DOI:10.7544/issn1000-1239.20200503
59(1):31-46,2022

基于指令流访存模式预测的缓存替换策略

王玉庆[1,2] 杨秋松[1] 李明树[1]
[1](中国科学院软件研究所基础软件国家工程研究中心 北京 100190)
[2](中国科学院大学 北京 100049)
(yuqing@nfs.iscas.ac.cn)

A Cache Replacement Policy Based on Instruction Flow Access Pattern Prediction

Wang Yuqing[1,2], Yang Qiusong[1], and Li Mingshu[1]
[1](*National Engineering Research Center for Fundamental Software*, *Institute of Software*, *Chinese Academy of Sciences*, *Beijing* 100190)
[2](*University of Chinese Academy of Sciences*, *Beijing* 100049)

Abstract Traditional cache replacement policies are mainly based on heuristics. In recent years, researchers have used prediction technologies to improve the performance of cache replacement. The application of prediction technologies is gradually becoming one research focus in cache replacement. Because the behaviors of loads and stores are complex, predicting these behaviors in caching systems is difficult with uncertainty. Some existing approaches have been proposed to resolve the problem with more and more complicated prediction algorithms. However, these methods cannot reduce uncertainty, and these methods cannot avoid the interference of out-of-order execution and cache prefetching at the same time. To solve these problems, we propose an approach to predict future memory reference, named IFAPP (instruction flow access pattern prediction). IFAPP recognizes loads and stores in programs based on the instructions flow predicted by branch prediction, and then IFAPP predicts the behavior of each of the loads and stores. IFAPP calculates reuse distance through predicted memory reference, and evicts the candidate with the largest reuse distance. IFAPP avoids the interference of out-of-order execution and cache prefetching. Besides, the objects of prediction are single load/store behaviors which are easy to predict. Both of these alleviate the uncertainty of caching predictions. The evaluations prove that IFAPP reduces the cache misses by 3.2% compared with LRU in L1D. Compared with BRRIP and BIP, IFAPP reduces the cache misses by 12.3% and 14.4% in L1D.

Key words branch prediction; cache replacement policy; ahead prediction; memory access sequence prediction; memory access pattern

摘 要 传统的缓存替换策略主要基于经验主义,近年来研究者们使用预测技术推测访存行为,提高缓存替换的准确性,预测技术的应用是当前缓存替换策略研究的热点.由于访存行为自身的复杂性,直接在缓存系统中预测访存行为是困难的,要面对很大的不确定性.当前已有的研究为了解决该问题,使用越来越复杂的预测算法来分析访存行为之间的关联.然而这种方式并未真正减小不确定性,同时现有的

收稿日期:2020-06-19;**修回日期**:2021-03-16
基金项目:"核高基"国家科技重大专项基金项目(2014ZX01029101-002);中国科学院战略性先导科技专项(XDA-Y01-01)
This work was supported by the National Science and Technology Major Projects of Hegaoji (2014ZX01029101-002) and the Strategic Priority Research Program of Chinese Academy of Sciences (XDA-Y01-01).

缓存替换策略很难避免乱序执行和缓存预取对访存行为分析过程的干扰.为了解决以上问题,提出了一种新的预测缓存访问序列的方法 IFAPP(instruction flow access pattern prediction),根据分支预测技术推测程序指令流,定位指令流中的访存指令,进而对其中访存指令的行为逐一进行预测.通过访存序列计算每个替换候选项的重用距离,将重用距离最远的候选项踢出.该方法可以避免乱序执行和缓存预取的干扰,预测对象是行为简单的独立访存指令,减少预测过程中所面对的不确定性.实验结果表明,该算法在一级数据缓存上比 LRU 算法平均减少 3.2%的缓存缺失.相比经典的基于缓存预测的 BRRIP 和 BIP 算法,该算法在一级数据缓存上分别减少 12.3%和 14.4%的缓存缺失.

关键词 分支预测;缓存替换策略;提前预测;访存序列预测;访存模式

中图法分类号 TP302

缓存是现代处理器中的一种重要机制,将常用数据从内存拷贝到缓存中,后续数据访问可以直接从缓存读取,从而减少慢速 DRAM 内存的访问次数,提升处理器性能.缓存的容量是有限的,在实际使用中发生缓存内容的替换是不可避免的,缓存的性能受缓存容量和缓存替换策略的影响很大.从优化缓存替换策略着手减少缓存缺失次数是近些年学术界的研究热点.

缓存替换策略是指缓存内容替换发生时,从所有的候选行中选择被替换行的算法.理论上最好的缓存替换策略是 Belady[1] 提出的 MIN.发生缓存替换时,MIN 算法总是将重引用距离最远的候选行从缓存中踢出,从而达到最优的替换性能.但是 MIN 算法替换时需要精确的未来访存序列,通常被认作是一种不可实现的算法.在实践中尝试应用 MIN 算法时,需要对访存行为进行抽象建模,利用模型来预测未来的访存行为.由于访存行为的多样性和复杂性,定义出一个准确的访存行为模型适用于所有的应用场景是很困难[2-3].当前的大部分缓存替换策略研究都是基于经验主义,比如基于时间局部性原理的最近最少使用(least recently used, LRU)算法和它的各种变种,将最近使用的数据保留在缓存中.但是基于经验主义的缓存替换策略都具有局限性,只适用于一部分特定的应用场景.

使用预测技术提高缓存替换的准确性是当前缓存替换策略研究的一种趋势.随着研究的深入,预测算法越来越复杂,以应对复杂多变的缓存访问场景.但即使深度学习这样的复杂算法,也很难保证能够适应所有的应用场景.同时,处理器中缓存系统接收到的访存序列会受到乱序执行机制的干扰,即便某条访存指令行为是简单的步长模式,该访存序列到达缓存系统的次序也是不确定的.因此,直接在缓存系统中预测访存序列是困难的,要面对很大的不确定性,而现有的缓存替换策略并不能真正减小这些不确定性.

访存序列预测过程中的不确定性,主要表现在 3 个方面:

1) 缓存访问行为的复杂性.这种复杂性更多体现在整体行为上,在缓存系统中接收到的访存序列是若干条指令执行后的结果,不同指令间的执行次序以及每条指令的行为模式都将对缓存系统最终接收到的访存序列产生影响.在这样复杂多变的访存序列基础上学习访存行为模式是困难的.

2) 在缓存系统中学习指令级的行为模式容易受到乱序执行机制的干扰.为了学习类似固定步长这样的访存模式,预测模块需要计算访存序列中相邻访问之间的差值.乱序执行会打乱访存执行的次序,使预测过程变得不准确.为解决该问题,文献[4]引入了额外的存储结构记录所有地址的访问.

3) 在开启了缓存预取机制的处理器中,缓存系统中接收到的访问序列中包含了预取请求.预取算法的投机性很高,一部分预取请求是无用的,并不能代表程序对数据的真实需求.若缓存替换策略不能识别预取项,就有可能保留不必要的预取数据在缓存中,从而影响到程序的执行性能.文献[5]分析了这一现象,并说明在开启预取的前提下,MIN 算法也达不到缓存替换的最优效果.

为了解决上述问题,本文提出了一种新的预测缓存访问序列的方法,通过程序的指令流来预测缓存访问序列.根据分支预测技术推测程序指令流,定位指令流中访存指令的位置,借助于访存指令模式的学习,对访存指令的地址逐一进行预测,生成访存序列.相比访存指令的行为,分支指令的行为(跳转方向和目标地址)更容易预测.分支预测技术的研究已经比较成熟,现有的分支预测算法[6]准确率很高.

现有基于预测技术的缓存替换策略[7-11]需要在

缓存系统中重新实现一套预测器,本文在处理器中已有的分支预测单元基础上扩展出对访存序列的预测.该方法预测对象具体到每条访存指令,降低了预测的复杂性;通过访存指令的提交序列进行模式学习,避免乱序执行的干扰;预测过程基于程序指令流,预测的访存序列只包含真实的程序需求,不受缓存预取机制的影响.通过上述方式本文方法减少了访存序列预测过程中所面对的不确定性.

基于预测出的访存序列,在选择替换候选项时将重引用距离最远的候选项从缓存中踢出.在进行缓存替换决策时,如果所有的候选项都未命中访存序列,将路号最小的候选项从缓存中踢出.新分配的缓存行有可能在下一次缓存替换时被选为踢出项,本文算法可以将一部分缓存数据保持在缓存中,因此对于缓存访问中的抖动和扫描等行为具有一定的抵抗性.

本文的主要贡献有 3 个方面:

1) 提出了基于指令流访存模式预测的缓存替换策略 IFAPP(instruction flow access pattern prediction).根据分支预测技术预测程序指令流,确定程序指令流中的访存指令,对每条访存指令的行为进行预测.该方法可以减少预测过程中所面对的不确定性,不依赖于特定分支预测算法,具有良好的可扩展性.

2) 提出了一种学习指令级访存模式的方法.对访存指令的执行结果进行记录,按照提交次序学习访存模式.同时改进了步长模式访存行为的识别,在普通步长模式基础上扩展对循环行为的支持.

3) 在 GEM5 模拟器上实现了原型系统,通过在 SPEC CPU2006 程序集中实验评估,验证了 IFAPP 缓存替换策略的有效性.同时实验证明了当分支预测和访存模式两者均充分学习的情况下,本文算法预测出的访存序列是比较准确的.

1 相关工作

理论上最优的缓存替换策略是 Belady 提出的 MIN 算法,每次将重用距离最远的候选项从缓存中踢出.基于该算法原理的缓存替换策略在实际应用中需要借助访存行为模型产生访存序列从而确定重用距离.一些早先的研究使用独立引用模型(independent reference model, IRM)[12]对访存行为进行建模,该模型假设所有的访存行为是互相独立的,并且单个访存行为具有固定的、已知的概率.应用该模型进行缓存替换时,每次将访问概率最小的候选项从缓存中踢出,等价于最不经常使用(least frequency used, LFU)算法.IRM 模型忽视了临近访存之间的相关性,在分析处理器缓存时表现不佳.近来,一些缓存系统研究[3,13-15]提出了更复杂的模型来分析访存行为,但是这些复杂模型只适用于对特定替换策略(如 LRU)进行分析.

由于现有的访存模型不能很好地反映真实程序的行为,大部分的缓存替换策略是基于经验主义的.基于经验主义的缓存替换策略分析应用场景中访存行为的模式,通过总结出的规律指导替换候选项的选择.比如大部分的程序都具有时间局部性,最近访问的缓存行大概率会被再次访问,由此衍生出了 LRU 算法和它的派生算法[16-18].与此相反,若程序不具有时间局部性,则采用最近最常使用(most recently used, MRU)替换策略更优.还有一部分研究认为经常被访问的缓存行很有可能会被再次访问,因此基于缓存行的访问频率构建替换策略[19-22].使用上述单一模式构建的缓存替换策略往往不能适用复杂的应用场景,后续的研究多采用混合策略,通过静态选择[19,22-23]或者动态选择[24-27]的方式从多种经验主义替换策略中进行选择.但是基于经验主义的缓存替换策略都具有局限性,只适用于一部分特定的应用场景.在实际应用中访存行为很可能同时具有多种规律,一部分访存行为具有时间局部性,另一部分与此相反.在此情形下,无论是 LRU 或者 MRU 都不能很好地工作.

使用预测技术提高缓存替换的准确性是当前缓存替换策略研究的一种趋势.预测技术一方面可以用于强化经验主义算法,另一方面也可以直接指导缓存替换.预测的内容可以是死块(dead block)[10,28-31],也可以是重用距离[7-11,32-33].随着研究的深入,缓存替换策略中使用的预测算法也变得越来越复杂,从最简单的基于饱和计数器的方法[7],发展到使用二级自适应的方法[10],直到最近的感知器算法[34]和深度学习算法[35].为了对复杂的缓存行为进行准确预测,除了预测算法本身在不断加强外,缓存访问行为中与重用距离相关的特性也不断被挖掘.访存指令的 PC(program counter)、访存的物理地址、访存序列历史信息等和重用距离相关的因素都被应用到预测算法中.如 MPPPB 算法[32]扩展了感知器算法[34],选择了 7 种与重用距离相关的因素作为算法的输入.预测技术的应用使缓存替换策略能够动态地学习访存行为的特征,增强了缓存替换策略的适应性.现有使用预测技术的缓存替换策略依然属于

经验主义的范畴,并不能从根本上弥补经验主义算法和 MIN 算法之间的差距.

缓存访问行为是复杂的,里边包含了大量的不确定性,在设计缓存替换策略时必须要考虑到如何处理这些不确定性[36].处理器中缓存系统接收到的访存序列会受到处理器乱序执行和缓存预取机制的干扰,这进一步增大了在缓存系统中预测访存行为的难度.当前的缓存替换策略均不能有效减少这些不确定性.

本文提出的缓存替换策略依然需要使用历史信息进行规律学习,与其他基于预测的缓存替换策略相比较,本文的预测对象比较简单.访存行为的复杂性更多体现在整体上而不是个体上,无论是分支指令行为还是指令级访存行为都是比较容易学习的.本文提出的算法预测访存指令流,进行指令级别的访存模式分析,可以天然规避预测过程中很多的不确定性,避免使用像深度学习这样的复杂算法.本文的实验结果证明预测出的访存序列是比较准确的,说明本文方法确实减小了缓存访问行为预测中的不确定性.

2 背景知识介绍

本文的算法基于处理器中的分支预测单元进行扩展,在指令级访存行为中比较常见的是步长访问模式.下文简单对这 2 方面的内容进行介绍.

2.1 分支预测

分支预测技术对程序中分支指令的跳转方向和跳转地址进行预测,减少分支指令刷新导致的处理器性能损失.研究表明,当前分支指令的跳转行为是受过往分支指令影响的.分支指令之间的相关性是分支预测的基础.分支预测器设计就是将分支指令之间的关系归纳为模式,通过模式识别的方式预测分支指令行为.分支预测器可以细分为方向预测器和地址预测器,分别预测分支指令的跳转方向和跳转地址.

以图 1 中的 TAGE-ISL 算法[6]为例,其方向预测器由多个预测模块组成,不同类型的分支指令由其专用预测模块处理.循环类型的分支会被循环预测器捕获;带有强偏向性的分支由 TAGE 中的 bimodal 部分进行预测;具有特定模式的分支则由 TAGE 中的全局预测器进行预测.该算法中还包含 2 个辅助器(统计纠正和立即更新),对主预测器 TAGE 的结果进行修正.

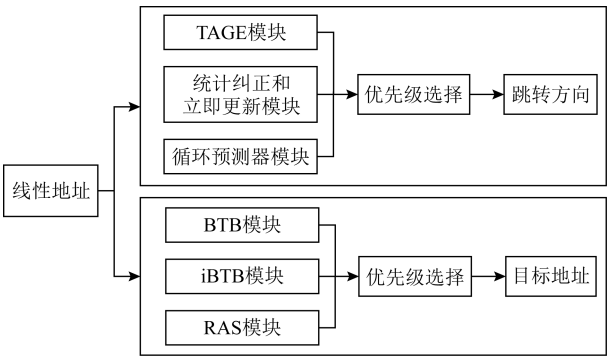

Fig. 1 The algorithm of TAGE-ISL

图 1 TAGE-ISL 算法

地址预测器的设计同方向预测器类似,也由多个专用预测器组成.若分支指令的跳转目标地址是固定的,则可以通过分支目标缓冲(branch target buffer, BTB)[37]中记录的上一次跳转地址进行准确预测;返回指令的返回地址是由 CALL 指令确定的,可以通过返回地址栈(return address stack, RAS)进行预测;若分支指令的跳转地址和某个模式是一一对应的,则通过间接分支目标缓冲(indirect BTB, iBTB)进行预测,其查询过程类似于方向预测器中的二级自适应算法.

BTB 主要作用在于解决 2 个问题:1)如何判断在某个位置分支指令是否存在;2)如何确定分支指令的类型以及跳转目标地址.这 2 个问题同样也是本文中进行读写访存指令预测时首先要解决的问题.因此本文参照 BTB 结构和功能设计访存目标缓冲(memory access target buffer, MATB),记录访存指令的 PC、访存指令类型和上一次访存物理地址.

2.2 步长模式访存行为

基于步长的数据访问模式是指连续的数据访问的间隔是一个固定值,例如依次访问 A,$A+K$,$A+2K$ 等.这种访问类型在应用程序中是十分常见的,当程序操作多维数组或者使用指针遍历元素大小相同的数据结构时,其数据访问都符合基于步长的模式.

在处理器中最常见的步长预测算法是 IBSP[38],其主要结构是访问预测表,如图 2 所示.访问预测表使用访存指令的线性地址进行索引,表中记录上一次访问的地址以及上 2 次访问的地址差.当访存指令未命中缓存时,使用访存指令的线性地址查询访问预测表.若标签命中,则使用当前的访存地址减去上一次访存地址,计算出当前的步长.若当前步长与访问预测表中记录的上一次步长相等,则说明当前

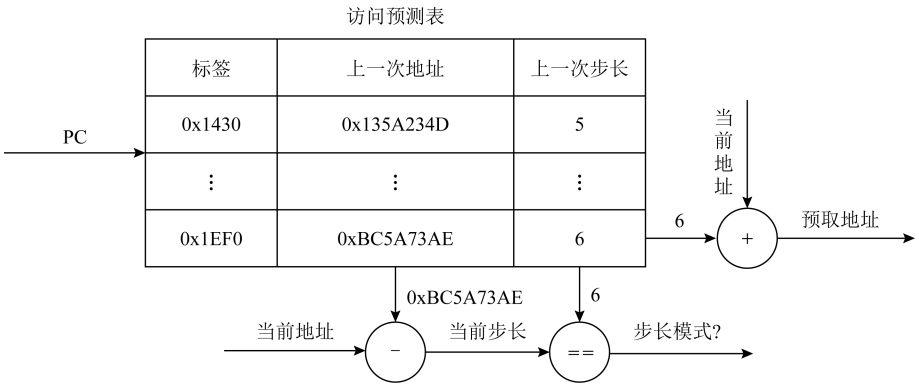

Fig. 2 Step predictor IBSP

图 2 步长预测器 IBSP

访存指令的行为符合步长模式.预测访存地址为当前访存地址加上当前步长.

3 基于指令流访存模式预测的缓存替换策略

本文提出的 IFAPP 缓存替换策略使用预测技术预测未来访存序列,使用同 MIN 算法相同的决策进行缓存替换——将重用距离最大的候选项踢出.未来访存序列的预测分为 2 个步骤:1)判断访存指令是否存在于指令流中,确定其发生的次序;2)为每条访存指令的每次执行进行独立的地址预测.第 1 个步骤依赖于分支预测技术对指令流进行预测,第 2 个步骤需要对每条指令的访存模式进行学习.

IFAPP 缓存替换策略基于分支预测技术,通过提前流水设计(ahead pipeline)[39]将分支预测过程做成单独的子流水.分支预测流水的预测结果是一个个指令块,其宽度为分支预测流水带宽.若指令块中存在跳转分支指令,则分支指令末尾字节的位置作为指令块的结束,这些指令块组成了程序的指令流.利用分支预测技术预测缓存访问序列需要解决 2 个问题:1)如何判断某个指令块中是否存在访存指令;2)如何确定访存指令的访存地址.

为了解决第 1 个问题,本文定义访存目标缓冲 MATB,接收处理器写回阶段反馈的访存指令信息.MATB 中记录流水线中执行过的访存指令的线性地址、访存指令的类型和访存物理地址等信息.通过检索 MATB,我们可以确认某个指令块中是否存在访存指令,以及该访存指令上一次执行所访问的物理地址.

为了解决第 2 个问题,本文在 MATB 的基础上添加访存指令类型识别功能.访存指令的行为是复杂的,某条访存指令的目标物理地址可能是随着时间变化的.访存物理地址不固定的访存指令是不能通过 MATB 进行准确预测的,因为 MATB 只能记录上一次执行的物理地址.本文算法将访存指令划分为不同的类型,不同的访存指令类型使用专用的预测器进行.本文可以识别多种常见指令级访存模式,对步长模式访问行为使用步长预测器进行地址预测;对具有时间关联性的访问行为使用时间关联预测器进行预测.对指令级访问模式的学习过程按照指令提交的次序进行,不会受到乱序执行的干扰.

通过分支预测和 MATB 我们可以确定访存指令的位置和先后次序;通过指令级访存模式的学习我们可以预测每条访存指令的目标地址.这两者结合起来就可以生成未来访存序列.该访存序列用于在缓存替换发生时计算重用距离,将重引用距离最远的候选项从缓存中踢出.访存序列中实现了读写指针,读指针随着取指过程自增,读写指针之间是有效预测项.计算候选项重用距离的过程从读指针开始遍历访存序列,查询候选项第 1 次出现的位置,该位置与读指针之间的距离即为候选项的重用距离.

本文的系统总体架构如图 3 所示.基于指令流访存模式预测的缓存替换策略系统分为以提前预测为主体的取指预测单元、以 MATB 为主体的访存预测单元和替换决策单元 3 个部分.取指预测单元和访存预测单元分别预测指令缓存和数据缓存的未来访存序列;替换决策单元进行重用距离的计算,选择候选项进行踢出.

取指预测单元在处理器的分支预测单元中添加提前预测功能,维护提前预测线性地址和提前预测分支历史信息.提前预测作为 CPU 中的一个子流水独立运行,同取指过程解耦合.提前预测的预测结果

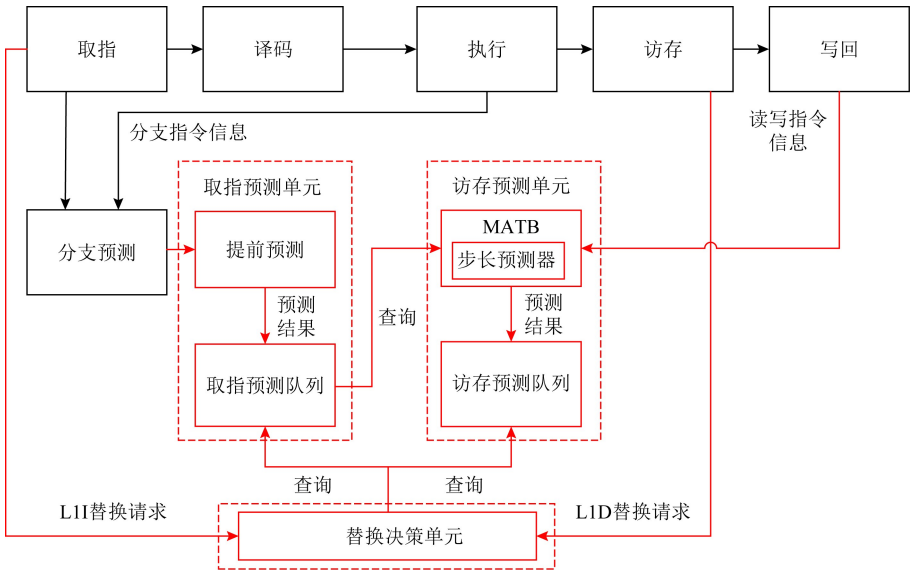

Fig. 3 IFAPP framework

图 3 IFAPP 框架

写入取指预测队列中,对取指单元中的一级指令缓存而言,取指预测队列中保存的就是未来一段时间的访存序列.提前预测过程中使用的地址是线性地址,预测结果在写入取指预测队列之前经过转译后备缓冲(translation lookaside buffer, TLB)的检索就能得到物理地址.

访存预测单元的主要功能是实现 MATB,记录流水线中执行过的访存指令信息.每当取指预测队列添加新项时,用该指令块的起始地址查询 MATB,预测指令块中的访存指令位置及其访存地址,并将预测结果写入访存预测队列.本文在 MATB 的基础上实现了步长预测器和时间关联预测器,能够对符合特定模式的访存指令进行预测.

替换决策单元的实现缓存替换的决策过程.取指预测队列和访存预测队列中的内容分别是一级指令缓存和一级数据缓存的未来访问序列.当一级指令缓存 L1I 发出替换请求后,替换决策单元接收到所有候选项的物理地址,使用这些物理地址检索取指预测队列,得到每个候选项的重用距离.将重用距离最大的候选项作为踢出项反馈给一级指令缓存.一级数据缓存 L1D 的替换请求处理与此类似.

下文中我们将详细介绍取指预测单元、访存预测单元和替换决策单元中的关键组件.

3.1 取指预测单元

取指预测单元是为了预测一级指令缓存的未来访问序列,本文的设计中在处理器中取指单元和分支预测单元基础上进行扩展,添加提前预测逻辑和取指预测队列.提前预测逻辑和取指预测队列用于缓存访问序列预测,提前预测逻辑的结果写入取指预测队列,并不影响取指过程.原有的取指单元和分支预测单元间的交互过程仍照常进行.

如图 4 所示,在分支预测单元中新增提前预测地址逻辑、提前预测历史、提前预测分支目标缓存(Ahead BTB, ABTB),提前预测的结果选择逻辑以及指令预测队列等模块,这些模块共同实现了取指预测单元.此处借鉴了分支预测设计中的提前流水设计,通过引入取指预测队列,将提前预测过程和取指过程解耦合.在取指单元处于阻塞状态时,依然会发送提前预测流水的有效信号,以此保证提前预测流水远远领先于取指单元.只有这样取指预测队列中才能够填充足够多的预测信息.当取指预测队列填满时,停止提前预测流水.提前预测逻辑与分支预测单元中的原有预测逻辑是互相独立的.

3.1.1 提前预测

提前预测的“提前”是相对于处理器中取指流水而言的.提前流水设计本意是解决慢速分支预测和取指速率不匹配的问题.在本文设计中我们故意加快分支预测的速率,以产生足够多的指令块支持访存序列预测过程.提前预测逻辑主要由提前预测地址逻辑、提前预测历史和提前预测分支目标缓存组成.

定义 1. 提前预测地址.提前预测地址逻辑生成提前预测流水的线性地址,表示为$\langle pc, op\rangle$,其中 pc 表示提前预测流水中的基地址,op 表示当前周期对基地址的更新操作.在初始化或者流水线刷新

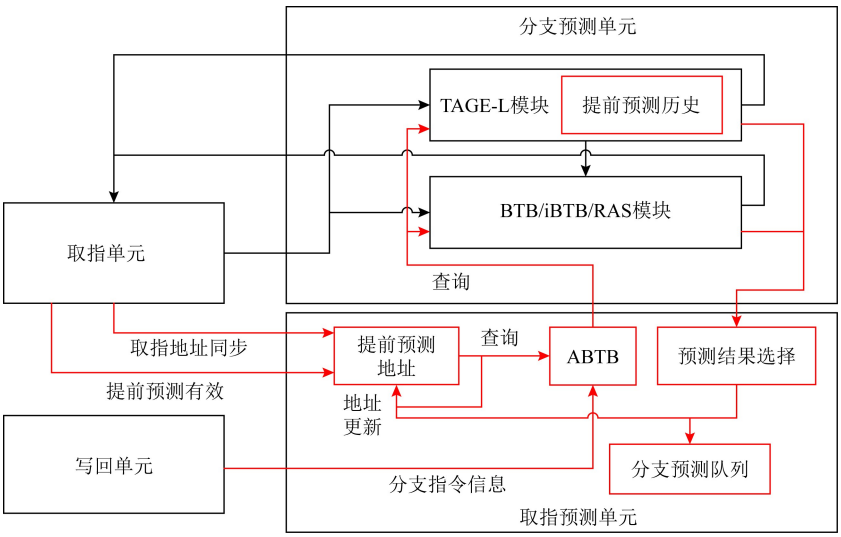

Fig. 4 The module of instruction fetch predictor

图 4 取指预测单元

时,pc 与取指单元中的线性地址进行同步.提前预测流水每周期以 pc 为起点,预测 2 个连续 64 B 的指令块.若预测过程未发现跳转的分支指令,pc 按照 128 B 对齐的方式进行自增.如果在提前预测流水识别到跳转的分支指令,则将 pc 更新为分支指令跳转目标地址.

定义 2. 提前预测历史.提前预测流水专用的历史并在提前预测流水中进行投机的更新.分支预测单元中已有的历史信息保持原有逻辑不变.以 TAGE-L 算法为例,其 TAGE 模块内会同时包含提前预测历史和原分支历史,两者在各自的预测流水中使用,是互相独立的.流水线刷新时不仅要更新提前预测线性地址,还需要对投机更新的提前预测历史信息进行回滚.当发生流水线刷新时,原分支历史会先进行回滚,然后将 2 个历史进行一次同步.

定义 3. ABTB 模块.提前预测地址并不是某个分支指令的地址,而是一个指令块的起始线性地址.分支预测单元中原有的 BTB 模块只能处理单个分支指令地址的检索,因此需要扩展一个 ABTB 模块,用来处理一个指令块中多条分支指令并行检索.ABTB 根据流水线后端反馈的分支指令信息进行分配.

ABTB 模块采用组相联结构,共 4 096 项.每一项的结构表示为〈$valid$,tag,$offset$,$branch\_type$,$target\_addr$,$phys\_addr$,$instlen$〉.其中 $valid$ 表示有效位;tag 表示标签;$offset$ 表示偏移;$branch\_type$ 表示分支类型;$target\_addr$ 表示预测跳转目标地址;$phys\_addr$ 表示行物理地址,在分支指令提交时根据其线性地址查询 TLB 获得,用于写入取指预测队列;$instlen$ 表示指令长度,用于计算分支指令的首字节线性地址.

ABTB 模块中以分支指令的结尾字节作为分支指令的地址,将分支指令的线性地址拆分为高位的标签和低位的指令块内偏移 2 部分.ABTB 在进行检索时,首先进行高位标签的比对,在标签相等的同时其块内偏移部分必须大于提前预测地址的低位才算命中.提前预测地址作为指令块的起始地址,分支指令的地址大于提前预测地址才能说明该分支指令位于这个指令块中.

提前预测流水的分支指令方向预测需要检索分支预测单元中原有的分支预测模块,这些模块是用分支指令的首字节地址进行索引的.每个指令块中可能包含多个分支指令,相对应的 ABTB 在提前预测流水检索的过程中可能会命中多项.每个命中项都是一条分支指令,其分支指令地址会再次检索分支预测单元中原有的分支预测模块(BTB,TAGE-L 等).命中 ABTB 的多条分支指令的预测结果汇总在一起,输入到预测结果选择模块.预测结果选择模块从分支指令中选择偏移最小的跳转分支作为预测结果写入取指预测队列.若指令块中存在跳转的分支指令,则提前预测地址作为指令块的起始地址,分支指令的地址作为指令块的结束地址写入取指预测队列.若指令块中不存在跳转的分支指令,则指令块内是连续的,其结束地址就是该 64 B 指令块的最后一个字节.

3.1.2 取指预测队列

取指预测队列用来存放提前预测的结果,在本文的设计中取指预测队列有 4 096 项.文献[40]指出预测序列长度是缓存容量的 8 倍以上时 MIN 算法才具有良好的性能,本文算法在一级缓存容量为 512 项的基础上乘以 8 作为取指预测队列深度.取指预测队列每一项的结构可以表示为〈*valid*,*line_addr*,*phys_addr*,*begin_offset*,*end_offset*,*taken*〉.其中,*valid* 表示有效位;*line_addr* 表示行线性地址;*phys_addr* 表示行物理地址;*begin_offset* 表示起始偏移;*end_offset* 表示结束偏移;*taken* 表示跳转位.行线性地址和起始偏移/结束偏移确定了 64 B 指令块内有效字节的范围.

取指预测队列以 64 B 指令块为单位记录提前预测流水的预测结果.64 B 的宽度是为了和取指单元访问一级指令缓存的带宽匹配.取指预测队列中实现了提交指针、读指针和写指针,如图 5 所示.当某个指令块中的所有指令都提交后,该指令块可以从取指预测队列中移除,此时提交指针加 1.当取指模块读取过某个指令块后,读指针加 1.提交指针和读指针之间指令块已经处于流水线中,读指针和写指针之间的指令块才是有效预测指令块.流水线刷新发生后,读写指针会回滚到分支刷新的位置或者提交指针的位置.

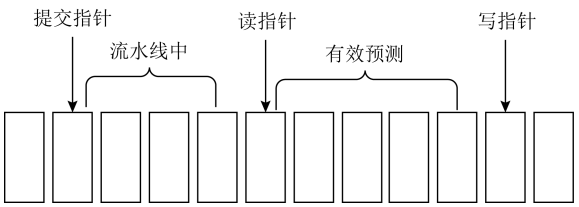

Fig. 5 Pointer in the queue of instruction fetch predictor

图 5 取指预测队列中的指针

在进行一级指令缓存的替换决策时,使用一级指令缓存反馈的候选项物理地址检索取指预测队列.此检索过程使用 64 B 对齐的物理地址进行匹配,检索从读指针开始,到写指针结束,第 1 个命中的条目项位置就是候选项的重用距离.若候选项未命中,则将重用距离设置为最大值.当出现多个候选项未命中时,选择缓存中路号最小的候选项作为踢出项.

3.2 访存预测单元

访存预测单元是为了预测一级数据缓存的未来访存序列,本文的设计中通过设计 MATB 来预测未来访存序列,并将结果放入到访存预测队列.取指预测队列中的每一个条目项代表一个指令块,每个指令块都包含起始偏移和结束偏移.各个指令块连起来构成了程序的指令流.每当取指预测队列添加新项时,用该指令块的起始地址查询 MATB,预测指令块中的访存指令位置及其访存地址,并将预测结果写入访存预测队列.访存预测队列条目项中包含取指预测队列编号,因此访存预测队列中的每一条访存指令,都能对应到取指预测队列中的一个条目项.建立这样的对应关系方便在流水线刷新时进行取指预测队列和访存预测队列的同步刷新.

3.2.1 访存目标缓冲 MATB

访存目标缓冲 MATB 存储结构有 4 096 个表项,MTAB 接收写回单元的反馈,记录已经提交的访存指令的线性地址以及访存目标物理地址,还有指令类型等信息.每一项的结构表示为〈*LineAddr*,*PhyAddr*,*InstType*〉.其中,*LineAddr* 表示访存指令的线性地址,线性地址分为标签和偏移 2 部分;*PhyAddr* 表示访存指令的目标物理地址;*InstType* 表明访存指令类型,*InstType* ∈ {*DirectInst*,*InDirectInst*},其中 *DirectInst* 表示直接访存指令,*InDirectInst* 表示间接访存指令.MATB 在查询过程中使用访存指令的线性地址进行命中判定,查询命中后输出访存指令的目标物理地址和访存指令类型.

MATB 的查询过程如图 6 所示.MATB 的命中判定是一个包含关系的判定,预测指令块是一个区间范围,我们需要判定访存指令的线性地址是否在这个区间范围内.取指预测队列新添加的项中包括指令块的起始地址和结束地址,可以转化为高位的标签和低位的起始偏移和结束偏移.使用指令块的标签和偏移查询 MATB,当一条访存指令的线性地址高位标签和指令块相同,并且低位偏移大于等于起始偏移,小于等于结束偏移的时候,即说明该访存指令位于该指令块中.

访存指令的线性地址在程序运行过程中一般是不变的,使用该线性地址检索 MATB 识别对应的访存指令应当是准确的.这一过程同分支预测算法中使用分支指令的线性地址检索 BTB 是类似的.分支指令占程序总指令数的比例大约是 1/4,分支预测技术的论文中对 BTB 的尺寸进行过实验,选用 4 096 个表项是比较合适的.本文假设程序中访存指令的比例同分支指令是类似的,在实现 MATB 时直接参照了 BTB 的尺寸.

命中 MATB 后,根据 MATB 中记录的访存指令类型选择访存目标物理地址.若是直接访存指令,

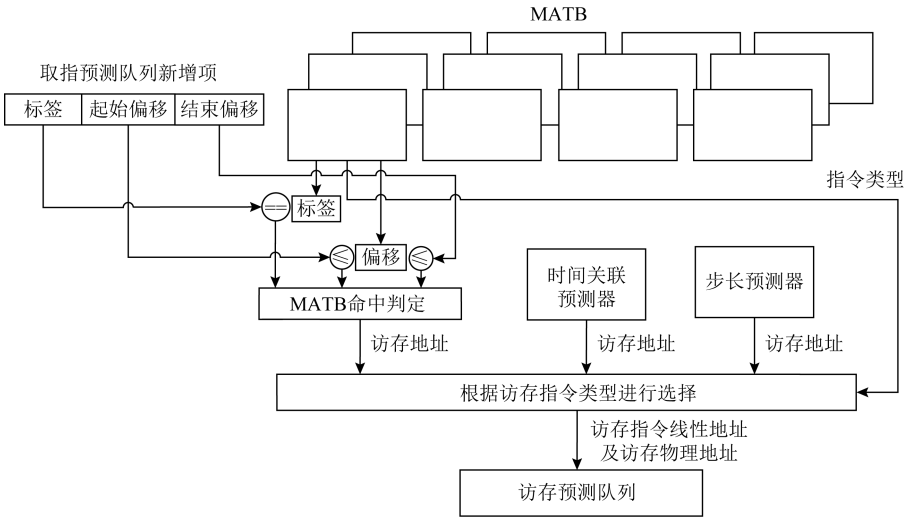

Fig. 6 Queries of MATB

图 6 MATB 的查询过程

则访存目标物理地址来自 MATB.若是间接访存,如果访存指令满足步长模式,则使用步长预测器获取物理地址;如果访存指令满足时间关联模式,则使用时间关联预测器获取物理地址.将访存指令线性地址和访存目标物理地址等信息写入访存预测队列中.MATB 的查询可能会命中多项,表明一个指令块中包含多条访存指令,这些访存指令按照指令块中出现的先后次序,将预测结果写入访存预测队列中.

3.2.1.1 步长预测器

在缓存系统中比较常见的访存模式是步长模式,为了支持对符合该模式的访存指令的目标地址进行预测,本文设计了步长预测器.本文中设计的步长预测器的设计参考了 IBSP[38] 算法,在其基础上进行了扩展.在实现中步长预测器是整合在 MATB 中的,扩展了每个 MATB 表项的属性.

步长预测器的结构如图 7 所示,相比 IBSP 算法,本文的步长预测器中扩展了可信计数器、当前模式、首地址以及最大计数器等内容.原本的 IBSP 算法只需要记录步长信息,因为数据预取操作是在当前访存地址的基础上计算出来的.针对$\{A,A+K,A+2K,\cdots,A+NK\}$这样的固定步长访存序列,本文的步长预测器不仅要记录其步长 K,同时也要记录序列的起始地址 A 以及最大计数值 N.为了增加步长预测器的准确性,增加了可信计数器和当前模式.当上一次步长和当前步长相同时,可信计数器加 1,否则可信计数器清零.当可信计数器的数值大于

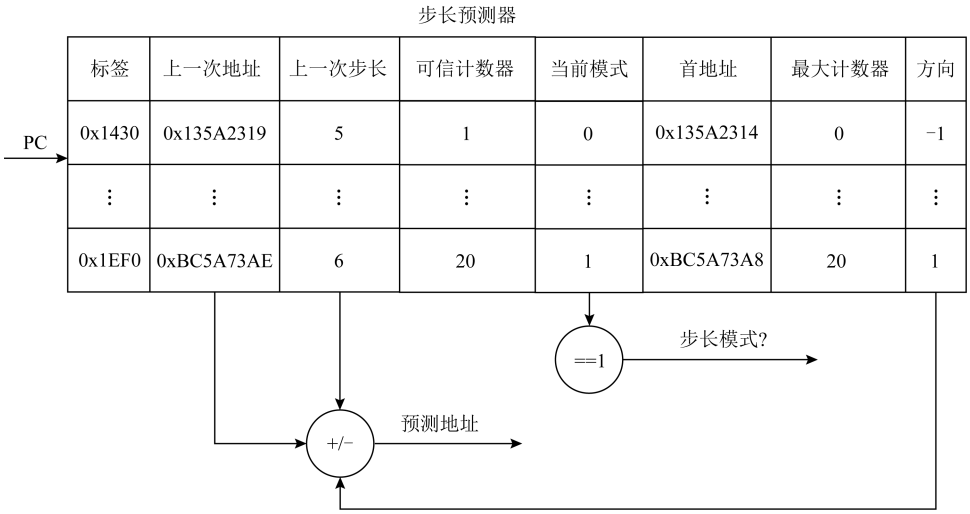

Fig. 7 Step predictor

图 7 步长预测器

某个阈值时,将当前模式位置1,表明当前访存指令的行为确实是符合步长模式的;若当前模式位为0,则表明当前尚处于学习阶段,学习阶段步长预测器不输出预测结果.步长始终是正值,方向位决定步长序列是递增还是递减.在MATB分配时首地址域设置为当前物理地址.在步长模式计数过程中,首地址域是不更新的,只更新上一次地址为当前最新值,通过这种方式将一个步长序列的首地址保存下来.当MATB为某条符合步长模式的指令进行更新时,发现当前地址恰好等于首地址,说明步长访存序列又重新执行了一遍.此时将可信计数器的值更新到最大计数器中,步长序列的头和尾都被记录下来.

步长模式识别算法如算法1所示:

算法1. 步长模式识别算法.

输入:提交的访存指令PC和物理地址$PhyAddr$;

输出:是否是步长模式.

① if(当前模式==步长模式);
② if(当前方向和步长与历史记录不符)
③ if($PhyAddr$==首地址 &&(最大计数器==0 ‖ 可信计数器≠最大计数器))
④ 最大计数器=可信计数器;
⑤ else
⑥ 清空步长相关信息;
⑦ end if
⑧ else
⑨ 可信计数器++;
⑩ if(最大计数器≠0 && 可信计数器>最大计数器)
⑪ 最大计数器=可信计数器;
⑫ end if
⑬ end if
⑭ else
⑮ if(当前方向和步长与历史记录不符)
⑯ 清空步长相关信息;
⑰ else
⑱ 可信计数器+1;
⑲ if(可信计数器>学习阈值)
⑳ 当前模式=步长模式;
㉑ end if
㉒ end if
㉓ end if

步长预测器的查询和MATB一致,发生在取指预测队列写入新的项之后.MATB命中说明该指令块中存在访存指令,同时可以获取该指令的步长预测器信息.若访存指令非步长模式,则预测物理地址同上一次物理地址相同;若访存指令为步长模式,则需要根据首地址和步长信息计算当前访存物理地址$AddrA$.此计算过程需要统计当前取指预测队列中该访存指令(指令块)已经出现的次数.

算法2. 步长预测器的访存地址预测.

输入:当前是否是步长模式、步长预测器表信息;

输出:预测地址.

① if(当前模式==步长模式)
② 根据编号检索取指预测队列,得到未提交的包含该指令的指令块个数N;
③ 预测地址=首地址+(N%最大计数器)×上一次步长;
④ if(最大计数器≠0)
⑤ NUM=(可信计数器+N+1)%(最大计数器+1);
⑥ 预测地址=首地址+方向×NUM×上一次步长;
⑦ else
⑧ NUM=1+N;
⑨ 预测地址=上一次地址+方向×NUM×上一次步长;
⑩ end if
⑪ else
⑫ 预测地址=上一次地址;
⑬ end if

3.2.1.2 时间关联预测器

时间关联访存行为[41]是指一段访存序列以固定的次序重复出现.例如我们观察到{A,B,C,D,A,B,C,D}这样的访存序列之后,{B,C,D}这样的序列有很大概率紧接着在A之后出现.指令级访存行为中也存在时间关联访存模式.

算法3. 时间关联行为识别.

输入:过去M次访存地址;

输出:时间关联信息.

① 将过去M次访存地址记录在$lastPhyVector[M-1:0]$中;
② if($lastPhyVector[3n-1:0]$匹配{A_1,A_2,…,A_n,A_1,A_2,…,A_n,A_1,A_2,…,A_n})
③ 时间关联模式=1;
④ 时间关联序列长度=n;
⑤ 时间关联序列=$lastPhyVector[n-1:0]$;
⑥ end if

时间关联预测器集成在 MATB 中.时间关联预测器记录每一条访存指令过去 M 次的访存地址,根据这 M 次历史信息识别时间关联访存行为.本文的算法中相同访存序列重复 3 次才能确认为时间关联模式,$lastPhyVector$ 的长度设置为 12,支持序列长度不超过 4 的时间关联访存行为.除此基本的时间关联行为模式之外,本文算法还支持形如$\{A,B,C,A+n,B+n,C+n,A+2n,B+2n,C+2n,\cdots\}$这样的步长——时间关联行为模式.

3.2.2 访存预测队列

访存预测队列用来存放检索 MATB 得出的预测结果.在本文设计中访存预测队列有 65 536 项,65 536 是在取指预测队列有 4096 项的基础上得出,每个取指预测队列项代表一个 64 B 指令块,我们假设其中最多有 16 条访存指令.每一项的结构表示为:$\langle valid, inst\_line\_addr, mem\_phys\_addr, memlen, inst\_queue\_index\rangle$.其中 $valid$ 表示有效位;$inst\_line\_addr$ 表示指令线性地址;mem_phys_addr 表示访存物理地址;$memlen$ 表示访存长度,根据访存物理地址和访存长度可以识别跨行的访存指令;$inst\_queue\_index$ 表示取指预测队列索引,便于在流水线刷新时进行取指预测队列和访存预测队列的同步刷新.

访存预测队列中也实现了提交指针,读指针和写指针.在进行一级数据缓存的替换决策时,使用一级数据缓存反馈的候选项物理地址检索访存预测队列,此过程同取指预测队列的检索类似.

3.3 替换决策单元

替换决策单元接收一级缓存发送的候选项物理地址,使用这些物理地址检索取指预测队列/访存预测队列,得到每个候选项的重用距离,将重用距离最大的候选项编号反馈给一级缓存.

若取指预测队列或者访存预测队列中没有足够多的有效预测结果,替换决策单元向一级缓存反馈相应信息,此时一级缓存使用默认替换策略进行候选项选择.在进行缓存替换决策时,如果所有的候选项都未命中访存序列,本文算法将路号最小的候选项从缓存中踢出.

4 实验与验证

为了验证本文提出方法的有效性,本节首先说明了实验环境和实验中使用的测试程序.然后从 2 个方面说明 IFAPP 算法的有效性:1)分析 IFAPP 算法的适用场景;2)对比 IFAPP 算法和其他算法的性能.

4.1 实验设置

本文采用 GEM5 模拟器[42]作为基础实验环境,使用 alpha 指令集.实验过程需要对 CPU 内部结构进行扩展,因此使用了 GEM5 中的 DerivO3CPU 细粒度模型.使用的分支预测算法为 TAGE-L.GEM5 的基本配置如表 1 所示:

Table 1 Configuration of GEM5

表 1 GEM5 配置

参数	配置值
cpu-clock/GHz	3.0
cpu-type	DerivO3CPU
bp-type	TAGE_L
l1i_size/KB	32
l1i_assoc	8
l2_size/KB	256
l2_assoc	8
mem_size/MB	4096
fetchBufferSize/B	64
tag_latency	2
data_latency	2
decodeWidth	8

表 1 中取指缓冲($fetchBufferSize$)的宽度和一级缓存访问的时延($tag\_latency/data\_latency$)影响 GEM5 模拟器取指阶段的执行速度.在设计中我们期望提前预测流水能够显著领先于取指流水,从而在取指预测队列和访存预测队列中填充足够多的预测信息.因此在实验评估环节我们会观测取指参数的变化对 IFAPP 算法的影响.

实验的测试集为 SPEC CPU2006 测试程序.完整的 SPEC CPU2006 测试程序在 GEM5 模型中可能需要运行几个月的时间,现在比较认可的做法是从 simulation point 开始运行.本文参考文献[43]中提供的采样点实验数据进行仿真.整型和浮点程序的检查点如表 2 和表 3 所示.

实验中从表 2、表 3 检查点开始执行 1 000 万条指令,统计 IFAPP 算法生效的次数和缓存替换发生的次数.

实验的对比算法为 LRU,BRRIP[7],BIP[24].BIP 算法是动态选择 LRU/MRU 插入模式的典型经验主义算法,BRRIP 是基于重用距离预测的典型算法.为了保证 IFAPP 算法的准确性,当前的实验设置为

只有当取指预测队列/访存预测队列中有超过 4 096 个预测结果时才使用 IFAPP 算法进行缓存替换，否则采用默认缓存替换策略(LRU).选择 4 096 这个阈值是因为文献[40]指出预测序列长度是缓存容量的 8 倍以上时 MIN 算法才具有良好的性能，按照当前 GEM5 配置一级缓存中包含 512 个缓存行，因此需要 4 096 个预测结果.

Table 2 Checkpoints of Integral Programs

表 2 整型程序检查点

CINT06	simpoint 点(亿条指令)
401.bzip2	3 792
445.gobmk	530
456.hmmer	3 313
458.sjeng	14 714
462.libquantum	15 117
464.h264ref	5 271
471.omnetpp	3 772
473.astar	3 287

Table 3 Checkpoints of Floating Programs

表 3 浮点程序检查点

CFP06	simpoint 点(亿条指令)
410.bwaves	20 179
433.milc	4 324
450.soplex	1 807
459.GemsFDTD	1 842
470.lbm	615

4.2 实验评估

在本节中首先分析 IFAPP 算法的适用场景，探讨流水线刷新和架构参数对 IFAPP 算法的影响；然后通过和对比 LRU，BRRIP，BIP 算法的性能进行对比，说明本文方法的有效性.

4.2.1 流水线刷新对 IFAPP 算法的影响

当分支预测错误产生刷新时，或者因为其他原因导致提交单元进行流水线刷新时，取指预测队列和访存预测队列都会响应该刷新.当前的实验设置为只有当访存预测队列中有超过 4 096 个有效预测结果时才使用 IFAPP 算法对一级数据缓存的替换请求进行处理，当处理器刷新比较频繁时 IFAPP 算法生效的次数会受到明显影响.为了评估流水线刷新对 IFAPP 算法的影响，我们首先统计不同 SPEC 程序中 IFAPP 算法的生效次数.

如表 4 所示，当处理器刷新比较频繁时(bzip2，h264ref，soplex，gobmk)，IFAPP 算法生效的次数占总替换次数的比例会显著减低.其中大部分的处理器刷新是由分支预测失败引起的.IFAPP 算法并不依赖于特定的分支预测算法，通过使用更准确的分支预测算法取代 TAGE-L，可以减小分支预测失败刷新的次数，增大 IFAPP 算法的生效次数，从而提高 IFAPP 算法的性能.

Table 4 Efficient Times of IFAPP with Different SPEC

表 4 不同测试程序 IFAPP 算法生效次数

SPEC 程序	L1D 替换发生次数	IFAPP 生效次数	分支指令数	分支预测失败刷新	处理器刷新
bzip2	85 794	6 433	1 087 601	97 401	118 691
GemsFDTD	224 256	213 058	218 701	15	2 019
astar	7 742	6 806	1 525 935	90	199
h264ref	34 579	298	1 101 246	15 840	19 814
hmmer	33 232	28 683	441 054	25	344
lbm	491 676	429 243	148 307	75	2 581
soplex	195 299	21 339	1 559 599	49 280	52 439
gobmk	34 701	1 246	1 518 238	98 340	102 635

4.2.2 架构参数对 IFAPP 算法的影响

影响 IFAPP 算法生效次数的因素除了刷新以外，还有提前预测流水领先于取指流水的程度.若模拟器中取指阶段执行速度比较快，提前预测流水只能轻微地领先于取指流水，也会导致访存预测队列中没有足够的预测结果.以 bzip2 测试程序为例，我们修改模拟器取指阶段的参数，分析取指执行速度与 IFAPP 算法生效次数的关系.

实验过程中，我们保持取指单元的带宽(*fetchBufferSize*)不变，修改一级缓存的访问时延(*tag_latency*/*data_latency*)和解码单元的带宽(*decodeWidth*).增大一级缓存的访问时延或者减小解码单元的带宽都会导致取指阶段的执行速度降低.根据表 5 可以看出，当我们逐渐增大缓存访问时延，并减小解码带宽，提前预测流水就会更明显地领先于取指流水，此时 IFAPP 算法生效的次数也相应地增长.在处理器设计中，缓存的访问时延和解码的带宽一般是固定的架构参数，并不能动态调整，此时通过

Table 5 Efficient Times of IFAPP with Different Parameters

表 5 不同参数下 IFAPP 算法生效次数

fetchBuffer Size/B	*tag_ latency*	*data_ latency*	*decode-Width*	L1D 替换次数	IFAPP 生效次数
64	2	2	8	85 794	6 433
64	5	5	4	85 085	6 623
64	10	10	4	83 885	8 405

增大提前预测流水的带宽(当前预测带宽为 128 B,2 倍于取指带宽)也可以起到相似的效果.

4.2.3 性能对比

在本节中我们对比了 LRU,BRRIP,BIP 算法和本文提出的 IFAPP 算法.

4.2.1 节中提到 IFAPP 算法受流水线刷新次数的影响比较明显,当 IFAPP 算法生效次数很少时,其缓存替换性能基本等同于 LRU 算法.在这样的 SPEC 程序中对比 IFAPP 算法和其他算法的性能没有实际意义.根据表 6 中的实验结果,我们选择 IFAPP 算法生效次数占比较高的 GemsFDTD,astar,hmmer,lbm,bwaves 这 5 个 SPEC 程序对比 IFAPP 算法和其他算法的性能.

Table 6 Relationship Between Efficient Times of IFAPP and Times of Cache Replacement

表 6 IFAPP 生效次数和缓存替换次数关系

SPEC 程序	缓存替换次数	IFAPP 算法生效次数
bzip2	84 576	10 384
GemsFDTD	231 176	226 389
astar	7 740	7 307
h264ref	34 579	1 250
hmmer	32 736	30 705
lbm	416 130	391 738
soplex	194 261	57 642
gobmk	34 430	1 613
bwaves	181 768	170 858
omnetpp	238 068	81
sjeng	12 862	0

如图 8 所示,IFAPP 算法相比 LRU 算法平均提高 3.2%的性能.与 BRRIP 算法和 BIP 算法相比,IFAPP 算法分别平均提高 12.3%和 14.4%的性能.在一级缓存中程序执行的局部性可以得到充分的体现,LRU 算法应用在一级缓存替换上已经可以取得相当好的效果.BRRIP 算法和 BIP 算法应用在一级缓存时,其性能要比 LRU 算法差.

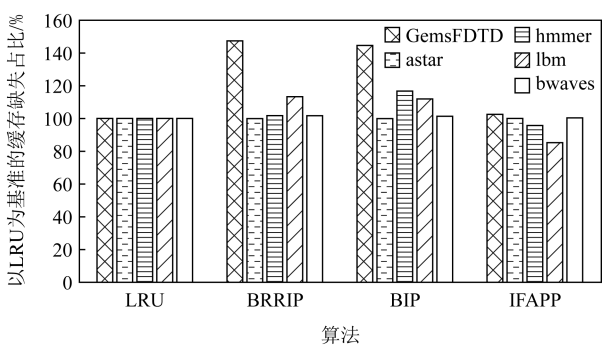

Fig. 8 Performance comparation of four algorithms

图 8 4 种算法性能对比图

IFAPP 算法在一级缓存能够比 LRU 算法更加有效是因为多数访存指令的行为并不复杂,IFAPP 算法可以对其访存地址进行准确预测.相比 LRU 算法,使用 IFAPP 算法进行缓存替换更能反映应用程序对数据的真实需求.如图 8 所示,IFAPP 算法在 lbm 程序上的性能明显优于 LRU.这是因为 lbm 程序中有很多访存指令行为符合步长模式,在整个采样片段中其地址一直自增.很多新分配的缓存行之后再也不会被访问,这样的行为是不利于 LRU 策略的.对于 IFAPP 算法来说,即便是新分配的缓存行,如果其重用距离过远也会被很快替换出去.因此 IFAPP 算法对于缓存访问中的抖动和扫描等行为具有一定的抵抗性.

本文统计了不同 SPEC 程序中 IFAPP 算法访存地址预测错误的次数,如表 7 所示.在百万量级的访存指令中,IFAPP 算法预测错误的次数很少.本文实验中使用的分支预测算法并是准确率最高的,所支持的访存模式也是最基本的,但 IFAPP 算法对访存序列的预测依然很准确.这说明预测访存序列的方法是很重要的,相比在缓存系统中直接预测访存序列,本文算法结合分支预测和访存模式学习这 2 种技术可以规避访存序列预测中的很多不确定性.实验结果证明了在分支预测和访存模式学习的共同作用下,IFAPP 算法可以减少访存序列预测中的不确定性.

Table 7 The Accuracy of Memory Address Prediction of IFAPP

表 7 IFAPP 访存地址预测准确性

SPEC 程序	访存指令数目	IFAPP 预测错误次数
GemsFDTD	3 249 255	2 421
astar	3 864 405	534
hmmer	3 851 259	6 395
lbm	2 702 928	48 796
bwaves	2 824 575	54 042

4.2.4 算法开销

IFAPP 算法的存储资源开销主要集中在 ABTB、MATB、取指预测队列和访存预测队列上.ABTB 有 4 096 个表项,每个表项 96 b,总共占用 48 KB 的存储空间.MATB 有 4 096 个表项.每个表项中包含了 71 b 的访存指令信息、69 b 的步长预测器和 523 b 的

时间关联预测器.时间关联预测器需要保存访存指令过去 12 次物理地址,占用空间较多.MATB 总共占用约 331 KB 的存储空间.取指预测队列有 4 096 项,每一项 86 b,占用 43 KB 的存储空间.访存预测队列有 65 536 项,每一项 94 b,占用 752 KB 的存储空间.

ABTB 和 MATB 的分配过程只需要几个时钟周期.取指预测队列和访存预测队列的分配过程在提前预测流水中进行,在本文的配置下提前预测流水明显领先于 CPU 正常流水,取指预测队列和访存预测队列的分配时间会被掩盖掉.IFAPP 算法的时间开销主要集中在使用一级缓存发送的候选项物理地址检索取指预测队列/访存预测队列,得到每个候选项的重用距离.实际应用中 65 536 项的访存预测队列应实现为 16 路的组相联结构,访存预测队列指针转换为组号/路号.在这个存储阵列上实现 4 个读端口,每周期可以读取 64 项(4×16),读取 4 096 项需要 64 个周期.加上后续的计数和比较过程,整个缓存替换决策大约需要 70 个周期.这个过程的时间开销是可以避免的,我们可以在缓存替换行为发生之前,提前检索取指预测队列/访存预测队列,计算好每个候选项的重用距离.

4.3 实验结论

通过 4.2.1 节的分析,我们可以看出流水线刷新对 IFAPP 算法的生效次数有明显的影响,这在一定程序上限制了 IFAPP 算法的应用场景.当前的算法设计为了保证准确性会在流水线刷新之后清空取指预测队列和访存预测队列.我们可以只进行写指针的回退,不进行数据的刷新,新的预测数据将覆盖原有数据,未覆盖的投机预测数据可以被保留.这样将降低 IFAPP 算法的准确性,但是可以降低流水线刷新对 IFAPP 算法生效次数的影响,扩宽 IFAPP 算法的应用场景.

本文提出的 IFAPP 算法在一级数据缓存上相比 LRU 算法提高了缓存替换的性能.这主要是因为借助分支预测和访存模式学习技术,我们可以准确地预测访存序列.通过使用更准确的分支预测算法,支持更多的访存模式,IFAPP 算法的性能还可以进一步提升.

5 总 结

本文提出了一种基于预测技术的缓存替换方法.根据分支预测技术推测程序指令流,定位指令流中的访存指令流,然后对每个访存指令的地址进行预测,生成访存指令预测信息.在访存指令预测信息的基础上应用 MIN 方法进行缓存替换.该方法可以避免乱序执行的干扰,预测对象是指令级行为模式,大大减少替换预测过程中所面对的不确定性.本文在 GEM5 模拟器上搭建了原型系统,分析了基于预测技术的缓存替换策略的适用场景,并通过性能对比实验说明了该算法的有效性,IFAPP 算法相比 LRU 算法平均提高 3.2%的性能,和 BRRIP 算法和 BIP 算法相比 IFAPP 算法分别平均提高 12.3%和 14.4%的性能.在后续工作中,我们将研究如何减少本文算法的开销,以及在多核环境下预测共享二级缓存的访问序列,并分析未来访问序列在缓存预取算法中的应用.

作者贡献声明:王玉庆提出研究思路,设计研究方案,进行实验和数据分析,起草论文;杨秋松设计研究方案,审阅论文;李明树参与论文审阅和最终版本修订.

参 考 文 献

[1] Belady L A. A study of replacement algorithms for a virtual-storage computer [J]. IBM Systems Journal, 1966, 5(2): 78-101

[2] Stefano T, Ahmed M H, Garetto M, et al. Temporal locality in today's content caching: Why it matters and how to model it [J]. ACM SIGCOMM Computer Communication Review, 2013, 43(5): 5-12

[3] Martina V, Garetto M, Leonardi E. A unified approach to the performance analysis of caching systems [C] //Proc of IEEE Conf on Computer Communications. Piscataway, NJ: IEEE, 2014: 2040-2048

[4] Ishii Y, Inaba M, Hiraki K. Access map pattern matching for high performance data cache prefetch [J/OL]. Journal of Instruction-Level Parallelism, 2011, 13 [2021-03-05]. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.395.9958&rep=rep1&type=pdf

[5] Jain A, Lin C. Rethinking Belady's algorithm to accommodate prefetching [C] //Proc of the 45th Annual Int Symp on Computer Architecture (ISCA'18). New York: ACM, 2018: 110-123

[6] Seznec A. Tage-sc-l branch predictors [C/OL] //Proc of the 4th JILP Workshop on Computer Architecture Competitions: Championship Branch Prediction. New York: ACM, 2014 [2021-03-05]. https://hal.inria.fr/hal-01086920

[7] Jaleel A, Theobald K B, Steely S C, et al. High performance cache replacement using re-reference interval prediction [C] //Proc of the 37th Annual Int Symp on Computer Architecture (ISCA'10). New York: ACM, 2010: 60-71

[8] Keramidas G, Pavlos P, Kaxiras S. Cache replacement based on reuse-distance prediction [C] //Proc of the 25th Int Conf on Computer Design. Piscataway, NJ: IEEE, 2007: 245-250

[9] Wu C J, Jaleel A, Hasenplaugh W, et al. SHiP: Signature-based hit predictor for high performance caching [C] //Proc of the 44th Annual IEEE/ACM Int Symp on Microarchitecture (MICRO-44). New York: ACM, 2011: 430-441

[10] Khan S M, Tian Yingying, Jiménez D A. Sampling dead block prediction for last-level caches [C] //Proc of the 43rd Annual IEEE/ACM Int Symp on Microarchitecture. New York: ACM, 2010: 175-186

[11] Das S, Aamodt T M, Dally W J. Reuse distance-based probabilistic cache replacement [J/OL]. ACM Transactions on Architecture and Code Optimization, 2015, 12(4). [2021-03-05]. https://dl.acm.org/doi/pdf/10.1145/2818374

[12] Aho A V, Denning P J, Ullman J D. Principles of optimal page replacement [J]. Journal of the Association for Computing Machinery, 1971, 18(1): 80-93

[13] Beckmann N, Sanchez D. Modeling cache performance beyond LRU [C] //Proc of the 2016 IEEE Int Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2016: 225-236

[14] Jelenkovic P R, Radovanovic A. Least-recently-used caching with dependent requests [J]. Theoretical Computer Science, 2004, 326(1): 293-327

[15] Sen R, Wood D A. Reuse-based online models for caches [J]. ACM Sigmetrics Performance Evaluation Review, 2013, 41(1): 279-292

[16] Jiménez D A. Insertion and promotion for tree-based pseudo LRU last-level caches [C] //Proc of the 46th Annual IEEE/ACM Int Symp on Microarchitecture (MICRO-46). New York: ACM, 2013: 284-296

[17] Wong W A, Baer J L. Modified LRU policies for improving second-level cache behavior [C] //Proc of 6th Int Symp on High-Performance Computer Architecture. Piscataway, NJ: IEEE, 2000: 49-60

[18] Smaragdakis Y, Kaplan S, Wilson P. EELRU: Simple and effective adaptive page replacement [J]. ACM SIGMETRICS Performance Evaluation Review, 1999, 27(1): 122-133

[19] Robinson J T, Devarakonda M V. Data cache management using frequency-based replacement [J]. ACM SIGMETRICS Performance Evaluation Review, 1990, 18(1): 134-142

[20] Hallnor E G, Reinhardt S K. A fully associative software-managed cache design [J]. Sigarch Computer Architecture News, 2000, 28(2): 107-116

[21] Qureshi M K, Thompson D, Patt Y N. The V-Way cache: Demand-based associativity via global replacement [C] //Proc of the 32nd Int Symp on Computer Architecture (ISCA'05). New York: ACM, 2005: 544-555

[22] O'Neil E J, O'Neil P E, Weikum G. The LRU-K page replacement algorithm for database disk buffering [J]. ACM SIGMOD Record, 1993, 22(2): 297-306

[23] Lee D, Choi J, Kim J, et al. LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies [J]. IEEE Transactions on Computers, 2001, 50(12): 1352-1360

[24] Qureshi M K, Jaleel A, Patt Y N, et al. Adaptive insertion policies for high performance caching [C] //Proc of the 34th Annual Int Symp on Computer Architecture (ISCA'07). New York: ACM, 2007: 381-391

[25] Megiddo N, Modha D S. ARC: A self-tuning, low overhead replacement cache [C] //Proc of the 2nd USENIX Conf on File and Storage Technologies (FAST'03). Berkeley, CA: USENIX Association, 2003: 115-130

[26] Bansal S, Modha D S. CAR: Clock with adaptive replacement [C] //Proc of the 3rd USENIX Conf on File and Storage Technologies (FAST'04). Berkeley, CA: USENIX Association, 2004: 187-200

[27] Subramanian R, Smaragdakis Y, Loh G H. Adaptive caches: Effective shaping of cache behavior to workloads [C] //Proc of the 39th Annual IEEE/ACM Int Symp on Microarchitecture (MICRO-39). New York: ACM, 2006: 385-396

[28] Kharbutli M, Solihin Y. Counter-Based cache replacement and bypassing algorithms [J]. IEEE Transactions on Computers, 2008, 57(4): 433-447

[29] Khan S M, Jimenez D A, Burger D, et al. Using dead blocks as a virtual victim cache [C] //Proc of the 19th Int Conf on Parallel Architecture and Compilation Techniques. Piscataway, NJ: IEEE, 2010: 489-500

[30] Lai A, Fide C, Falsafi B. Dead-block prediction & dead-block correlating prefetchers [C] //Proc of the 28th Annual Int Symp on Computer Architecture. New York: ACM, 2001: 144-154

[31] Liu Haiming, Ferdman M, Huh J, et al. Cache bursts: A new approach for eliminating dead blocks and increasing cache efficiency [C] //Proc of the 41st Annual IEEE/ACM Int Symp on Microarchitecture(MICRO-41). New York: ACM, 2008: 222-233

[32] Jimenez D A, Teran E. Multiperspective reuse prediction [C] //Proc of the 50th Annual IEEE/ACM Int Symp on Microarchitecture (MICRO-50). New York: ACM, 2017: 436-448

[33] Lin Junmin, Wang Wei, Qiao Lin, et al. A cache replacement policy based on reuse distance prediction and stream detection [J]. Journal of Computer Research and Development, 2012, 49(5): 1049-1060 (in Chinese)
(林隽民, 王炜, 乔林, 等. 一种基于重用距离预测与流检测的高速缓存替换算法[J]. 计算机研究与发展, 2012, 49(5): 1049-1060)

[34] Teran E, Wang Zhe, Jimenez D A. Perceptron learning for reuse prediction [C/OL] //Proc of the 49th Annual IEEE/ACM Int Symp on Microarchitecture (MICRO-49). New York: ACM, 2016 [2021-03-05]. https://ieeexplore.ieee.org/document/7783705

[35] Zhan Shi, Xiangru Huang, Jain A, et al. Applying deep learning to the cache replacement problem [C] //Proc of the 52nd Annual IEEE/ACM Int Symp on Microarchitecture (MICRO-52). New York: ACM, 2019: 413-425

[36] Beckmann, Sanchez D. Maximizing cache performance under uncertainty [C] //Proc of 2017 IEEE Int Symp on High Performance Computer Architecture (HPCA). Piscataway, NJ: IEEE, 2017: 109-120

[37] Lee J, Smith A J. Branch prediction strategies and branch target buffer design [J].IEEE Computer, 1984, 17(1): 6-22

[38] Baer J L, Chen T F. An effective on-chip preloading scheme to reduce data access penalty [C] //Proc of the 1991 ACM/IEEE Conf on Supercomputing. Piscataway, NJ: IEEE, 1991: 176-186

[39] Seznec A, Fraboulet A. Effective ahead pipelining of instruction block address generation [C] //Proc of the 30th Annual Int Symp on Computer Architecture. New York: ACM, 2003: 241-252

[40] Jain A, Lin C. Back to the future: Leveraging Belady's algorithm for improved cache replacement [C] //Proc of the 43rd Annual Int Symp on Computer Architecture (ISCA'16). New York: ACM, 2016: 78-89

[41] Bakhshalipour M, Lotfikamran P, Sarbaziazad H. Domino temporal data prefetcher [C] //Proc of the 2018 IEEE Int Symp on High Performance Computer Architecture (HPCA). Piscataway, NJ: IEEE, 2018: 131-142

[42] Binkert N, Beckmann B, Black G, et al. The GEM5 simulator [CP/OL] (2018-08-29) [2021-03-05]. http://www.m5sim.org/Main_Page

[43] Ganesan K, Panwar D, John L K. Generation, validation and analysis of SPEC CPU2006 simulation points based on branch, memory, and TLB characteristics [C] //Proc of SPEC Benchmark Workshop Computer Performance Evaluation. New York: ACM, 2009: 121-137

Wang Yuqing, born in 1987. PhD candidate. His main research interests include computer architecture, operating system and cache system.

王玉庆,1987 年生.博士研究生.主要研究方向为计算机体系结构、操作系统和缓存系统.

Yang Qiusong, born in 1977. PhD, professor, PhD supervisor. His main research interests include operating system, software engineering and system security.

杨秋松,1977 年生.博士,教授,博士生导师.主要研究方向为操作系统、软件工程和系统安全.

Li Mingshu, born in 1966. PhD, professor, PhD supervisor. Fellow of CCF. His main research interests include operating system, software engineering and distributed system.

李明树,1966 年生.博士,教授,博士生导师,CCF 会士.主要研究方向为操作系统、软件工程和分布式系统.