处理器分支预测研究的历史和现状

冯子军 肖俊华 章隆兵

摘要 在过去十几年中，分支预测技术一直是提高处理器性能的重要方法，工业界和学术界对之进行了大量研究。分支预测的本质是克服指令控制相关，提高指令并行度。随着研究的不断深入，当前学术界认为分支预测是一个指令学习的过程，这就使得对分支预测的研究出现了新的趋势。本文对分支预测技术的历史和研究现状进行了归纳，以便从总体上了解分支预测技术的发展过程。

1 引言

过去的十几年里，分支预测技术一直是提高通用处理器性能的重要方法。分支预测的本质是克服指令控制相关，提高指令并行度，从而使得处理器的性能得到提高。在这方面学术界和工业界都进行了大量的研究和实践，分支预测的重要性体现在以下几个方面：

首先，现在的通用处理器大多采用深度流水线和宽发射机制，分支预测是两者的关键支撑技术。虽然前两年英特尔（Intel）的奔腾 4（Pentium4）靠深度流水提高主频一直为人所诟病，但是同时应该注意到没有一定深度的流水，处理器频率就不可能太高，也就不会有很高的性能。目前 x86 处理器一般都有 20－30 级的流水线。此外展望新世纪体系结构时候，耶鲁.帕特（Yale.Patt）预测宽发射将成为单芯片集成 10 亿晶体管主要解决方案之一[1]，而且宽发射也是提高单片处理器性能的重要手段之一，最近英特尔的新处理器Conroe就从原先奔腾的三发射提高到四发射，使得处理器性能提高 30％也是例证[2]。

简单的分析表明：在当前流行的深流水线宽发射体系结构中，分支预测率会严重影响取指带宽的利用率。在 5 发射 10 级流水线条件下，预测准确率为 90%时，带宽会浪费 47%；而如果准确率提高到 96%则带宽浪费可降低到 26%（一般处理器设计为 2 到 8 发射，此处为了计算方便假定 5 发射）。

另外，分支预测技术不仅在高性能通用处理器中采用，而且在嵌入式处理器也广泛采用，所以作为一个处理器设计者，我们应该知道当前存在的分支预测的各种算法及其优缺点，这样才能对功耗和性能进行权衡。事实上，一种分支预测机制可能在某些应用中可以提高运算效率，但在另一些应用可能效果就不明显，因此设计者需要对不同应用采用不同的分支预测解决方案。

还应该指出，很好地理解分支指令对微处理器的设计至关重要，分支指令是计算机不同于计算器的最重要区别，使得计算机得以超越简单的数字计算功能转变为可以完成各种复杂任务和运算的信息处理装置。分支指令决定了程序从取指令到执行指令的路径，对分支指令的特性和行为理解深刻就可以帮助处理器设计者来平衡处理器结构。在程序里面分支指令组合起来就形成了程序分支行为，后面的介绍将说明不同分支预测机制的提出就是根据这些程序行为来进行设计和改进的。分支行为是取指单元设计必须考虑的关键因素，而掌握更复杂的分支特性，比如分支相关，就能很好地帮助我们对不同应用选择和改进分支预测。

综上所述，可以知道分支预测的重要性，本文就处理器的分支预测技术的过去、现在和将来做一下总结和展望。本文首先介绍分支指令的性质和分支行为的一些属性，其次介绍分支预测的发展历程和主要的分支预测方法，然后介绍分支预测的最新进展，最后预测未来处理器设计分支可能会出现的问题和发展趋势。

2 分支行为

分支预测器设计的本质是在对分支指令行为认识的基础上，提出分支指令的预测机制，从而

减少分支惩罚，也就是分支预测误预测导致的流水线等待。一组分支指令组合起来就成为程序的分支行为。程序分支行为非常复杂，国外学术界作了大量的研究。这里只介绍一些基本的特点，以便更好地了解后面重点说明的预测机制。下面我们就分别简单介绍一下分支指令属性和程序分支的行为。

2.1 分支指令的基本属性

分支指令总的来说有三个基本属性：分支指令的类型、分支指令发生的频率和分支指令的成功率。分支指令的类型可以分为条件分支指令和无条件分支指令，由于分支指令目标地址的不同，无条件分支又可以进一步分为立即分支指令，间接分支指令和返回分支指令。立即分支指令就是分支的地址就在分支指令中，一般都是直接跳转，比如跳转（jump）这类指令；间接分支就是分支的目标地址不在分支指令中，而是从其他寄存器中取得；返回型分支的分支目标地址是从链接寄存器（Link register）或者堆栈中得到，一般是程序返回使用。

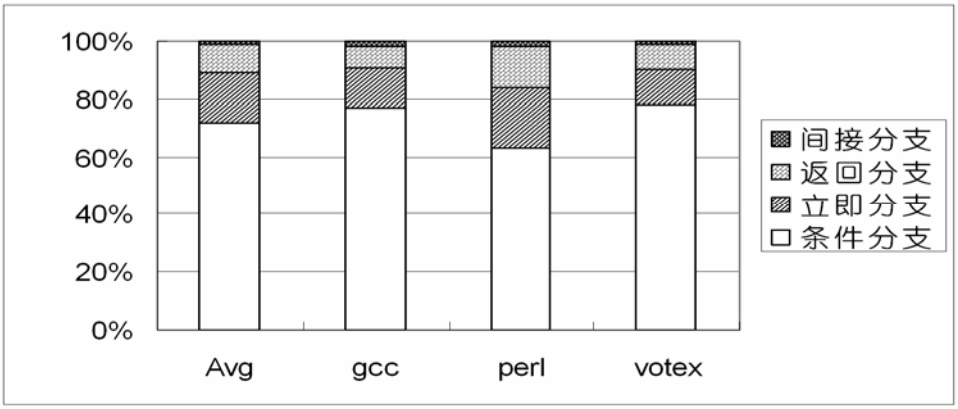

图 1 不同分支类型所占 SPEC 程序比例

如图 1 所示，统计分析SEPC程序结果表明，分支指令中 72%是条件分支，17％是无条件立即跳转指令，10％是返回指令，1％是间接分支指令。其中立即分支跳转可以采用BTB（Branch Target Buffer，分支预测缓冲区）这种方式精确预测，返回型跳转可以采用返回地址栈(RAS[1])精确预测[5] ，间接分支跳转的预测一直没什么好的办法，在[6]中进行了讨论。所以分支预测大量的工作是进行条件分支预测。

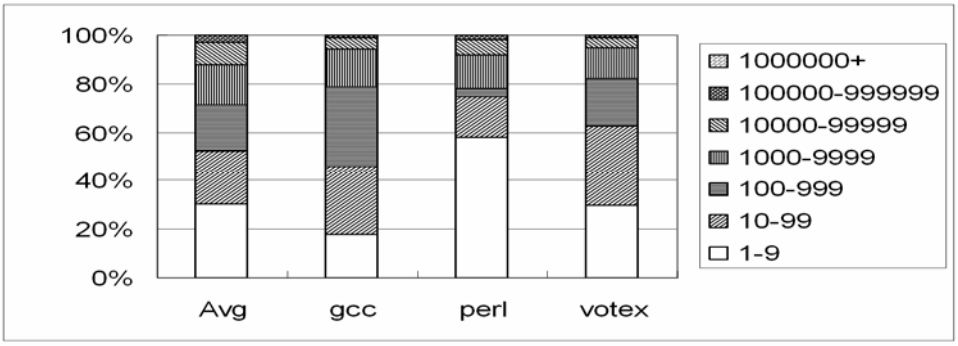

图 2 分支指令执行的频率分布

图 2 说明了 SPEC 程序条件分支预测执行的频率分布。从图中我们可以看到，大部分条件分支在程序运行过程中，仅仅执行几次，平均来说所有分支指令的 53%仅仅执行 99 次或者更少，只有 11％的分支执行了超过 10000 次或者更多。

[1] Return Address Stack

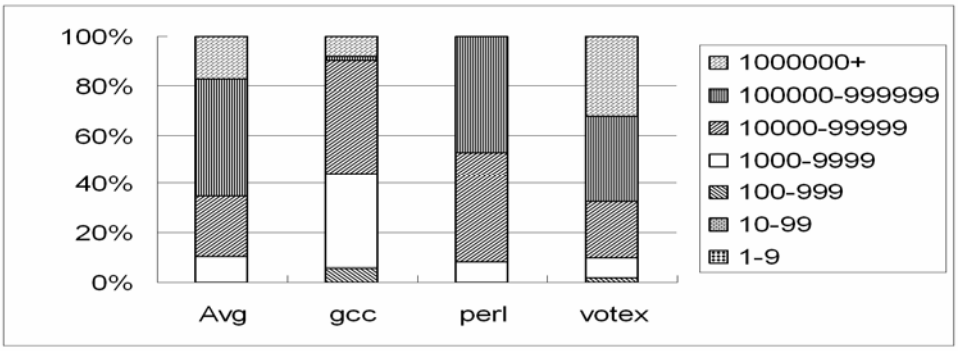

图 3 分支执行的频率占整个 SPEC 分支的权重

图 3 说明了不同分支指令执行在整个程序分支的权重，上面所说的执行 99 次或者更少的分支指令有 53%，图 2 所统计的执行 99 次或者更少的分支指令占 spec 程序的 53%，但是这 53%的指令在 spec 程序分支执行比重中只占 0.2%，在柱状图上都显示不出来。而图 2 所示的占 11%的分支指令占据了所有分支执行的 87%，也就是 10%的指令占据了程序 90%的运行时间。

图 4 统计了分支预测的分支成功率的分布。分支成功率是分支指令成功跳转除以整个程序运行的分支总数。从图上统计可以看出来大概 28%的指令总是跳转或者总不跳转，另外有 32%的指令，跳转的几率总少于 5%或者多于 95%总跳转，也就是说 60%的分支指令具有强烈的一个跳转方向，所以可以推出，具有挑战性的工作是预测其余 40%的分支跳转。

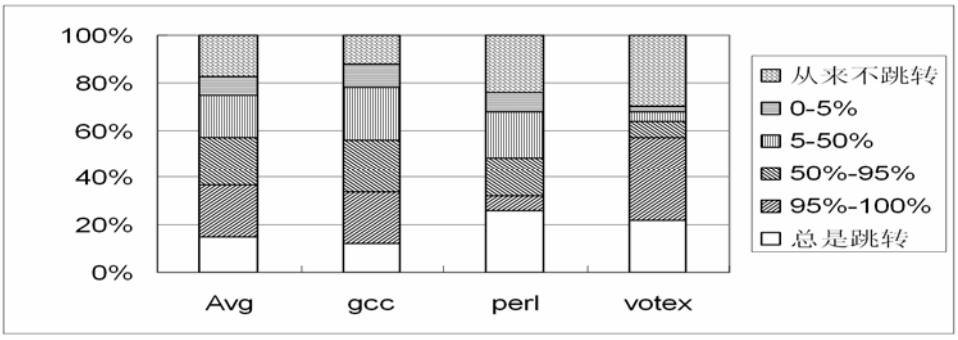

图 4 分支成功率权重分布

2.2 程序的分支属性

我们所运行的程序中有大量的分支指令，这些指令组合起来就表现为程序的分支行为，不同应用程序的分支行为非常复杂。国际上对分支预测的研究一般都主要针对SPEC CPU（简称SPEC）的程序，因为SPEC程序是专业测试机构精心挑选出来的、在各个行业有代表性应用的一组程序[3]，一般国际上都是通过对比SPEC的分数来说明处理器性能。对SPEC测试程序的分析表明 5 到 7 条指令里面就有 1 条分支指令，而处理器一般都是一次取 4 到 6 条指令，也就是说几乎每次都可能取到一条分支指令，这进一步说明了对分支指令研究的重要性，甚至有文章认为分支预测将来可能是处理器性能提高的瓶颈[4]。总体而言程序分支行为有以下特点：

循环结构在现在程序中大量存在。大量的研究表明，循环结构的分支指令行为在程序中是最为常见、重复出现的一种分支模式。一般我们可以用一个字符串来记录程序循环模式，比如一个循环 10 次的程序可以用 1110100110 来表示，其中 0 代表不跳转，1 代表跳转。前文已对分支指令的属性作了说明和分类，而多条分支指令组合成程序的分支行为，可以根据循环的模式来进行分支预测，这些模式在 SPEC 程序几十亿指令的执行过程中会重复出现。

程序分支属性存在系统性（或者联想性）。这种形式不重复出现，但是大量存在，例如 SPEC

的 parse 程序分析一个单词“chines”，它自然会预测到下一个字母是 e，[7]的研究中表明 30%的分支有这样的系统推测行为。但是由于程序的行为非常复杂，目前这方面研究还没什么太大进展。

程序分支之间有相关性。相关性一般多出现在程序的if-else分支[8]。研究表明如果一段程序中，2 条同样类型的分支指令有相反的分支结果，一般这两条分支指令是相关的。这就告诉我们如果预测成功其中一条分支指令，就可以知道另外一条分支指令的预测结果。以前的研究[7]表明 60％的分支指令具有相关性，这也成为后面分支预测的主要基础。

3 分支预测的预测机制

分支预测按输出结果可以分为两个问题：第一是分支指令方向预测，就是预测分支指令是跳转（taken）还是不跳转（not taken）；第二是分支指令地址的预测，分支地址只有在流水线指令执行阶段才能计算出来，为了避免等待，需要在译码阶段进行预测。一种常见的解决方法是采用 BTB[8]。

分支预测按工作方式可以分为静态和动态两种。静态是在程序编译时候进行分支预测，主要是对大量的循环有效。动态与静态的主要区别是，动态分支预测器可以在程序运行时根据分支指令执行情况进行调整，而静态分支预测在编译时完成后就不再改变了。

3.1 基本分支预测器

简单静态分支预测

最简单的静态分支预测有两种情况，要么预测分支总是执行跳转(taken)，要么预测分支总是不执行(not taken)。这种方法实现简单且预测速度快，但是静态分支预测对不同程序表现不一样，如果程序循环比较多，预测成功率就高，反之预测精度就低，预测精度一般都在 40%-60%之间。

BTFN 预测(Back Taken, Foreword Not taken)

针对静态分支预测的缺点，有人随后提出一种改进的方法：假如分支地址是正增加的那么就预测不跳转，如果分支地址是往回跳的就预测分支跳转，这种算法对单循环很有效。

以上两种预测机制的优点就是硬件开销很小，所以在半导体工业早期集成度还不是很高的时候被广泛使用，比如IBM360/370[9]。现在的应用程序大概每 5－7 条指令就有一条分支指令，这个精度根本不能满足处理器对高性能的需求，所以现在的高性能处理器基本没有采用的了，但是嵌入式处理器还有采用。

Profile 预测

在静态分支预测之后，学术界开始了在编译器上开始了基于分支的profile研究[10]，即先运行一遍程序把分支跳转的信息记录下来，然后把这些信息告诉编译器，在此基础上再进行编译，通过增加代码来增加程序空间局部性，这样在静态分支预测基础上能提高分支预测准确率。

1 位分支预测（last time）

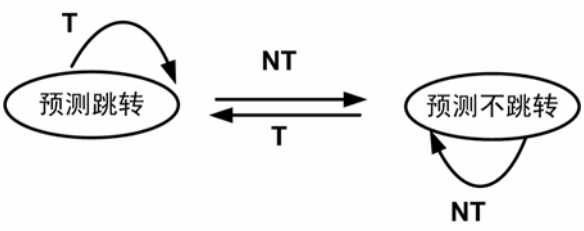

图 5 一位分支预测

静态分支预测不能满足要求，为了提高处理器性能又提出了动态分支预测器，其中最简单的就是 1 位分支预测器[11] 。基本思想就是当前分支预测和上次分支预测有同样的结果，所以也叫前次（last time）分支预测。1 位分支预测器通过一个动态位来记录分支结果，0 表示分支不跳转，1 表示分支跳转，分支指令通过地址来索引一个 1 位的表以预测分支是否成功，当分支指令提交的时候根据实际的方向修改表。

（BHR，Branch History Register），用来记录最近 k 条分支的分支结果。第二级被称作模式历史表（PHT，Pattern History Table) 通常历史记录在一个表里，PHT 的表组成可以多种多样，不过研究表明采用 2 位分支预测器最好。用 BHR 来索引 PHT，也就是根据我们前面程序分支特性介绍，2 级分支预测的基本思想就是用分支的历史模式来索引一个 2 位分支预测器。2 级分支预测精度在模拟情况下很好，达到了 97%。测试向量当时采用的是 SPEC CPU89。不过在后来的研究表明，SPEC CPU89 这套测试程序分支指令并不是很难被预测，甚至 2 位分支预测采用 SPEC CPU89 的测试向量，测试精度都能达到 90％左右。

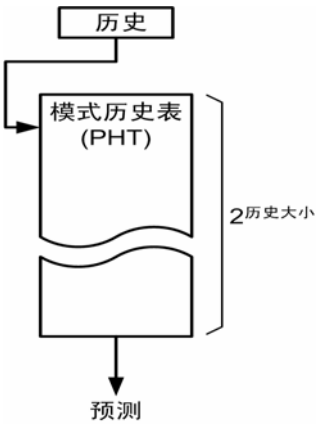

图 8 基于历史的分支预测

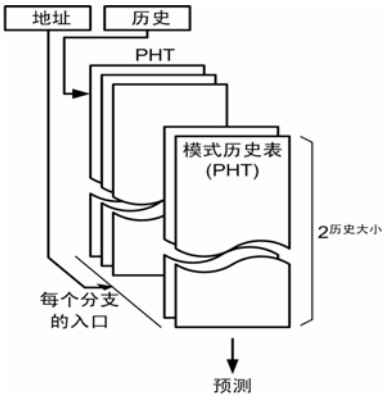

图 9 全局分支预测器

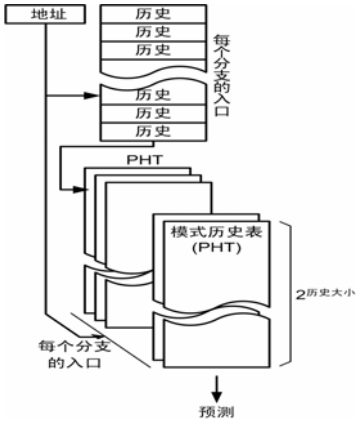

图 10 局部分支预测器

基于历史的分支预测

在发现上面分支相关问题后，S.潘、K.索和J.拉赫梅（S. Pan. K. So, and J. Rahmeh）提出了一种新的预测机制[12]，如图 8 所示，用一个全局历史寄存器去索引一个 2 位分支预测表，预测结果由 2 位分支预测表给出。这种分支预测的机理是，假设以前的n条分支指令和当前分支指令相关。但是这种分支预测机制，没有考虑地址，只考虑所关心的分支的历史，所以效果不是很明显。

全局分支预测

全局分支预测如图 9 所示。与局部分支预测不一样，全局分支的基本原理是，通过记录分支的前 n 次历史预测分支，来预测当前分支的输出结果。第一级是一个 BHR 通过移位记录历史，第二级是一个 2 位分支预测。这样的预测机制对 if-else 这样的指令相关预测非常有效，用踪迹驱动（trace driven）的模拟器研究数据表明：同样的代码，全局分支预测要比 2 位分支预测好。尤其现在高级语言里面有大量的 if-else 语句，因此用全局分支预测效果就会比较好。

局部分支预测

如图 10 所示，每一个分支指令地址都分别记录在一个BHR中，这些BHR就构成了转移历史表（BHT，branch history table)。传统的看法认为对于科学计算测试向量而言，局部分支预测要好于全局分支预测[15]，这是因为科学计算测试程序中含有大量的循环，每一个循环有一个BHT。但正因如此分支预测表的硬件开销很大，另外由于表一般很大，整个分支预测器训练的过程就比较长，对小程序效果反而会变差。

组合分支预测器

2 级分支预测出现后 ，大量研究表明不同的分支预测只能对某类的分支行为有效，而没有万能的分支预测。这种情况下有人就把不同分支预测组合起来，根据分支行为来选取不同分支预测器，这就是如图 11 的混合分支预测器[16]。

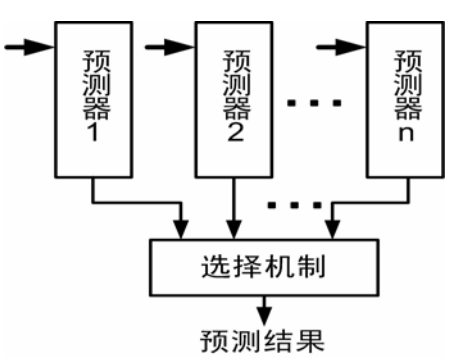

图 11 混合分支预测器

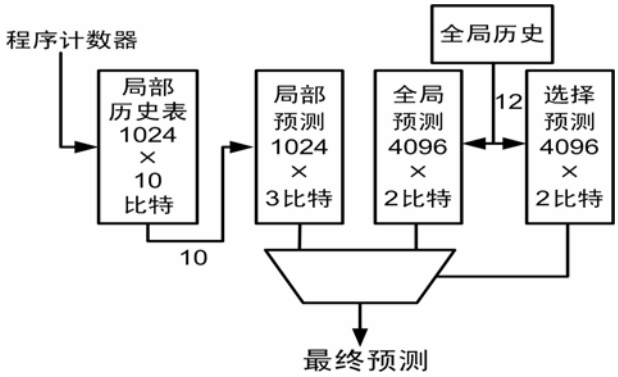

图 12 alpha21264 分支预测器

如图 11 所示，混合分支预测器可以选用多种分支预测器，然后通过一个选择机制进行预测，混合预测器的选择机制就成为一个需要考虑的问题。如图 12 所示，Alpha21264 就是采用局部 2 级分支预测器和全局 2 级分支预测进行混合的分支预测。全局用 12 条分支历史来索引 4k 项的 PHT；局部分支预测用一个 2 级分支预测，第一级是一个 1024 项的 10 位历史，分别用 0 和 1 表示跳转和不跳转，第二级是一个 1024 项的 3 位饱和分子预测器。这种预测机制使得对浮点预测达到了 1000 条指令 1 条预测错误，定点 1000 条指令有 11 条预测错误。

以上基本上是传统分支预测器的预测机制，下表是以前工业界处理器采用分支预测机制的情况。

	处理器	分支预测器
静态分支预测	Intel 8086	无分支预测
	Intel 486	总是跳转
	Sun superSparc	总是不跳转
	HP 7x00	BTFN
	早期的 PowerPC	Profile
动态分支预测	Alpha21064、AMD K5	1 位分支预测
	PowerPC604 、 MIPS R10000	2 位分支预测
	Pentium Pro、Pentium II	2 级分支预测
	Alpha 21264	组合分支预测

表 1 一些典型商品处理器的分支预测机制

4 分支别名干扰问题

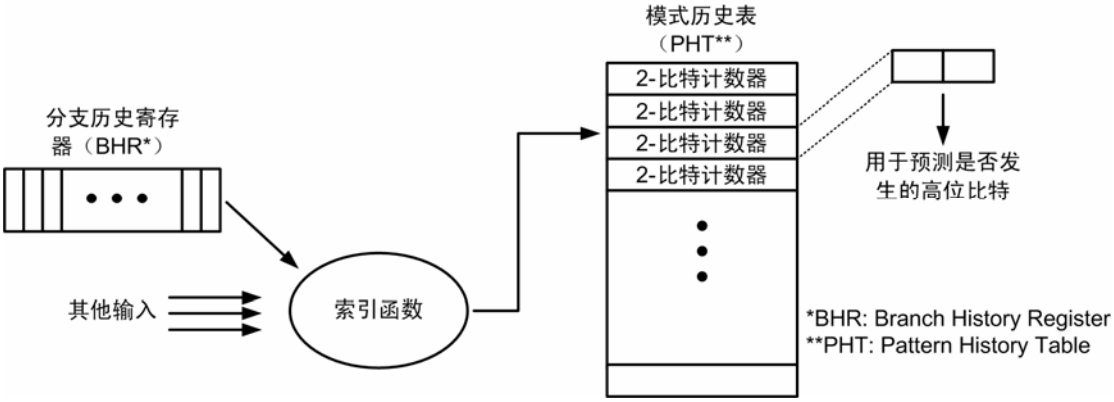

图 13 分支别名干扰问题

自从 2 级分支预测器出现以来，预测精度一般都在 92％左右，无论 BHT 和 PHT 表如何增大，效果也不是很明显。分支精度能不能提高成为当时超标量处理器性能提高的关键。经过大量研究表明，分支精度不能提高的主要原因是不同分支地址访问同一个 PHT，造成分支干扰。

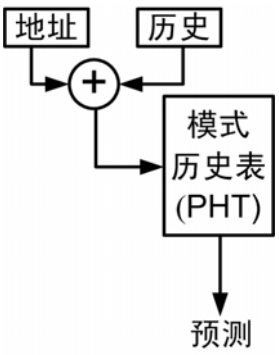

图 14 Gshare 分子预测器

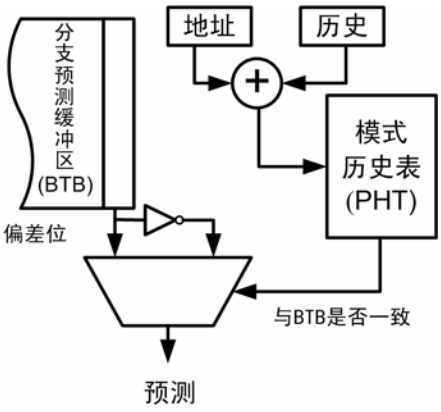

图 15 Agree 分支预测

Gshare 和 Gselect 分支预测器

克服这一困难最简单的方法是DEC西部实验室麦克法林（McFarling）提出来的Gshare分支预测器[16]，如图 14 所示。它把分支地址和分支历史进行异或运算，然后去索引PHT，这样的好处就是离散了原先的BHR。另外还有一种Gselect方法，分别取地址的高位和BHR的低位组合去索引PHT，实验结果表明Gshare比Gselect好一点[16]。

Agree 分支预测器

如图 15 所示，在BTB中设置了一个偏差位，Gshare的分支预测器输出从是否跳转转化为是否与BTB输出的结果一致，Agree的基本思想还是我们前面所说的分支指令 83%是具有强烈跳转或者不跳转的偏差特性的。在[17]文中说明仿真显示Agree分支预测器对SPEC的gcc测试程序有8%-30%的提高[17]。而且Agree的分支预测器在HP的处理器中得到实际应用。

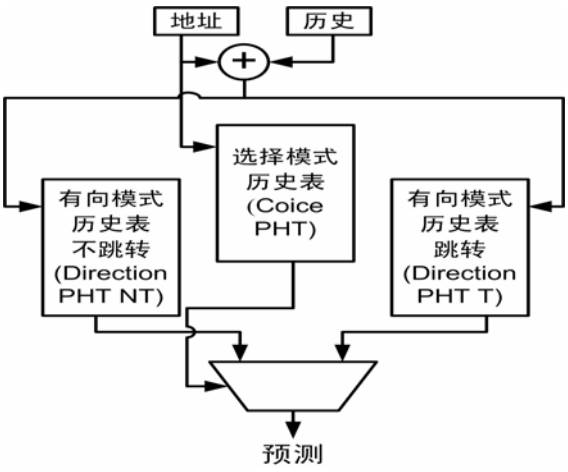

图 16　Bi-Mode 分支预测

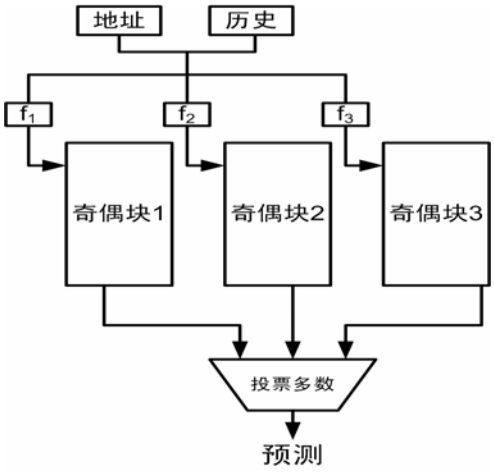

图 17　Skewed 分支预测

Bi-Mode 分支预测器

如图 16 所示，Bi-mode和Agree分支预测的思想一致，不过它是把容易发生跳转和不跳转的分布放入不同的PHT[18]。它由 3 部分组成，一部分用来选择PHT，另外两部分表示PHT的方向，分别为跳转和不跳转，PHT的方向被全局历史索引。

Skewed 分支预测器

如图 17 所示，Skewed的分支预测的主要思想是，分支的别名冲突并不是因为PHT太小，仿真发现PHT再大也不能降低别名冲突，而是组相连度不够。Skewed分支预测器把PHT分为 3 个奇偶块，通过哈希访问每一个块的 2 位分支预测[19]，假如分支预测错误 3 个块都更改，如果分支预测正确则只有一个块更改。Skewed分支预测器有 1/3-2/3 的容量冗余，而且对于上下文切换，分支的训练很慢。

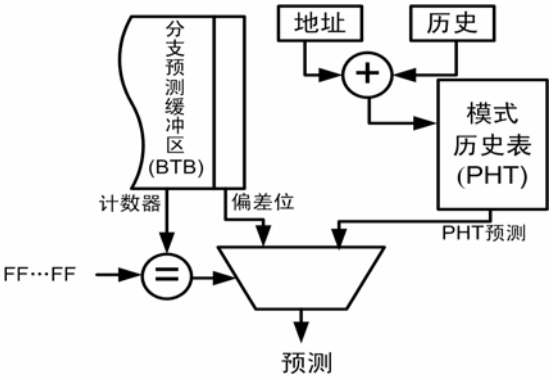

图 18 Filter 预测器

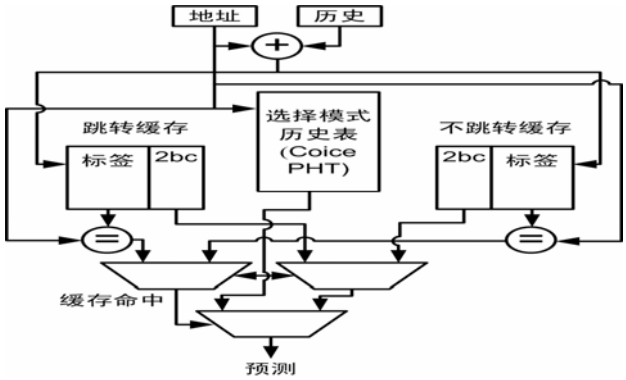

图 19 YAGS 分支预测器

Filter 分支预测器

如图 18 所示，Filter分支预测的主要思想是减少PHT中的冗余信息。PHT中大量的分支预测是高度偏向某个方向的，Filter机制是想把这些信息用一位分支预测器记录在BTB中。当一条分支指令处理后，如果分支方向和偏向位一致，则 1 位分支预测器增 1；否则 1 位分支预测器置为 0，偏向位翻转。假如 1 位分支预测器饱和，则表明该分支高度偏向一个方向，偏向位被设置为预测方向，分支指令采用PHT进行预测。Filter分支预测机制通过减少PHT的冗余消除了大量别名冲突，然而对高度偏向的分支指令的预测效果会差[20]。

YAGS 分支预测

如图 19 所示，YAGS（Yet Another Global Scheme）是在总结上述预测器的基础上进行组合的一种机制。由于 PHT 包含了大量的冗余信息，而实际上分支预测器只需存储历史的偏移就可以，而不必存储所有分支的所有内容。YAGS 采用前面介绍的 Bi-mode 结构存储这些偏移，只需增加标签

（tag）索引,而不用存储大量分支无关信息。不像传统缓存的标签，YAGS 的标签只是地址的低位。文章[21]仿真说明 YAGS 比前面的机制预测精度高而且硬件开销也不是很大。

5 分支预测的现状

历史总是重复出现，当年 2 位分支预测再也不能提高分支预测精度，发现了主要原因是没有考虑分支相关，从而提出了 2 级分支预测解决这个问题，使得预测精度大大提高。与此类似，当 2 级分支预测遇到瓶颈的时候，发现了主要原因是分支别名问题，也就是多地址访问同一个分支历史表，造成分支干扰，学术界又提出了大量的解决方案，多种分支预测器被提出，比如 Agree，Bi-mode 等，这个研究热情一直持续到 90 年代末。

伴随着指令集并行为代表的超标量处理器的成熟，传统分支预测也日趋成熟。当人们觉得分支预测已经很难再有进步的时候出现了新的转机，主要有如下原因：第一，随着新应用的出现及新处理器结构的提出，需要有新的预测机制，如下面介绍的SMT结构的分支预测等；第二，处理器功耗越来越重要，处理器性能和功耗的需要进行权衡，这就需要更好的理解处理器分支预测和功耗的关系；第三，对提高处理器分支预测精度的需求依然很大，比如分支精度提高 0.5%，10000 条分支指令就能够使得预测成功率提高 50 次，减少 50 次流水线的刷新，对性能提升很有好处。近几年国际上学术界开始举办分支预测大赛，神经网络分支预测器成为当前的研究热点，这方面文章在近年的Micro[2]，ISCA[3]上出现[22][23]。

SMT 分支预测器

随着超标量处理器的成熟，挖掘指令级并行性越来越困难，同时多线程（SMT，Simultaneous Multi-Threading）结构能有效挖掘线程级并行性，在现代处理器中得到广泛应用。由于 SMT 的基本特点是同时从多个线程取指令，如何设计分支预测器是个问题。是每个线程采用独立的分支预测器，还是多个共享一个分支预测器？如果多个线程共享一个分支预测器，会不会发生线程间干扰现象？这些问题都需要研究。

文章[24]指出，如果多个线程共享一个分支预测器，而且保持 PHT 或者 BTB 的大小和线程数目成比例增长，那么各个线程之间的干扰就很小。文章[25]研究了 SMT 结构分支预测器的组织结构，把 SMT 的分支预测器分为 4 类，包括:多个线程共享一个分支预测、每个线程一个分支预测器、每个线程共享 PHT 分别有自己 BHT、每个线程共享 BHT 分别有自己的 PHT。最后仿真结果表明，每个线程分别有一个分支预测器比多个线程共享一个分支预测器效果要好，但差别不是很明显。

Power Aware（节约功耗导向）分支预测

2000 年以来，半导体工业进入了深亚微米，集成度越来越高，处理器规模越来越大。这就使得处理器功耗急剧上升，此外芯片过热使得芯片封装很困难，这样 Power-aware 的处理器设计成为研究热点。由于分支预测器对处理器性能影响很大，如何平衡它的精度和功耗是一个值得研究的问题。文章[26][27]研究了通过分支预测来减少漏电功耗；[28]研究了分支预测和功耗的关系。

神经网络预测器

随着分支预测研究的深入，学术界认识到分支预测的本质是机器学习问题，而机器学习已经在模式识别、自动控制等领域进行过大量研究，其中神经网络是目前机器学习的有效方法。罗格斯（Rutgers）大学的丹尼尔.A.希梅内斯（Daniel. A. Jimenez）最先在 2001 年HPCA会议上将神经网络引入了分支预测领域[28]。文章[28]研究表明，对于 4K的硬件开销，采用SPEC CPU2000 测试程序，神经网络分支误预测率要比Gshare降低了 26%，比组合分支预测降低了 12%。将神经网络引入分支预测领域，使得分支预测研究又开始活跃，许多新的分支预测机制被提出[29]。分支预测

[2] Annual International Symposium on Microarchitecture
[3] International Symposium on Computer Architecture

和神经网络都有很多内容，可以参考文章[30]。

分支预测竞赛

由于应用程序在变，程序行为也发生变化，为了进一步提高分支预测的精度以提高处理器性能，国际上从 2004 年开始每两年举行一次分支预测大赛[31]。这个竞赛对每一个参赛者的约束是：在(64K+256)Bit的分支预测大小取得最好的分支预测精度，而不用考虑功耗、价格、延迟等影响。提交的分支预测器通过 21 个踪迹（trace）来比较，踪迹又分为 2 种，一种是给每个竞赛者用的分发的踪迹，还有一种是不给竞赛者而组织者进行测试的，每 3 个踪迹又分为一组，分别为SPECfp、SPECint、Internet、Multimedia、Productivity、Server和Workstation 7 组代表不同应用。

在第一届分支预测竞赛上，美国的华人教授高洪亮（译音，Hongliang Gao）取得未分发踪迹的第一名，而分发踪迹的第一名由法国的安德列（André）教授获得。他们的分支预测正确率都在 98％左右（作为对比，gshare 大概在 95％左右），而分支预测正确率的 1％提升对于处理器性能的提高都很大。

Hongliang Gao的分支预测是采用神经网络预测器，由此也引发了最近分支预测研究和神经网络结合的热潮。相比而言安德列（Alpha EV8 分支预测器的主要设计者[32]）的分支预测器更实用一些，通过优化分支指令长度得到了很好效果。

6 结束语

综上所述，分支预测的发展一直伴随着高性能处理器的发展，分支预测器对处理器性能提高起着重要作用，一直都是处理器设计的重中之重。目前分支预测技术发展遇到了一个瓶颈，其最高精度达到 95-98%，很难进一步提高。但是随着大量新应用程序的出现，又有新的程序分支行为，以及对硬件开销、功耗等因素考虑，分支预测技术将会有一些新的发展。例如，SPEC组织正在进行功率测试标杆程序（Power benchmark）的研究；另外随着半导体摩尔定律的延续，分支预测器可以越做越大（比如Alpha 21464 分支预测器有 352KBit[32]，几乎和一级缓存规模相当）；此外在分支预测器设计中如何平衡功耗是个大问题。目前神经网络分支预测器是学术界的研究热点，可以预见近期这方面研究将有可能取得大的进展。

参考文献：

[1] YN Patt, SJ Patel, DH Friendly, and J. Stark. One Billion Transistors, One Uniprocessor, One Chip, IEEE Computer, pp. 51 -- 57, Sept. 1997

[2] Introduction to Intel Core Duo Processor Architecture， Volume 10 Issue 02 May 15, 2006

[3] The Standard Performance Evaluation Corporation http://www.specbench.org/

[4] P. Ranganathan, and N. Jouppi. The relative impact of memory latency, bandwidth and branch limit to microprocessor performance. In Proceedings of the 1s Workshop on Mixing Logic and DRAM: Chips that Compute and Remember (held inconjunction with the 1997 International Symposium on Computer Architecture), June 1997.

[5] D. Kaeli and P. Emma, “Branch history table predictions of moving target branches due to subroutine returns,” in 18th Annu. Int. Symp. Computer Architecture, 1991.

[6] P.-Y. Chang, E. Hao, and Y. N. Patt, “Predicting indirect jumps using a target cache,” in Proc. 24th Annu. Int. Symp. Computer Architecture,1997.

[7] M. Evers, S. J. Patel, R. S. Chappell, and Y. N. Patt, “An analysis ofcorrelation and predictability: What makes two-level branch predictors work,” in 25th Annu. Int. Symp. Computer Architecture, 1998.

[8] DA.Patterson and JL Hennessy. Computer Architecture: A Quantitative Approach 3rd. ed. Morgan Kaufmann, San Mateo, 2003.

[9] D. W. Anderson et al. The IBM System/360 model 91: Machine philosophy and instruction handling, IBM Journal (Jan. 1967), 8-24.

[10] J. A. Fisher and S. M. Freudenberger, “Predicting conditional branch directions from previous runs of a program,” in 5th Int. Conf. Architectural Support for Programming Languages and Operating Systems,1992.

[11] J. E. Smith, “A study of branch prediction strategies,” in 8th Int. Symp. Computer Architecture, 1981.

[12] S. Pan. K. So, and J. Rahmeh. Improving the accuracy of dynamic branch prediction using branch correlation. In Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems, 1992, pp.76-84.

[13] T.-Y. Yeh and Y. N. Patt, “Two-level adaptive branch prediction,” in 24th ACM/IEEE Int. Symp. Microarchitecture, 1991.

[14] “Alternative implementations of two-level adaptive branch prediction,” in 19th Annu. Int. Symp. Computer Architecture, 1992.

[15] C. Yound and M. Smith. Improving the accuracy of static branch prediction using branch correlation. In Proceedings of the 6thInternational Conference on Architectural Support for Programming Languages and Operating Systems, October 1994, pp. 232-241.

[16] S. McFarling, “Combining branch predictors,” Digital Equipment Corporation, WRL Tech. Note TN-36, 1993.

[17] E. Sprangle, R. Chappell, M. Alsup, and Y. Patt. The Agree predictor: Amechanism for reducing negative branch history interference. In Proceedings of the 24th Annual International Symposium on Computer Architecture, May 1997.

[18] C.-C. Lee, I.-C. Chen, and T. Mudge. The Bi-Mode Branch Predictor. Proc. MICRO 30, Dec. 1997.

[19] P. Michaud, A. Seznec, and R. Uhlig. Trading conflict and capacity aliasing in conditional branch predictors. In Proceedings of the 24th Annual International Symposium on Computer Architecture, May 1997.

[20] P. Chang, M. Evers, and Y. Patt. Improving branch prediction accuracy by reducing pattern history table interference. In Proceedings of the International Conference Parallel Architecture and Compilation Techniques. October 1995.

[21] A. Eden and T. Mudge, “The YAGS branch predictor,” 31th Ann. IEEE/ACM Symp. Microarchitecture (MICRO-31) Dec 1, 1998

[22] Daniel A. Jiménez, *Fast Path-Based Neural Branch Prediction*, Proceedings of the 36th Annual International Symposium on Microarchitecture (MICRO-36), December 2003

[23] A. Gandhi, H. Akkary, and ST Srinivasan. Reducing branch misprediction penalty via selective branch recovery. In Proceedings of The Tenth International Symposium on High-Performance Computer Architecture, pages 254--264, December 2004.

[24] Branch Prediction and Simultaneous Multithreading. SCbastien Hily* and AndrC Seznec. IRISA/INRIA. Proceedings of the 1996 Conference on Parallel Architectures and Compilation Techniques (PACT '96)

[25] Matt Ramsay, Chris Feucht, and Mikko H. Lipasti, Exploring efficient SMT branch predictor design. Workshop on Complexity-Effective Design , in conjunction with ISCA, 2003. 26

[26] Z. Hu, P. Juang, P. Diodato, S. Kaxiras, K. Skadron, M. Martonosi,and D.W. Clark, “Managing Leakage for Transient Data: Decay and Quasi-Static Memory Cells,” Proc. 2002 Int’l Symp. Low Power Electronics and Design, pp. 52-55, Aug. 2002.

[27] Z. Hu, P. Juang, K. Skadron, D. Clark, and M. Martonosi, “Applying Decay Strategies to Branch Predictors for Leakage Energy Savings,” Proc. 2002 Int’l Conf. Computer Design, pp. 442-445, Sept. 2002.

[28] JIM´ENEZ, D. A. AND LIN, C. 2001. Dynamic branch prediction with perceptrons. In Proceedings of the Seventh International Symposium on High Performance Computer Architecture, 197–206.

[29] Daniel A. Jiménez, Piecewise Linear Branch Prediction, Proceedings of the 32nd International Symposium on Computer Architecture (ISCA-32), June 2005

[30] Daniel A. Jiménez and Calvin Lin, *Neural Methods for Dynamic Branch Prediction*, ACM Transactions on Computer Systems, Vol. 20, No. 4, November 2002.

[31] http://www.jilp.org/cbp

[32] Design Tradeoffs for the Alpha EV8 Conditional Branch Predictor. to appear in Proceedings of the 29th IEEE-ACM International Symposium on Computer Architecture, 25-29 may 2002

作者简介：

冯子军： 中科院计算技术研究所微处理器研究中心，博士生

肖俊华： 中科院计算技术研究所微处理器研究中心，博士生

章隆兵： 中科院计算技术研究所微处理器研究中心，副研究员