基于控制流的混合指令预取

沈 立,王志英,鲁建壮,戴 葵
(国防科技大学计算机学院,湖南长沙 410073)

摘 要: 取指令能力的高低对微处理器的性能有很大影响.指令预取技术能够有效地降低指令 Cache 的访问失效率,提高微处理器的取指令能力,进而提高微处理器的性能.本文提出了一种基于程序控制流的混合指令预取机制,它采用顺序预取和非顺序预取相结合的方式将指令提前读入指令 Cache.模拟结果显示,该方法能够有效地提高指令 Cache 访问的命中率,并具有实现简单,无效预取率低等特点.

关键词: 控制流图; 指令预取; 分支预测
中图分类号: TP363 **文献标识码**: A **文章编号**: 0372-2112 (2003) 08-1141-04

Hybrid Instruction Prefetch Based on Control Flow

SHEN Li, WANG Zhi-ying, LU Jian-zhuang, DAI Kui
(*National University of Defense Technology, School of Computer, Changsha, Hunan* 410073, *China*)

Abstract: Instruction supply can influence processor performance greatly. Instruction prefetching is an effective mechanism to reduce instruction cache miss rate. It proposes an instruction prefetching mechanism based on program control flow graph, while sequential prefetch and non-sequential prefetch are both employed to fetch instructions in advance. Experimental results show it can increase instruction cache hit rate effectively. It predicts program behavior statically so that it does not need any complex hardware to predict and restore.

Key words: control flow graph; instruction prefetch; branch prediction

1 引言

指令 Cache 能够弥补微处理器和存储器之间的性能差距,减少平均访存延迟,提高微处理器的取指令能力,但指令 Cache 访问失效会造成严重影的性能损失.指令预取技术通过分析程序行为,提前把即将被访问的指令从存储器中读入指令 Cache,达到降低指令 Cache 访问失效率的目的.许多研究[3,4]均表明,高效的预取技术能够大幅度提高取指令部件的性能.

高质量的指令预取技术应解决以下三个问题:

(1) 由于分支指令的存在,运行时所执行的指令流并非总是顺序的,当控制流发生转移时,如何确定被预取指令的地址?

(2) 无效预取包括两类:读入已经在 Cache 中的指令块;指令块被读取入 Cache 后,在被替换出 Cache 之前没有被访问过.无效预取对提高性能毫无疑义,如何判断并减少无效预取?

(3) 如果预取时间太早,可能会造成无效预取;如果预取时间太晚,又无法确保在使用指令之前将其读入指令 Cache.如何确定预取指令的流出时间?

衡量指令预取技术性能高低的标准主要包括:指令 Cache 失效率、无效预取比率以及能否在使用指令之前完成预取(本文用预取完成率来表示),本文也将使用这三项指标对预取技术进行评价.

2 相关工作

近十几年来,人们对指令预取技术进行了大量的研究.最简单的指令预取技术为 Next-*N*-Line 预取.这是一种顺序预取技术,每次预取当前指令块之后的 *N* 个块,通常在 $N=4$ 或 8 时效果最好.其优点是实现简单,但效率不高,体现在无效预取指令多,并且不能有效地对长距离跳转、过程调用、过程返回、间接跳转等分支指令进行预取.

Target-Line[6]预取使用一个预测表确定预取地址,表中记录了指令地址和对应的预取地址.每个 Cache 块可能包含多条指令分别对应预测表中某个项.这种技术可能无法保证有充足的时间完成预取,其效果的好坏直接受到预测表大小的影响,但其无效预取率低于 Next-*N*-Line 预取.

Wrong-Path 预取[1]根据条件分支指令进行预取,但是此次预取并不是为本次执行进行的,而是为了确保在下次执行该分支指令时,目标指令已经被读入指令 Cache.这种技术实

收稿日期:2002-09-16;修回日期:2003-03-08
基金项目:863 高技术研究发展计划(No. 2001AA111050)

现代价低,一般有足够的时间完成预取,但可能有较多的预取是无效的.

Markov 预取[2]最初是为了实现数据预取提出的,每当访问 Cache 失效时进行预取操作.它维护一个预测表,表中记录了每个失效地址及其所对应的预取地址,失效地址和预取之间是一对多对应关系.一般说来,采用这种方法,发出预取操作的时间足够早,但无法保证预取是否有效.

总的说来,这些技术能够在一定程度上降低指令 Cache 失效率,但主要问题在于:发出预取的时间不够确定,无法确保有足够的时间完成预取,同时也无法保证预取是否有效.本文提出了一种基于程序控制流图的指令预取技术,该方法采用顺序和非顺序预取相结合的方式进行指令预取,在降低指令 Cache 失效率的同时,能够保证发出预取操作的时间足够早.另外,这种方法能够保证预取的有效性,降低无效预取率.

3 基于控制流的指令预取

3.1 控制流图

基本块是程序中一个顺序执行的语句序列,其中只有一个入口和一个出口.对一个基本块而言,执行时只能从其入口进入,从其出口退出.对于一个给定的程序,可以把它划分为一系列的基本块.基本块的入口可以是程序的第一条指令,或者能由分支指令转移到达的指令,或者紧跟在条件分支指令之后的指令.而出口可以是分支指令,或者过程返回指令.本文将一个基本块执行结束后下一个被执行的基本块称为它的后继基本块,对于 *RISC* 指令集,一个基本块最多只有两个后继基本块,其中需要发生转移才能到达的为非顺序后继,另一个为顺序后继.

程序的控制流图 *CFG* 可以定义为:$CFG=\langle N, E\rangle$, N 为程序中所有基本块的集合,每个基本块表示一个结点, E 为基本块之间的所有有向边的集合,每条有向边表明一个可能的控制流方向,如图 1 所示.由于循环指令的存在, *CFG* 是一个有环有向图.

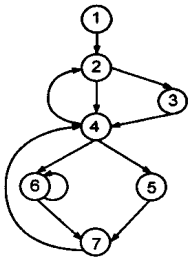

图 1 程序流图

采用 Next-*N*-Line 技术,每当取指令部件读一个指令块,指令 Cache 会自动从存储器读入与之相邻的下 N 个块,因而可以保证将基本块的顺序后继提前读入指令 Cache.但分支指令、函数调用的存在使得控制流经常会发生非顺序改变,这就需要准确推测程序控制流的方向,并根据预测结果进行预取[5].程序控制流图 *(CFG)* 指明了所有可能的控制流方向,本文采用 Next-*N*-Line 与非顺序预取相结合的方式,通过 Next-*N*-Line 技术预取顺序后继指令 $(N=4)$,而利用非顺序预取技术预读入非顺序后继指令.Next-*N*-Line 功能能够很容易地在指令 Cache 中实现,但非顺序预取需要一定编译技术和硬件设计的支持.

3.2 非顺序预取

CFG 中各结点的出口指令指明了它的非顺序后继结点,可以根据该指令确定被预取指令的地址.如果有的话,预取指令是结点的第一条语句,这样可以在执行一个结点的同时进行预取,保证有充足时间完成预取.预取时每次读入 1 个指令块即可,这样可以减少不必要的非顺序预取.如果该指令块最终被执行,Cache 能够通过顺序预取读入后继的指令块.

根据出口指令类型不同,我们对指令集进行了扩展,定义了一组预取指令,其语法如表 1 所示.这里 addr 是分支指令的转移目标地址,第一类 fetch 指令按照与条件分支指令和直接无条件分支指令相同的方式执行.而 Stack 表示返回地址栈,函数调用的返回地址被依次压入其中并按照 FILO 的顺序弹出.

表 1 预取指令语法格式

预取命令	描 述
fetch addr	针对条件分支指令和直接无条件分支指令,预取指令地址为 addr
fetch R_i	针对间接无条件分支指令,预取地址保存在 R_i 中
fetch stack	针对过程返回指令,预取地址保存在栈顶单元中

非顺序预取的流程为:程序开始运行时,取指令部件读入第一个指令块并将其中的指令送入译码部件译码,如果译码结果为预取指令,则向指令 Cache 发送非顺序预取操作;其他类型的指令则送入指令 buffer,供微处理器后端执行.这样能够在执行当前结点指令的同时预取其非顺序后继结点.一个结点执行结束后,根据其运行结果读入并执行正确的后继结点以及其中可能的非顺序后继指令.依次类推,直至整个程序结束.具体流程如图 2 所示,微处理器采用前后端分离的结构,图中只包括与非顺序预取相关的内容.

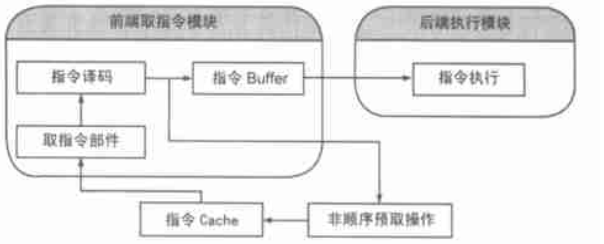

图 2 非顺序预取流程图

值得注意的是,尽管非顺序预取和顺序预取的触发条件不同(当取指令部件读入一个指令块时 Cache 将预取它的顺序后继指令块;而非顺序预取操作由预取指令触发),它们也有可能发生冲突.为避免冲突,我们设计非顺序预取的优先级高于顺序预取,即当执行非顺序预取操作时暂停顺序预取操作,待非顺序预取结束后,继续进行顺序预取.

另外,在进行实际操作之前,还应判断要预取的指令是否已经在 Cache 中,如果在就取消预取操作.这种过滤机制利用 lockup-free Cache[8]实现.这种 Cache 的实现开销低,当指令 Cache 中存在空闲端口时就可以进行检测[7].当以下几种情况发生时 Cache 端口会空闲:(1) 访问指令 Cache 失效;(2) 指令窗口满;(3) 指令 Buffer 满.另外,采用多体/多端口指令 Cache 也能实现检测,但实现开销太大.因此我们选用了 lockup-free Cache.

3.3 预取指令生成

根据 3.1 节所述,当转移成功时,基本块的出口语句都能

够非顺序地改变程序计数器(PC)的值.在 RISC 机器中,一个基本块可以有一个或两个后继基本块,这些后继基本块的地址以及相应的预取指令都可以根据出口语句的类型确定.

一般说来,基本块出口语句可以分为四类:条件分支指令、直接无条件分支指令、间接无条件分支指令和过程返回指令.根据这四类指令的不同特征可以选择不同的预取指令.

(1)条件分支指令:根据条件,转移可能成功也可能失败,因此有两个后继基本块.编译时可以获得其非顺序后继的地址 addr,预取指令为:fetch addr.

(2)直接无条件分支指令:转移总是成功,只有一个后继基本块,在编译时可以获得转移目标地址 addr,预取指令为:fetch addr.

(3)间接无条件分支指令:转移总是成功,只有一个后继基本块,转移目标地址保存在寄存器 R_i 中,预取指令为 fetch R_i.

(4)过程返回语句:转移总是成功,只有一个后继基本块,这类语句通常出现在过程调用返回时,由于过程调用可能出现嵌套,可以用一个栈来保存过程调用的返回地址(即预取地址),每次从栈的顶端弹出预取地址,预取指令为 fetch stack.

由于 Cache 能够预取顺序的 N 个指令块,假设指令块大小为 B 字节,那么当被预取指令的偏移量小于等于 $N \times B$ 时,预取任务已经由 Cache 完成,不必插入预取指令.这样做有两个好处:保证充足的预取时间,以及减少无效的非顺序预取.另外,对于循环结构的控制流,由于其中的指令具有非常高的时间局部性,而且循环结构中的指令由 Cache 顺序预取,因此无需预取其非顺序后继指令块.通过这两项过滤措施能够减少无效预取操作.

4　试验结果及分析

本节将按照第 1 节中提出三项指标分析该指令预取机制的性能.

顺序预取由 Cache 完成,每读入一个指令块,自动确定下 N 个指令块的地址并预取.非顺序预取指令的地址在编译时即可确定,而且在编译时能够保证预取地址的准确.执行一个基本块第一条指令的同时,对其非顺序后继基本块进行预取,这样可以保证有比较充足的时间完成预取.

采用这种机制,第二种无效预取将不可避免:当分支转移成功时,顺序预取的指令块无效,而当转移失败时,非顺序预取的指令块无效,但由于每次只预取 1 个指令块,无效预取的损害被降至最低.

4.1　实验环境和测试程序

我们选择了以下的模拟环境:使用两级 Cache,指令 Cache 为 lockup-free Cache,容量为 32K 字节,相联度为 4,块大小为 32 字节,它的另外一项功能是能够进行 Next-N-Line 预取(N=4);二级 Cache 为数据指令混合 Cache,容量为 1M 字节;取指令部件的取指带宽为 32 字节/cycle,两级 Cache 之间的带宽为 8 字节/cycle;Cache 的替换策略为 LRU,采用写回方式.微处理器前端的指令 buffer 最多可以保存 16 条指令.

我们选择 SPECint95 基准程序作为测试程序,如表 2 所示.所有测试程序均用 GCC 编译,没有经过任何其他优化.编译的同时向程序中加入非顺序预取指令.

表 2　测试程序

测试程序	I-Cache 失效率	二级 Cache 失效率
Gcc	6.4 %	0.012 %
Perl	2.7 %	0.009 %
Vortex	10.8 %	0.011 %
Go	2.1 %	0.010 %
m88ksim	1.2 %	0.021 %
Li	1.9 %	0.007 %
Compress	3.2 %	0.011 %
Ijpeg	2.1 %	0.015 %

4.2　实验结果及分析

表 3 列出了针对 SPECint95 测试程序的模拟结果,从表中可以看出,使用以基本块为单位的预取技术,能够在保证无效预取率比较低的前提下,有效提高指令 Cache 的访问命中率.所有 SPCEint95 测试程序的平均预取完成率为 97.025 %,说明采用这种方法,能够保证有足够多的时间完成预取.

表 3　模拟测试结果

测试程序	I-Cache 失效率	预取完成率	无效预取率
gcc	0.65 %	97.2 %	6.9 %
perl	0.99 %	96.5 %	7.7 %
vortex	0.56 %	97.4 %	10.1 %
go	0.35 %	98.5 %	4.3 %
m88ksim	0.13 %	96.7 %	7.2 %
Li	0.19 %	98.3 %	3.3 %
Compress	0.23 %	97.1 %	6.1 %
Ijpeg	1.01 %	94.5 %	9.3 %

表 4 从指令 Cache 失效率、无效预取率和预取完成率三个方面比较了本文讨论的几种指令预取技术的性能,这里 Next-N-Line 预取 N=4,Target 预取预测表大小为 32 项.从中可以看出,采用本文提出的预取方法 CFGP,指令 Cache 失效率最低,预取完成率和无效预取率也比较理想.

表 4　几种预取策略的性能比较

预取策略	I-Cache 失效率	预取完成率	无效预取率
Next-N-Line	3.10 %	81.00 %	30.10 %
Target-Line	1.50 %	87.00 %	19.70 %
Wrong-Path	0.90 %	90.10 %	17.30 %
Markov	1.30 %	98.20 %	15.20 %
CFGP	0.51 %	97.03 %	6.87 %

5　结束语

本文提出了一种基于程序控制流的混合指令预取技术,该技术结合使用顺序和非顺序预取方法,当控制流可能发生转移时,采用非顺序方法预取它的非顺序后继指令块,否则顺序预取其后的指令块.由于预取都建立在静态分析程序控制流图的基础上,因而可以避免使用复杂的分支预测硬件,降低了硬件复杂度.另外还通过一定的过滤机制减少无效非顺序预取操作.模拟结果显示,该方法能够有效地降低指令 Cache 的访问失效率,无效预取比例低,效果比较理想.

参考文献:

[1] J Pierce, T Mudge. Wrong-path instruction prefetching [A]. The 29th International Symposium on Microarchitecture [C]. Paris: IEEE Computer Society Press, 1996. 165 - 175.

[2] D Joseph, D Grunwald. Prefetching using markov predictors [A]. The 24th Annual International Symposium on Computer Architecture [C]. Denver: ACM Press, 1990. 252 - 263.

[3] N Jouppi. Improving direct-mapped cache performance by the addition of a small fully associative cache and prefetch buffers [A]. Proceedings of the 17th Annual International Symposium on Computer Architecture [C]. Barcelona: ACM Press, 1990. 388 - 397.

[4] C Xia, J Torrellas. Instruction prefetching of systems codes with layout optimized for reduced cache misses [A]. The 23rd Annual International Symposium on Computer Architecture [C]. Philadelphia: ACM Press, 1996. 271 - 282.

[5] I K Chen, C C Lee, T N Mudge. Instruction prefetching using branch prediction information [A]. In 1997 International Conference on Computer Design [C]. Austin: IEEE Computer Society Press, 1997. 593 - 601.

[6] J E Smith, W-C Hsu. Prefetching in supercomputer instruction caches [A]. Proceedings of the 1992 ACM/ IEEE conference on Supercomputing [C]. Minneapolis: IEEE Computer Society Press, 1992. 588 - 597.

[7] G Reinman, B Calder, T Austin. Fetch directed instruction prefetching [A]. In Proceedings of the 32nd Annual International Symposium on Microarchitecture [C]. Haifa: IEEE Computer Society Press, 1999. 16 - 27.

[8] D Kroft. Lockup-free instruction fetch/ prefetch cache organization [A]. In 8th Annual International Symposium of Computer Architecture [C]. Minneapolis: IEEE Computer Society Press, 1981. 81 - 87.

[9] K Farkas, N Jouppi. Complexity/ performance tradeoffs with non-blocking loads [A]. In 21st Annual International Symposium on Computer Architecture [C]. Chicago: IEEE Computer Society Press, 1994. 211 - 222.

作者简介:

沈 立 男,1976 年生于陕西宝鸡,主要从事微处理器系统结构,编译优化技术等方面的研究.

王志英 男,1956 年生于山西,主要从事计算机系统结构研究.

鲁建壮 男,1977 年生于河北,主要从事微处理器系统结构,芯片设计等方面的研究.