HIP:一种基于提示值索引的间接转移预测技术

谢子超,史秦青
(北京大学信息科学技术学院,北京 100871)

摘 要: 随着模块化程序和面向对象语言的发展,间接转移预测已成为影响处理器性能的瓶颈.本文提出了一种基于提示值索引的间接转移预测技术(Hint-indexed Indirect-branch Prediction,HIP).该技术将间接转移指令目标地址保存在 BTB 中,每个目标地址使用一个提示值与之对应.HIP 技术在进行间接转移预测时,首先使用已有的分支方向预测器区分不同的间接转移场景,并获得其中存储的提示值.该提示值与指令地址进行计算,生成一个虚拟地址,用于索引存储间接转移目标地址的 BTB 项.实验表明,该技术可以显著提高性能,而且不需额外的大容量存储结构.与常用的基于 BTB 的结构相比,HIP 技术可将基础处理器性能提升 20.38%.与已有基于硬件的方法相比,HIP 比 VPC 方法性能提高 8.66%,并且可以获得与 48KB TTC 预测器相同的性能提升效果.HIP 还可以将处理器能耗平均降低 14.34%.

关键词: 微处理器;间接转移预测;高能效
中图分类号: TP302 **文献标识码:** A **文章编号:** 0372-2112 (2012) 08-1523-09
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2012.08.005

HIP:A Hint-Indexed Indirect-Branch Prediction Technique

XIE Zi-chao,SHI Qin-qing
(*School of Electronics Engineering and Computer Science*, *Peking University*, *Beijing* 100871, *China*)

Abstract: Indirect-branch prediction becomes a performance bottleneck for modern processors as object-oriented programs are more commonly used recently. This paper proposes a Hint-indexed Indirect-branch prediction (HIP) technique. It first uses the existing branch direction predictor to distinguish different indirect-branch occurrences, and treats the corresponding data as hints. Then, those hints are calculated with the indirect-branch addresses to generate the virtual addresses, which are used to access the BTB for the predicted indirect-branch targets. Our evaluation shows that HIP could achieve attractive performance improvement without large dedicated storages. Compared to that of a commonly-used BTB, it improves average performance by 20.38%. Compared with previously proposed hardware-based predictors, HIP improves performance by 8.66% over that of the VPC predictor, and it achieves the equivalent performance improvement provided by a 48KB Tagged Target Cache (TTC) predictor. The energy consumption is also reduced by 14.34% over the baseline.

Key words: microprocessors; indirect-branch prediction; energy-efficiency

1 引言

现代高性能处理器利用分支指令预测技术,开发指令级并行,从而提高处理器性能.之前的研究主要集中在开发高准确率的直接转移指令(Direct Branch)预测技术[1].而对于间接转移指令预测(Indirect-Branch Prediction),通常很难达到较高的准确率.这是因为间接转移指令的分支目标地址保存在其指令指定的寄存器中.该寄存器中的值可能会随着程序的执行而发生各种变化,即可能存在多个分支目标地址.现有的分支目标缓冲器(Branch Target Buffer,BTB)仅能预测间接转移指令最近一次的转移目标,因此该类指令预测准确率低[2,3].近年来,随着面向对象语言程序的广泛应用[4],间接转移指令使用得更加广泛.该类型指令通常用于实现虚函数指针,Switch-Case 语句以及函数指针等常见数据结构.此外,Intel Pentium M[5]处理器中也已经加入了专用的间接转移预测部件.因此,实现具有高间接转移预测准确率的分支预测结构对处理器性能的提升具有重要意义.

随着功耗逐步成为芯片设计的主要限制因素,能效性逐渐成为处理器设计的重要指标[6].芯片设计必须在性能和能耗间进行权衡,选择那些能够以最小能耗取得最佳性能的方法.基于此种考虑,利用或改进处理器

收稿日期:2011-03-04;修回日期:2012-02-09
基金项目:国家 863 高技术研究发展计划(No.2006AA010202)

中已有结构来提升处理器性能成为一种重要的设计方法[2].该方法在提升性能的同时,不会引入额外的能耗损失,从而达到高能效的设计目标.

本文提出了一种基于提示值索引的间接转移预测技术(Hint-indexed Indirect-branch Prediction,HIP).该技术将间接转移指令目标地址保存在 BTB 中,每个目标地址使用一个提示值与之对应.HIP 技术在进行间接转移预测时,首先使用已有的分支方向预测器区分不同的间接转移场景(Branch Occurrences),并获得其中存储的提示值(Hint).该提示值与指令地址进行计算,生成一个虚拟地址(Virtual BTB Address,VBA),用于索引存储间接转移目标地址的 BTB 项.该技术通过对已有 BTB 和分支方向预测器功能的重定义实现间接转移预测,无需添加大容量的专用目标存储器,也无需附加额外的编译支持.

本文实验结果表明,相比传统 BTB 结构,HIP 技术可将 4 发射 16 级流水线的高性能处理器的每千条间接转移指令预测失效数(Misprediction Per Kilo Instructions,MPKI)从 3.69 降至 0.99,性能提高 20.38%.与其他基于硬件的间接转移预测技术相比,HIP 可以达到和 48KB 的 TTC(Tagged Target Cache)预测器[3]相同的性能提升效果,并且与 VPC 预测器[2]相比,其性能提高了 8.66%.

2 研究背景

本节首先介绍基于硬件的间接转移预测技术的基本原理,之后介绍该类技术的相关研究工作.

2.1 预测原理

已有的基于硬件的间接转移预测技术均利用分支转移历史信息(即最近若干条分支指令的执行结果,Global History Register,GHR)来区分不同的间接转移场景,从而预测分支目标地址.本文选取 SPEC CPU 2006[7]中的 Perlbench 程序中的典型片段(图 1)作为实例说明该类技术的基本原理.程序片段中函数指针(sortsvp)对应一条间接转移指令(jsr $ 26,($ 27),0).随着程序的执行,位于程序中第 13 行的条件转移指令执行的不同结果,会导致不同的间接转移目标($ 27 的值不同),即产生不同的函数指针.例如,当第 13 行条件为真时,函数指针为 S_qsortsv;否则为 S_mergesortsv.由此,根据已执行过的条件转移指令的历史信息(执行结果),预测器就可区分不同的间接转移场景,从而对预测目标地址进行推测.

```
1:  void
2:  Perl_sortsv(pTHX_ SV **array, size_t nmemb, SVCOMPARE_t cmp)
3:  {
4:      void (*sortsvp)(pTHX_ SV **array,
5:                      size_t nmemb,
6:                      SVCOMPARE_t cmp,
7:                      U32 flags)
8:        = S_mergesortsv;
9:      SV *hintsv;
10:     I32 hints;
11:
12:     hints = SORTHINTS(hintsv);
13:     if (hints & HINT_SORT_QUICKSORT) {
14:         sortsvp = S_qsortsv;
15:     }
16:     else {
17:         sortsvp = S_mergesortsv;
18:     }
19:
20:     sortsvp(aTHX_ array, nmemb, cmp, 0);
21: }
```

```
        ldah $29, 0($26)
        lda $29, 0($29)
        ldah
$1, S_qsortsv($29)
        ldq $3, 0($0)
        lda
$27, S_qsortsv($1)

$L276:
        jsr $26, ($27), 0

$L273:
        ldah
$1, S_mergesortsv($29)
        lda
$27, S_mergesortsv($1)
        br $31, $L276
```

图1 Perlbench中间接转移指令示例

2.2 相关技术

已有的一些使用专用部件的间接转移预测技术[3,8]可以有效提升间接转移预测准确率.Chang 等人提出在 BTB 外引入一个大容量的 TTC 结构[3],单独存储不同间接转移场景下的间接转移目标地址.TTC 结构的思想类似两级分支预测方法[9],其使用分支转移历史信息(GHR)区分不同间接转移场景.当取得一条间接转移指令时,TTC 预测器使用 PC 和分支转移历史的异或值作为索引,获得预测目标地址.当该间接转移指令提交时,使用正确的目标地址更新对应的 TTC 项.

通过组合使用多个目标地址预测器,Driesen 等提出了另一种额外使用大容量存储单元的预测技术—Cascade 预测器[8].对于可以简单预测(只有一个目标地址)的间接转移指令,使用简单的一级预测器预测;而对于拥有多个目标地址的间接转移指令,使用复杂的二级预测器进行预测.Seznec 和 Michaud 提出了与该思想非常类似的 IT-TAGE 预测器[10].该预测器由一个基础预测器以及一系列能够捕捉非常长历史信息的预测表组成.在预测时,选择历史信息最长的预测表中的命中项值作为输出结果.

VPC 预测技术[2]使用已有的条件转移预测部件来进行间接转移预测.VPC 将一条间接转移指令的每个目标地址等价表示成一个虚拟的条件分支转移指令的转移目标地址.当进行间接转移指令预测时,VPC 每个周期以不同虚拟分支指令循环访问条件分支预测器.该循环访问直到一条虚拟指令被预测为发生,或达到循环访问的上界时才停止.

3 HIP 预测技术

HIP 技术的核心是使用提示值(Hint)作为索引进行间接转移预测,即将 Hint 作为间接转移预测目标地址的一种中间表示.该技术使用 Hint 将间接转移场景到目标地址的映射分为两步,如图 2 所示:一是将每个间接转移场景映射到 Hint,二是将每个 Hint 映射到间接转移目标地址.使用 Hint 索引预测流程有如下两个优点:①利用 Hint 可以快速获得其所指向的数据,相比于直接对目标地址进行预测的方法仅多一步操作;②Hint 索引的数据可以离散的排布在存储空间中,因此可将这些数据的存储融入现有的结构中.

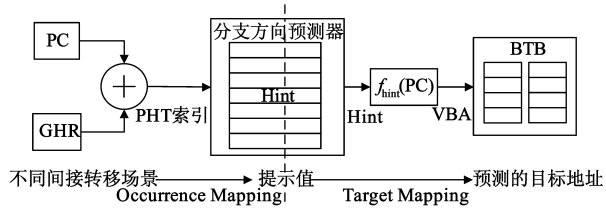


图2　HIP中间接转移场景和目标地址之间的映射关系

3.1　基本结构

HIP技术将间接转移指令的目标地址保存在BTB中.区别于条件转移指令,一条间接转移指令会占用多个BTB项,从功能上将这些项划分为目标地址项和分配项.目标地址项用于存储该间接转移指令的目标地址.每个目标地址项由Hint生成的虚拟地址(VBA)进行索引.随着程序的运行,HIP根据该条间接转移指令出现过的目标地址,动态的分配目标地址项.相关研究表明,大多数程序中一条间接转移指令的目标地址不会超过16个[2](进一步的目标地址数分析将在本文试验中予以讨论).基于此种情况,HIP给一条间接转移指令最多分配16个目标地址项.如果一条间接转移指令的目标地址超过16个,HIP将替换已有的目标地址项,以记录最新遇到目标地址.分配项用于记录每个间接转移指令目标地址项的使用情况.该信息占用分配项的低16位.为了在预测时尽快确认BTB中是否存在该间接转移指令的目标地址项,分配项在BTB中使用PC直接索引.

由于BTB中一条间接转移指令最多占用16个目标地址项,HIP使用4位数据作为Hint以生成索引每个目标地址项的VBA(位数与PC相同),由此建立了每个Hint与目标地址之间的一一映射关系(Target Mapping).Hint相当于计算VBA的函数指针,即不同的Hint表示对PC的不同计算方式 VBA = f_{Hint}(PC).一种简单的VBA计算方式f_{Hint}(PC)是将PC的最高4位和最低4位分别与Hint相加.这种使用Hint对PC进行相应变换以生成BTB索引的方式,保证了BTB已有的访问机制不变.处理器加入HIP机制后,无论是否进行间接转移预测,BTB总是根据输入地址进行索引和Tag比较,最后给出命中结果及命中地址.该实现结构维持原有的BTB访问机制不变,降低了HIP机制的设计复杂度,使该方法在实际处理器设计中更易被实现.

HIP使用分支转移历史来区别每一间接转移指令的不同转移场景,每种场景对应一个Hint,即建立转移场景与Hint的映射关系(Occurrence Mapping).通过对现有分支预测器功能的分析和重定义,可以将Hint保存在基于历史的分支方向预测器的表项中.现有商用处理器[5]通常使用基于PHT(Pattern History Table)的组合方向预测器(Hybrid Branch Predictor),但PHT中每项表示转移方向的2位饱和计数器对于间接转移预测是无效的(永远预测为转移).HIP利用这些空隙存储与间接转移场景相对应的Hint,以替代2位无效的饱和计数值.4位Hint需要使用2个PHT表项保存.为了索引生成的简便,HIP将Hint分别保存在XOR(PC,GHR)和XOR(PC,GHR << 1)索引的项中.

3.2　预测流程

本节基于普遍使用的以PHT为基础的组合方向预测器和BTB结构,描述HIP预测流程.

第一个周期,HIP同时访问BTB和分支方向预测器.如果BTB命中,意味着该间接转移指令被执行过,命中项即为该指令的分配项.否则,预测中止并暂停取指,直到间接转移目标地址在流水线中计算完成.本周期PHT得到的结果是Hint的低2位.此时,HIP使用了如下加速机制加快预测速度:本周期首先假设Hint的高2位为0,得到一个临时的Hint (TempHint),并根据f_{TempHint}(PC)获得临时虚拟地址(TempVBA).

第二个周期,HIP使用TempVBA访问BTB,并改变索引再次访问方向预测器.如果BTB命中,则存储的目标地址作为预测目标地址(TempPredTarget),将被发射到流水线和指令缓存中.如BTB失效,则直接忽略本周期BTB的输出值.HIP需要改变索引第二次访问PHT,以获得Hint的高2位,即将原有的PHT索引左移1位,并在最末位补0.两次PHT访问结果组成了完整的4位Hint.需要注意的是,如果此时Hint高位为0,则完整的Hint与TempHint相同,其所对应的目标地址在本周期已经被取出并被发射.在这种情况下,本周期即结束此次间接转移预测 (快速预测模式,Fast-Pred).

第三个周期,HIP使用完整的4位Hint访问BTB.如果BTB命中,HIP以所得目标地址(PredTarget)作为预测目标地址(完整预测模式,Full-Pred),并取消之前发出的预测目标地址(如果已有地址被发射到流水线和指令缓存中).如果BTB失效,HIP暂停预测直到从流水线中获得实际的目标地址(或继续使用最近预测的目标地址).综上预测流程,HIP预测最多需要3个周期.

3.3　更新流程

如果间接转移指令提交时发现预测正确,HIP对BTB的操作与条件转移指令更新BTB时相同,即更新目标地址项所在Set的替换信息(通常为LRU信息).一旦发现预测错误,HIP需要更新存储在方向预测器的Hint,使其与存储正确目标地址的BTB项相对应.HIP中会产生如下两种错误情况:

(1)Hint错误:BTB中一个目标地址项已存放了正确的间接转移地址,但Hint映射到了错误的目标地址

项.

(2)无意义 Hint:没有目标地址项存储正确目标地址,因此 Hint 无意义.

为了区分这两种情况,HIP 更新算法需要根据分配项中记录,遍历 BTB 中该间接转移指令全部已使用的目标地址项.遍历时,一旦出现某个 VBA 访问 BTB 失效的情况,说明该目标地址项已被其他分支指令替换,此时需要及时更新记录项中的内容.如果某个目标地址项中存储地址与正确目标地址相同(Hint 错误情况),则更新 Hint 指向该位置.如果遍历全部的地址仍没有发现与正确地址匹配的项(无意义 Hint 情况),则需要根据分配项中的信息,分配一个新的目标地址项或替换一个目标地址项来记录正确的目标地址.如果新分配目标地址项,则需要更新分配项中内容.如果替换已使用的目标地址项,HIP 选择一项进行替换(替换算法将在本文实验中进行讨论).更新 BTB 后,HIP 需修改相应 Hint,使其与保存正确目标地址的 BTB 项相对应.

由于在传统 BTB 中一个周期只能访问一个 BTB 项,因此 HIP 的遍历过程最多需要 16 个周期,这增加了更新算法的时间复杂度.然而实际上由于 HIP 预测准确率高(见本文实验部分),因此需要遍历更新的情况较少.另一方面,HIP 只会访问已经分配的项,因此大部分情况下,不会访问过多的 BTB 项,更新所需周期数较少.该更新过程与 VPC 技术[2]的更新方法类似,但实现更为简单.VPC 更新时,要判断正确的目标地址是否在 BTB 中已经保存,则需要遍历该间接转移指令相关的各个目标地址项,直到找到包含正确目标地址的目标地址项或达到循环访问的上界(在其论文中该值评估为 12).而 HIP 同样需要遍历保存目标地址的目标地址项,但只遍历已分配的目标地址项,因此通常情况下,遍历的表项更少,如表 1 所示.举例而言,如果一个间接转移目标地址数共为 3 (依次为 A,B,C),BTB 中已经记录了 2 个地址 A 和 B,此时需要新分配一个 BTB 项存储 C 地址.VPC 技术此时需要对 BTB 遍历 12 次,达到循环访问的上界后发现仍没有搜索到 C 地址,才会在 BTB 中分配一项保存 C 地址.而 HIP 只需要遍历 2 次即可(因为此时 HIP 中该指令的分配项记录该间接转移指令当前只使用了 2 个目标地址项).

需要注意的是,在分支预测中加入 HIP 机制后,条件转移指令在更新时,可能替换已保存的间接转移预测地址.当条件转移指令预测错误且 BTB 中未包含该指令的目标地址时,需要在 BTB 中为该目标地址分配一个 BTB 项.此时,BTB 会在该指令地址索引的组(Set)中根据 BTB 替换策略(通常为 LRU)选择替换一项.如果选择替换的目标地址保存的是间接转移的目标地址,则会将已保存的间接转移预测地址替换.然而,此种情况对处理器性能的影响基本可以忽略:HIP 中进行间接转移预测时,同样会更新此次访问的目标地址项所在组的 LRU 信息,且操作与条件转移指令使用 BTB 的更新操作相同.此更新操作保持现有 BTB 工作机制不变,即在每个 Set 中根据 LRU 信息保存最为常用的分支指令的目标地址.如果一条条件转移指令在 BTB 更新时,最终选择替换该 Set 中一个间接转移目标地址项,则说明该 BTB 项在本 Set 中最近最少被使用,理应被替换.因此此时对该项的替换不会对性能产生明显影响.本文实验部分也对该方面进行了相应评测.

表 1 HIP 与 VPC 更新周期数统计

程序	间接转移指令总更新次数	HIP 总更新周期	HIP 平均每次更新周期	VPC 平均每次更新周期
gcc	89917	150423	1.67	2.90
crafty	62210	117859	1.89	3.50
eon	30769	31428	1.02	2.01
perlbmk	176148	953554	5.41	7.76
gap	828	449	0.54	0.50
sjeng	76978	171130	2.22	2.45
perlbench	471218	2920330	6.20	6.01
gcc06	45554	95581	2.10	2.58
povray	64613	51480	0.80	1.02
richards	6529	10538	1.61	2.00
average	38699	70141	1.81	2.39

3.4 硬件开销和复杂度

HIP 基于现有的分支预测部件的改动方案,如图 3 所示.

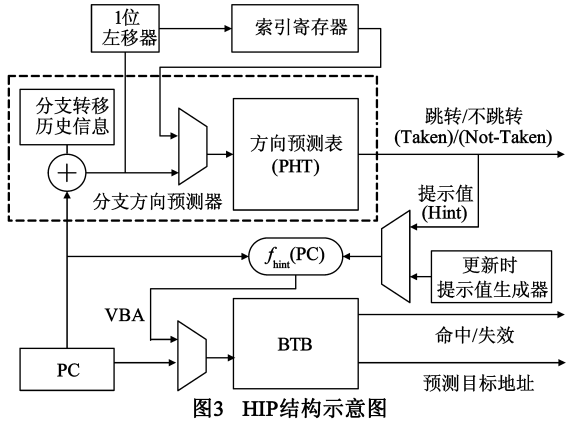


图3 HIP结构示意图

(1)增加 Hint 相关逻辑:增加 1 个 Hint 寄存器,用来在预测和更新时保存 4 位 Hint;增加 1 个循环计数器,用以在更新时根据分配项中信息产生遍历的 Hint

(2)增加 PHT 二次索引的相关逻辑:增加 1 个一位左移移位器,产生第二次 PHT 访问索引;PHT 输入端增加 1 个多选器,选择 PHT 访问索引

(3)增加 VBA 相关逻辑:增加 1 个 ALU,用以根据 Hint 和 PC 计算 VBA;BTB 输入端增加 1 个多选器,选择

PC 或 VBA 作为访问地址

(4)增加分配项寄存器,以便在整条流水线中携带该项的信息

HIP 基于已有分支预测器进行拓展,只对现有结构进行了简单修改:首先,HIP 不需要大容量存储器或复杂表结构来存储目标地址和 Hint.其次,HIP 方案中用于预测和更新的组合逻辑简单.另外,HIP 方案不会影响预测部件的关键路径:VBA 以及 PHT 索引均提前一个周期生成,因此不在分支预测的关键路径上.

4　实验评估

本文扩展了 SimpleScalar 模拟器[11]进行 HIP 效果评测,并使用 Wattch 扩展功耗模型[12]评测能耗开销.用于评测的基础处理器结构参数,如表 2 所示.由于大多数现有高性能商业处理器中均使用预译码(Predecoding)技术,本文采用和 VPC 类似的方法[2],即重用这些附加位来辨识间接转移指令.本文重点针对以下间接转移指令敏感的测试集进行评测:5 个 SPEC CPU 2000 INT benchmarks,4 个 SPEC CPU 2006 benchmarks[7]以及 1 个 C++ benchmark—Richards 程序[13]*.这些测试集当所有间接转移指令均准确预测时,可使 IPC(Instruction Per Cycle)提升 5%以上.同时,本文还针对 HIP 技术对 SPEC CPU INT 2000 中间接转移指令不敏感程序产生的影响进行了评估.

表 2　基础处理器配置参数

参数	配置值
频率	1GHz
流水线	16 级,4 取指/发射/执行/提交 512 项 RUU,128 项 LSQ
分支转移预测器	使用预译码判断 Branch 指令 BTB:4 路组相联,4096 项 组合分支预测器(32K gshare,8K bimodal and selector) 32 项返回地址栈(RAS) 分支指令失效代价为 15 个周期
L1-Cache	ICache:32KB,8 路组相联,每行 32 字节,2 周期访问延时 DCache:32KB,8 路组相联,每行 32 字节,2 周期访问延时
L2 Unified Cache	1MB,16 路组相联 每行 64 字节,10 周期访问延时
主存	150 周期平均访问延时

本文使用 SimPoint[14]选择输入集合中有代表性的片段进行评测.每个评测程序运行 100M 条 Alpha 指令,并使用 Compaq C++ V6.5-042 for CompaqTru64 UNIX V5.1B Rev.2650 进行编译.

4.1　HIP 技术对分支指令 MPKI 及性能影响

基础处理器和使用 HIP 技术后间接转移 MPKI 比较,如图 4 所示.相对基础处理器,HIP 平均可以将间接转移 MPKI 从 3.69 降至 0.99,显著提升了间接转移指令的预测准确率.

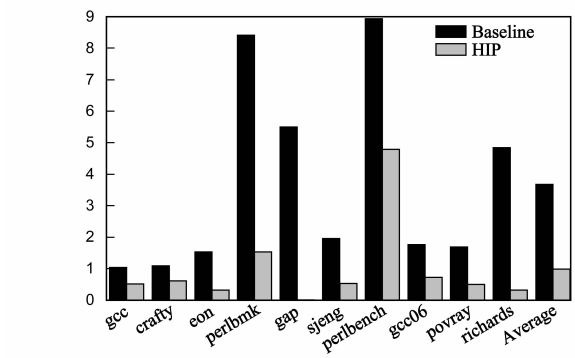


图4　HIP对间接转移MPKI的影响

表 3　HIP 对条件转移 MPKI 的影响

程序	基础处理器 MPKI	HIP MPKI
gcc	9.47	9.66
crafty	8.21	8.35
eon	6.59	6.59
perlbmk	3.66	4.66
gap	1.52	2.12
sjeng	13.56	13.66
perlbench	3.27	3.87
gcc06	8.29	8.59
povray	2.51	2.51
richards	4.83	5.13
Average	6.19	6.51

HIP 对条件转移指令 MPKI 的影响,如表 3 所示,其将条件转移平均 MPKI 从 6.19 提升至 6.51.由于 HIP 技术复用 PHT 中饱和计数器以保存间接转移预测使用的不同 Hint,因此其会改变原本用于条件转移指令预测的饱和计数器的值,即产生别名冲突(Alias).例如,HIP 将提示值高位"11"存入用于某个条件转移预测的饱和计数器中,替换原有值"00",则会改变原有预测结果(将预测跳转而非不跳转),由此可能带来预测错误.但实验结果表明,HIP 对条件转移预测造成的负面影响小于间接转移预测准确率提升的积极影响,分支指令总 MPKI 从 9.88 降至 7.50.

分支指令总 MPKI 的下降导致处理器 IPC 的提升.HIP 对 IPC 的提升比例,如图 5 所示.由于处理器频率不受影响,IPC 指标即反应了其对性能的影响.

为了深入分析 HIP 技术,本文也对理想的间接转移预测方案以及两种特殊 HIP 结构进行评测.由于 HIP 技术复用 PHT 保存组路指针并复用 BTB 保存间接转移

* 本文选取了与相关最新研究[15]相类似的测试集进行了评测.虽然已有研究也对基于 Java 的 DaCapo 测试集进行了评测(运行该类测试集需要在 Java 虚拟机上执行).但由于不同的 Java 虚拟机对于间接转移指令的处理有明显差异,因此对于 Java 测试集的评测不具有普遍性,所以本文不选取该类测试集进行评测.

目标地址,因此这两个复用结构可能会影响 HIP 的预测准确率.图 5 所示的两种特殊 HIP 实现方式,分别为使用单独 PHT 结构(保存提示值)同时复用 BTB 结构(ShareBTB_SinglePHT)以及使用单独 BTB 结构(保存间接转移目标地址)同时复用 PHT 结构(SharePHT_SingleBTB).实验结果表明,100%准确的理想间接转移预测能提高性能的上界是 25.41%,而 HIP 可使性能提高 20.38%.HIP 技术 ShareBTB_SinglePHT 结构的 IPC 提升效果接近于理想的间接转移预测机制,平均可以使 IPC 提升 23.67%.而 HIP 的 SharePHT_SingleBTB 结构只略高于普通 HIP 结构,平均可以使 IPC 提升 20.78%.以上两个实验表明,BTB 复用对于间接转移预测的影响较小,HIP 中的主要限制因素在于 PHT 中的别名冲突.

本节还对间接转移指令不敏感的程序使用 HIP 技术可能产生的影响进行了评测.实验数据如表 4 所示.实验结果表明,HIP 对这些程序的性能没有负面影响.这是由于以下两个原因:

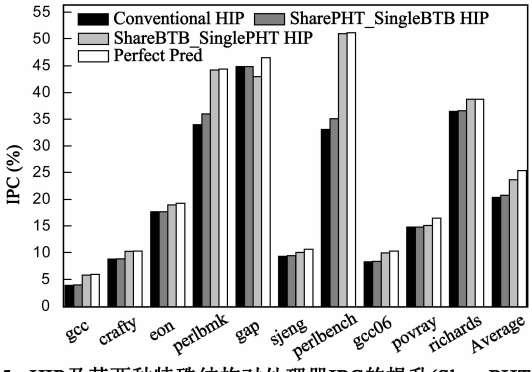


图5 HIP及其两种特殊结构对处理器IPC的提升(SharePHT_SingleBTB为只复用PHT不复用BTB,ShareBTB_SinglePHT为只复用BTB不复用PHT)

表 4 间接转移指令不敏感程序使用 HIP 技术后的效果

程序	间接转移预测准确率 Baseline	间接转移预测准确率 HIP	条件转移预测准确率 Baseline	条件转移预测准确率 HIP	加入 HIP 后 IPC 提升
gzip	89.33%	91.12%	90.67%	90.66%	0.42%
vpr	99.90%	99.91%	90.92%	90.92%	0.00%
mcf	95.98%	96.23%	92.32%	92.32%	0.00%
parser	93.23%	93.27%	90.13%	90.13%	0.00%
vortex	86.23%	88.99%	96.00%	95.99%	1.40%
bzip2	70.79%	70.90%	89.11%	89.09%	0.00%
twolf	40.08%	53.58%	84.22%	84.22%	0.33%

(1)HIP 技术只有在取得间接转移指令时,才会执行 HIP 预测流程.因而,其对条件转移指令的预测流程没有影响.

(2)HIP 技术根据遇到的间接转移指令的目标地址,在 BTB 和 PHT 中动态的分配间接转移预测所使用的项.当间接转移指令比例极少时,这种分配的项也极少,因而不会给条件转移预测带来明显的性能影响.

4.2 HIP 更新策略评测

HIP 在更新时,针对间接转移指令目标地址项的替换,可以采用多种替换策略,本节主要选取具有代表性的 Pseudo-LRU 策略(以下简称 LRU 策略)和随机策略进行评测.两种替换策略的实验结果,如表 5 所示.实验结果表明,这两种 HIP 替换策略对处理器性能的影响基本相同.这是由于:1)平均 90%以上的间接转移指令的目标地址不超过 16 个,即 HIP 可分配足够的空间以存储程序中新出现的间接转移目标地址,因此大部分情况下不会出现替换已记录的目标地址项的情况.在该情况下,HIP 只需在空闲的目标地址项中为间接转移指令分配一项.因此,随机策略和 LRU 策略没有差别.2)只有当间接转移目标地址超过 16 个时,不同替换策略理论上才会略有差别.此时,随机策略可能会替换掉经常使用的目标地址项,但统计表明此情况出现的概率平均不到 1%.因此随机策略可以获得与 LRU 策略相同的性能提升.

另一方面,如果使用 LRU 策略,则需要使用较多的位数来记录 LRU 信息.现有 HIP 中使用一个 32 位的 BTB 项作为分配项,记录目标地址项的使用情况.如果要记录 LRU 信息,则需要再额外占用 2 个 BTB 项以存储 64 位 LRU 信息($16x\log_2 16$),这就增加了 HIP 在更新时的复杂度:即需要多访问 2 次 BTB,将全部的 LRU 信息取回后,再分配新的目标地址项.因此,鉴于使用 LRU 策略的效果并不明显,且会增加了设计复杂度,HIP 在更新时选择实现更为简单的随机策略.

4.3 Hint 位数评测

本节针对已选择的基于 Alpha ISA 测试程序进行评测,发现平均 80%以上的间接转移指令未超过 8 个地址,如图 6 所示.若将 Hint 设置为 3 位(即为一条间接转移指令最多分配 8 个目标地址项),则处理器性能影响如表 5 所示.

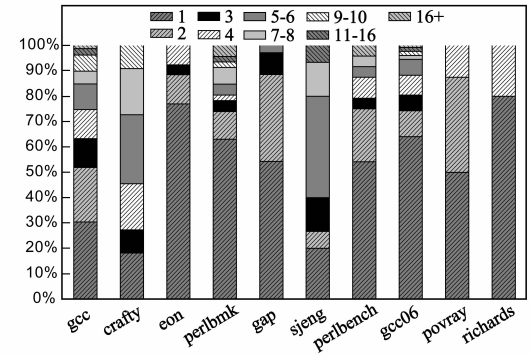


图6 间接转移目标地址项的数目分布

实验结果表明,如果将目标地址项设置为 8,对一般程序间接转移 MPKI 和 IPC 的影响可以忽略,而对 perlbmk、perlbench 目标地址数量较多的程序,有着较为明显的负面影响.当 Hint 设置为 3 位时,存储每个 Hint 仍然需要占用 PHT 中 2 项(每个 PHT 项为 2 位饱和计数器,3 位 Hint 分别占用索引的 2 位和索引的低位).每次间接转移预测时,仍需要访问 PHT 2 次获得全部的 Hint 值.因此,缩小可分配的目标地址项数目不能加快 HIP 预测.综上所述,从选择对象的普遍性考虑,选取 16 个目标地址项具有一定的普适性.

4.4　不同微体系结构 HIP 效果评测

4.4.1 双端口 PHT 结构

本节评测 PHT 为双端口结构时 HIP 的效果.PHT 为双端口时,HIP 可以在一周期内获得全部 Hint.该实验的评测如图 7 所示.实验结果表明,增加 PHT 的双端口访问对 HIP 性能的提升效果并不明显,增加双端口后比单端口机制只提升了 2.99%.增加双端口 PHT 访问实际上只对 HIP 中 Full-Pred 模式起到了加速作用.而 HIP 机制中已存在加速机制(Fast-Pred 模式),而且本文实验表明 Fast-Pred 模式平均占全部预测的 69.66%,已经有很好的加速作用.考虑到增加 PHT 双端口对 HIP 的加速效果很小,并且增加了设计复杂度,因而没有必要使用该机制.

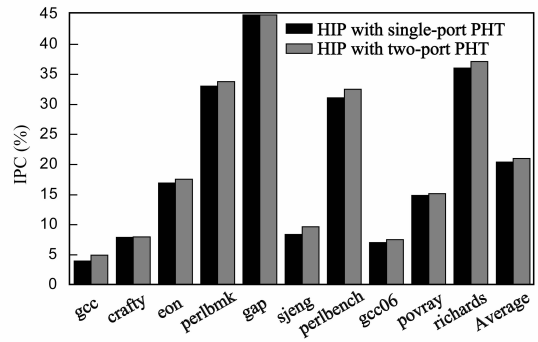


图7　单端口PHT与多端口PHT下HIP的性能提升效果

4.4.2　不同 BTB 容量

由于 HIP 将间接转移指令的目标地址保存在 BTB 中,因此有必要针对使用不同容量 BTB 时 HIP 技术的效果进行评估.BTB 容量分别为 512、1K、2K、4K 项时,评测结果如表 6 所示.由于 HIP 中每条间接转移指令仅占用与其动态执行目标个数相同的

BTB 项,因此即使使用较小的 BTB,HIP 也能获得很高的间接转移指令预测准确率以及性能提升.

表 5　不同更新策略下以及不同 Hint 位数时,HIP 的 IPC 提升效果

程序	gcc	crafty	eon	perlbmk	gap	sjeng	perlbench	gcc06	povray	richards	Average
随机策略	3.73%	7.68%	16.89%	31.72%	44.76%	7.86%	30.37%	6.99%	14.87%	35.72%	20.38%
Pseudo-LRU 策略(未计入 LRU 存储代价)	3.96%	8.88%	18.61%	35.36%	46.10%	9.35%	33.94%	8.43%	15.16%	38.83%	21.03%
Pseudo-LRU 策略(计入 LRU 存储代价)	3.92%	8.76%	17.61%	34.33%	45.32%	9.35%	33.81%	8.36%	15.00%	36.91%	20.54%
3 位 Hint	2.50%	6.92%	16.89%	24.22%	44.76%	7.52%	22.19%	6.63%	14.86%	35.72%	17.53%
4 位 Hint	3.73%	7.68%	16.89%	31.72%	44.76%	7.86%	30.37%	6.99%	14.87%	35.72%	20.38%

表 6　不同 BTB 容量下 HIP 方案的间接转移 MPKI 和 IPC 提升效果

BTB 项数	MPKI			IPC 提升	
	基础处理器	HIP(随机替换)	HIP(LRU 替换)	HIP(随机替换)	HIP(LRU 替换)
512	3.69	1.01	1.00	16.99%	17.00%
1024	3.51	1.01	1.00	18.23%	18.27%
2048	3.40	0.99	0.99	19.77%	19.80%
4096	3.27	0.99	0.99	20.38%	20.54%

4.5　与其他间接转移预测方案的比较

与相关工作类似[2,15],本节将 HIP 预测器与前文提到的一种最佳的、基于硬件实现的间接转移预测器 TTC[3]以及高能效预测器 VPC[2]的效果进行对比.各种间接转移预测器对间接转移 MPKI 和 IPC 的提升,分别如图 8、图 9 所示.

本文测试了不同容量的 4 路组相联 TTC 预测器,从 256 项到 64K 项.每一个 TTC 项都包括 4 字节的目标地址以及 2 字节的标签域,容量分别从 1.5KB 至 384KB.实验结果表明,HIP 可以取得和 48KB TTC 预测器同等的性能,而至少 6 个程序(gcc,crafty,eon,gcc06,povray,richards),HIP 的性能提升效果与 96KB TTC 预测器等同.

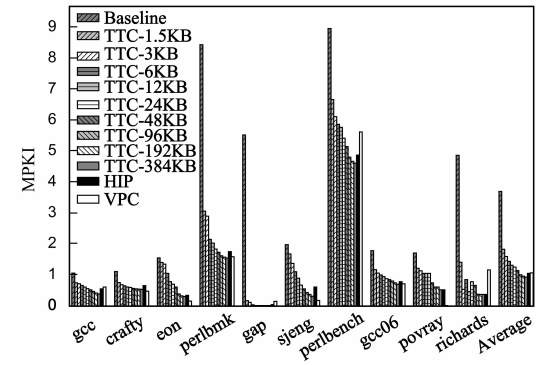


图8　HIP与其他间接转移预测技术的间接转移MPKI比较

HIP 可以取得和 VPC 相类似的间接转移预测准确率,且在性能上优于 VPC 预测 8.66%.这是因为 HIP 相

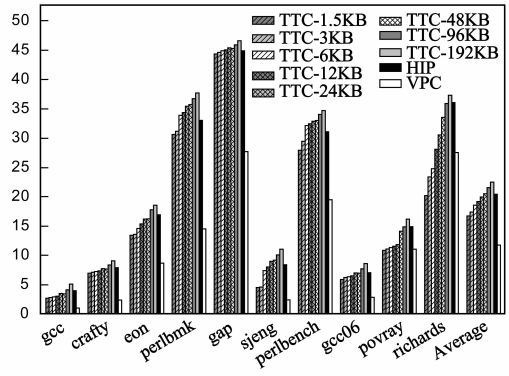


图9 HIP与其他间接转移预测技术IPC比较

比于 VPC,其预测速度更快,并且性能提升效果受动态间接转移目标地址数目和流水线长度的影响更小.HIP 中 $f_{(\mathrm{Hint})}$(PC)的 BTB 索引生成方式与 VPC 的策略在预测速度上有着明显不同:VPC 是采用类似链表的方式,对生成的索引进行依次访问;而 HIP 中是根据 Hint 的不同,直接计算获得索引目标地址项的索引.实验评测发现,VPC 平均需要 5-6 个周期得到预测结果,而 HIP 平均只需 2.20 个周期.在更新时,HIP 效率也要高于 VPC:如表 1 所示,HIP 根据分配项中的记录只访问已分配的项,而 VPC 在 BTB 未记录当前目标项时需要进行 12 次遍历.

4.6 能耗评测

本节使用 Cacti 工具[16]获得能耗开销的评估参数,工艺设置为 65nm,处理器主频为 1GHz.加入 HIP 技术后处理器能耗降低 14.34%,并且其效果优于其他间接转移预测方案,如图 10 所示:与硬件开销大的 TTC 预测不同,HIP 无需添加更大的专用存储部件,因而不会引入额外的硬件能耗开销.由于其具有高的预测准确率,因而使用 HIP 时流水线排空次数少,可以有效降低能耗.另外,由于 HIP 预测时只访问已有的分支预测器,不会增加每周期的最大功耗.

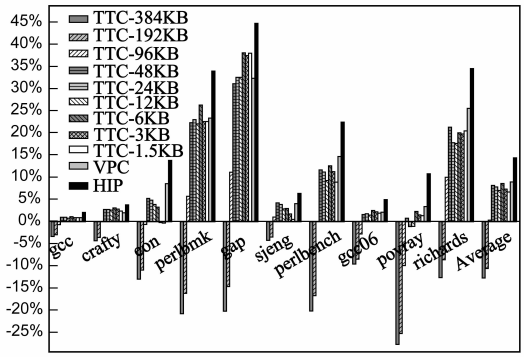


图10 HIP与其他技术的能耗降低比例比较

5 结论

本文提出并评测了一种基于提示值索引的间接转移预测技术.该技术利用现有分支预测部件,在 BTB 中保存间接转移目标地址,并通过结构重定义将与每个间接转移场景相对应的 Hint,融合并存入分支方向预测器中.该技术无需专用的大容量硬件和额外的编译器支持.评测结果表明,相对于基础结构,HIP 不仅显著提升了间接转移预测准确率,而且降低处理器能耗开销,达到了一种性能提高和能耗减少的双赢局面.

参考文献

[1] 沈立,王志英,鲁建壮,戴葵.基于控制流的混合指令预取[J].电子学报,2003,31(8):1141 – 1144.
Shen Li, Wang Zhi-ying, Lu Jian-zhuang, Dai Kui. Hybrid instruction prefetch based on control low [J]. Acta Electronica Sinica, 2003, 31(8): 1141 – 1144. (in Chinese)

[2] H Kim, J A Joao, O Mutlu, C J Lee, Y N Patt, R Cohn. VPC prediction: Reducing the cost of indirect branches via hardware-based dynamic devirtualization [A]. Proc of 34th Ann. Int'l Symp. Computer Architecture [C]. San Diego, CA, USA: ACM Express, 2007. 424 – 435.

[3] P-Y Chang, E Hao, Y N Patt. Target prediction for indirect jumps [A]. Proc of 24th Ann Int'l Symp Computer Architecture [C]. Denver, Colorado, USA: ACM Express, 1997. 274 – 283.

[4] B Calder, D Grunwald, B Zorn. Quantifying behavioral differences between C and C++ programs [J]. Journal of Programming Languages, 1995, 2(4): 323 – 351.

[5] S Gochman, et al. The Intel Pentium M processor: Microarchitecture and performance [J]. Intel Technology Journal, 2003, 7(2): 21 – 59.

[6] O Azizi, A Mahesri, BC Lee, SJ Patel, M. Horowitz. Energy-performance tradeoffs in processor architecture and circuit design: a marginal cost analysis [A]. Proc of 37th Ann Int'l Symp Computer Architecture [C]. Saint-Malo, France: ACM Express, 2010. 26 – 36.

[7] SPEC. Standard performance evaluation corporation [DB/OL]. http://www.spec.org, 2011.

[8] K Driesen, U Hölzle. The cascaded predictor: Economical and adaptive branch target prediction [A]. Proc of the 31st Annual Int'l Symp. Microarchitecture [C]. Dallas, Texas, USA: ACM Express, 1998. 249 – 258.

[9] T-Y Yeh, Y N Patt. Two-level adaptive training branch prediction [A]. Proc of the 24st Annual Int'l Symp. Microarchitecture [C]. Albuquerque, New Mexico, USA: ACM Express, 1991. 51 – 61.

[10] A Seznec, P Michaud. A case for (partially) TAgged GEometric history length branch prediction [J]. Journal of Instruction-Level Parallelism (JILP), 2006, 8(1): 1 – 23.

[11] T Austin, E Larson, D Ernst. SimpleScalar: An infrastructure

for computer system modeling [J]. IEEE Computer, 2002, 35 (2): 59 – 67.

[12] D Brooks, V Tiwari, M Martonosi. Wattch: A framework for architectural-level power analysis and optimizations [A]. Proc. of 27th Ann Int'l Symp Computer Architecture [C]. Vancouver, BC, Canada: ACM Express, 2000. 83 – 94.

[13] M. Wolczko. Benchmarking Java with the Richards benchmark [DB/OL]. http://research. sun. com/people/mario/java_benchmarking/richards/richards. html, 2011.

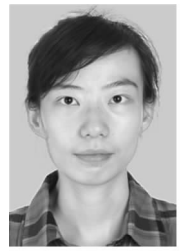
[14] E Perelman, G Hamerly, B Calder. Picking statistically valid and early simulation points [A]. Proc of the 12th Int'l Conference on Parallel Architectures and Compilation Techniques [C]. New Orleans, Louisiana, USA: ACM Express, 2003. 244 – 255.

[15] M U Farooq, L Chen, L K John. Value based BTB indexing for indirect jump prediction [A]. Proc of the 16th High Performance Computer Architecture [C]. Bangalore, India: IEEE Society, 2010. 1 – 11.

[16] S Thoziyoor, N Muralimanohar, J H Ahn, N P Jouppi. CACTI 5.1 [R]. Technical Report HPL-2008-20, 2008.

作者简介

谢子超　男,1984 年生于北京市. 现为北京大学信息科学技术学院博士后. 主要研究方向为微处理器微体系结构设计.
E-mail: xiezichao@gmail.com



史秦青　女,1988 年生. 曾获北京大学信息科学技术学院学士学位. 现为美国马里兰大学博士生.
E-mail: qshi@umd.edu