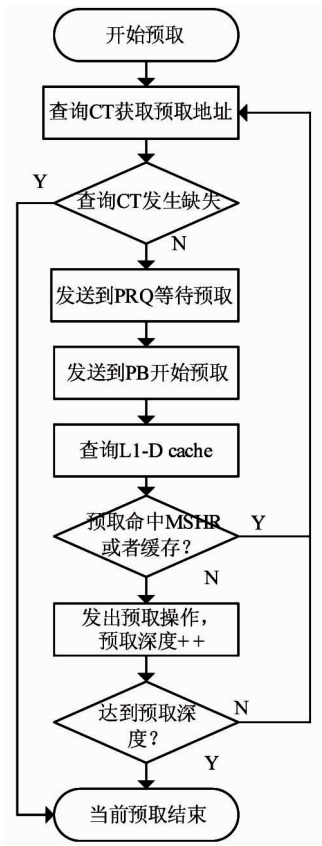
图 5　预取深度为 1 时，反馈预取对比无反馈预取个数平均增加 4.27%

2.2　反馈预取系统实现

前文提到，对于链表结构的某个生产者指令而言，其所有消费者指令之间的执行优先级相同，但是时间优先级不同，该特点是链表结构特有，对于其他数据结构和其他预取器而言并不适用，这是反馈预取机制能适用于链表结构的根本原因。

本文所述的链表结构反馈预取机制，在 Roth 等[3,4]的基础上增加了 Cache 查询反馈信号、当前生产者寄存器（current producer register，CPR）和当前消费者寄存器（current consumer，CCN），见图 2。理论上，反馈信号可以从 CT 直连到 L1-DCache 的单独端口上，并且不需要 CPR 和 CCN 记录历史预取信息，查询 CT 取得的预取地址可以直接查询 L1-D Cache。但是 Cache 端口是非常宝贵的系统资源，为了降低对 L1-D Cache 的端口需求，降低系统设计复杂度，本文设计反馈模块和 PB 共用同一个 Cache 端口。其中反馈预取模块用于把每一次发出的预取命令查询 Cache 的结果反馈给预取引擎，预取引擎据此判断是否应该对同一个生产者指令继续产生预取命令，CPR 用于记录当前生产者指令的 PC，CCN 用于记录当前消费者指令的 PC。

本文设计的反馈预取系统由硬件自动实现，由 3 部分组成，即自动识别链表结构的硬件模块、预取引擎、反馈模块。其中前二者的实现类似于 Roth 等[3,4]提出的方法，本节主要介绍反馈模块的结构，反馈预取的流程见图 6。例如，当执行图 1 中 9a 指令时，通过查询 CT 取得其消费者指令的 PC 9b，取得其访存地址并把预取命令发送到 PRQ 中，进而在 Cache 端口空闲时把预取命令发送到 PB 中，PB 查询 L1-D Cache，发现命中 MSHR 或者 Cache，则把命中信号回传 CT 继续进行查询，获取 4a 指令的访存地址继续进行预取。



在原有预取器基础上，增加从 L1-D 到 CT 的反馈信号，把对于预取命令查询 Cache 的结果反馈给 CT 用于指导进一步预取操作

图 6　反馈预取器工作流程

3　实验方案

为了验证反馈预取的有效性，本文实现了 Roth 等[3,4]提出的无反馈预取机制，基于此进一步实现了针对链式访存的反馈预取机制，并选择参数每周期执行指令数（instructions per cycle，IPC）和 L1-D Cache 缺失率进行性能对比。

本实验方案基于 Sniper-6.1 模拟器[12]，该模拟器是面向 X86 体系结构时钟精确的模拟器平台，经过跟真实硬件平台 Intel Core2 校准，其特点是模拟精度高，通过取样方式进行模拟加速，模拟结果可信度较高。本实验通过 Intel Pin-2.14 工具获取程序执行 Trace（包括访存指令编码、访存指令执行结果等），并输入模拟器以时钟精确模式进行性能评估。本实验的模拟器主要配置参数如表 2 所示，其中预

取器中 PPW 大小为 128 项,CT 为 256 项。实验采用的标准测试程序是 Olden,测试采用的程序参数详细描述如表 2 所示,编译器使用 gcc-5.4.1。

表 2　CPU 模拟器参数配置表

结构	参数
Frequency	2.66 GHz
Dispatch Width	4
ROB	96 entry
MSHR	16 entry
Branch Predictor	Pentium _ m Penalty 15cycle
L1-I Cache	Perfect
L1-D Cache	Size: 64 kB Associativity: 4 Cache line size: 64 B Replacement policy: LRU
L2	Size: 245 kB Associativity: 8 Cache block size: 64 B Replacement policy: LRU
PPW	128 entry
CT	256 entry

4　反馈预取性能评估

实验结果如图 7 和图 8 所示,图 7 集中展示了没有预取时程序 IPC、无反馈预取(即 Roth 等[3,4]提出的预取机制)时程序 IPC、本文提出的反馈预取的程序 IPC,其中,针对反馈预取,详细分析了预取深度 1 ~4 的情况下 IPC 变化。以上数据以无反馈预取 IPC 为 1 进行归一化处理。从上述数据可以看到,相比于没有预取,无反馈预取可以获得较好的性能提升,但相比于反馈预取还有提升空间。反馈预取机制相比于无反馈预取机制 IPC 平均提升 8.14%。当预取深度为 1 时,从程序 IPC 的角度看,对于 bisort、health、treeadd 等,IPC 提升从 1.3% 到 52.2% 不等。但是对于 tsp、voronoi IPC 分别有 5.2%、0.9% 的降低,图 3 中数据可以解释其性能下降的原因,对于 voronoi 和 tsp,在预取深度为 1 时,反馈预取有效性相比无反馈预取有效性有所下降,进

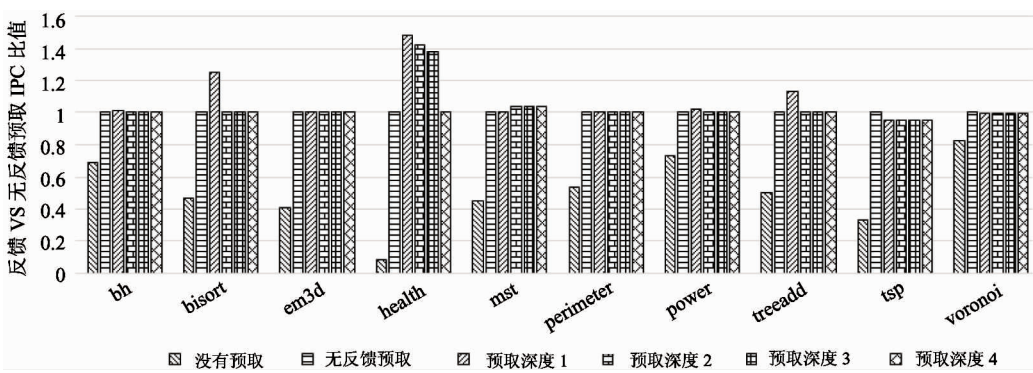


图 7　预取深度为 1 时反馈预取对比无反馈预取的 IPC 平均提升 8.14%

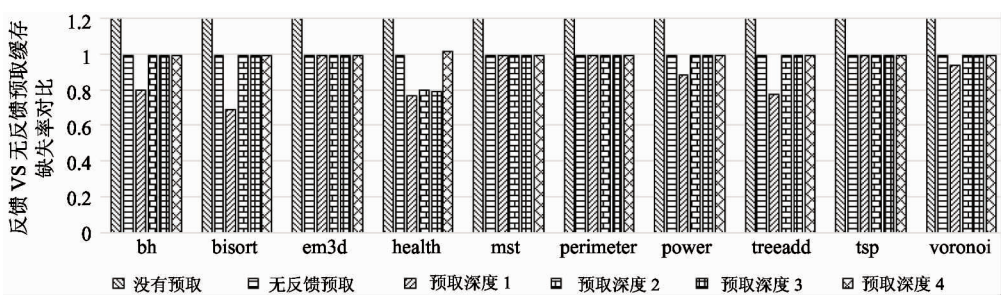


图 8　预取深度为 1 时反馈对比无反馈预取 L1-D Cache 缺失率平均降低 11.18%

而导致其性能有所下降。图 7 展示了 L1-D Cache 相关性能表现,其中以无反馈预取的 L1-D Cache 缺失率为 1 进行归一化,可以看到,反馈预取相比于无反馈预取 L1-D Cache 缺失率都有所降低,其降低值从0.3% 到 43.5% 不等,缓存缺失率平均降低 11.18%。

5 相关工作

与本文相关的工作主要包括两方面,一方面是针对链表结构的预取,另一方面是反馈预取机制。其中针对链表结构的预取可以细分为软件、硬件两类,其中硬件预取方式还可以细分为从处理器发出的预取和从存储系统发出的预取两种方式。以下对上述相关工作进行总结。

5.1 链表结构预取

链表结构的预取目前主要有软件预取和硬件预取两种方式。在软件预取方面,Luk 等[2] 提出一种编译算法使用类型信息识别链表结构,并在特定位置插入预取命令提高程序性能,该方法有可能产生冗余指令并发出无效预取命令。Chilimbi 等[8] 为了解决链表结构导致的性能问题,提出一种 Cache 数据放置策略,把相邻的链表结构尽可能地放入相同或者相邻的 Cache 块,提高空间局部性和算数相邻性,同时使用缓存着色技术消除冲突问题。该方法有可能导致较大的识别链表结构的开销,并且需要明确知道 Cache 详细微结构基础上实现。Karlsson 等[1] 为了减少链表访存指令的启动停顿时间,建立了预取指针数组,数组中指针指向链表结构中前几个节点,在访问节点之前通过预取指针数组发出预取。该方法可能导致大量 Cache 污染。Yang 等[7] 提出由软件控制通过预先执行机制加速链式访存的方法,通过其他线程的预先执行,把链式访存所需数据提前放入 Cache 中加速链式访存真正执行时的访存速度。

硬件预取方面主要是硬件自动识别的基于依赖链的预取,Roth 等[3] 对链表结构建立生产者消费者模型,在程序运行时动态识别访问链表结构的访存指令,并收集链表结构访存指令之间的依赖关系。基于空间局部性的原理,该文章假设生产者消费者之间的关系会在一定程度上比较稳定,因此当程序再次执行链表结构访存的生产者时,则在程序运行的同时发出对其消费者的预取命令。Roth 和 Sohi[4] 针对 Roth 等[3] 方法进行优化,有些情况下链表结构预取命令执行时间长于程序真正访存指令执行时间,从而导致预取操作无法完全和程序正常访存操作在时间上重叠。Roth 和 Sohi[4] 提出通过添加跳转指针进行更早的预取,可以进一步在 Roth 等[3] 的基础上针对一些特定场景提高预取性能。本文所提出的反馈预取和 Roth 等[3,4] 方法是互补的,通过反馈预取命令查询缓存的信息给预取引擎,可以进一步提高预取性能。Zhang 等[5] 通过编译器或者程序员对程序中链表结构进行标记,通过硬件机制对链式数据进行预取,该方法使用的标记信息需要在内存中连续,并且需要软硬件系统实现,复杂度较高。Yang 和 Lebeck[6] 基于 Roth 等[3,4] 方法基础上,提出在每一级 Cache 处添加流水线与 TLB,并把链表结构访存指令发送到每一级 Cache 的预取引擎处执行,在生产者指令产生的目的寄存器的值从 Cache 取出后立刻发出下一条预取命令,相比于 Roth 等[3,4] 提出的预取机制可以更早地取得链表结构的执行结果,性能更好。Jimenez 等[10] 提出一种把预取引擎置于存储处理器系统的预取机制,预取引擎在物理位置上紧邻目录并且与其互联,同时连接处理器 TLB。该预取引擎相比处理器预取器而言更接近数据从而获取更好的性能。Yang 和 Lebeck[6] 及 Srinath 等[11] 所述方法需要在 Cache 或互联网络中添加 TLB 或者 Cache,硬件开销大、实现复杂。

5.2 反馈预取

反馈预取系统的目的是根据程序行为进一步提高预取器性能或者减少因为预取导致的性能下降。Jimenez 等[10] 通过分析 IBM POWER7 处理器,发现预取的参数对于预取器性能影响较大,因此提出自适应的预取方法,主要包括两个步骤,在预取探索阶段,设定不同的预取器参数记录哪一个预取会有最好的 IPC,然后将其用于真正运行阶段。在运行时也会记录多个阶段的平均 IPC 并根据此进一步修正

预取器的参数。Srinath 等[11]通过类似取样方式取得预取器精度、Cache 污染程度、预取延迟程度等信息,动态调整预取器的参数,达到预取器性能最优的目的。上述反馈预取都是根据长时间取样结果重新配置预取器调整预取参数,开销较大,且无法针对每一个预取操作进行反馈预取。

6 结 论

本文详细分析了现有针对链表结构的预取有效性,发现其预取有效性高达 98.2%,但这些预取命令只占所有查询 L1-D Cache 预取命令的 24%,约 76% 的预取命令会因为命中 L1-D Cache 而终止执行。基于此,本文针对链表数据结构的特点,提出了反馈预取的方法,在前人工作基础上进一步将链表数据结构访存性能平均提高约 8.14%,降低 L1-D Cache 缺失率 11.18%。此外,本文提出的反馈预取方法与以往预取方法并不冲突,可以协同工作。

目前已有的反馈预取机制[10,11],其反馈方式需要通过对一段时间内针对程序性能进行取样后进行预取参数的调整,而且这些反馈机制并不针对链表结构。本文反馈预取机制可以针对每一次预取操作进行反馈预取,可以保证反馈信号的及时性,同时不需要重新配置预取器的预取深度,而预取器的重新配置会对预取性能造成较大影响。目前本文所用 Sniper 模拟器及反馈预取器实现代码已经开源,见 https://github.com/qianlong-zhang/sniper

参考文献

[1] Karlsson M, Dahlgren F, Stenstrom P. A prefetching technique for irregular accesses to linked data structures. In: Proceedings of the 6th International Symposium on High-Performance Computer Architecture[C]. Touluse, France, 2000. 206-217

[2] Luk C, Mowry T. Compiler-based prefetching for recursive data structures[J]. *ACM SIGOPS Operating Systems Review*, 1996, 31(9): 222-33

[3] Roth A, Moshovos A, Sohi G. Dependence based prefetching for linked data structures[J]. *ACM SIGOPS Operating Systems Review*, 1998, 32(5): 115-126

[4] Roth A, Sohi G. Effective jump-pointer prefetching for linked data structures[C]. In: Proceedings of the 26th International Symposium on Computer Architecture, Atlanta, USA, 1999, 111-121

[5] Zhang Z, Torrellas J. Speeding up irregular applications in shared-memory multiprocessors: memory binding and group prefetching[C]. In: Proceedings of the 22nd International Symposium on Computer Architecture, Margherita Ligure, Italy, 1995. 188-99

[6] Yang C, Lebeck A. Push vs. pull: data movement for linked data structures[C]. In: Proceedings of the 14th International Conference on Supercomputing, Dallas, USA, 2000, 176-186

[7] Yang C, Lebeck A, Tseng H, et al. Tolerating memory latency through push prefetching for pointer-intensive applications[J]. *ACM Transactions on Architecture and Code Optimization (TACO)*, 2004, 1(4): 445-475

[8] Chilimbi T, Larus J, Hill M. Improving pointer-based codes through cache-conscious data placement[R]. Technical report CS-TR-98-1365. Madison: University of Wisconsin, 1998

[9] Hughes C, Adve S. Memory-side prefetching for linked data structures[J]. *Journal of Parallel and Distributed Computing*, 2005, 65(4): 448-463

[10] Jimenez V, Cazorla F, Gioiosa R, et al. Adaptive prefetching on POWER7: improving performance and power consumption[J]. *ACM Transactions on Parallel Computing*, 2014, 1: 1-25

[11] Srinath S, Mutlu O, Kim H, et al. Feedback directed prefetching: improving the performance and bandwidth-efficiency of hardware prefetchers[C]. In: Proceedings of the IEEE 13th International Symposium on High Performance Computer Architecture, Scottsdale, USA 2007, 63-74

[12] Carlson T, Heirman, Eeckhout L. Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation[C]. In: Proceedings of the 2011

International Conference for High Performance Computing, Networking, Storage and Analysis (SC), Seattle, USA, 2011, 1-12

Feedback prefetch for linked data structure

Zhang Qianlong[*,**], Hou Rui[***], Yang Sibo[****], Zhang Lixin[*]

([*] Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

([**] University of Chinese Academy of Sciences, Beijing 100049)

([***] Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093)

([****] School of Software & Microelectronics, Peking University, Beijing 100871)

Abstract

The pre-existing prefetch technique for linked data structure (LDS) is analyzed in detail, and the influence of prefetch depth to the performance of application is reveal with detail experiments. Besides, the pattern of single producer instruction of memory access corresponding multiple consumer instructions of memory access is analyzed, and the shortness of existing prefetch technique is pointed out. To further improve the prefetch performance for LDS, a feedback prefetch technique is proposed, which can give the feedback of cache query outcome to the prefetch engine if cache hit occurs, then another prefetch command is created for that producer instruction of memory access. The feedback prefetch technique can cooperate with other prefetch techniques for LDS. Experiment results show that, compared with prefetch technique without feedback the feedback prefetch technique can increase the IPC by 8.14%, and reduce the L1-D Cache miss rate by 11.18% with negligible hardware area increase.

Key words: linked data structure (LDS), pointer chasing, data prefetch, feedback prefetch, hardware prefetch