支持 Trace 预构的分支目标提取

杜贵然, 窦 勇, 徐 明, 周兴铭
(国防科技大学计算机学院, 湖南长沙 410073)

摘 要: 程序中大量存在着分支指令, 分析发现大多数执行分支的偏移量可从指令中直接得到. 为了支持 Trace 预构, 我们提出了分支目标提取机制——BTP, BTP 扫描预取的指令块, 提取分支指令及分支目标. Trace 预构机制根据 BTP 的扫描结果, 预先构造程序的执行踪迹. 对 SPECint95 测试程序的模拟实验表明: BTP 能够有效识别目标地址, 一级指令 Cache 访问的不命中率显著下降, 程序的性能也相应提高.

关键词: 分支目标提取; 踪迹预构
中图分类号: TP311.13 文献标识码: A 文章编号: 0372-2112 (2002) 02-0156-04

Branch Target Profiling——Supporting Trace Preconstruction

DU Gui-ran, DOU Yong, XU Ming, ZHOU Xing-ming
(*Department of Computer Science, Changsha Institute of Technology, Changsha, Hunan 410073, China*)

Abstract: The offsets of most branches can be evaluated from instruction and current PC. Branch target profiling (BTP) mechanism scans prefetched instruction blocks, and generates targets of direct branches. The trace preconstruction mechanism observes the processor's instruction dispatch stream to detect opportunities for jumping ahead of the processor. However, branch hazards prevent trace preconstruction from branch target disambiguity. With the help of BTP, Trace preconstruction mechanism constructs pseudo traces from prefetched instructions. To SPECint95 benchmark suite, BTP can generate branch target accurately, reduces cache access miss rate efficiently, and prompts program performance.

Key words: branch target profiling; trace preconstruction

1 引言

提高微处理器的指令级并行性(ILP)有三个瓶颈: 指令读取的瓶颈、指令执行的瓶颈、操作数获取的瓶颈. 其中, 获取指令的瓶颈是最根本的瓶颈, 没有指令就无法确定操作数, 也不能执行. 当前的微处理器大都将指令的处理划分为流水线, 以提高执行部件的吞吐率, 增加执行带宽. 而存储器的发展并没有相应的减少访问的延迟. 在这样的情况下, 已经很难保证一级 Cache 能够在一个时钟内向译码部件提供指令, 从而使指令译码的等待延时相应增加. 因此, 解决指令读取是加快指令执行的关键问题.

指令预取技术实现指令块的预先读取, 是消除指令读取瓶颈的一个有效的手段. 但程序中大量存在的分支指令制约了指令预取的有效性. 先进微处理器广泛采用分支预测技术[4]提高指令预取的准确性, 受进程切换、程序固有的控制逻辑、以及预测部件自身的能力等的影响, 分支预测不可能完全正确, 限制了指令预取的实际效能.

Trace Cache[5]作为一种低延迟高带宽的指令读取机制受到体系结构设计师的广泛重视, 并为商用微处理器所采纳. Trace Cache 机制保存程序执行过的动态指令踪迹, 当一条保存的执行路径被重复执行时, Trace Cache 能够一次提供动态指令流的多个逻辑连续块. 而在静态程序中这些逻辑连续块可能是存储在物理上不连续的块中.

受 Trace 的动态特性, 及 Trace Cache 的容量限制, 容易导致 Trace Cache 访问的不命中. Trace 预构技术[2](Trace preconstruction)是 Trace Cache 不命中时的补救办法, 它根据分支目标缓冲区的信息, 预先得到程序的运行踪迹, 以弥补分支预测失败导致的额外开销. Jacobson 等人提出的 Trace 预构机制针对循环结束点、子程序调用返回点之后的程序进行预构, 限制了预构的范围, 不能全面支持指令处理部件读取指令的进一步要求. 同时, Trace 预构的性能也与程序的结构紧密相关, 对于子程序调用嵌套较深和循环嵌套层次较多的程序, 预构 Trace 缓冲区的容量限制了能够保存的 Trace 数目, 使预构部件的功能得不到充分发挥, 从而削弱了预构机制的有效性.

本文提出了分支目标地址提取(BTP: Branch Target Profiling)机制来支持 Trace 的预构. BTP 机制对程序将要执行的指令块进行过滤, 并计算分支的目标, 协助 Trace 的预构并指导指令的读取. 基于 BTP 的 Trace 预构能够动态适应程序的逻辑, 减少对分支历史的依赖, 扩大预构的范围, 提高预构的准

收稿日期: 2001-01-16; 修回日期: 2001-07-09
基金项目: 九五国防预研项目(No. B801); 国家自然科学基金重点课题(No. 69933030); 湖南省院士基金

确度. 本文的结构: 第 2 节介绍分支目标提取原理及实现策略, 第 3 节说明基于分支目标提取的 Trace 预构方法, 第 4 节介绍我们的模拟方法并分析模拟结果, 第 5 节是本文的总结以及进一步的工作.

2　分支目标提取原理及实现策略

2.1　分支指令分析

根据程序中分支指令的目标信息可获得性来分类, 把分支指令分为两类: 第一类是指令中没有直接的转移目标地址信息的指令, 如 SimpleScalar[11] 结构(类似于 MIPS 的 ISAII 指令集) 的寄存器转移型指令 JR 和 JALR, 指令中没有直接给出转移目标的信息, 而是以寄存器的内容作为转移目标; 第二类是转移目标可在指令中得到的指令, 如条件分支指令 BEQ, 在指令操作数字段给出了目标地址的偏移量. 对第一类指令, 仅当控制转移指令执行完毕才能确定转移目标, 如果要预取转移指令后的指令, 一般可采用分支预测部件给出的预测地址作为预取目标地址. 对第二类指令, 一般来讲, 当得到分支指令后就能比较准确地得到分支两个方向的目标地址, 为实现指令预取提供了条件. 分析 SPECint95 程序, 我们发现程序中大量存在着第二类分支指令. 如表 1 所示.

表 1　SPECint95 程序中的分支指令(数据: 执行的第 1– 101M 指令)

	第一类分支数	第二类分支数	第二类分支比例(%)	第一类分支命中率	分支预测命中率	平均基本块指令数
gcc	1306518	15003840	91.9896	0.7646	0.8650	4.7720
compress	856395	10704940	92.5926	1.0000	0.8321	5.0937
go	794675	10343834	92.8655	0.8953	0.8004	6.4147
ijpeg	2245100	13873601	86.0715	0.9994	0.9585	5.7093
li	2518166	14680619	85.3585	0.8260	0.9013	4.3490
m88ksim	3171604	15441200	82.9601	0.9995	0.9712	4.5451
perl	1370714	18126384	92.9697	0.9981	0.9706	4.8242
vortex	1846991	13424733	87.9058	0.9894	0.9598	6.3066

从表中可以看出, Cint95 程序集的分支指令约占实际执行指令的五分之一. 在所有执行的这些分支中, 每 10 条中大约有 9 条的目标地址可以通过当前程序计数器的内容与分支指令中的偏移量移位相加来得到.

为此, 我们设计了 BTP 部件用来得到分支目标地址. BTP 部件是为Trace 预构而设计的分支指令预处理部件.

2.2　分支指令提取方法

BTP 的基本思想: 顺序扫描预取的指令块, 获取指令块中的分支指令, 并将第二类分支的转移偏移量与扫描的程序计数器相加, 得到分支的目标, 以此目标作为下一次预取的依据. BTP 的基本构成单元有: 指令块缓冲区, 分支目标地址表(BTT: Branch Target Table), 分支指令匹配部件(BMU: Branch Matching Unit)等. BTP 的结构示意如图 1. 下面简单介绍这些部件.

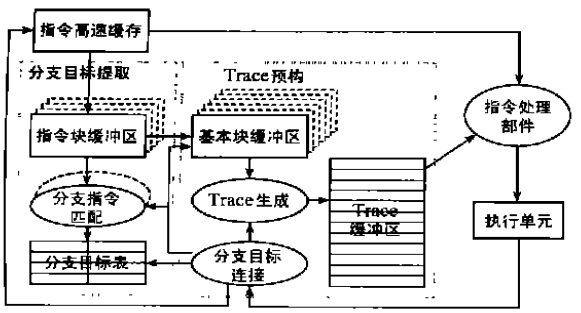

图 1　分支目标提取和 Trace 预构示意图

图 1 中的“指令块缓冲区”保存从指令 Cache 中读取的多个指令块, 指令块仍以程序地址索引. 指令块缓冲区中的指令有两个用途: 一是作为分支匹配部件的输入, 二是作为构建基本块的指令源. 指令块缓冲区的容量可根据预构的深度来决定. 一般而言, 若要预构含两个基本块的 Trace, 预构的 Trace 最多可以包含一条分支指令, 而分支指令向下可能会有两个方向, 其中一个方向的基本块是分支不转移时将执行的指令块, 另一个方向的基本块是分支转移时将执行的指令块. 假设这两个基本块不在同一个 Cache 块中, 在这种情况下, 指令块缓冲区应该有能力同时提供两个基本块的指令. 除了存储正在被分支匹配部件分析的指令块以外, 从预构的角度, 还应该保留已完成分支指令匹配但尚未填入基本块缓冲区的指令. 在以下的讨论中, 假设前瞻两级分支做Trace 的预构, 所以设计的指令块缓冲区容量为四块. 需要说明的是: 四个指令块的容量可以支持含两个以上分支的Trace 预构. 包含更多分支的 Trace 预构将会对存储系统提出更高的要求.

选择指令块缓冲区的长度与二级 Cache 的块长度一致. 考虑到当一级 Cache 访问不命中时, 将读取二级 Cache 的指令块, 但一般的系统设计中, 一级 Cache 的块长度要比二级 Cache 小, 所以当一级 Cache 不命中时, 可将二级 Cache 中的一个完整块拷备到指令块缓冲区, 同时, 又不会因为块长度的关系, 导致一级 Cache 的污染.

表 2　分支目标地址表(BTT)的结构

表项名称	表项说明
C_addr	分支指令的地址
C_type	分支指令类型
B_next	分支下一条指令的地址
B_not_takenB_btt	不转移路径上的下一个分支指令表项
B_not_takenB_bb	不转移路径上的基本块表项
B_target	分支指令的目标地址
B_taken_btt	转移路径上的下一个分支指令表项
B_taken_bb	转移路径上的基本块表项
valid	有效位标志
others	其它标志

BTT 表记录分支指令地址和分支的目标地址. BTT 表由两个结构相同的二维表格组成, 分别对应同一个分支的两个

不同方向,以避免访问冲突,如表 2 所示.每个表项包含的主要内容有:分支指令地址、分支的下一条指令地址、分支目标地址、不转移路径上的下一个分支指令地址、不转移路径的基本块指针、转移路径上的下一条分支指令地址、转移路径上的基本块指针、控制指令类型以及有效标志等.根据 Trace 预构所设计的最大分支指令数,并考虑到对跳转指令所作的优化,每个表设计为能够容纳 4 个分支,这样,BTT 表共有 8 个表项,能容纳 8 条控制转移指令.BTT 表的内容主要供分支连接部件查询,用来生成指令执行树并预构 Trace.

分支指令匹配部件(BMU: Branch Matching Unit)实现指令块的扫描、匹配操作,并从指令块中提取分支指令及分支目标地址.BMU 的输出有两个:其一是基本块,其二是控制指令.BMU 由两路并行指令块过滤单元构成,分别地对两个指令块进行控制转移指令过滤,每一路一次并行过滤四条指令.如果四条指令中没有控制指令,则下一个时钟继续匹配后续指令;如果有控制指令,则提取程序顺序的首条控制指令的地址和指令中包含的分支目标地址信息(如果存在)并填入相应的 BTT 表;如果匹配到达指令块边界,则发出指令块读取请求.当匹配发现控制指令时,过滤单元除了填写当前 BTT 表项的未完成的域外,还将请求一个空闲的 BTT 表项,用于记录这个新的分支的有关信息.如果该控制指令是目标地址不能从指令中获得的第一类控制指令,我们的补救措施是访问分支目标缓冲区(Branch Target Buffer[3]),将返回的分支预测的方向或目标地址直接填入对应表项.分析 Cint95 程序的统计结果(见表 1),这一类指令的规律比较明显,保存的目标地址历史信息具有很高的准确度.当 BMU 过滤到这类指令时,如果 BTB 中没有相应的表项,则该 Trace 的预构终止,如果 BTB 命中,则以预测的目标地址作为预取地址.

BMU 的工作过程:当请求的指令块读出时,启动对控制转移指令的过滤,如果过滤发现新的分支指令时,计算分支的目标,若分支目标不在当前指令块缓冲区命中,则发出指令块请求.当某条分支指令及分支方向都已确定时,触发 Trace 预构操作.分支指令匹配的工作过程如图 2 所示.

分支目标连接部件(BTLU: Branch Target Linking Unit)实现指令流的构造.BTLU 将获得的分支目标信息和基本块信息按照程序顺序连接为路径,作为 Trace 构造的依据.

指令预取并不无限制地进行,而是受最后递交的分支与当前匹配块间的分支指令数和基本块缓冲区的容量限制.对每个基本块的第一次过滤操作以 BTLU 提供的信息开始,过滤的起点可能是指令块的任意位置,不一定是指令 Cache 块的起点.过滤的同时完成对基本块的填充.

3 Trace 预构的实现

分支目标提取得到有效的信息后,Trace 预构部件(TPU: Trace Preconstruction Unit)根据 BTT 表和基本块缓冲区中的内容构建 Trace.TPU 由基本块缓冲区(BBB: Basic Block Buffer),Trace 缓冲区,BTLU 及 Trace 生成部件等构成.TPU 和 BTP 是紧密相连的,结构如图 1 所示.BBB 存储分支指令之后跟随的指令基本块,以基本块为单位便于 Trace 的构造.Trace 缓冲区保存预构结束的 Trace.BTLU 根据内核执行的分支方向作废所有不会执行到的分支方向上的 Trace 预构,清除无用表项,并向正确路径上的分支发出新的 Trace 预构请求.Trace 生成部件完成 Trace 的填充.

Trace 预构的工作流程:每当 BTT 表的一个表项填充完成时,就确定了分支指令之后的两个基本块的信息.BTLU 向 Trace 生成部件发出预构请求,预构部件获得这些信息后,根据指令执行树开始或继续 Trace 的预构.预构部件通过 BTT 表的基本块指针索引 BBB 得到基本块并填充到 Trace 中.达到 Trace 边界的预构 Trace 存入 Trace 缓冲区.已完成预构的基本块将置为空闲.

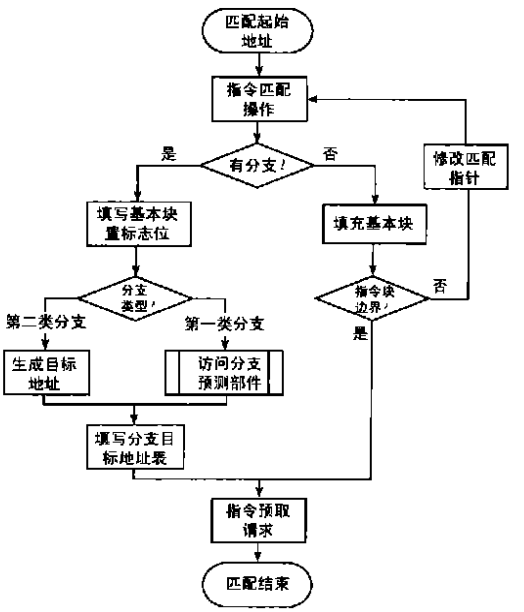

图 2 分支指令匹配流程图

4 模拟方法及结果分析

我们选择学术界广为接受的威斯康星大学的 SimpleScalar 高性能模拟器来验证基于 BTP 的 Trace 预构技术的有效性.该模拟器运行稳定,具有较好的可移植性,界面清晰,文档比较齐备,可免费从网上获取.SimpleScalar 定义了一种 SimpleScalar 体系结构,它类似于 MIPS ISAII 指令集,可通过模拟器自带的 GCC 交叉编译器生成执行代码.此外,工具套件中还包括了预编译的 Cint95 执行代码.模拟器以执行驱动方式运行,能准确记录每条指令在流水线上的执行过程,支持乱序执行和前瞻发射,以及最多两级 Cache 结构.

模拟器的配置:指令一级 Cache 容量 16kB,512 路,直接映射,块长 32B,延迟 1 个时钟;数据一级 Cache 容量 16kB,128 组,4 路组相联,块长 32B,延迟 1 个时钟;共用二级 Cache,容量 256kB,1024 组,4 路组相联,块长 64B,延迟 12 个时钟;主存容量不限,第一块延迟 36 个时钟,后续块间隔一个时钟;分支预测器是双态饱和计数器,容量 2K 项,512 组四路相联的 BTB 表,返回地址栈容量 16;4 条整数流水线,一条为乘/除法(延迟:乘法 3/除法 20),流水延迟 1 个时钟;4 条浮点流水线,一

条为乘/除法(延迟: 乘法 4/ 单精度除 12/ 双精度除 24), 流水延迟 1 个时钟; 存储端口, CPU 访问 2 个可读写, 分支目标提取部件预取指令端口 1 个.

BTP 的配置: 指令块缓冲区 4 块, 每块容量 64B; BTT 表记录最多 8 条分支信息; 2 路 BMU 并行匹配指令块, 每个时钟分别处理四条指令的匹配; BBB 的容量为 16 块, 每块长度 64B, 预构Trace 缓冲区 8 个, 缓冲区容量为 16 条指令, BTLU 和 Trace 生成部件各一个, 指令 Cache 预取请求队列长度为 8.

模拟执行了 Cint95 测试程序集, 以 Cint95 的 ref 目录下的数据作为测试程序的输入, 其中, cc1 的输入为 cccp. i; ijpeg 的输入为 penguin. ppm; go 的输入为 9 9 null. in; li 的输入为 ctak. lsp; perl 的输入为 test 目录下的 jumble. pl(ref 目录下的输入不够执行 101M 指令). 统计测试程序模拟执行的第 1M 到 101M 条指令作为结果进行对比. 表 3 列举了 6 个程序的统计结果. 从表中可以看出, 实现了 BTP 后, 指令预取部件访问一级指令 Cache 的命中次数显著增加了(由于实现了指令预取, 读取的指令数比实际递交的指令数要多) . 一级 Cache 访问不命中率显著下降, 从而减少了处理部件的指令等待延迟, 结合 Trace 预构机制, 程序的平均加速比提高了 12%.

表3　部份 SPECint95 程序的统计结果(第 1– 101M 指令)

程序	一级 Cache 命中次数		一级 Cache 不命中率		减少不命中率(%)	提高 IPC(%)
	基准	预构	基准	预构		
cc1	105403957	129813273	0 0555	0 0476	14. 2342	17. 0751
go	106706631	128502843	0 0430	0 0377	12. 3256	21. 8245
ijpeg	103205276	129958882	0 0002	0 0001	50. 0000	0. 0803
li	106478704	130765600	0 0082	0 0067	18. 2927	1. 3408
perl	101777845	130929254	0 0470	0 0371	21. 0638	11. 3896
vortex	101217839	129663099	0 0679	0 0562	17. 2312	20. 0870

5　总结及进一步的工作

对一级 Cache 的不命中率进行分析, 发现 BTP 虽然能够提高分支目标的准确度, 但对第一类分支仍然无法得到目标地址, 利用分支预测部件附带的 BTB, 对提高分支预测的准确率有明显效果. 如果考虑分支目标提取, 能够把分支预测的分支类型进行分类, 对目标能够提取的分支只记录分支的可能方向, 对目标不能从指令中得到的分支, 记录分支的目标信息, 则能够提高分支预测的针对性, 并降低分支预测的实现代价.

此外, 由于 Trace 预构访问了一级 Cache, 虽然对一级 Cache 访问不命中时采用了指令块缓冲区来保存预取指令的策略, 但仍有一部分指令会进入一级指令 Cache, 如果这些指令在后续的处理中不会执行, 则将导致一级 Cache 的污染, 并可能导致有用指令被无用指令清除, 因此, 对 Trace 预构的指令预取策略, 可考虑实现一级 Cache 的刺探, 二级 Cache 的更新策略, 减少有用指令与无用指令竞争 Cache 的问题.

参考文献:

[1] Burger D, Austin T M. The simplescalar tool set, Version 2. 0 [R]. Technical Report CS-TR-97-1342, Computer Sciences Department, University of Wisconsin-Madison, 1997.

[2] Jacobson Q, Smith J E. Trace preconstruction [A]. Proc. of 27th Inter. Symp. on Computer Arch [C]. Vancouver, Canada, 2000: 37– 46.

[3] Lee J K L, Smith A J. Branch prediction strategies and branch target buffer design [J]. IEEE Computer, 1984, 21(7) : 6– 22.

[4] McFarling S. Combining branch predictors [R]. Digital WRL Technical Note TN-36, June 1993.

[5] Rotenberg E, Bennett S, Smith J E. Trace cache: a low latency approach to high bandwidth instruction fetching [A]. Proc. of 29th Inter. Symp. on Microarchitecture [C]. Paris, France, 1996: 24– 35.

作者简介:

杜贵然　1969 年生于四川泸县, 博士. 主要研究领域为先进微处理器, 高性能计算机体系结构. E-mail: duwang@ nudt. edu. cn

窦　勇　1966 年生于吉林市, 博士. 副研究员, 主要研究领域为并行处理, 计算机体系结构.

徐　明　1964 年生, 博士, 教授, 主要研究领域为高性能计算机体系结构, 并行与移动数据库.

周兴铭　1938 年生, 教授, 博士生导师, 中国科学院院士, 主要研究领域为高性能计算机体系结构, 并行与移动数据库, CSCW.