基于类型预测的甚块预测器

苟鹏飞 喻明艳 杨 兵 李清波 王诗博
(哈尔滨工业大学微电子中心 哈尔滨 150001)

摘 要 高性能的甚块预测器是保证EDGE体系结构性能的关键手段.为研究性能更好的甚块预测器,文中通过仿真实验发现甚块的出口类型独立于甚块的出口个数和甚块的动态执行结果而存在.以此为据,提出了基于类型预测的甚块预测器.该预测器摈弃了甚块出口号,直接对甚块出口类型进行预测.随后,根据对甚块出口类型可预测性的分析,通过实验证明甚块出口类型与历史和路径信息相关.仿真结果显示,与经典的基于出口预测的甚块预测器相比,文中提出的基于类型预测的甚块预测器能够将每千条指令误预测次数平均降低约10%.

关键词 甚块预测器;分支预测器;EDGE体系结构;出口类型预测;可预测性
中图法分类号 TP338 **DOI号**: 10.3724/SP.J.1016.2012.01539

Type-Only Hyperblock Predictor

GOU Peng-Fei YU Ming-Yan YANG Bing LI Qing-Bo WANG Shi-Bo
(*Microelectronic Center, Harbin Institute of Technology, Harbin* 150001)

Abstract Since EDGE architecture is hyperblock-based, high performance hyperblock predictors are crucial to guarantee promising performance of EDGE. Based on the analysis of hyperblocks' exit types, we find that the exit type of a hyperblock is independent on the exit ID, motivating us to propose a type-only hyperblock predictor that predicts exit types without exit ID. Analysis of the predictability of exit types for hyperblocks proves that exit types are correlated to histories and/or paths, permitting us to incorporate aggressive prediction techniques into type-only hyperblock predictors to harvest better performance. Compared with conventional exit-based hyperblock predictors, experiments show that MPKI (Mispredicts Per Kilo Instructions) of our proposal is able to outperform by 10%.

Keywords hyperblock predictor; branch predictor; EDGE architecture; exit type predcition; predictability

1 引 言

近年来,诸多文献都已表明,多核/众核技术实际上是微处理器设计者在面临处理器发展瓶颈后的一种无奈之举[1-2],其是否代表未来的方向还有待时间的检验[3-5].根据Amdahl定律[3],处理器的整体性能将仍然在很大程度上受制于其单线程处理能力.因此,进一步探索单处理器设计空间,依然是推动微处理器发展的动力之一.EDGE(Explicit Data Graph Execution)体系结构[6-7]是一种近年来学术界出现的能够改善单处理器能耗、设计复杂度和

收稿日期:2011-07-12;最终修改稿收到日期:2012-02-12.**苟鹏飞**,男,1983年生,博士研究生,研究方向为高性能计算机体系结构.E-mail: pengfeidaxia@gmail.com.**喻明艳**,男,1967年生,教授,中国计算机学会(CCF)会员,研究领域为超大规模集成电路设计、高性能计算机体系结构、模拟集成电路设计等.**杨 兵**,男,1976年生,博士,研究方向为高性能计算机体系结构、高性能编译器等.**李清波**,男,1985年生,硕士研究生,研究方向为高性能计算机关键路径分析.**王诗博**,女,1989年生,本科生,研究方向为指令块预测器.

性能的下一代体系结构. EDGE 体系结构通过块执行(Block-atomic Execution)和指令间显式通信(Direct Instruction Communication)的方式,为下一代体系结构提供了不同于传统 RISC/CISC 体系结构的思考角度. EDGE 体系结构的基本概念[8]、微结构细节[9]、实现方式[10]和整体性能评估[11]并不是本文的研究范围,因此不作详细介绍.

EDGE 体系结构以甚块(Hyperblock)而不是单个指令为基本执行单元[6],所有改变机器状态(寄存器和存储器)的行为都以甚块为单位发生. 甚块[12]是一种单入口、多出口(Single Entry, Multiple Exits)并且包含多个基本块的指令集合. 甚块使用谓词化(Predication)技术,将甚块内基本块之间的控制流转化为数据流,从而保证甚块中能够容纳尽可能多的指令. 甚块执行完毕时,通过出口(Exit)处的跳转指令,跳转到下一个甚块. 而甚块内部的指令则按照纯粹的数据流相关性执行. 为提高性能,EDGE 体系结构通过甚块控制流推测技术,为执行引擎同时提供多个推测的(Speculative)甚块,实现指令窗口的充分填充,从而最大限度地保证对指令级并行的发掘. 甚块控制流推测技术则是通过甚块预测器对下一甚块地址的预测来完成的.

在之前的研究成果中,EDGE 处理器使用基于出口预测的甚块预测器[13]. 该方法在预测时,首先使用出口预测器预测当前甚块的出口号(Exit ID),随后在该出口号基础上完成目标(Target)预测. 这种基于出口预测的方法同样被 Multiscalar[14] 等处理器使用,是目前学术界对指令块进行预测时的通行方法. 该方法中的出口预测与传统分支预测技术中的跳转方向预测相类似,因此,当前的研究人员将绝大部分精力投入到了设计出口预测器中. 然而,由于甚块具备单入口、多出口特性,使得出口预测器需要解决"多选一"问题,而不是传统分支预测器所面临的"二选一"问题,因此,诸多应用于传统预测器的二值预测技术需要经过不同程度的修改,才能够满足甚块出口预测的需要[13,15-16]. 从实验结果来看,尽管使用了诸多激进的预测技术,出口预测器仍然导致了甚块预测器中约 50%的误预测[13,16],是甚块预测器的性能瓶颈.

为设计性能更佳的甚块预测器,本文在对甚块行为及其出口类型进行统计分析后,提出了一种不使用出口预测器,直接对甚块出口类型进行预测的方案,称之为基于类型预测的甚块预测器(Type-only Hyperblock Predictor). 由于不使用出口预测器,简化了甚块预测步骤,从而能够提供更好的甚块预测性能. 总体来说,本文有如下 3 点贡献:

(1) 针对 EDGE 体系结构中的甚块预测问题,根据实验分析和统计结果,提出了一种不使用出口预测器,直接对甚块出口类型进行预测的方案.

(2) 对甚块出口类型的可预测性进行了研究,发现甚块出口类型与历史和路径信息相关.

(3) 构建基于类型预测的甚块预测器,并分析了其性能. 实验结果表明,在针对甚块出口类型的特点使用 TAGE 预测技术后,本文提出的基于类型预测的甚块预测器相对于基于出口预测的甚块预测器,将每千条指令误预测数(Mispredicts Per Kilo Instructions, MPKI)平均降低了约 10%.

本文第 2 节将对应用于 EDGE 体系结构的甚块预测器基本概念及研究现状进行简单介绍;第 3 节将描述甚块出口和出口类型的关系,分析甚块出口类型的统计特性,并对甚块类型预测的合理性进行阐述;第 4 节将给出基于类型预测的甚块预测器结构;第 5 节将介绍本文所使用的仿真方法和基本实验环境;在第 6 节中,将对甚块类型的可预测性进行分析,并比较基于类型预测的甚块预测器与基于出口预测的甚块预测器的性能.

2 背景知识和相关工作

分支预测器(Branch Predictor)是一种为高性能处理器提供推测执行能力的部件. 分支预测器通过预测分支指令的跳转方向,并配合相应的分支目标预测技术,在分支指令执行前获取其跳转结果,为处理器提供了无停顿的推测指令流,增大了处理器的取指能力、指令窗口填充能力,并相应地保证了性能.

与传统的、工作于指令粒度上的分支预测器一样,指令块预测器(Next-block Predictor)是一种为基于块的处理器(Block-atomic Processors)提供控制流推测能力的部件. 虽然指令块预测器有不同的研究背景[13-14,17-20],但其基本思想是一致的,即为每个指令块提供跳转目标预测,使得基于块的处理器可以同时取入多个推测的指令块. 通常来说,在基于块的处理器中,指令块由编译器根据特定的约束将多个基本块组合而成[6,21]. 类似的,EDGE 体系结构为了尽可能地增大指令块大小,使用了甚块技术. 甚块是编译器生成的具有谓词化指令(Predication)的"单入口、多出口"指令块. 甚块将多个传统指令集中

的基本块(Basic Block)集合到一起，因此其内部通常具有多条执行路径，不同的执行路径对应不同的出口.执行开始时，控制流从甚块唯一的入口进入，触发甚块的执行.执行结束时，甚块从其多个出口中，选择且仅选择一个出口作为控制流的转移点，开始下一甚块的执行.为了区分甚块中不同的出口，编译器为甚块的每个出口都分配唯一的出口号(Exit ID).出口号在本质上是编译器对甚块内部不同执行路径的近似[22]，用以表征构建甚块时被条件转换(If-Conversion)技术湮没的相关性.因此，从这个角度来看，用出口号能够从理论上还原某条执行路径的历史，从而使得对该路径的跳转结果进行预测变得可能.由于在对甚块进行预测时，最终的执行路径还未知，因此需要首先对出口号进行预测.甚块“多出口”的特性则对出口号预测提出了与传统分支预测器不一样的要求：

(1)由于具有多个出口，因此传统分支预测器的“2 选 1”预测技术需要被修改为“多选 1”预测技术.

(2)由于具有多种出口类型，因此甚块预测器需要对出口类型进行预测.

目前应用于 EDGE 体系结构中的甚块预测器使用了如图 1 所示的结构，该结构在 TRIPS 原型芯片中得到了应用[11].图 1 中，预测过程由两个步骤构成：出口预测和目标地址预测.首先，使用出口预测器预测甚块在执行完成后跳转出口的出口号.随后，该出口号与指令块地址一起，被用于预测该出口的类型.紧接着，根据出口的类型访问相应的分支目标缓存，产生最终的跳转地址.图 1 中的出口预测器使用了能效比较高的全局/本地锦标赛预测器，而文献[16]中则评估了多种二值预测技术的出口预测潜力，给出了多种不同的预测器结构.对于目标地址预测部分，除基于粘滞位表(Hysteresis)的出口类型预测

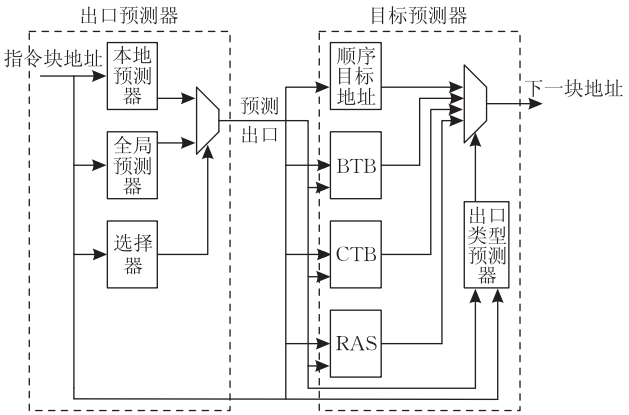


图 1　TRIPS 原型芯片中使用的指令块预测器结构[13]

器外，还有 3 种针对不同出口类型的跳转目标缓存，分别为：分支目标缓存(BTB)、函数调用目标缓存(CTB)和函数返回地址缓存(RAS).此外，如果跳转地址是当前甚块的下一个甚块(In Program Order)，则是顺序跳转目标，可以直接产生跳转地址.

这种先预测出口，再针对特定出口预测跳转目标的策略，是甚块预测中应用最广泛，也是最直观的结构之一[13-14,20].也正因为如此，当前针对甚块预测器的研究几乎都将目光聚焦到了出口预测上，而仅仅为出口类型预测提供非常有限的资源和极为简单的结构.本文将尝试打破这种格局，将目光转移到甚块出口类型预测上，并证明这种方法相对于基于出口预测的方法更加有效.

3　甚块出口类型预测技术

出口预测是当前甚块预测器的重要组成部分.甚块预测器将出口预测得到的出口号与指令块地址相结合，预测该出口的类型，并进一步选择相应的分支类型缓存，输出甚块的跳转地址.由于出口号本质上是编译器对甚块内部执行路径的近似，因此对出口号的使用实际上是利用甚块内部的执行历史来预测跳转类型和跳转目标.但是，由于出口号本身需要经过预测得到，因此一旦这种“推测(Speculative)”的出口号(或者说推测的甚块内部执行历史)出现误预测，对最终跳转目标的影响将呈现错误叠加的效果.从已发表的文献可以看出，约 50%左右的甚块误预测均由出口误预测导致[13,15-16]，体现了出口预测器的低效性.本节对出口预测的缺点进行了分析，并在此基础上，提出了一种摈弃出口预测器，直接进行类型预测的甚块预测方案.

3.1　出口预测器所面临的问题

在甚块预测器中，甚块跳转目标最终由针对不同类型的分支目标缓冲产生.与传统的二值分支预测不同，出口预测器预测得到的出口号并不能直接给出该甚块跳转目标的信息.其原因在于，传统的二值分支预测(主要是针对条件分支指令的预测)如果预测结果为“不跳转(Not-Taken)”，则跳转目标即是该分支指令的下一条指令，无需访问分支目标缓存.因此，传统的二值分支预测器在解决“2 选 1”问题后得到的答案中，有可能直接包含分支跳转目标信息.而对于甚块预测器的出口预测而言，解决“多选 1”问题后，仅仅得到了“推测”的出口号信息，除非进一步通过出口号预测该出口的类型，否则无法

从出口号本身获得甚块的跳转目标.正因如此,出口预测的作用体现在如下两点中:(1)使用预测得到的出口号区分同一甚块中不同出口的分支类型;(2)使用预测得到的出口号区分同一甚块中不同出口的跳转目标.结合图1中甚块预测器的典型结构,出口预测器的功能体现在下述的甚块预测器工作步骤中:

首先,出口预测器使用甚块地址、历史信息、地址路径信息等,预测某一甚块的出口号.

其次,将预测得到的出口号与甚块地址相结合,访问出口类型预测器,得到该出口的类型.

最后,根据出口的类型,选择相应的分支目标缓冲,并使用出口号和甚块地址对其进行索引,得到最终的跳转地址.

从上述步骤中可以看到,出口预测虽然无法直接产生最终的甚块跳转地址,但却影响着出口类型预测和目标预测的结果.一旦出口预测器性能低下,将极大地损害甚块预测器的整体性能.不幸的是,由于出口预测器需要完成"多选1"的使命,其性能相对于传统分支预测器的下降,是显而易见的.其原因在于传统的二值分支预测技术需要经过一定程度的修改才能被应用于出口预测器.这些修改包括:将每次预测产生的历史信息从1位变为多位[13-14,17]、改变预测结果的编码方式(增加计数器位数[13-14,17]、使用PPE预测方法[16])、修改预测器的决策方案(例如将OGEHL预测器的加法树更改为多数投票[16])等.最为重要的是,由于出口预测器的相关信息(执行历史、地址路径等)需要在甚块粒度上获取,相对于基于基本块的分支预测器,其历史信息有可能会丢失.这些因素使得出口预测器的性能变得并不尽如人意,从而使得甚块预测器的整体性能受到影响.

值得注意的是,虽然"多选1"的出口预测器存在上述弊端,但之前的研究人员几乎都倾向于在甚块预测器中保留出口预测器,并尝试使用各种激进的预测技术来改善其性能.这些尝试建立在这样的事实基础之上:甚块中的每个出口分别属于其内部不同的执行路径,由不同的基本块构成,出口号表征了这种湮没在甚块中的路径历史信息.对甚块出口的预测与传统分支预测器对基本块控制流的预测,在本质上是一致的,因此通过改善出口预测的性能来提高预测器整体性能,是十分合理的.

结合甚块预测器本身的工作流程可以发现,即使有上述事实的存在,出口预测器仍然需要额外的条件才能高效地工作:甚块中每个出口都拥有不同的出口类型和不同的跳转目标,通过出口号(或者说甚块内部的执行路径)对其进行区分是必要的.当前的研究人员通常都默认该条件成立,但事实上,由于编译器、指令集、体系结构等诸多因素的影响,情况并非如此.本文将说明在EDGE体系结构中甚块的出口类型与出口号并无直接联系,并且能够在不需要出口预测器的情况下直接对出口类型进行预测.

3.2 甚块出口类型特征

本节将对甚块的出口类型进行分析,以所使用的TRIPS指令集为基础对甚块进行分类,其细节以及相应的仿真方法和测试程序集在第5节中有详细描述.由于使用了TRIPS指令集,因此每个甚块最多拥有4种出口类型,分别为:顺序目标(Sequential Target)、普通分支目标(Branch Target)、函数调用目标(Function Call Target)和函数返回目标(Function Return Target)[10].某个甚块可能具有多个出口,且每个出口的类型可能不同.但由于在执行时,对该甚块的每一次调用(动态实例)都只能使用一个出口,并产生一种特定的出口类型,因此,为分析EDGE体系结构中甚块的出口类型特征,本文首先将甚块按照其动态实例所使用的出口类型分为6大类.分别为

类型1~4. 甚块的所有动态实例(Dynamic Instances,即执行时的多次调用)都只产生同一种类型的出口.根据TRIPS指令集[10],这将定义4类甚块,分别为只产生顺序目标的甚块、只产生普通分支目标的甚块、只产生函数调用目标的甚块、只产生函数返回目标的甚块.

类型5. 甚块的动态实例既可能产生顺序目标,也可能产生普通分支目标.

类型6. 其它的情况.

图2展示了11个SPEC CPU2000程序中,对于动态实例个数排名前100的甚块,按照上述分类方法分类后不同类型甚块所占的比例.从图2中,可以得到

(1)平均约有近80%的甚块在执行过程中只产生某种特定的出口类型.也就是说,无论这些甚块从哪个出口跳转,其出口类型都将始终保持一致.

(2)平均约有近20%的甚块在执行过程中,即可能产生顺序目标,亦可能产生普通分支目标.这些甚块的行为与普通的条件分支指令非常类似.即如果这些甚块产生顺序目标,那么其跳转地址将可以直接得到,无须访问任何分支目标缓存,这与普通条件分支指令的"不跳转"情况类似;如果这些甚块产

生普通分支目标，那么最终的跳转目标地址将通过访问 BTB 得到，这同样与普通条件分支指令的“跳转”情况类似.

(3) 其它类型的甚块数目微乎其微，小于 1%.

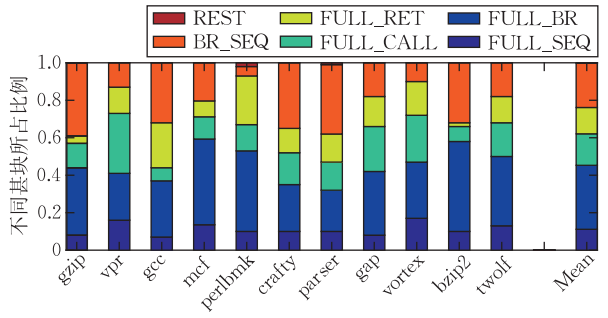


图 2　甚块的出口类型特征统计(其中，FULL_SEQ、FULL_BR、FULL_CALL、FULL_RET 分别表示动态实例只产生某种特定类型出口的甚块；BR_SEQ 表示动态实例中既产生顺序目标又产生普通分支目标的甚块；REST 为其它所有情况)

从上述现象能够得出这样的结论：尽管文献[16]中的数据说明甚块的出口号分布较为均匀①，但其出口类型具有较强的特征，且该特征自成体系. 进一步观察可以看到，其中大部分甚块(约 80%)的偏向性极强. 由于这种极强的偏向性，这些甚块的出口类型在不需要出口预测器的情况下，通过简单的、不使用相关信息的粘滞位表就能够很好地预测. 本文设计了实验来初步验证上述结论. 实验中将 TRIPS 原型芯片甚块预测器中的出口预测器删除，直接使用简单粘滞位表(仅使用甚块地址来索引该表，不使用历史信息)来预测甚块出口类型，与原有的 TRIPS 原型芯片预测器进行性能比较. 原有的 TRIPS 原型芯片预测器配置与文献[13]中相同. 两种方案均为出口预测器和类型预测器分配 32 KB 的资源，并使用每千条指令误预测数(Mispredictions Per Kilo Instructions，MPKI)作为量化标准，这也是评估预测器性能的常用指标[13,23]. 比较结果如图 3 所示，图中给出了对 11 个 SPEC CPU2000 整型程序的 MPKI 剖析，将 MPKI 归结为不同的来源，分别为：出口预测器(EXIT)、出口类型预测器(TYPE)、普通分支缓存(BTB)、函数调用目标缓存(CTB)和函数返回缓存(RAS).

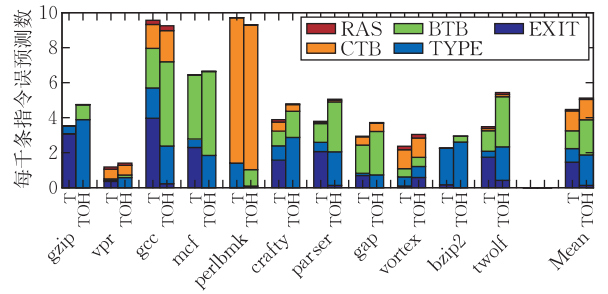


图 3　使用简单粘滞位表对甚块进行预测的结果(其中 T 为 TRIPS 原型芯片甚块预测器；TOH 为无出口预测器且仅使用简单粘滞位表进行出口类型预测的甚块预测器)

从图 3 中可以看到，相对于原有的 TRIPS 原型芯片预测器而言，删除了出口预测器且只用了简单粘滞位表进行甚块出口类型预测的方案，其 MPKI 平均仅仅上升了 8% 左右. 更为重要的是，原有的 TRIPS 原型芯片预测器中由出口预测器和类型预测器导致的 MPKI，在无出口预测器的方案中，转换为了几乎等量的由类型预测器导致的 MPKI，且有约 10% 的下降. 该现象说明，甚块的出口类型在无出口预测器的情况下就能够进行预测，并且凭借简单的仅用甚块地址进行索引的粘滞位表就能够获得较好的预测效果. 这与前述 EDGE 体系结构中约 80% 甚块的出口类型具有极强的偏向性是相符的. 另外约 20% 的甚块由于其行为与条件分支指令行为类似，因此，可以大胆假设这些甚块的出口类型与甚块“间”历史或地址路径信息(而不是出口号)相关，使用更为复杂的相关性预测技术能够对出口类型进行较好的预测. 本文将在第 6 节中通过实验结果来证明这个假设.

总而言之，本节的实验数据说明，甚块的出口类型呈现出独立于出口号的特征，这样的特征使其可在不使用出口号时，直接被出口类型预测器所预测. 因此，前述甚块中每个出口都拥有不同的出口类型，通过出口号对其进行区分是必要的这个假设，实际上在 EDGE 体系结构中并不成立. 进而可以认为，至少对于甚块的类型预测来说，出口预测器并不是必要的.

值得注意的是，图 3 中，虽然删除出口预测器后甚块出口和出口类型的预测性能提高了，但总体 MPKI 还是有约 8% 的上升，这是由 BTB 所导致的. 由于在无出口预测器的方案中，仅使用了甚块地址来索引 BTB，因此 BTB 无法区分同一个甚块产生的不同跳转目标. 这种现象说明，甚块中不同的出口确实会产生不同的跳转目标；甚至同一出口，同样的出口类型，也会由于间接跳转指令的存在[10,16]，产生不同的跳转目标. 虽然这个现象部分说明前述甚块中每个出口都拥有不同的跳转目标，通过出口号对其进行区分是必要的这个假设成立，但该问题实际可以通过使用相关性分支目标缓冲技术来解决

① 实验数据也说明出口号的分布较为均匀.

(Correlative BTB)[14,24-25],即将甚块地址和全局历史或地址路径信息结合到一起来索引 BTB.由于这并不是本文的研究目标,因此并不对这个问题作详细阐述.尽管如此,本文随后的实验将证明即使不使用相关性分支目标缓冲,本文所提出的方案依然具有较好的性能.

3.3 如何看待甚块类型预测的合理性

为更形象和深入地理解无出口预测的甚块类型预测,需要进一步分析甚块的跳转行为.实际上,可以将甚块的跳转行为看作具有间接跳转目标的条件分支指令,如图 4 所示.从图中可以看到,无论甚块拥有多少出口,从哪个出口跳转,最终呈现的结果无外乎两种情况:(1)跳转到该甚块在程序上的下一甚块;(2)跳转到其它甚块.前一种情况,即是出口类型预测中预测到顺序分支目标的情况;而后一种情况,可以进一步细分为 3 种不同的分支类型,并通过不同类型的分支目标缓冲(在 TRIPS 指令集中,分别是 BTB、CTB、RAS)来得到跳转目标.因此,从甚块是否为顺序分支目标这个角度来看,可以将其看作具有"跳转"或"不跳转"行为的条件分支指令.而从甚块跳转时需要根据不同的执行上下文来确定跳转目标的角度来看,则可以将其看作间接跳转指令.以这样的角度为基础,直接使用甚块出口类型构成的历史则与条件分支指令的跳转历史相似.该历史表征了甚块的跳转行为,可被出口类型预测器用于更准确地预测甚块出口类型,从而在不需要出口号的情况下完成甚块跳转地址的预测任务.这就能够避免传统甚块预测器中低效的出口预测器对出口类型预测性能的影响,直接将重点聚焦到甚块类型预测上,实现问题的简化,提供更多的性能提升可能.

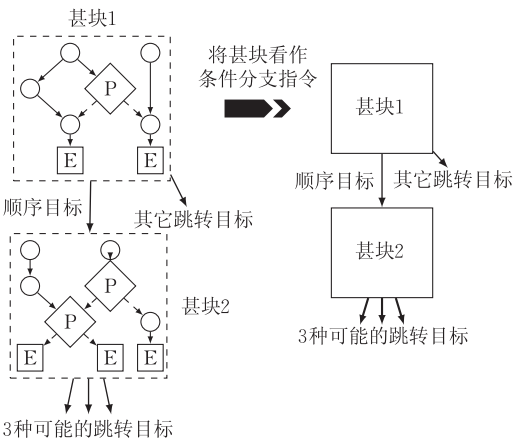


图 4 将甚块看做间接跳转的条件分支指令

从分支历史相关性角度来看,取消出口预测事实上是在预测过程中忽略了甚块"内"的相关信息,这种相关信息对被预测的甚块而言是"推测"的,不确定的;而直接使用甚块的出口类型构成历史,则是使用甚块"间"的相关信息来指导预测器的工作,这种相关信息是既有的、确定的.本文随后的实验将证明,使用甚块"间"的相关信息能够获得更好的甚块预测性能.

本节通过上述分析得到了如下 3 点结论.首先,传统甚块预测器中的出口预测器是整体性能的瓶颈,并且仅仅产生预测过程的中间结果,不包含直接的甚块跳转目标信息;其次,甚块的出口类型具有独立于甚块出口号的特性,并具备被出口类型预测器直接预测的潜力;再次,甚块的出口类型表示了甚块"间"的相关性,能够在没有出口号的情况下完备地描述甚块的跳转行为.综合这 3 点因素,可以认为,在没有出口预测器的情况下直接对甚块的出口类型进行预测,是一种可行的、并具有性能提升潜力的方案.鉴于此,本文将在该思路的指导下,提出基于类型预测的甚块预测器结构.

4 基于类型预测的甚块预测器结构

4.1 总体结构

本节根据上述分析结果,提出一种基于类型预测的甚块预测器结构.由于甚块出口类型也有多种可能(TRIPS 指令集定义了 4 种),因此需要使用与出口预测器类似的多值预测器.具体的做法是,使用"4 选 1"类型预测器从顺序目标、普通分支目标、函数调用目标和函数返回目标中确定当前的预测结果,并根据预测结果访问相应的分支目标缓存,得到最终的跳转目标.这种类型预测器所使用的历史信息与传统分支预测器中所使用的二值历史信息类似,由每个甚块产生的出口类型组成,表征了甚块"间"的执行历史.与传统甚块预测器所使用的出口号历史信息不同,这种历史信息仅仅保留甚块间的相关性,忽略了甚块内的执行路径信息.由于一共需要预测 4 种类型,因此每一个甚块会产生 2 位历史.地址路径则是由执行过程中甚块的地址构成,表征了执行轨迹上已执行的甚块地址.

该方案依然使用多值预测结构,可能会由于多值预测本身的缺点造成性能损失,但由于直接使用了甚块"间"相关信息对出口类型进行预测,因此并不会受到出口预测器的干扰.另一方面可以将有限

的资源全部投入到出口类型预测器中，提高改善性能的可能性. 具体的预测器结构如图 5 所示. 图中使用与出口预测器类似的多值预测器，在历史或地址路径信息的帮助下直接对出口类型进行预测. 如果甚块类型预测器预测结果为顺序目标，则直接产生预测的跳转目标，如果是其它 3 种类型，则根据相应的分支目标缓冲产生预测结果.

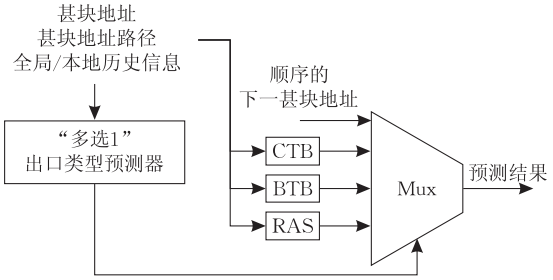
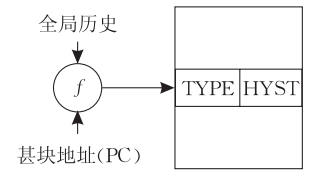


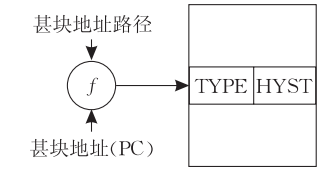
图 5　基于类型预测的甚块预测器结构

4.2　使用历史信息和甚块地址路径的甚块类型预测器

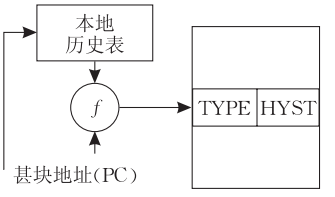
图 5 中的多值预测器可用类似经典 gshare[26] 预测器的结构实现，如图 6 所示. 该结构将甚块地址和全局/本地历史信息（或者甚块地址路径信息）相结合来索引相关性表，是一种利用历史和地址路径相关性进行分支预测的经典结构. 图 6(a)～(c)分别为全局历史、全局路径和本地历史预测器，分别使用相应的历史和地址路径信息与甚块地址按照一定的方法（通常是异或）结合后，对样本历史表（Pattern History Table）进行索引①. 表中每一项由两部分组成，第一部分存储出口类型（TYPE），即当前访问该项能够得到的预测结果；另一部分则是使用饱和计数器实现的粘滞位（HYST），用于更新预测结果. 值得注意的是，该结构中历史信息由每一个甚块产生的出口类型构成，而地址路径信息则由执行轨迹上的甚块地址构成. 与传统分支预测器或出口预测器一样，图中的 3 种预测器方案可以两两组合为锦标赛预测器. 另外，在对甚块出口类型的历史和路径相关性进行研究时，这 3 种预测器也是最具代表性的结构.



(a) 全局历史预测器



(b) 全局路径预测器



(c) 本地历史预测器

图 6　全局历史预测器、全局路径预测器和本地历史预测器

4.3　使用 TAGE 结构的甚块类型预测器

在传统的二值分支预测器中，TAGE（TAgged GEometric history）预测器[23] 是硬件可实现的性能最好的预测器之一. 文献[13]和文献[16]中就利用 TAGE 预测技术，实现了基于出口预测的甚块预测器. 对出口类型预测器来说，TAGE 预测技术同样能够提供新的设计思路.

TAGE 预测技术使用数个不同的历史样本表，分别由呈几何级数增长的历史长度索引，并通过精巧的更新机制和标签（TAG）匹配机制，使得与不同历史长度相关的分支指令可以被相应长度的历史所预测，从而避免不同历史样本之间的别名冲突. 根据 TAGE 预测技术的特点，针对 3.2 节中描述的甚块出口类型特性，本文使用 TAGE 预测技术设计了一种 TAGE 甚块出口类型预测器，可用于图 5 中的多值预测，其结构如图 7 所示. 该 TAGE 甚块类型预测器由简单粘滞位表 T_0（仅使用甚块地址进行索引，历史长度为 0）以及一组历史长度呈几何级数增长的历史样本表 $T_1 \cdots T_n$ 组成，该几何级数历史长度由 $L(j)=\alpha^{(j-1)}L(1)$[23] 计算得到. 与经典 TAGE 预测技术类似，表 $T_1 \cdots T_n$ 中每一项由 3 部分组成：预测结果（TYPE）、粘滞位（HYST）和标签（TAG），T_0 表中则没有标签位. 更新机制和标签匹配机制与文献[23]中相同. 根据 TAGE 预测技术的特点，结合图 2 中给出的甚块出口类型特性，TAGE 甚块类型预测器可以达到如下效果：(1) 甚块中 80% 偏向性极强、跟历史信息无关的甚块，将由简单粘滞位表 T_0 来预测；(2) 剩余 20% 与历史相关的甚块，将分别由 $T_1 \cdots T_n$ 中相应的表来预测. 这就使得不同特性甚块之间发生别名冲突的可能性大大减小，从而提高预测性能. 随后的章节中，将对本节所提出的甚块类型预测器进行性能评估.

① 虽然不同预测器结构有所不同，但通常都有类似样本历史表（PHT）的结构.

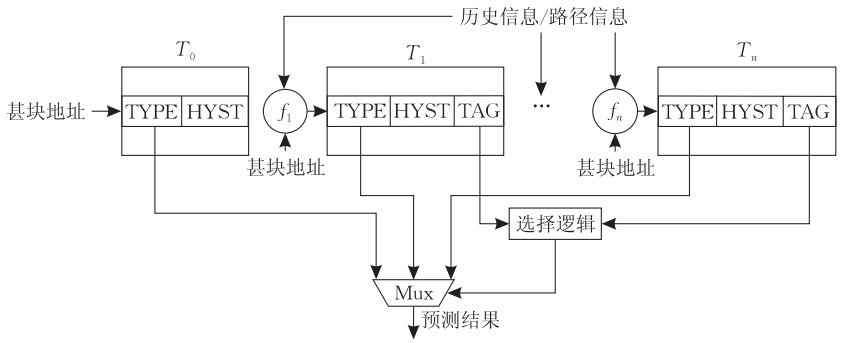


图 7 基于 TAGE 技术的甚块出口类型预测器

5 仿真环境和评估方法

由于本文的诸多分析都基于实验方法得到,因此,在进行详细的性能评估前,本节将首先描述所采用的仿真环境和性能评估方法. 本文所有的实验数据都将在本节所描述的仿真工具、基本配置参数和测试程序集基础上得到.

5.1 仿真工具

本文使用 M5_EDGE[27]模拟器作为仿真工具. M5_EDGE 是基于 M5[28]模拟器,使用 C++/Python 构建的 EDGE 体系结构仿真工具,支持 TRIPS 指令集[10],具有高层次的 4 级时序模型,能够在微结构层面对 EDGE 体系结构进行快速的设计空间探索. 得益于 M5_EDGE 良好的面向对象和模块化特性,本文能够方便地对甚块预测器进行建模,并在统一的仿真框架和基准配置参数下进行比较. 在仿真时,所使用的基准配置如表 1 所示,文中所涉及到的预测器都将在本框架下实现. 为了消除处理器其它部分对预测器的影响,操作数网络(Operand Network)、谓词化指令(Predicated Instruction)、Cache 和访存顺序(Memory Access Order)都被配置为理想状态,即仿真时操作数网络延迟为 0、甚块中的谓词化指令在执行前就已经获取了结果、Cache 总是命中、访存顺序在执行前就已经获得.

表 1 基准处理器配置参数

取指	每周期 16 条指令(一个 Cache Line)
指令映射策略	静态映射,并有 1 个周期的映射延迟
指令分派开销	1 周期
指令窗口大小	1K 指令(8 个 TRIPS 的 Hyperblock)
操作数网络延迟	完美
执行引擎	每周期最多 16 条指令能被同时执行
谓词化	完美
提交延迟	1 周期
Cache	完美
Memory 访问顺序	完美

5.2 指令集、编译策略和测试程序

本文选择 TRIPS 指令集的原因在于:其一,TRIPS 指令集是开发较为完善的 EDGE 指令集,工具链较为完善;其二,TRIPS 指令集在 M5_EDGE 中有较好的支持. 在编译时,本文使用了 TRIPS 工具链中 tcc 的-Omax 优化选项[29]. 对于有出口预测的仿真,编译器将为每个甚块的出口按照正常的策略分配出口号;对于只有类型预测的仿真,编译器将为每个甚块的出口都分配出口号 0.

本文使用 SPEC CPU2000 中能够被 TRIPS 工具链正确编译的 11 个整形测试程序作为测试基准. 之所以这样选择是因为整形程序对于控制流推测的要求较高,能够更好地体现甚块预测器特性,而浮点程序对于甚块预测器的性能则不是那么敏感. 在仿真时,选择了 ref 输入集,并使用 Simpoint[30]仿真方法,以便将仿真时间缩短到可接受的范围之内.

6 实验结果分析与性能评估

本节将在第 5 节所示的实验方法基础上,使用不同方案来实现甚块出口类型预测器,并观察、分析和比较其性能. 首先,本节将使用无冲突(Interference-free)的全局预测器、地址路径预测器和本地预测器对甚块出口类型的可预测性及其与历史/地址路径信息的相关性进行分析. 随后,将比较本文提出的基于类型预测的甚块预测器和传统的基于出口预测的甚块预测器的性能. 由于本文主要关注二者性能的差异,因此本节实验所使用的分支目标缓冲大小均固定,分别为 BTB 拥有 4096 个入口、CTB 拥有 128 个入口、RAS 拥有 64 个入口,每个入口的具体实现与文献[13]中保持一致.

6.1 甚块出口类型的可预测性分析

使用无冲突(Interference-free)表对分支指令的可预测性进行研究是一种由来已久的方法[31-32].

文献[16]中，作者使用类似的方法对 EDGE 体系结构中基块出口的可预测性进行了研究.通过使用无冲突表，预测器可以在表中为每一个样本(pattern)分配独立的入口(Entry)，从而使得不同的样本之间不存在别名(Aliases)冲突，将历史或地址路径对预测能力的影响最纯粹地反映出来.

图 8 中给出了使用图 6 中 3 种预测器时，11 个 SPEC CPU2000 整型程序在历史/地址路径信息的长度从 0 比特变化到 63 比特时的平均 MPKI，其中每种预测器都使用了无冲突表.对于全局历史预测器和地址路径预测器，历史/路径和基块地址使用了两种方式进行组合.一种为使用 gshare 中给出的异或(XOR)方式，另一种为拼接(Concatenated)方式，即从基块地址和历史/路径信息中各取一部分，互不干扰地组合起来形成表的索引.本地预测器中，历史和路径按照异或方式结合.地址路径信息使用执行轨迹上基块地址的低 4 位组合而成，而历史信息中每个基块产生 2 比特历史，对应着基块出口类型的 2 位编码(TRIPS 指令集中共 4 种出口类型).

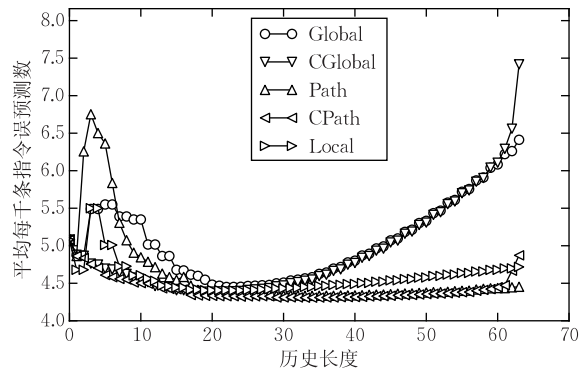


图 8　对于基于类型预测的基块预测器，使用无冲突样本表时全局历史、全局路径和本地预测器的 MPKI 变化情况(其中，Global/CGlobal 分别为使用异或策略和拼接策略的全局历史预测器；Path/CPath 分别为使用异或策略和拼接策略的地址路径预测器；Local 为本地预测器)

图 8 中的结果显示：当历史长度为 0 时，即仅使用基块地址对表进行索引时，所有预测器的 MPKI 相等，约为 5.07.由于本实验中的预测器使用无冲突表，因此这也是使用简单粘滞位表进行基块出口类型预测的极限值.从第 3.2 节中对基块出口类型分布特征的分析可知，当历史长度为 0 时预测器主要对约 80%具有极强偏向性的基块提供准确预测.而与图 3 中使用简单粘滞位表(32 KB)进行基块出口类型预测的结果(平均 MPKI 为 5.15)相比，该极限值仅仅下降了约 1.6%.这说明使用大小为 32 KB 的粘滞位表已经足够对约 80%具有极强偏向性的基块出口类型进行预测.如希望进一步提升预测率，只能寄望于改善另外约 20%基块的预测性能.

随着历史长度的增长，有两种不同的趋势.其一，对于使用拼接方式进行索引的预测器，MPKI 随着历史长度的增长持续而稳定地下降.这说明两层含义：(1)基块出口类型与全局历史/地址路径信息相关；(2)该相关性可以被出口类型预测器捕捉.其二，对于使用异或方式进行索引的预测器，MPKI 在短暂的下降(历史长度小于约 2 bit)后，突然上升，直到历史长度继续增加到一定程度(全局历史预测器为大于 13 bit，地址路径预测器为大于 9 bit，本地历史预测器为大于 5 bit)，MPKI 才重新下降到比不使用历史/路径相关信息(0 bit 历史长度)时更小的程度.这说明，使用异或索引方式时，在历史/地址路径长度较短的情况下，预测器能够捕获部分基块的历史/地址路径相关性.当历史/地址路径长度增加时，在某个区间段内，由基块出口类型自身特点和异或索引方式共同作用导致了较多的样本重复(即虽然历史不同且基块地址不同，但异或之后的索引却相同)，使得 MPKI 出现了上升趋势.这样的上升趋势在历史/路径长度增大到一定程度时得到了抑制，这说明当历史长度足够时，出口类型预测器依然有能力利用历史/地址路径相关性获得性能提升.

当历史长度继续增加时，所有预测器都在历史长度 15 bit 到 25 bit 之间达到性能极限，MPKI 的最小值为 4.32(使用拼接索引的全局路径预测器，历史长度 20)，相对于历史长度为 0 时有约 14%的下降.之后，所有预测器的 MPKI 都出现不同程度的上升，且全局历史预测器相对于地址路径预测器上升更为明显.这说明基块出口类型在本文的实验条件下，与距离太远的基块(对于本地预测器，则是相距太久的动态实例)之间的相关性并不明显，且地址路径信息相对于历史信息更能反映较远距离基块的相关性.另外，使用拼接方式进行索引的预测器在历史长度增加到 60 之后，MPKI 有较为明显的突然增加.这主要是由于最大索引长度固定为 64，从而使得基块地址信息在历史和地址路径信息太长后造成了丢失，导致索引样本的差异化减弱，增加了误预测次数.这也说明，基块类型不能单纯使用历史或地址路径信息进行预测.

总体来看，对全局预测器而言，使用地址路径信息比使用历史信息性能更优.地址路径预测器的 MPKI 最小为 4.32(使用拼接索引的全局路径预测

器),而全局历史预测器的 MPKI 最小为 4.39,上升了约 1.5%.整体而言,地址路径预测器的 MPKI 亦小于全局历史预测器.另外,本地预测器性能则介于地址路径预测器和全局历史预测器之间.综上所述,本节通过分析使用无冲突表的全局历史/地址路径预测器和本地历史预测器在不同历史长度下的性能,能够得出如下结论:

(1)甚块出口类型与甚块"间"的历史/地址路径信息(全局/本地)相关,在忽略甚块"内"的相关信息(出口号)后,利用这些甚块"间"的相关信息能够获得预测性能的提升.这证明了第 3.2 节的猜测:对于行为与条件分支指令类似的甚块(约占 20%),其出口类型与甚块"间"的历史/地址路径信息相关.

(2)在预测甚块出口类型时,使用地址路径信息比使用全局历史信息更有效,且本地历史预测器也能够获得可观的预测准确率.

6.2 基于类型预测的甚块预测器与基于出口预测的甚块预测器性能比较

上节的分析证实了甚块出口类型与甚块"间"的历史/地址路径信息相关,因而可以利用这些信息来提升其预测性能.本节将从性能的角度比较传统的基于出口预测的甚块预测器和本文所提出的基于类型预测的甚块预测器.比较时,对于基于出口预测的甚块预测器,将使用文献[13]中所提出的全局/本地锦标赛出口预测方案,其具体结构如图 1 所示.这也是基于出口预测的甚块预测器在相对简单的结构下能达到较好性能的方案.对于基于类型预测的甚块预测器,将在图 5 所示结构的基础上,使用全局/本地锦标赛预测器实现其中的"多选 1"类型预测器,直接对 4 种甚块出口类型进行预测.比较时为出口预测器和类型预测器分配的资源将从 1KB 变化到 512KB,资源的大小根据每个表的入口数和每个入口的比特数来确定,具体的配置信息如表 2 所示.

表 2 中,G/L EXIT 为基于出口预测的全局/本地锦标赛甚块预测器,其中 $G/L/C$ 分别为其中的全局历史长度、本地历史长度和选择器历史长度,相应的样本表入口数为 $2^{G/L/C}$,H 则为该预测器中用于在出口预测后进行类型预测的粘滞位表入口数;H TYPE为基于类型预测的仅使用简单粘滞位表的甚块预测器,其中 H 表示其粘滞位表的入口数;P/L TYPE 为基于类型预测的全局路径/本地锦标赛甚块预测器,其中 $L/G/C$ 分别为本地历史长度、地址路径长度和选择器历史长度,相应的样本表入口数为 $2^{L/G/C}$.另外,对于预测出口的样本表来说,其每个入口由 3 比特出口号信息和 1 比特粘滞位构成;对于预测出口类型的样本表来说,其每个入口由 2 比特出口类型信息和 1 比特粘滞位构成;对于选择器样本表来说,每个入口由 3 比特粘滞位构成.

表 2 性能比较配置表

	G/L EXIT				H TYPE	P/L TYPE		
	G	L	C	H	H	L	G	C
1KB	10	7	10	512	2KB	8	10	10
2KB	11	8	11	1K	4KB	9	11	11
4KB	12	9	12	1K	8KB	10	12	12
8KB	13	10	13	2K	16KB	11	13	13
16KB	14	11	14	2K	32KB	12	14	14
32KB	15	12	15	4K	64KB	13	15	15
64KB	16	13	16	4K	128KB	14	16	16
128KB	17	14	17	4K	256KB	15	17	17
256KB	18	15	18	4K	512KB	16	18	18
512KB	19	16	19	4K	1MB	17	19	19

图 9 中给出了针对 11 种 SPECCPU2000 测试程序的平均 MPKI.作为对比,基于类型预测的甚块预测器使用了两种方案:(1)地址路径(使用拼接索引方式)/本地预测器(PATH/LOCAL_TYPE)锦标赛方案,这也是在对甚块出口类型的可预测性进行分析后,得出的最优组合;(2)使用简单粘滞位表(即历史长度为 0,仅使用甚块地址进行索引)进行甚块出口类型预测的方案(HYST_TYPE),该方案用于展示无相关信息时,甚块出口类型预测能力的变化.另外,对于基于类型预测的地址路径/本地锦标赛甚块预测器,还引入了使用无冲突表(IF_PL_TYPE)时的情况(该方案中历史长度与图 9 中普通方案相同,但使用无冲突表),以便观察使用锦标赛预测器时甚块出口类型预测的极限.

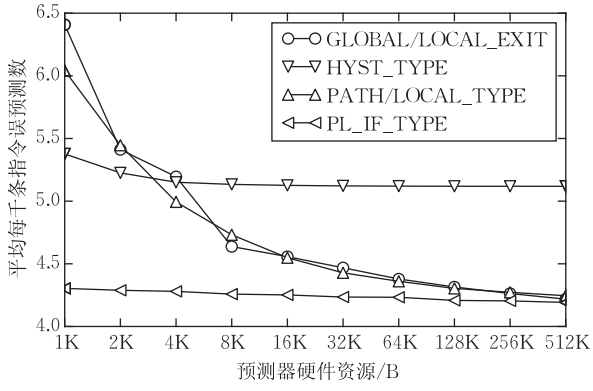


图 9 基于类型预测的甚块预测器和基于出口预测的甚块预测器性能比较(其中,GLOBAL/LOCAL_EXIT 为基于出口预测的全局路径/本地锦标赛甚块预测器;HYST_TYPE 为基于类型预测的仅使用简单粘滞位表的甚块预测器;PATH/LOCAL_TYPE 为基于类型预测的地址路径/本地锦标赛甚块预测器;PL_IF_TYPE 为使用无冲突表的基于类型预测的地址路径/本地锦标赛甚块预测器)

图 9 中的结果说明：首先，当资源从 1 KB 变化到 512 KB 时，对基于类型预测的地址路径/本地锦标赛甚块预测器而言，其 MPKI 和基于出口预测的全局历史/本地锦标赛甚块预测器几乎相等. 除在 1 KB 时，基于类型预测的方案能够将 MPKI 降低约 8%外，其余情况下，两者性能持平. 其次，在资源最小(1 KB)时，基于类型预测的简单粘滞位表甚块预测器性能最好，相对于基于出口预测的全局历史/本地锦标赛甚块预测器，MPKI 降低了约 15%，相对于基于类型预测的地址路径/本地锦标赛甚块预测器，MPKI 降低了约 10%. 其原因在于，当资源受限时，由于甚块出口类型本身较强的偏向性(约 80%的甚块只产生一种特定的出口类型)，无历史信息、仅使用甚块地址对甚块出口类型进行预测的简单粘滞位表依然能够获得可观的预测准确率. 而此时，由于资源较少，历史样本表入口数受限，因此如果使用地址路径/本地锦标赛甚块预测器进行出口类型预测，会因为相关信息的加入而导致太多不必要的别名冲突，从而增加了误预测次数. 对于基于出口预测的甚块预测器而言，资源受限使得出口误预测次数增多，并因此导致更加低效的出口类型预测和跳转目标预测，使得其总体性能在资源受限时最差. 但是，使用简单粘滞位表的方案由于不使用相关信息，因此其虽然在资源很小时能够提供较好的性能，但随着资源的增大很快就陷入饱和(资源大于 4 KB 后)，从而无法提供持续的性能增长. 最后，基于类型预测的地址路径/本地锦标赛甚块预测器与使用无冲突表的方案相比，其 MPKI 还有较大差距，该现象在资源小于 128 KB 时极为突出. 这一方面说明在资源受限时，地址路径/本地锦标赛方案确实由于别名冲突的增多而导致性能下降；另一方面，该现象也说明，如果能够寻找到更高效的结构，则可能将 MPKI 进一步降低.

6.3　使用 TAGE 预测器提升甚块类型预测器性能

正如前述，对于甚块出口类型的预测而言，如果能够找到更优秀的方案，那么其性能还有较大的上升空间. 图 7 中给出的 TAGE 甚块类型预测器正是这样一种潜在的方案. 为分析 TAGE 甚块出口类型预测器的性能，本节在图 5 中所示的基于类型预测的甚块预测器结构基础上，用 TAGE 甚块出口类型预测器实现其中的“多选 1”预测器. 另外，传统的 TAGE 预测器通常使用全局历史/地址路径进行索引，而为了利用本地历史信息，本节仿照锦标赛预测器的方式，将 TAGE 类型预测器和本地类型预测器结合到一起. 作为对比，引入了在文献[13]中使用的基于出口预测的 TAGE/本地甚块预测器(在具体实现时，该预测器同样将 TAGE 出口预测器和本地出口预测器以锦标赛预测的方式结合). 这两种预测器的具体配置如表 3 所示. 其中，所有的配置都使用了典型的有 5 个表的 TAGE 结构[23]. 表中，T/L TYPE 是基于类型预测的 TAGE/本地甚块预测器，其中 L/C 分别为本地历史长度和选择器历史长度，相应的样本表入口数为 $2^{L/C}$，$\alpha/l(1)$ 分别为 TAGE 预测器历史长度计算参数，相应样本表 $T_1\cdots T_n$ 的大小由计算得到的历史长度按照与本地历史预测器和选择器相同的方式得到，D 为 TAGE 预测器中默认表 T_0 的入口数；T/L EXIT 是基于出口预测的 TAGE/本地甚块预测器，其中 $L/C/\alpha/l(1)/D$ 参数之含义与 T/L TYPE 中一致，样本表的入口数计算方式亦相同，H 为在出口预测之后，用于类型预测的简单粘滞位表入口数. 在本配置中，本地历史预测器和选择器的样本表入口配置与表 2 中一致. TAGE 预测器的样本表中，如果是进行出口类型预测，则每个入口由 2 比特出口类型信息、2 比特粘滞位和 9 比特标签位构成，其中表 T_0 无标签位；如果进行出口预测，则每个入口由 3 比特出口号信息、2 比特粘滞位和 9 比特标签位构成.

表 3　性能比较配置表

	T/L TYPE					T/L EXIT					
	L	C	α	$l(1)$	D	L	C	α	$l(1)$	D	H
1 KB	9	10	1.2599	3	512	7	8	1.205	4	512	512
2 KB	10	11	1.2599	4	512	8	9	1.4938	3	512	512
4 KB	11	12	1.2164	5	1 K	9	10	1.542	3	1 K	1 K
8 KB	12	13	1.542	3	1 K	10	11	1.4422	4	1 K	1 K
16 KB	13	14	1.5874	3	2 K	11	12	1.375	5	2 K	2 K
32 KB	14	15	1.4812	4	2 K	12	13	1.2599	7	2 K	2 K
64 KB	15	16	1.671	3	4 K	13	14	1.2331	8	4 K	4 K
128 KB	16	17	1.4422	5	4 K	14	15	1.2114	9	4 K	4 K
256 KB	17	18	1.4736	5	8 K	15	16	1.1934	10	8 K	8 K
512 KB	18	19	1.415	6	8 K	16	17	1.1784	11	8 K	8 K

性能比较结果如图 10 所示. 作为参照，图中引入了上节中基于出口预测的全局历史/本地锦标赛甚块预测器和基于类型预测的地址路径/本地锦标赛甚块预测器. 从图 10 中可以看到，基于类型预测的 TAGE/本地甚块预测器对性能的提升非常明显. 首先，相对于基于类型预测的地址路径/本地锦标赛预测器，该方案将 MPKI 在资源为 1 KB 时降低了约 18%. 虽然该差距随着资源的增多而逐渐缩小，但使用 TAGE 技术的方案一直保持着优势，直到 512 KB 时两者的 MPKI 才趋于相等. 其次，相对于基于出口预测的全局历史/本地锦标赛甚块预测

器,总体趋势相同.即在资源为1KB时该方案能将MPKI降低约23%,直到512KB时,两者MPKI才趋于相等.最后,相对于基于出口预测的TAGE/本地甚块预测器,总体趋势亦相同.在资源大于256KB后,基于出口预测的TAGE/本地甚块预测器略占优势,MPKI低4%左右.

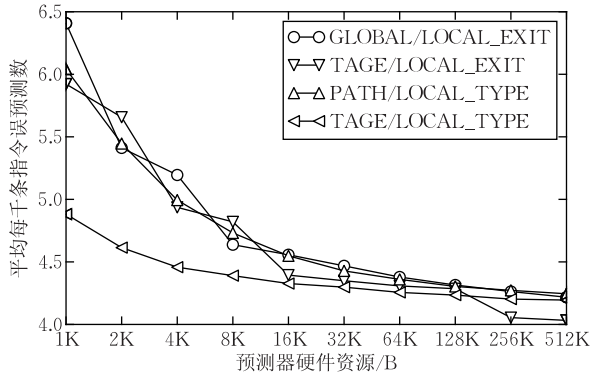


图10 使用TAGE技术的预测器比较(其中,GLOBAL/LOCAL_EXIT为基于出口预测的全局历史/本地锦标赛甚块预测器;TAGE/LOCAL_EXIT为基于出口预测的TAGE/本地甚块预测器;PATH/LOCAL_TYPE为基于类型预测的地址路径/本地锦标赛甚块预测器;TAGE/LOCAL_TYPE为基于类型预测的TAGE/本地甚块预测器)

总体来说,基于类型预测的TAGE/本地预测器在资源较小时效果较好,平均将MPKI降低了约10%.随着资源的增大,基于类型预测的TAGE/本地预测器一直保持优势.基于出口预测的TAGE/本地甚块预测器仅在资源大于256KB后才能达到更低的MPKI.而1KB到64KB之间通常是处理器能够为甚块预测器分配硬件资源的合理范围,因此,基于类型预测的TAGE/本地预测器在较低资源时的性能领先具有较强的现实意义.

总而言之,上述试验结果说明TAGE技术能够提供更好的甚块出口类型预测性能,从而是一种较好的、适用于直接预测甚块出口类型的技术.这是因为TAGE技术能够很好地与甚块出口类型的特性相契合,能根据甚块出口类型特性的不同选择相应的条件进行预测.上述试验结果也说明,相对于基于出口预测的甚块预测器,基于类型预测的甚块预测器能够提供更低的MPKI,从而是一种更好的保证处理器甚块控制流推测能力的方案.

7 结 论

简而言之,本文所提出的直接对甚块出口类型进行预测的方案,简化了甚块预测步骤,提高了甚块的预测能力.值得注意的是,虽然本文以EDGE体系结构作为研究,但本文对甚块出口类型的分析和结论对其它甚块体系结构中的甚块预测器设计来说,同样具有指导和借鉴意义.

本文后续的研究工作包括:(1)进一步探索甚块出口类型的特性,开发性能更优的甚块预测器;(2)将基于类型预测的甚块预测器应用到实际的EDGE处理器设计中,提升处理器的总体性能;(3)将本文的研究成果扩展到其它甚块体系结构中,为设计下一代处理器提供新的思路.

参 考 文 献

[1] Shalf J, Asanovic K, Patterson D et al. The manycore revolution: Will HPC community lead or folow? SciDAC Review, 2009, 1(14): 40-49

[2] Asanovic K, Bodik R, Demmel J et al. A view of the parallel computing landscape. Communications of the ACM, 2009, 52(10): 56-67

[3] Hill M D, Marty M R. Amdahl's law in the multicore era. IEEE Computer, 2008, 41(7): 33-38

[4] Eyerman S, Eeckhout L. Modeling critical sections in Amdahl's law and its implications for multicore design//Proceedings of the 37th Annual International Symposium on Computer Architecture. Saint-Malo, France, 2010: 362-370

[5] Esmaeilzadeh H, Blem E, Amant R St et al. Dark silicon and the end of multicore scaling//Proceedings of the 38th Annual International Symposium on Computer Architecture. San Jose, CA, USA, 2011: 365-376

[6] Burger D, Keckler S W, McKinley K S et al. Scaling to the end of silicon with edge architectures. IEEE Computer, 2004, 37(7): 44-55

[7] Nagarajan R, Kushwaha S K, Burger D et al. Static placement, dynamic issue (SPDI) scheduling for edge architectures//Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques (PACT'04). Antibes Juan-les-Pins, France, 2004: 74-84

[8] Sankaralingam K, Nagarajan R, Liu H et al. Exploiting ILP, TLP, and DLP with the polymorphous trips architecture//Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA'03). San Diego, CA, USA, 2003: 422-433

[9] Sankaralingam K, Nagarajan R, McDonald R G et al. Distributed microarchitectural protocols in the trips prototype processor//Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture. Orlando, Florida, USA, 2006: 480-491

[10] McDonald R, Burger D, Keckler S W et al. TRIPS processor reference manual. Department of Computer Science, University of Texas at Austin: Technical Report TR-05-19, 2005

[11] Gebhart M, Maher B A, Coons K E et al. An evaluation of the trips computer system//Proceedings of the 14th Interna-

tional Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'09). Washington, DC, USA, 2009: 1-12

[12] Mahlke S, Lin D, Chen W et al. Effective compiler support for predicated execution using the hyperblock//Proceedings of the 25th Annual International Symposium on Microarchitecture (MICRO 25). Portland, Oregon, 1992: 45-54

[13] Ranganathan N, Burger D, Keckler S. Analysis of the trips prototype block predictor//Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 2009). Boston, MA, USA, 2009: 195-206

[14] Jacobson Q, Bennett S, Sharma N et al. Control flow speculation in multiscalar processors//Proceedings of the 3rd International Symposium on High-Performance Computer Architecture. San Antonio, Texas, USA, 1997: 218-229

[15] Ranganathan N, Nagarajan R, Jiménez D et al. Combining hyperblocks and exit prediction to increase front-end bandwidth and performance. Department of Computer Science, University of Texas at Austin: Technical Report TR-02-41, 2002

[16] Ranganathan N. Control flow speculation for distributed architectures[Ph. D. dissertation]. The University of Texas at Austin, United States, Texas, 2009

[17] Hao E, Chang P Y, Evers M et al. Increasing the instruction fetch rate via block-structured instruction set architectures//Proceedings of the 29th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-29). Paris, France, 1996: 191-200

[18] Jacobson Q, Rotenberg E, Smith J. Path-based next trace prediction//Proceedings of the Thirtieth Annual IEEE/ACM International Symposium on Microarchitecture. Research Triangle Park, North Carolina, USA, 1997: 14-23

[19] Patel S, Lumetta S. Replay: A hardware framework for dynamic optimization. IEEE Transactions on Computers, 2001, 50(6): 590-608

[20] Zmily A, Kozyrakis C. Block-aware instruction set architecture. ACM Transactions on Architecture and Code Optimization, 2006, 3(3): 327-357

[21] Sohi G, Breach S, Vijaykumar T. Multiscalar processors//Proceedings of the 22nd Annual International Symposium on Computer Architecture. Santa Margherita Ligure, Italy, 1995: 414-425

[22] Esmaeilzadeh H, Burger D. Hierarchical control prediction: Support for aggressive predication//Proceedings of the 2nd Workshop on Parallel Execution of Sequential Programs on Multi-Core Architectures (In Conjunction with ISCA 2009). Austin, Texas, USA, 2009: 71-80

[23] Seznec A, Michaud P. A case for (partially)-tagged geometric history length predictors. Journal of Instruction Level Parallelism, 2006, 8(1): 1-23

[24] Chang P Y, Hao E, Patt Y. Target prediction for indirect jumps//Proceedings of the 24th Annual International Symposium on Computer Architecture. Denver, Colorado, USA, 1997: 274-283

[25] Driesen K, Holzle U. The cascaded predictor: Economical and adaptive branch target prediction//Proceedings of the 31st Annual ACM/IEEE International Symposium on Microarchitecture (MICRO-31). Dallas, Texas, USA, 1998: 249-258

[26] McFarling S. Combining branch predictors. Digital Western Research Laboratory, Palo Alto, CA, USA: Technical Report TN-36, 1993

[27] Guo Pengfei, Li Qingbo, Jin Yinghan et al. M5 based edge architecture modeling//Proceedings of the 2010 IEEE International Conference on Computer Design (ICCD). Amsterdam, Netherlands, 2010: 289-296

[28] Binkert N L, Dreslinski R G, Hsu L R et al. The M5 simulator: Modeling networked systems. IEEE Micro, 2006, 26(4): 52-60

[29] Yoder B, Burrill J, Mcdonald R et al. Software infrastructure and tools for the trips prototype//Proceedings of the 3rd Annual Workshop on Modeling, Benchmarking and Simulation. San Diego, CA, USA, 2007: 1-10

[30] Van Biesbrouck M, Calder B, Eeckhout L. Efficient sampling startup for simpoint. IEEE Micro, 2006, 26(4): 32-42

[31] Evers M, Patel S, Patt Y. An analysis of correlation and predictability: What makes two- level branch predictors work//Proceedings of the 25th Annual International Symposium on Computer Architecture. Barcelona, Spain, 1998: 52-61

[32] Loh G. A simple divide-and-conquer approach for neural-class branch prediction//Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques (PACT 2005). Saint Louis, MO, USA, 2005: 243-254

GOU Peng-Fei, born in 1983, Ph. D. candidate. His research interests include high performance computer architecture research.

YU Ming-Yan, born in 1967, professor. His research interests include VLSI design, high performance computer architecture, analogy circuit design, etc.

YANG Bing, born in 1976, post-doctor. His research interests include high performance computer architecture and high performance compiler techniques.

LI Qing-Bo, born in 1985, M. S. candidate. His research interest is in critical path analysis of high performance computers.

WANG Shi-Bo, born in 1989, undergraduate. Her research interest is in next-block predictor design.

Background

EDGE (Explicit Data Graph Execution) architectures, with features of block-atomic fetch/execute/commit and explicit instruction communication, are one of the most promising alternatives for future processors. Contemporary EDGE processors, such as TRIPS and TFlex, usually maintain block-atomicity at the granularity of hyperblocks. Hyperblocks, formed at compile time, are single-entry, multiple-exit blocks with possibly predicated instructions. To guarantee efficient execution models, EDGE processors adopt control-flow speculation techniques to speculatively fetch multiple hyperblocks onto execution substrates. Since the unit of fetch/execute/commit is hyperblock, control-flow misspeculation resolving and recovery should be done at hyperblock boundaries. Therefore, EDGE architectures rise challenges to conventional instruction-centric control-flow speculation techniques, especially to the branch prediction, which is at the heart of control-flow speculation.

To face these challenges, the next-block predictor was introduced by Ranganathan et al to the research community, enabling high performance control-flow speculation for EDGE architectures. Basic idea of the next-block predictor is to employ the compiler-approximate intra-block paths such as exit IDs to predict branch targets for each hyperblock. At prediction time, the next-block predictor predicts exit ID of a hyperblock by an exit predictor. To incorporate conventional branch predictor techniques into exit predictors, researchers have to make modifications, due to inherent differences between the hyperblock exit prediction and the conventional branch prediction. After producing the predicted exit ID, which is inherently the speculative intra-block path, the next-block predictor predicts the branch type of this exit, then accesses different target buffers in accordance with the predicted branch type. Target buffers tuned for different branch types finally produce the predicted address of the next hyperblock. Published achievements reveal that though a bunch of aggressive techniques has been evaluated, more than 50% mispredicts are still induced by exit ID mispredicts, indicating exit predictors are the performance bottleneck, leaving spaces for future optimizations.

In this paper, our studies show that without exit IDs (intra-block path informations), inter-block informations such as branch type histories are effective enough in predicting hyperblock branch types. As a result, based on these findings, we propose a type-only next-block predictor, which eliminates the exit predictor and directly predicts branch types using non-speculative inter-block informations. By employing TAGE technique in branch type predictors, our proposal outperforms published exit-based next-block predictors by 10% in MPKI (Mispredicts Per Kilo Instructions) across a wide resource budget range for SPEC CPU2000 integer benchmarks.