文章编号:1001 - 2486(2005)01 - 0060 - 04

判定执行的功耗优化*

王永文,张民选
(国防科技大学 计算机学院,湖南 长沙 410073)

摘 要:判定执行消除分支指令,有助于提高性能,但执行额外的指令会造成能量浪费。尽早作废无效的判定指令,可以减少能量消耗。针对 Itanium 2 处理器,修改流水线功能划分,提前读取谓词的值,提出谓词相关情况下的流水线停顿方法。模拟结果表明,提前读取谓词并作废无效指令,能减少能量浪费,提高能量效率。

关键词:判定执行;流水线;相关;低功耗设计
中图分类号:TP302.7 文献标识码:A

Power Optimization for Predicated Execution

WANG Yong-wen, ZHANG Min-xuan
(College of Computer, National Univ. of Defense Technology, Changsha 410073, China)

Abstract: Predicated execution improves the performance by eliminating branches but wastes energy due to extra instructions. Nullifying FALSE-predicted instruction earlier can save energy. Based on the Itanium 2 microprocessor, we modify the pipeline function; advance the read of predicate and present methods to stall the pipeline on predicates dependency. Simulation results show that the advanced predicate reading and FALSE-predicated instruction nullifying can reduce energy waste and improve the efficiency of energy.

Key words: predicated execution; pipeline; dependency; low-power design

判定执行(predicated execution)是指令基于谓词的有条件执行[1],它消除分支指令,将控制相关转换为数据相关。判定执行给指令附加一个执行条件,称作谓词(predicate)。谓词的值由谓词定义指令根据比较运算的结果设定,谓词的值决定指令的执行状态:谓词为真的判定指令正常执行,谓词为假的判定指令的结果必须要作废,不能修改体系结构状态。

判定执行有利于性能的提高。首先,判定执行把条件分支指令从动态指令流中消除,减少了处理器每个时钟周期遇到的分支的数目,消除了处理器在分支发射率上的限制。其次,判定执行消除了分支预测失败的开销,可以极大提高性能。据统计,判定技术能够平均消除 27% 的分支以及 56% 的分支预测失败[1]。然而,判定执行不利于降低功耗,因为这种机制执行了程序所不需要的无效指令,引起不必要的能量浪费。

先前的研究工作主要考虑如何提高判定执行的性能,而没有考虑其功耗特性[2~4]。本文主要研究判定执行的功耗特性,认为能量浪费是由无效指令的执行引起的,并通过提前作废无效指令的方法来减少能量浪费。

1 判定执行模型

当前已经有若干体系结构实现了判定执行,IA - 64[5] 是 Intel 和 HP 公司联合定义的 64 位高性能微处理器体系结构,它采用 EPIC 设计哲学,利用了超标量和 VLIW 两种体系结构的优势,强调编译器和体

* 收稿日期:2004 - 09 - 17
基金项目:国家自然科学基金资助项目(60273069,90207011)
作者简介:王永文(1977—),男,博士生。

系结构共同开发指令级并行。Itanium 系列微处理器是 IA－64 体系结构的商品化实现,表现出强大的性能和良好的发展前景。我们将以 IA－64 体系结构作为基准模型。

IA－64 体系结构给每条指令增加一个谓词操作数,谓词的值决定指令的执行状态。另外增加了一组谓词定义指令,根据源操作数的比较结果来设定谓词的值。Itanium 处理器定义了 64 个一位的谓词寄存器,用于保存指令的执行谓词。所有的指令都被读取并执行,只有谓词为真的指令的结果被保存,谓词为假的指令的结果将被丢弃。

Itanium 2 处理器设计有 8 级流水线[6],如表 1 所示。Itanium 2 处理器的谓词定义指令在 EXE 站执行。判定指令在 REG 站读取谓词,在 EXE 站执行。只要谓词定义指令在相关的判定指令的前一周期发射,就可以保证判定指令在执行前能得到正确的谓词值。

表 1 Itanium 2 处理器的流水线

Tab. 1 Pipeline of Itanium 2 microprocessor

流水站	操作
IPG	分支预测,取指地址生成,L1 I-Cache 访问
ROT	指令包旋转
EXP	指令分派,功能部件端口映射
REN	寄存器重命名
REG	读取寄存器
EXE	ALU,L1D cache 访问
DET	异常检测
WRB	写回

Itanium 2 的流水线设计减少了谓词定义指令和判定指令之间的相关距离,有利于编译器产生紧凑的代码。但无效指令必须等到 REG 站才能识别,增加了能量的浪费。如果提前读取谓词寄存器,又会增加谓词定义指令和判定指令的相关距离,引起性能的下降。为此,需要一种折衷的方案。

2 尽早作废无效指令

假设采用类似于 Itanium 2 的流水线结构,主要关心流水线后端的 EXP、REN、REG、EXE 站。由于只有谓词值确定之后才知道判定指令是否无效,所以尽早作废无效指令的前提是尽早得到谓词的值。本节将讨论在不引起明显性能损失的条件下尽早作废无效指令的技术。

2.1 流水线修改

在 Itanium 2 处理器中,只有完成谓词寄存器重命名之后,才能读取谓词的值,所以谓词的值最早只能在 REG 站得到。为了提前得到谓词的值,需要提前进行谓词寄存器的重命名。

重新分配了流水线的功能,将谓词寄存器的重命名和读取提前一站,即在 EXP 站进行谓词寄存器的重命名,在 REN 站读取谓词寄存器。这样做的根据主要有两点:第一,Itanium 指令系统中谓词寄存器的编码字段位置是固定的,在分派时非常容易对谓词寄存器字段进行译码。第二,谓词寄存器重命名的操作十分简单,可以与指令主操作码的译码同时进行,因此不会影响时钟频率。

对流水线进行修改后,谓词定义指令需要提前两个周期才能确保判定指令在 REN 站能够读到谓词寄存器。现有的二进制程序是针对原来的流水线结构生成的,判定指令可能无法在 REN 站得到谓词寄存器的值,因此需要相应的数据相关检测逻辑,以保证程序的正确性。

2.2 数据相关检测

在 REN 站,如果谓词为假,则可以判定指令是无效的并将它作废。该指令与其他指令之间的数据相关也随即消除。如果谓词为真,则该指令要继续处理。但如果不能确定指令的谓词值真假,这时无法确定指令是否无效,也不能确定指令与其他指令之间的数据相关性是否真正存在。处理这种情况有两种方法,第一,让指令在 REN 站暂停,直到谓词值可用后再根据谓词的值决定应该作废还是继续执行,

称这种方法为 REN 停顿。第二,让指令进入 REG 站,在 REG 站再次读谓词。

假设允许指令进入 REG 站。如果谓词为假,则可以判定指令是无效的并将它作废。该指令与其他指令之间的数据相关也随即消除。如果谓词为真,则该指令要继续处理。但如果不能确定指令的谓词值真假,这时无法确定指令是否无效,也不能确定指令与其他指令之间的数据相关性是否真正存在。处理这种情况有两种方法,第一,让指令在 REG 站暂停,直到谓词值可用后再根据谓词的值决定应该作废还是继续执行,称这种方法为 REG 停顿。第二,让指令进入 EXE 站,在 EXE 站再次读谓词,这时谓词的值肯定是可用的,称这种方法为无停顿。

Itanium 2 处理器采用的方法与无停顿类似,只是在 REN 站不读谓词寄存器,而在 REG 站即使谓词为假也不作废指令。修改流水线后要求谓词寄存器提供更多的读端口,这样会增加谓词寄存器的复杂度。

比较 REN 停顿、REG 停顿和无停顿三种方法,显然 REN 停顿的功耗最小,停顿也最多;无停顿的性能最好,功耗也最大;REG 停顿方法介于两者之间。至于性能和功耗的差异究竟有多大,关键因素在于谓词最早什么时候可用,将在第 3 节通过模拟实验来定量分析。

2.3 无效指令的作废

作废无效指令的目的是降低部件的功耗,可以采用功能部件使能、门控时钟或门控电源的方法。由于它们都是成熟的低功耗设计技术[7],本文不再赘述。

特别需要注意的是,谓词为假的谓词定义指令不能简单地作为无效指令处理。因为对谓词定义指令而言,谓词不是指令状态的标志,而是一个真正的源操作数,谓词的值将会影响谓词定义指令的结果。所以,流水线要对这类指令进行专门的控制。

3 模拟实验与结果分析

3.1 模拟方法

使用扩展的 IMPACT 模拟器[8],它是一个周期精确的性能和功耗分析工具,图 1 给出了模拟器的结构。模拟器按照 Itanium 2 处理器的体系结构参数[6]配置。实验使用 7 个 SPEC 程序和 4 个 MediaBench 测试程序,它们能反映出高性能微处理器应用的主要特点。

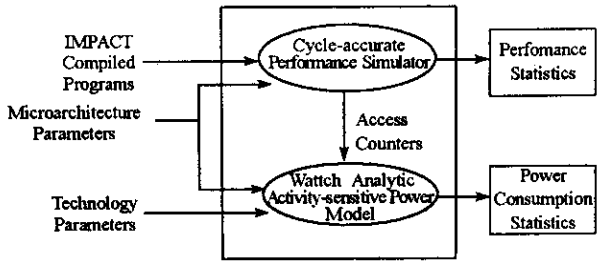

图 1 性能/功耗模拟器结构示意图

Fig. 1 Organization of power/performance simulator

3.2 性能与功耗评价

从性能的角度考虑,采用 REN 停顿,性能损失较为严重,而采用 REG 停顿,几乎没有性能损失,如图 2 所示。这是因为指令调度保证 EXE 站的谓词是肯定可用的;多数测试程序的谓词在 REG 站也是可用的,只有 129. compress 程序中有 1% 的判定指令在 REG 站无法得到可用的谓词;但是多数判定指令无法在 REN 站得到谓词,只能停顿。采用无停顿方法,性能无损失,因为指令没有因为流水线的改变而产生新的暂停,所以图中没有显示。

从能量的角度考虑,采用 REN 停顿和 REG 停顿都减少了能量消耗,如图 3 所示,前者能量浪费更少,主要原因是在 REN 流水站确定的无效指令不用读取寄存器,从而减少了能量。由于除了 129. compress程序的极少量指令外,所有测试程序的指令都能在 REG 站读到谓词的值,即只有极少数无效指令会在 EXE 站浪费能量,所以采用无停顿方法的能量消耗与采用 REG 停顿的相当,图中也没有给

出。

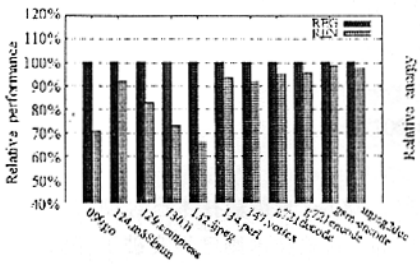

图2 相对性能

Fig. 2 Relative performance

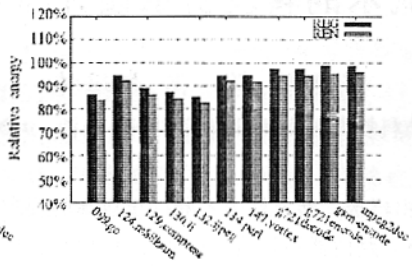

图3 相对能量

Fig. 3 Relative energy

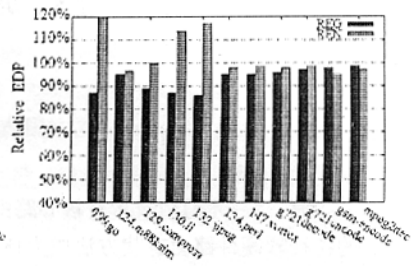

图4 相对能量延迟积

Fig. 4 Relative EDP

为了综合评价性能和功耗两个指标,使用能量和延迟的乘积(EDP)来衡量处理器的能量效率,EDP越小,则能量效率越高;EDP越大,则能量效率越低。如图4所示,采用REG停顿方法,能获得更高的能量效率,而采用REN停顿方法,由于性能损失严重,某些测试程序的能量效率比原流水线还要低。

4 结 论

本文认为无效指令造成了判定执行的能量浪费,提前作废无效指令是减少能量浪费的有效方法。基于Itanium 2微处理器的流水线,修改了流水线划分功能,提出谓词相关条件下的流水线停顿方法。模拟结果表明,采用REN停顿和REG停顿都能有效地减少能量消耗,但采用REN停顿,性能损失严重,而采用REG停顿,几乎没有性能损失,可以提高处理器的能量效率。

参 考 文 献:

[1] Mahlke S A, Hank R E, Bringmann R A, et al. Characterizing the Impact of Predicated Execution on Branch Prediction [A]. Proceedings of the 27th Annual International Symposium on Microarchitecture[C], San Jose, CA, 1994:217-227.

[2] McCormick J E. Supporting Predicated Execution: Techniques and Tradeoffs [D]. University of Illinois at Urbana-Champaign, 1996.

[3] Carter L. Compiler and Hardware Predicated Dependency Analysis and Scheduling[D]. University of California, San Diego, 2002.

[4] Darsch A, Seznec A. Out-of-order Predicated Execution with Translation Register Buffer [R]. Technical Report PI-1573, IRISA, 2003.

[5] Huck J. Introducing the IA-64 Architecture[J]. IEEE Micro, 2000, 20(5): 12-23.

[6] McNairy C, Soltis D. Itanium 2 Processor Microarchitecture[J]. IEEE Micro, 2003, 23(3).

[7] 张民选,王永文. 一种消除无效操作降低ASIC功耗的设计技术[J]. 计算机工程与科学, 2002, 24(6).

[8] Wang Y, Zhang M. Impact XP: An Integrated Performance/Power Analysis Framework for Compiler and Architecture Research [J]. Chinese Journal of Electronics, 2004, 13(2): 250-253.