维普资讯 http://www.cqvip.com

谓词执行及其关键技术浅析

沈 立 鲁建壮 王志英
(国防科大计算机学院 长沙410073)

Predicated Execution and its Key Techniques

SHEN Li LU Jian-Zhuang WANG Zhi-Ying
(Computer College, National University of Defense Technology, Changsha, Hunan, 410073)

Abstract As a program transformation technique, control dependence can be converted to data dependecnce by predicatted execution with some architectural support. Predicated execution and its key techniques are discussed in this paper, and IA-64 is selected as an example to indicate the realizaation of predicated execution. At last, questions need to be studied fatherly are proposed.

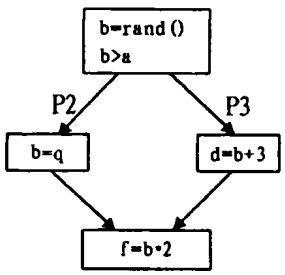
Keywords Guarded execution, Predicated execution, IA-64

1 引言

在超标量和 VLIW 微处理器的设计中，指令间的相关，尤其是控制相关和数据相关，严重限制了指令级并行(ILP)的开发，从而限制了微处理器性能的进一步提高。条件执行技术(Guarded Execution)能够将控制相关转化为数据相关来处理。具体来说，它能够将控制相关于一条分支指令的其他指令转换为数据相关于该分支条件的条件指令。条件指令与常规指令的不同之处在于它含有显式的条件指示符，其语义为：首先计算指令执行条件，如果条件为真，则执行该指令中的操作，否则将其作为空操作处理。条件执行技术实质上是一种程序变换技术，变换后的程序无论对编译优化还是对硬件调度都有很大好处，但需要专门硬件机制的支持。目前只有 ARM 指令集与 IA-64指令集支持条件执行。

条件执行技术有许多优点：它能够消除大部分控制相关，扩大基本块，使得编译调度更加充分；由于分支指令数目减少，简化了硬件设计，也减少了分支预测错误时的性能损失。但它也有一些缺点：由于它使得更多的指令可以提前发射执行，进一步增加了处理器执行控制部件的复杂度以及处理器中寄存器的数目和复杂度；因为它将不同分支路径的指令变成了一个基本块中的指令，但又没有保留各个分支路径的其他信息，编译很难权衡调度哪些指令。

谓词执行技术(Predicated Execution)[1]是显式并行技术(EPIC)的一个重要的组成部分，是条件执行技术的一种实现，它为每条指令增加一个源操作数(即谓词)作为指令执行条件，当谓词为真时执行指令中的操作，否则将其转换为空操作处理。图1给出了两段代码，(a)为普通代码，(b)为转换后的代码，这里 P_2 和 P_3 为谓词，(b)中的第二条指令是谓词设置指令。可以看出，转换后分支指令已被谓词设置指令所替换，原有的四个基本块被合并为一个基本块，合并后的基本块称为超块(HyperBlock)[3]。超块只有一个入口，就是它的第一条指令，但可以有多个出口，同一超块内部各基本块之间的的所有控制相关都被消除，编译调度更加充分。通过引入谓词使得编译器有了更大的调度空间，在扩大基本块体积的同时，也扩大了数据和控制前瞻的范围。



(a) 普通代码

```
b=rand()                  if true // b=random number
P2, P3 cmpp.un.uc b>a     if true // if b>a then P2=true, P3=false
                                  else P2=false, P3=true
b=q                       ifP2 // ifP2 is true, b=q else nullify
d=b+3                     ifP3 // ifP3 is true, d=b+3 else
                                  nullify statement
f=b*2                     if true // f=b*2
```

(b) 转换后的代码

图1 谓词执行技术代码实例

谓词执行技术能够有效地增加基本块大小，从而增加基本块中的指令级并行性(ILP)。研究表明，合并多个控制能够将指令级并行性增加到原来的3倍[9]，而且能够有效地提高其他代码变换技术如软流水(soft pipeling)[10]和模调度(module scheduling)[11]的性能。另外，它能够有效减少控制流的变化，从而减少指令 Cache 的失效。但它也存在一定的不足，如：需要专门指令集的支持，需要增加寄存器文件的读/写端口，需要编译器提供专门支持，执行的无效指令(执行但不被确认的指令)数增加。

本文主要讨论了谓词执行技术及其关键问题，以 IA-64 为例分析了谓词执行技术实现所需的编译技术和体系结构支持，并提出了若干需要进一步研究的问题。

维普资讯 http://www.cqvip.com

2 谓词执行的关键问题

谓词执行技术需要完成的工作包括：选择若干个基本块，根据基本块所在的分支路径为它们指定谓词，消除各基本块之间的分支指令并将它们转换为谓词定义指令。为此，需要解决两个关键问题：(1)怎样为每个基本块指定谓词?(2)将谓词定义指令放置在程序中什么位置?

每个基本块对应的谓词由它所在的分支路径和 PDG 中的所有分支条件决定，因而每一个问题是一个纯粹的图论问题。每段代码都可以根据其自身的控制相关表示为相应的程序相关图(Program Dependence Graph，PDG)[2]，图中每一个顶点都表示一个基本块，各顶点对应的谓词应由它所在的边决定。这个问题就可以转换为如何为 PDG 中的每个顶点指定谓词。假设 T 是 PDG 中所有分支转移条件的集合，T 中有 n 个元素，那么 PDG 中所有谓词的集合 $P=\{t_1 \wedge t_2 \wedge \cdots \wedge t_n\}$。如果 N 是 PDG 中所有顶点的集合，那么这个问题等价于函数 $R:N\rightarrow P$。

第二个问题不仅要考虑每个基本块所在的分支路径，还要考虑能够确定分支转移条件的最早位置。另外，在插入谓词定义指令时还需考虑由于引入谓词带来的数据相关，应尽早计算出各基本块所对应谓词的值。使用问题一中的假设，这个问题等价为函数 $K:P\rightarrow\langle n_1,n_2\rangle$，这里 $n_1,n_2\in N$，其含义为将谓词 p 的定义指令放在顶点 n_1 中，而将谓词 $\neg p$ 的定义指令放在顶点 n_2 中。

解决这两个问题的经典算法是 If-conversion 转换算法，该算法能根据每个程序的 PDG 确定函数 R 和 K，详细情况见参考文[4]。

3 谓词执行的支持技术

实现谓词执行技术并充分发挥其优点能够大幅度提高微处理器的性能，但这需要编译技术和体系结构的共同支持。

3.1 编译技术支持

研究表明，通过 If-conversion 转换并消除所有分支指令总能提高数字应用程序的性能[7]，而对非数字应用程序，由于它具有基本块体积更小、分支指令可预测性更差等特点，选择什么样的分支指令进行转换将直接影响其性能[3]。因此，在编译时应选择那些执行频率高且可预测性差的分支指令进行 If-conversion 转换才能提高性能，同时这样也有助于提高分支预测的准确率，减少因分支预测失败造成的性能损失。选择什么样的分支指令进行 If-conversion 转换是编译需要解决的第一个问题，它等价于如何选择来自不同分支路径的基本块组成超块。

超块质量的高低将直接影响编译调度的效果，是否将一条分支路径合并入某个超块通常需要考虑以下因素：资源利用率、无效指令、分支路径固有的指令冲突以及执行频率。超块由多个分支路径合并得到，来自各分支路径的指令将共享处理器的所有硬件资源，因而可能造成更多的结构冲突，影响指令级并行的开发；超块中只有一部分指令的执行结果会被确认，其余将被丢弃，这些指令就是无效指令，它会造成资源浪费，妨碍处理器性能的提高；如果某一分支路径内部存在过多的指令相关，那么必将影响整个超块的调度结果；分支路径的执行频率是分支预测和前瞻执行(Speculative Execution)的重要依据，也可以作为选择分支路径并入超块的依据。

编译器需要解决的另一个问题是：If-conversion 转换应在编译的哪个阶段完成。一般说来，If-conversion 既可以在指令调度之前完成，也可以和指令调度一起完成。在指令调度前完成 If-conversion，编译器能够充分利用谓词表示提供的信息，有助于更充分地完成指令调度优化；而在指令调度的同时完成 If-conversion，能够根据程序轮廓信息，如分支执行频率和分支转移概率等，更好地选择分支指令进行转换和消除，因而能够更好地解决第一个问题。

3.2 体系结构支持

要实现谓词执行技术，传统的超标量/VLIW 体系结构必须进行一定的修改。首先应该修改指令格式，为每个指令增加一个源操作数(即谓词)，并提供专门的存储单元保存每条指令所对应的谓词。最常用的方法是提供一个多端口谓词寄存器文件，每个寄存器的长度都是1位。

一些指令集提供了比较赋值指令，可以根据一定的条件为寄存器赋值，这种指令可用作谓词定义指令，但它一次只能对一个寄存器赋值。条件分支指令的存在使得某些基本块的谓词之间存在一定的联系，例如图1中的 P_2 和 P_3，显然 $P_2=\neg P_3$ 成立，即谓词 P_2 和 P_3 可以同时确定，采用这种指令指定谓词的值显然效率不高。所以还应该扩展指令集，增加专门的谓词定义指令，提高谓词定义操作的效率。

最后，则应修改指令流水线，实现谓词执行的语义，即：当谓词为真时，执行指令的操作，否则，将指令转换为空操作处理。这既可以在译码阶段实现，也可以在写回阶段实现。在译码阶段实现时，译码部件根据谓词决定流出指令的操作还是空操作；而在写回阶段实现时，则是根据谓词决定指令的执行结果是否写回。一般说来，在译码阶段实现可能会对实际性能产生影响，因为谓词作为源操作数会带来更多的数据相关；而在写回阶段实现则有可能造成更多的无效指令，即指令虽然被执行但其结果却没有被确认，这样会造成一定程度的资源浪费。

当然，在不同的体系结构中，它们的具体实现也不相同。

4 实例

支持谓词执行的体系结构有很多，包括 Cydra5[5]、HPL Playdoh[6]和 HP/Intel IA-64[6]等，本节我们以 IA-64为例讨论谓词执行技术的实现。

IA-64提供了64个1位的谓词寄存器[12]，其中 PR_0(Predication Register)总是为1。由于对谓词寄存器访问频繁，谓词寄存器文件提供了多个独立的访问端口，包括15个读端口、11个写端口以及若干支持谓词寄存器传输指令的读写端口。为支持软流水，16～63等高48个寄存器是可以进行旋转的。

IA-64的指令编码如图2所示，每个指令字都是128位，包含3个指令槽和1个5位的指令模板，模板字段用来传递编译器提供的调度和并行化信息，每个指令槽中放置一条41位的指令，其低6位指明了该指令要访问的谓词寄存器。

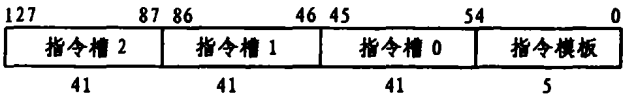


图2 IA-64的指令编码格式

IA-64对指令集[13]进行了扩展，增加了谓词定义指令和谓词传输指令。谓词定义指令支持10种浮点和整数的比较关系以及5种比较结果的写回方式，其中整数比较指令能够同时修改两个谓词寄存器；为了提高循环分支指令的预测准确率，

维普资讯 http://www.cqvip.com

它允许分支部件修改谓词寄存器 PR_{63}。谓词传输指令能够在谓词寄存器和通用寄存器之间传输数据，而且多个谓词寄存器可以并行传输。这两类指令的格式分别如图3与图4所示。

(*qp*)cmp. *crel*. *ctype* *p1*,*p2*=*r2*,*r3*	register 类型
(*qp*)cmp. *crel*. *ctype* *p1*,*p2*=*imm8*,*r3*	imm8类型
(*qp*)cmp. *crel*. *ctype* *p1*,*p2*=*r0*,*r3*	parallel_inequality 类型
(*qp*)cmp. *crel*. *ctype* *p1*,*p2*=*r3*,*r0*	pseudo-op
(*qp*)fcmp. *frel*. *fctype*. *sf* *p1*,*p2*=*f2*,*f3*	Floating-Point 类型

图3 IA-64谓词定义指令格式

图3中 *qp* 是该指令对应的谓词寄存器，*crel*/*frel* 指明具体的比较关系，如果比较结果为真则谓词值为1，否则其值为0。谓词的值根据 *ctype*/*fctype* 指定的写回方式写回目的谓词寄存器 *p1*和 *p2*。

(*qp*)mov *r1*=*pr*.	from 类型
(*qp*)mov *pr*=*r2*,*mask*17	to 类型
(*qp*)mov *pr*. *rot*=*imm*44	to-rotate 类型

图4 IA-64谓词传输指令格式

图4列出了三类谓词传输指令，from 型指令能够将谓词寄存器 PR_i 的值复制到通用寄存器 R_1 的第 i 位；to 型指令将通用寄存器经屏蔽后的结果拷贝到谓词寄存器中，这里 mask 即屏蔽码；to-rotate 型指令将28位的立即数左移16位形成44位，再将符号扩展的结果复制到旋转谓词寄存器中。

IA-64的流水线[12]共分十级：IPG、FET、ROT、EXP、REN、WLD、REG、EXE、DET、WB，EXP 段进行谓词寄存器的旋转和重命名，REG 段读取谓词寄存器并检测数据相关，EXE 段完成谓词定义指令的执行并产生结果，同时对于条件执行的指令（分支指令除外）读取其谓词的值并将其向后逐级传送，分支指令则在 DET 段读取相应的谓词，在 WB 段根据谓词判断指令结果是否提交并进行相应的提交/作废操作。

IA-64的编译器针对超块结构进行了大量的优化调度工作，它首先将超块转换为 PSSA（Predicated Static Single Assignment）格式[8]的中间代码，然后对中间代码进行调度优化，最后在超块的所有出口增加补偿代码以保证获得正确的执行结果。针对超块进行的优化调度包括：谓词前瞻（Predicated Speculation，PSpec）和降低控制高度（Control Dependence Reduction，CHR）[8]。PSpec 技术能够在确定谓词值之前瞻执行指令，而 CHR 技术能够尽可能早地确定谓词的值，从而减少需要进行谓词前瞻的指令。经统计，通过以上调度技术，能够将执行时间缩短12%到68%[8]。

小结 谓词执行技术是条件执行技术的实现，作为一种程序变换技术，它能够将程序中的控制相关转换为相对于控制条件的数据相关。谓词执行技术有两个关键问题：如何为每个基本块指定谓词，以及在何处放置谓词操作指令；充分发挥谓词执行技术的优点，需要编译技术和体系结构的共同支持；最后，本文以 HP/Intel IA-64为例讨论了谓词执行技术的实现。

谓词执行技术还存在一定的不足或需要进一步研究的方面，主要包括：

1. 谓词寄存器是一个多端口寄存器，硬件实现开销太大，如何进行简化？

2. 如何在指令执行条件中增加分支路径信息，这样有助于提高编译指令调度的效率。

3. If-conversion 能够转换并消除分支指令，可以减少分支预测错误，减少因分支预测失败造成的性能损失。如何根据程序轮廓信息选择分支指令进行转换？

4. If-conversion 转换可以在指令调度前完成，也可以和指令调度一起完成，两种方式各有利弊。如何结合 If-conversion 和指令调度这两个过程才能获得最高性能？

这些问题都需要进一步研究。

参 考 文 献

1 Park J C H, Schlansker M. On Predicated Execution: [Technical Report HPL-91-58]. HP Labs, May 1991

2 Ferrante J, Ottenstein K, Warren J D. The Program Dependence Graph and Its Use in Optimization. ACM Trans. Program. Lang. Syst. 1987, 9(3): 319～349

3 Mahlke S A, Lin D C, Chen W Y, Hand R E, Bringmann R A. Effective compiler support for predicated execution using the hyperblock. In: Proc. of the 25th Intl. Symposium on Principles of Programming Languages, 1983. 177～189

4 Allen J R, Kennedy K, Portfield C, Warren J. Conversion of Control Dependence to Data Dependence. In: Conf. Record of the 10th Annual ACM Symp. On Principles of Programming Languages, 1983. 177～189

5 Rau B R, Yen D W L, Towle R A. The Cydra 5 departmental supercomputer. IEEE Computer, 1989, 22: 12～35

6 Dehnert J C, Hsu P Y, Rau J P. HPL playdoh architecture specification: Version 1.0: [Tech. Rep. HPL-93-80]. Hewlett-Packard Laboratories, Palo Alto, CA 94303, Feb. 1994

7 Dehnert J C, Hsu P Y, Bratt J P. Overlapped loop support in Cydra 5. In: Proc. of the 3rd International Conference on Architectureal Support for Programming Languages and Operating Systems, April, 1989. 26～38

8 Carter L, Simon E, Calder B. Carter L, Ferrante J. Path Analysis and Renaming for Predicated Instruction Scheduling. International Journal of Parallel Programming, special issue, 2000, 28(6)

9 Lam M S, Wilson R P. Limits of Control Flow on Parallelism. In: Proc. of the 19th Annual Intl. Symposium on Computer Architecture, Gold Coast, Australia, May 1992. 46～57

10 Rau B R, Glaeser C D. Some Scheduling techniques and an Easily Schedulable Horizontal Architecture for High Performance Scientific Computing. In: Proc. of the Fourteenth Annual Workshop on Microprogramming, Oct. 1991. 183～198

11 Rau B R. Iterative Modulo Scheduling: An Algorithm For Software Pipelining Loops. In: Proc. of the 27th Annual International Symposium on Microarchitecture, San Jose, California, 1994. 63～74

12 Intel® IA-64 Architecture Software Developer's Manual Volume 3: Instruction Set Reference Revision 1.1, July 2000

13 Sharangpani H, Arora K. Itanium Processor Microarchitecture. IEEE Computer Sociey, Sep. —Oct. 2000. 24～43