

高性能代价比的两层关联间接转移预测器设计

袁楠^{1),2)} 范东睿¹⁾

¹⁾(中国科学院计算技术研究所系统结构重点实验室 北京 100190)

²⁾(中国科学院研究生院 北京 100049)

摘 要 随着面向对象语言程序、动态链接库(DLL)等的普遍应用,间接转移指令的使用越来越频繁.两层关联间接转移预测器预测准确度高,但实现硬件代价较高,因此并不实用.文中深入分析了两层关联间接转移预测器中产生误预测的原因,通过改进索引方法、压缩存储等实用方法减小硬件实现代价.实验结果表明,通过这些方法的改进,在133K比特硬件存储代价下,使用一组SPEC CPU2000测试程序进行评估,间接转移误预测率为9.6%,仅比两层关联预测器理想误预测率高2.3%,而4路组相联BTB预测器的误预测率为31%.

关键词 间接转移;两层预测器;误预测

中图法分类号 TP302 DOI号: 10.3724/SP.J.1016.2008.01898

Design of Cost-Effective 2-Level Correlation Indirect-Branch Predictor

YUAN Nan^{1),2)} FAN Dong-Rui¹⁾

¹⁾(Key Laboratory of Computer System and Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

²⁾(Graduate University of Chinese Academy of Sciences, Beijing 100049)

Abstract Indirect branches occur frequently in Object-Oriented Language (OOL), and in Dynamically-Linked Libraries (DLL), two programming environments rapidly increasing in popularity. 2-level correlation indirect-branch predictor tracks branch behavior well but costs unaccepted hardware budget. Through analyzing principle of mis-prediction in 2-level correlation indirect-branch predictor, the authors propose several cost-effective methods like improving index function and storage compressing to reduce hardware cost of 2-level correlation indirect-branch predictor. With 133K-bit hardware storage budget, for a set of benchmarks comes from SPEC CPU2000, miss-prediction rate drops from 31% on 4-way set-associative BTB to 9.6% on proposed predictor, which is 2.3% larger than the optimal mis-prediction rate.

Keywords indirect branch; 2-level correlation predictor; mispredict

1 引言

在宽发射、深度流水的超标量处理器体系结构中,准确的转移指令预测技术能够为处理器提供连续的指令流输入,弥补了转移指令转移方向和转移目标地址尚未计算出来时,处理器无法继续执行而

造成的处理器周期的浪费.因此,转移预测技术是挖掘指令级并行性(ILP)的重要手段.但是,一旦发生转移指令误预测,处理器预执行的错误路径上的几十条甚至上百条指令将被丢弃,所有猜测执行的工作将被取消,造成处理器时钟周期的浪费.因此,提高转移指令预测准确度是转移指令预测器设计和应用的关键目标.

转移指令按照是否存在转移条件以及转移目标地址是否在指令中指明分为 4 类:无条件直接转移指令、条件直接转移指令、无条件间接转移指令和条件间接转移指令。根据实际需求,在多数体系结构的指令集上,仅存在前 3 种转移指令,例如 x86 指令集。本文所说的间接转移指令,均指无条件的间接转移指令,而条件转移指令泛指条件直接转移指令(无条件直接转移指令不需要预测)。

间接转移指令在程序中出现的频率相对于直接转移指令出现的频率较低。间接转移指令大量应用于函数指针,或者 Switch-Case 型语句的跳转表^[1]。随着面向对象语言程序、动态链接库(DLL)以及虚拟机(virtual machine)等技术的普遍应用,程序中运行时时刻决定动态地确定转移目标地址的特性越来越显著,间接转移指令的使用将越来越频繁。因此,间接转移指令的预测准确度对处理器性能的影响将越来越大^[2]。

相比于条件转移指令,间接转移指令预测难度更大。首先,预测多比特表示的目标地址(通常为 32 位,64 位)比预测单比特表示的转移方向(如“1”表示转移,“0”表示不转移)通常需要更多的信息,使用更复杂的硬件。其次,不同的条件转移指令间可能具有很高的转移方向相同的概率(同为“转移”或“不转移”),将转移方向趋向性相同的转移指令归类并利用这一特点预测,可大量抵消别名问题带来的误预测次数,提高转移方向预测命中率^[3-6];由于不同的间接转移指令转移目标地址几乎不同,预测间接转移指令时难以利用趋向性的特点。

两层关联转移预测器^[7]利用转移指令与转移历史行为之间的关联性预测条件转移指令的转移方向,其原理也同样适用于间接转移指令预测^[2,8-10]。相对于传统的 BTB(转移目标地址缓存)预测器^[11-12],两层转移预测器可以大大提高间接转移指令的预测准确度。可是,为了达到较高的预测准确度,两层间接转移历史预测器通常需要较复杂的逻辑电路、较大的存储空间,成为制约两层关联转移预测器在商用处理器上实用化的重要原因。

本文在两层关联转移预测器的基础上研究预测准确度高、硬件代价小的间接转移预测方法。第 2 节介绍国际上间接转移指令预测器的研究进展,这些预测器大都以提高预测命中率为主要目的,与实用仍有一定差距;第 3 节介绍本文实验中使用的测试程序和间接转移预测器的模拟环境和方法;第 4 节分析两层关联间接转移预测器产生误预测的原因,

并研究预测表索引的 Hash 构造算法、压缩转移目标地址的方法;第 5 节对全文进行总结。

2 相关工作

参考文献[11]介绍了 BTB 预测器及多种结构组织和配置对性能的影响。最基本的 BTB 预测器存储每条转移指令最近一次的转移目标地址,并使用该地址作为对应转移指令下一次的预测值。一旦 BTB 预测器误预测时,存储的转移目标地址被替换为新的转移目标地址。BTB 利用了间接转移指令转移行为的局部性,即一条间接转移指令一段时间内仅向一个转移目标地址发生转移。参考文献[12]对 BTB 预测器进行了改进,为了适应 C++ 程序中虚函数调用特性,为每项添加 2 比特状态机位使预测目标地址在两次连续的误预测后才被替换。

参考文献[13]针对一种特殊的间接转移指令——函数调用返回指令,根据函数调用和返回转移指令成对的程序行为提出返回地址栈(RAS)预测函数返回指令的目标地址。

参考文献[9]探索了多种结构的两层关联转移预测器对于间接转移指令预测准确度。用于间接转移预测的两层关联预测器使用转移目标地址作为转移历史的路径,代替了两层条件转移预测器中使用转移方向作为转移历史。实验表明,理想预测器情况下,GA_p 预测器相比 PA_p 预测器具有更好的预测性能。参考文献[2]通过改变目标地址更新方法和时机、PHT 表的大小和相联度、历史寄存器的长度,研究了实际各种有限存储空间配置下的两层间接转移预测器的性能。研究中注意到了不同转移指令与不同长度的转移历史的关联性,并建议使用两个不同长度(例如一长一短)历史寄存器混合预测,以在相同的硬件代价下具有更高的预测准确度。

参考文献[8]提出了类似于两层关联转移预测器的结构的目标地址缓存(Target Cache)预测器,其显著的特征就是历史寄存器不仅记录间接转移指令的转移历史,还记录其它转移指令的转移历史,如条件转移指令、函数调用/返回指令等。实验证明了间接转移指令的转移行为与不同类型转移指令之间的关联。

参考文献[10]为了提高硬件利用率,将目标地址唯一的间接转移指令或者目标地址变化不频繁的间接转移指令,与其余较难预测的间接转移指令通过过滤器区分开,并分别使用不同的预测器进行预

测. 预测器混合使用了 GAp^[8] 和双路径历史预测器^[9], 并使用了过滤器区分不同转移行为类型的转移指令.

参考文献[14]利用数据压缩领域中的预测方法, 提出了能够自适应部分匹配历史路径的 Markov 预测器, 使不同的间接转移指令能够动态地选择不同长度的转移历史相关联, 提高预测准确度.

参考文献[15]认为混合预测器中各预测器的预测结果的静态优先级关系没有充分利用混合预测器的潜力, 因此增加选择逻辑能够动态地选择预测器的输出结果.

3 实验方法

本文使用 SPEC CPU2000 中的一组定点测试程序 REF 规模输入集下(见表 1)在 32 位地址空间的 x86 处理器上执行约 500 亿条指令^①产生的动态间接

转移指令序列驱动转移预测模拟器作为实验手段.

为了简化模拟实验过程, 预测模拟器仅使用间接转移作为输入, 直接转移指令不参与间接转移预测. 其次, x86 体系结构中, 间接转移指令可能会引起段间的转移, 造成段控制寄存器的切换, 处理器整个流水线将被清空, 预测失去意义, 因此, 预测模拟器仅模拟预测段内的间接转移行为.

我们实现了全相联的 BTB 预测模拟器. 模拟表明, 即使 BTB 项数增至无限多时, 本文选择的这组 SPEC CPU2000 定点测试程序仍然具有较高的间接转移误预测率(见表 1). 显然, BTB 预测器不能很好地捕捉这些测试程序中间接转移指令的转移行为. 对于 SPEC CPU2000 中的其他定点测试程序, 使用 BTB 预测器已经能够达到 5% 以下的平均误预测率, 间接转移目标的局部性较好. 后文中, 我们使用表 1 中这几个间接转移行为复杂的程序及测试输入集, 检验间接转移预测器的性能.

表 1 SPEC CPU2000 定点测试程序

测试程序	程序输入	编程语言	间接转移指令	间接转移目标地址	间接转移次数	平均转移间隔	90% 转移	95% 转移	BTB 误预测率/%
gcc	200.i	C	392	1540	38918186	775	55	74	62.14
	scilab.i		362	1391	164370046	304	52	68	60.31
	166.i		373	1281	157962060	294	51	69	60.00
gap	ref.in	C	224	324	421268950	118	13	15	43.98
vortex	Lendian1.raw	C	163	361	33167460	1506	7	9	8.38
	Lendian2.raw		163	361	75520978	661	7	9	27.40
	Lendian3.raw		163	360	31392696	1591	6	8	6.20
eon	cook	C++	177	415	72376631	690	16	19	22.95
	rushmeier		179	420	72849514	686	15	18	27.22
	kajiya		183	418	69115126	723	17	21	25.01
crafty	crafty.in	C	136	317	88213544	566	5	6	53.03

表 1 中, “间接转移指令”表示程序在运行时静态转移指令的个数, 同一测试程序在不同输入的情况下程序行为不一定相同. “间接转移目标地址”表示程序执行过程中转移目标地址的个数. “间接转移次数”表示了程序执行过程中发生间接转移的次数. “平均转移间隔”表示程序执行时相邻两条间接转移指令的平均 x86 指令间隔, 主流 x86 体系结构处理器所能容纳的同时活动指令数一般在 100~200 条微指令之间, 从表中可以看到, 间接转移指令平均执行间隔通常大于 200, 因此利用序列驱动的转移预测模拟器模拟方法虽然没有模拟精确的超标量处理器结构行为, 但模拟准确度误差不会很大. “90% 转移”和“95% 转移”表示间接转移指令序列中, 分别有多少条转移次数最多的转移指令构成了 90%, 95% 的间接转移次数. 可以看到, 少数的转移指令构成了转移次数主要部分. “BTB 误预测率”表示使用无限

多项的全相联 BTB 预测器时, 对应测试程序和输入的误预测率.

4 实验结果及分析

根据转移指令转移行为与全局或者局部历史转移行为的关联性, 参考文献[16]列举并分析了各种配置下两层关联转移预测器结构, 应用于条件转移指令转移方向的预测. 在基本的两层间接转移预测器中, 全局/局部转移历史寄存器顺序存储转移目标地址的历史路径信息, 转移目标地址表存储转移目标地址, 如图 1 所示. 预测时, 转移指令地址和全局/局部转移历史寄存器内容经过 Hash 操作(通常为异或)后的值索引转移目标地址表, 产生预测结果,

① gcc 输入集为 200.i 在约 300 亿条指令时程序运行完毕.

并使用预测目标地址更新转移历史寄存器(假设预测结果准确). 因此预测正确时,不做任何更新. 预测错误时,则需要恢复预测错误前的转移历史寄存器内容,并更新转移目标地址表中对应表项为正确的转移目标地址.

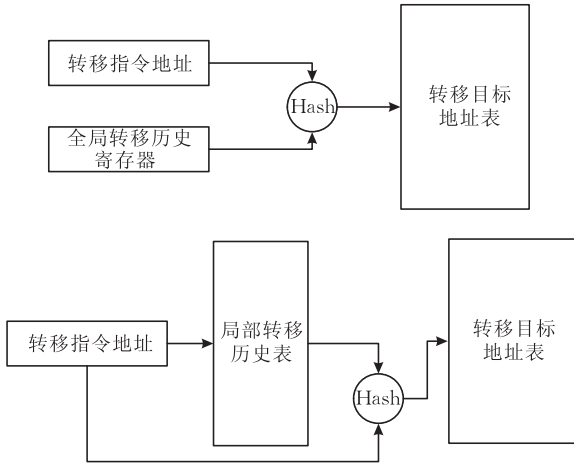


图 1 全局历史的间接转移两层关联转移预测器 GAp(上)与局部历史的间接转移两层关联转移预测器 PAp(下)

本节首先分析了两层关联间接转移预测器误预测的原因,并模拟无限硬件资源得到上述两层间接转移预测器理想的预测能力. 接着,在实际情况下,预测器在有限的硬件资源代价的约束下误预测率上升,我们提出若干合理利用硬件资源提高预测准确度的改进方法,并分析了这些方法和理想预测能力的差别.

4.1 两层间接转移预测器的理想误预测率

两层关联间接转移预测器中产生误预测的原因可分为 3 种:

(1) 预测器初始时引起的误预测(启动误预测). 某条转移指令前几次转移,由于在预测器看来还没有形成有规律的、稳定的转移历史模式,引起误预测. 启动误预测次数是转移指令和转移历史路径相结合的二元组的个数. 直观上,预测器转移历史路径长度越长,启动误预测越明显.

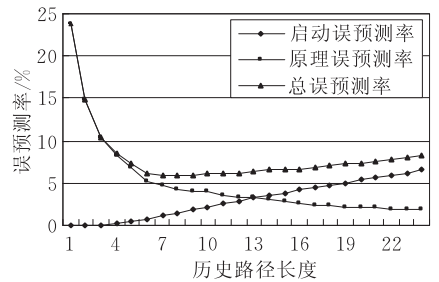
(2) 预测器预测行为不符合转移指令转移规律引起的误预测(原理误预测). 根据两层关联转移预测器的原理,一个转移指令在相同的转移目标历史模式下应转移到同一个目标地址,反之预测器不能正确地捕捉转移行为,发生误预测.

(3) 预测器输入和存储的信息不充分引起的误预测(别名误预测). 实际预测器的构造中,为了减小

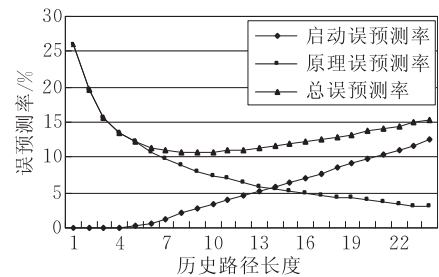
硬件代价,仅使用每个转移目标地址值的少数比特作为历史信息的表示,并将转移指令地址和历史信息经过某种 Hash 运算(如异或)后得到转移目标地址表的索引值. 转移指令地址或者历史信息本来不相干的预测经过这种缩减了的信息表达方式和 Hash 运算之后可能得到相同的转移目标地址表索引值,引起相互干扰产生误预测.

对于一个有固定历史路径长度的两层预测器,启动误预测和原理误预测是预测器本质上固有的,不能够消除. 别名误预测情况可以通过记录更完整的历史信息、使用更好的 Hash 算法以及使用足够的转移地址目标表项来改善.

我们分别实现了不同历史路径长度的理想的 GAp 和 PAp 预测器模拟器,使用无限的硬件资源保证仅出现前两种误预测情况,如图 2 所示.



(a) 基于全局历史的两层预测器(GAp) 误预测率(理想预测器)



(b) 基于局部历史的两层预测器(PAp) 误预测率(理想预测器)

图 2 理想两层预测器的误预测率

由图 2 看到,启动误预测率曲线在较短历史路径长度下相对平坦,说明与转移指令关联的转移目标的个数并不多(此处并不代表与转移指令关联的转移目标不多,因为转移指令可能与一个转移目标的多次转移有关联),造成历史路径相同的概率较大. 之后启动误预测率曲线随着预测器历史路径长度的增加而近似线性地增长,说明间接转移与转移目标历史的关联性较好,否则将会造成超线性的增长. 给定转移历史路径长度,理想预测器的启动误预测次数的一个重要意义就是能大体反映出至少需要

多少转移目标地址表项,以避免存储空间不足引起的别名误预测.原理误预测率体现了转移与转移目标历史的关联性.由原理误预测率曲线观察到,多数转移与 1~10 次历史转移目标地址相关联,之后误预测率减小的趋势越来越平缓,说明与转移历史关联度较深的转移越来越少.综合两者,两层关联间接转移预测器在历史长度为 7~9 时达到最佳的预测率.

当历史长度较短时,同样历史长度的 PAp 预测器相对于 GAp 预测器更准确.当历史长度逐渐增大时,GAp 预测器误预测率更小,说明转移指令的转移行为与全局转移目标历史的关联性更强.实际上,一部分间接转移指令活跃时没有其它间接转移指令发生转移,即在一段时间内该转移指令的局部转移历史就是全局转移历史,这种情况下 GAp 与 PAp 预测能力是相同的.因此,GAp 预测器有一定预测转移行为与局部转移路径历史关联性更强的转移指令的能力,反之 PAp 预测器并不具备 GAp 预测器的预测能力.

4.2 索引的 Hash 构造方法

构造实际预测器时,为了减小硬件代价,仅选取每个转移目标地址值的部分比特作为历史信息的表达(为了方便,后文称根据转移目标地址转换后的历史信息的表示为“路径表示”).传统方法使用转移目标地址低 n 位作为路径表示.因此,不同的转移目标地址只要低 n 位相同就具有相同的路径表示,造成预测间的相互干扰,产生别名误预测.下面我们通过研究路径表示产生别名的概率,探索减小别名误预测的方法.

定理 1. 两个转移的目标地址的 n bits 路径表示相同的概率最小为 $\frac{1}{2^n}$,且与路径表示的方法无关.

证明. n bits 表示的值一共有 2^n 种取值.假设 n bits 值的 2^n 种取值出现的概率为 $P_0, P_1, P_2, \dots, P_{2^n-2}, P_{2^n-1}$, 有 $P_0 + P_1 + P_2 + \dots + P_{2^n-2} + P_{2^n-1} = 1$. 那么,两次转移目标地址的 n 比特路径表示相同的概率为

$$P_{\text{alias}} = P_0 \times P_0 + P_1 \times P_1 + \dots + P_{2^n-2} \times P_{2^n-2} + P_{2^n-1} \times P_{2^n-1} \quad (1)$$

设数组 $A_i = \frac{1}{2^n} - P_i$ ($0 \leq i \leq 2^n - 1$), 则 A_i 为第 i 种取值概率与平均取值概率 $\frac{1}{2^n}$ 的差,显然 $A_0 + A_1 + A_2 + \dots + A_{2^n-2} + A_{2^n-1} = 0$. 代入式(1),有

$$\begin{aligned} P_{\text{alias}} &= P_0 \times P_0 + P_1 \times P_1 + \dots + \\ &P_{2^n-2} \times P_{2^n-2} + P_{2^n-1} \times P_{2^n-1} \\ &= \left(\frac{1}{2^n} - A_0\right)^2 + \left(\frac{1}{2^n} - A_1\right)^2 + \dots + \\ &\left(\frac{1}{2^n} - A_{2^n-2}\right)^2 + \left(\frac{1}{2^n} - A_{2^n-1}\right)^2 \\ &= \frac{1}{2^n} + (A_0^2 + A_1^2 + \dots + A_{2^n-2}^2 + A_{2^n-1}^2) \quad (2) \end{aligned}$$

因此,当 $A_0 = A_1 = \dots = A_{2^n-2} = A_{2^n-1} = 0$, 即 $P_0 = P_1 = \dots = P_{2^n-2} = P_{2^n-1} = \frac{1}{2^n}$ 时,两次转移目标地址的 n bits 路径表示相同的概率最小为 $\frac{1}{2^n}$. 可以看到,最小概率与路径表示中每比特取值概率有关,与如何选取路径表示的方法无关.得证. 证毕.

推论 1. 2^n 种取值概率的方差 $S^2 = \frac{1}{2^n} (A_0^2 + A_1^2 + \dots + A_{2^n-2}^2 + A_{2^n-1}^2)$, 根据式(2)可知,路径表示取值概率方差越小,即越趋近于平均分布,两次路径表示别名概率越小.

推论 2. 两个历史路径长度为 h 的历史路径串含 h 个 n bits 的路径表示,根据定理 1 别名的概率最小为 $\frac{1}{2^{h \times n}}$. 根据推论 1, 整个历史路径比特串取值概率趋近于平均分布时,历史路径比特串别名的概率最小. 固定比特长度的历史路径串下,无论如何变化路径表示的比特长度和历史长度,别名产生的最小概率是一定的.

根据定理 1 和推论 1, 2, 减小别名出现的概率有两个方法: (1) 由于有限比特宽度的路径表示产生别名的最小概率一定, 因此要求路径表示的比特数尽量多, 使得产生别名的最小概率尽量小; (2) 要求转移目标地址的路径表示取值的概率趋近于平均分布, 或者使整个历史比特串取值概率趋近于平均分布. 使用第一种方法, 需要增加路径表示的长度, 虽然可能减小产生别名的概率, 但需要增加更多的硬件资源. 使用第二种方法, 则需要改进路径表示方法和历史比特串表示方法, 使得它们的取值符合平均分布.

常用的 GAp 或者 PAp 预测器的 Hash 函数将转移指令地址值与转移历史寄存器值拼接(G-Select)产生转移目标地址表的索引. 参考文献[17]提出了 G-Share 算法, 将转移指令地址值和转移历史寄存器值异或产生转移目标地址表的索引, 并指出在同等硬件代价下, 应用 G-Share Hash 算法的预测器预测准确度比 G-Select Hash 算法高, 但没有分析

原因. 当转移指令个数有限尤其是少数转移指令发生大量转移的情况下, 使用 G-Select 算法拼接后的 Hash 结果中由转移指令地址组成的部分取值不均匀, 而将转移指令地址与历史信息异或之后能一定地改善索引取值分布的均匀性, 降低别名误预测率. 定理 1 以及推论 1,2 启示我们从 Hash 函数这个层次看待问题, 传统的使用低 n 位作为路径表示的方法, 和使用多年的 G-Share 算法也可能有改进余地. 那么, 有选择地引入更多的比特信息进行 Hash 运算, 是否更有利于索引取值概率的平均分布?

在几乎不增加硬件代价的情况下, 我们提出“路径折叠”和“历史折叠”两种 Hash 方法, 使得有限比特的路径或历史表示取值更加趋近平均分布.

路径折叠方法. 即将转移目标地址值的低 $2n$ bits 长度的串 AB (A 和 B 的长度均是 n bits) 的 A 和 B 两部分异或形成 n bits 长度的串 C 作为路径表示, 以替代原仅使用低 n bits 长度的串 B 进行路径表示的方法. 如转移目标地址为 $0xf0001234$, 取 $n=4$, 则比特串 A 为 $0x3$, 比特串 B 为 $0x4$, 路径折叠后的路径表示为 $0x7$. 当转移目标地址的末 n bits 取值概率分布不是很平均时, 路径折叠方法让转移目标地址中更多的比特位参与了运算, 重新分布了 n bits 路径表示的取值概率, 这样更有利于趋向于平均分布.

历史折叠方法. 即使用转移目标地址的低 $2n$ bits 表示路径, 在使用 Hash 函数计算目标地址存储单元索引值时, 将长度为 $2n \times h$ (h 表示历史路径长度) bits 的整个历史路径串等分为长度为 $n \times h$ bits 的两个串进行异或操作得到长度为 $n \times h$ bits 的串. 当转移目标地址本身分布不均匀时, 如某个转移目标地址大量出现, 历史折叠方法将一次转移的路径表示和另外一次转移的路径表示进行异或, 使得历史路径串取值概率分布更为平均. 因此, 历史折叠方法在多数情况下比路径折叠方法具有更小的取值别名概率的潜力.

图 3 展示了 GAp 预测器在不同路径表示方法下的误预测率实验结果. 仅使用转移目标地址的低 4 位表示路径, 并再采用 G-Share 算法产生索引直接相联的转移目标地址存储区的 GAp 预测器, 与完整的转移目标地址表示路径的理想 GAp 预测器相比有 4%~5% 的误预测率差. 而后两种 GAp 预测器在几乎不增加硬件代价的情况下 (仅增加了异或逻辑以及历史路径寄存器宽度, 存储空间大小完全相同), 选用目标地址的低 8 位表示路径, 使用历史

折叠或者路径折叠方法将历史路径比特串长度降至一半, 然后与转移指令地址异或 (G-Share 算法) 产生索引. 历史路径长度大于 3 时, 采用折叠方法的 GAp 预测器的误预测率已经非常接近理想值, 说明别名误预测通过折叠的 Hash 算法几乎得到消除. 图 3 中还可以观察到历史折叠方法的误预测率曲线有小幅度的“跳跃现象”: 当历史长度为奇数时, 历史折叠方法优于路径折叠方法, 而在历史长度为偶数时, 历史折叠方法的误预测率有一个微小上升, 差于路径折叠方法. 这个现象出现的原因是历史路径长度为偶数时进行历史折叠会导致另一种别名的出现, 例如转移指令的转移周期为 4, 转移目标序列为 $A, B, C, D, A, B, C, D, \dots$ 当长度为偶数 4 的转移历史路径为 $ABCD$ 和 $CDAB$ 时, 由于进行历史折叠后产生的索引值是一样的, 因而会产生相同的预测. 但显然, 这里应该预测出不同的转移目标: A 和 C . 奇数长度的历史路径的预测器使用历史折叠则不存在这一问题.

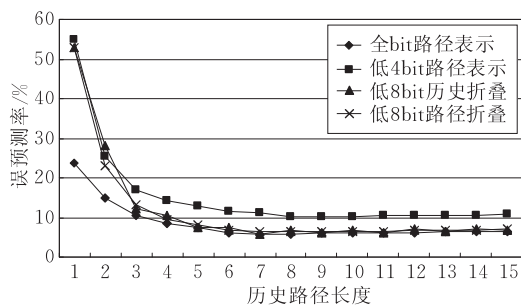


图 3 GAp 预测器历史在不同的路径表示方法下的误预测率

当然, 还有多种有效的 Hash 算法使索引取值概率趋近于平均分布, 例如多次折叠, 这里仅分析了简单的两种.

4.3 预测器存储空间压缩

间接转移指令预测器一般根据某种算法选择一个被存储着的历史上出现过的转移目标地址作为预测目标地址. 转移指令目标地址一般为 32 位或 64 位, 因此间接转移预测器需要使用大量的存储空间存储多个转移指令目标地址. 例如, Pentium 4 处理器中使用了 4K 项的 BTB 预测器, 仅存储转移目标地址就需要 $4K \times 32\text{bits} = 128K\text{bits}$ 的存储空间. 存储空间大会带来访问延迟、面积、功耗大等诸多问题, 本节我们讨论通过简单、实用的压缩减小预测器存储空间的方法.

4.3.1 利用指令的空间局部性压缩转移目标地址

不论是什么类型的转移指令预测器, 转移指令

地址都是预测器的输入,转移目标地址是预测器的输出.我们可以充分利用转移指令地址和目标地址之间的关系,并在预测器中使用更少的比特存储这种“关系”,而预测目标地址能够由转移指令地址以及存储的这种“关系”生成,达到压缩预测器存储空间的目的.很自然地,转移指令地址和转移目标地址之间的距离是一种简便、直观的关系表达方法.图 4 展示了本文所选定的测试程序和 Linux 内核代码中

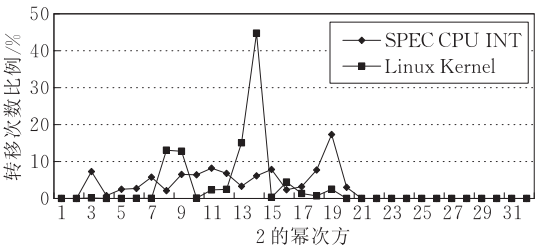


图 4 转移指令地址和转移目标地址之间的距离关系
(横坐标为两个地址所不同的最高比特位置)

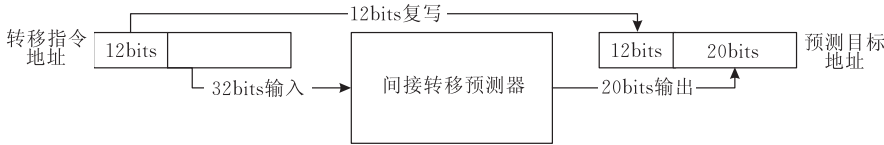


图 5 简化后的间接转移预测器通用结构

需要指出的是,应用程序通过间接转移指令调用动态链接库函数或者操作系统内核函数时,转移指令和转移目标地址的距离则会相对较远.因此,利用转移指令和转移目标的空间关系来压缩存储的方法并不适用,改进将作为下一步的工作.

4.3.2 增加转移目标地址索引表

传统两层关联转移预测器中,历史路径比特串长度与存储目标地址的存储空间大小是 2 的幂次方关系.如使用 4bits 表示转移目标路径,历史长度取值为 5 时,为了满足历史路径串到存储空间地址的直接映射关系,就需要 $2^{4 \times 5} = 1\text{M}$ 项转移目标地址的存储空间,这在处理器设计实现时是不可接受的.使用相关度更高的缓存区^[2]时,也会增加额外的 TAG 存储空间,造成了一定浪费.

前文已提出,理想预测器情况下的启动误预测数的一个现实意义就是表明了给定历史路径长度下两层关联转移预测器需要多大的目标地址存储空间才能容纳所有的转移目标地址.根据图 2 中测试程序的启动误预测数,最优历史路径长度下 10000 项左右大小的转移目标地址存储空间就能达到很好的预测率.因此,我们选择长度为 14bits 的转移目标地址表索引值,按照每个目标地址 20 比特存储空间

间接转移指令地址和转移目标地址之间的不同的距离占总间接转移次数的比例.可以看到,所有间接转移指令地址和目标地址的高 12 位完全一样,即所有的转移指令地址和目标地址之间的距离都在 1MB 以内.从程序行为上来看,例如 Switch-Case 型语句中,转移指令与各分支紧邻,转移指令与转移目标之间的距离不长.

基于以上分析,预测器仅产生预测目标地址的低 20 位,而高 12 位直接由转移指令地址的高 12 位复写得到,如图 5 所示.这样,转移预测器能够使用更少的存储空间.如在 BTB 预测器中,使用存储目标地址的低 20 位代替存储整个 32 位目标地址,大大降低了存储空间大小.我们相信,64 位地址空间的程序这个现象将更加明显,并在构建预测器时能够节省更多的存储空间.当转移指令地址和转移目标地址的距离超过 1MB 时,处理器最终识别为转移预测错误并清空流水线上错误路径指令.

(见第 4.3.1 节)计算,共需要 $2^{14} \times 20\text{bit} = 320\text{Kbits}$ 的存储空间,这个存储空间仍然相当大.由表 1 可以看到,测试程序集中每个程序转移目标地址仅有 300~420 项左右(gcc 是个例外,有 1500 项),可以想象,一个时间段内活跃的转移目标地址还要小于等于这个数.也就是说,转移目标地址表其实只需要几百个表项就够了,构造 2^{14} 项的转移目标地址表是非常浪费的.这里,我们使用了两级索引结构的预测器结构,在转移目标地址表 Y 前增加了转移目标地址索引表 X,用于产生表 Y 的索引值,如图 6 所示.这样,虽然单个表项较窄的转移目标地址索引表 X 表项较多,由于单个表项较宽的转移目标地址值表 Y 表项数少,从整体上达到节省预测器存储空间的目标.由于采用了两级索引机制,导致预测器电路产生预测结果延迟增长,在高主频的超标量处理器中,可能需要多拍完成.索引表和目标地址表可采用简单的直接映射方式降低延迟. Alpha EV8 处理器虽然极端地追求指令级并行性,但它的锦标赛预测器仍然采用了多级索引和仲裁结构,需要多拍输出预测结果.随着超标量处理器前端流水级划分越来越细、越来越多(如 Pentium 4 的前端流水线为 6 级),多拍输出的转移预测器也能很好地贴合前端流水线.

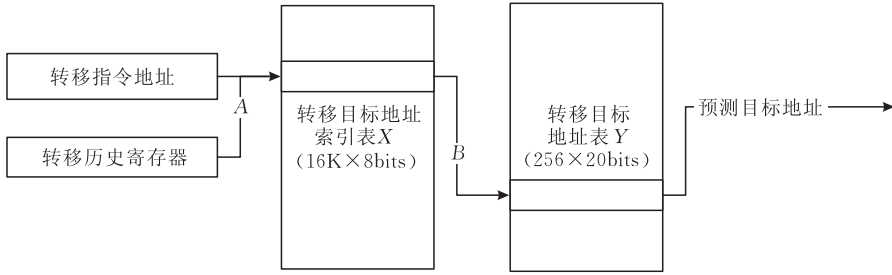
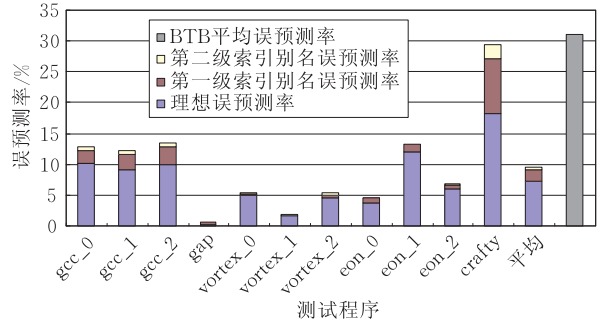


图 6 带索引表的 GAp 预测器的构造

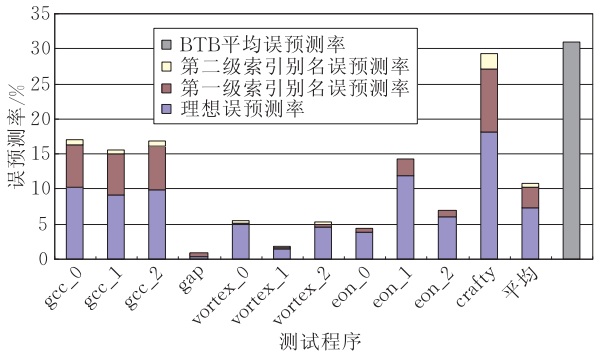
根据以上分析,我们结合改善的索引算法和压缩方法,在两层关联间接转移预测器的基础上,提出一个新的间接转移预测器:转移目标的路径表示由 6bits 构成,历史路径长度为 5;这样共 30bits 的历史路径串按照路径折叠或者历史折叠方法得到 15bits 的值;将该值异或转移指令地址后,取低 14 位比特值 A 按照直接映射方法索引转移目标地址索引表 X ;由 X 输出的 8 比特值 B 按照直接映射方法索引转移目标地址表 Y 得到 20bits 目标地址低位,拼接转移指令地址的高 12 位产生 32bits 预测目标地址。预测器预测正确时,情况同基本的两层预测器。转移预测错误时,除更新转移路径历史寄存器,还要将正确转移目标地址的低 16 位等分异或后产生新的 8 位索引值 B' ,根据发生预测时的索引值 A' 更新转移目标地址索引表 X 对应项为新值 B' ,并根据索引值 B' 更新转移目标地址表对应项为转移目标地址的低 20 位比特值。

该预测器需要 $2^{14} \times 8\text{bits} + 256 \times 20\text{bits} = 133\text{Kbits}$ 的存储空间,与 Pentium 4 处理器 BTB 预测器转移目标地址存储空间基本相当。最后,我们在实验中分析了我们提出的预测器的误预测率原因,并与几乎相同硬件代价下的 4 路组相联的 BTB 预测器性能相比较。

图 7 为上述预测器的模拟实验结果。可以看到,使用历史折叠方式的 GAp 预测器第一级索引别名平均增加了 1.8% 的误预测率,路径折叠方式的 GAp 预测器第一级索引别名平均增加了 2.9% 的误预测率,而第二级索引别名使两个预测器的误预测率仅上升了 0.4%~0.5%。上述 GAp 预测器硬件代价小、实现简单,历史折叠方法下平均误预测率仅为 9.6%,与理想预测器下 7.3% 的误预测率接近。经过对比实验,上述预测器的误预测率远远优于相同硬件代价下传统的 4 路组相联 BTB 预测器(4K 项,每项 32 位)上 31% 的平均误预测率。



(a) GAp 预测器误预测率(历史折叠)与 BTB 预测器误预测率比较



(b) GAp 预测器误预测率(路径折叠)与 BTB 预测器误预测率比较

图 7 GAp 预测器误预测率及组成

5 结 论

本文分析了两层关联间接转移预测器产生误预测的原因:启动误预测、原理误预测和别名误预测,在此基础上消除别名误预测得到了理想两层关联转移预测器的预测能力。本文随后提出改进索引的 Hash 构造方法、压缩存储空间代价两类方法,在有限硬件代价下尽可能消除别名误预测,保证两层关联间接转移预测器预测准确度接近理想预测准确度的同时,有效地降低硬件实现代价,从而使两层关联间接转移预测器更为实用。实验结果表明,通过以上方法的改进,约 133K 比特硬件存储代价下,一组 SPEC CPU2000 测试程序的间接转移误预测率由

4 路组相联 BTB 预测器上的 31%, 降低至 9.6%, 仅比两层关联转移预测器理想误预测率高 2.3%。所采用的两类方法也具有很强的通用性, 可适用于各种不同的预测器。

参 考 文 献

- [1] Uh Gang-Ryung. Effectively exploiting indirect jumps[Ph. D. dissertation]. Florida State University, Tallahassee, Florida, 1997
- [2] Driesen K, Holzle U. Accurate indirect branch prediction//Proceedings of the International Symposium on Computer Architecture. Barcelona, Spain, 1998: 167-178
- [3] Sprangle E, Chappell R S, Alsup M, Patt Y N. The agree predictor: A mechanism for reducing negative branch history interference//Proceedings of the 24th International Symposium on Computer Architecture. Denver, Colorado, USA, 1997: 284-291
- [4] Lee Chih-Chieh, Chen I-C K, Mudge T N. The bi-mode branch predictor//Proceedings of the 30th International Symposium on Microarchitecture. North Cardinal, USA, 1997: 4-13
- [5] Michaud P, Sez nec A, Uhlig R. Trading conflict and capacity aliasing in conditional branch predictors//Proceedings of the 24th International Symposium on Computer Architecture. Denver, Colorado, USA, 1997: 292-303
- [6] Eden A N, Mudge T. The YAGS branch prediction scheme//Proceedings of 31st International Symposium on Microarchitecture. Dallas, Texas, USA, 1998: 69-77
- [7] Yeh T, Patt Y. Two-level adaptive training branch prediction//Proceedings of the International Symposium on Microarchitecture. Albuquerque, New Mexico, 1991: 51-61
- [8] Chang P, Hao E, Patt Y. Target prediction for indirect jumps//Proceedings of the International Symposium on Computer Architecture. Denver, Colorado, USA, 1997: 274-283
- [9] Driesen K, Holzle U. Limits of indirect prediction. University of California, Santa Barbara; Technical Report TRCS97-10, 1997
- [10] Driesen K, Holzle U. The cascaded predictor: economic and adaptive branch target prediction//Proceedings of the 31st International Symposium on Microarchitecture. Dallas, Texas, USA, 1998: 249-258
- [11] Lee J, Smith A. Branch prediction strategies and branch target buffer design. IEEE Computer Magazine, 1984, 17(1): 6-22
- [12] Calder B, Grunwald D. Reducing indirect function call overhead in C++ programs//Proceedings of the 21st Annual Symposium on Principle of Programming Languages. Portland, Oregon, USA, 1994: 397-408
- [13] Kaeli D, Emma P. Branch history table prediction of moving target branches due to subroutine returns//Proceedings of the 18th International Symposium on Computer Architecture. Toronto, Ontario, Canada, 1991: 34-42
- [14] Kalamatianos J, Kaeli D. Predicting indirect branches via data compression//Proceedings of the 31st International Symposium on Microarchitecture. Dallas, Texas, USA, 1998: 272-281
- [15] Chu Yul, Ito M R. An efficient indirect branch predictor//Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing. 2001: 394-402
- [16] Yeh Tse-Yu, Patt Yale N. A comparison of dynamic branch predictors that use two levels of branch history//Proceedings of the 20th Annual International Symposium on Computer Architecture. San Diego, California, United States, 1993: 257-266
- [17] McFarling S. Combining branch predictors. WRL Technical Note TN-36, Digital Equipment Corporation, June 1993



YUAN Nan, born in 1982, Ph. D. candidate. His current research interests include parallel architecture, algorithm, and runtime system.

FAN Dong-Rui, born in 1979, Ph. D. , associate professor. His main research interests include low-power design and processor microarchitecture.