基于负载瞬时IPC性能的同时多线程处理器取指策略

何立强[1),2)] 刘志勇[2)]

[1)](内蒙古大学计算机学院 呼和浩特 010021)

[2)](中国科学院计算技术研究所 北京 100080)

摘 要 同时多线程处理器在每时钟周期从多个线程读取指令执行,极大地提高了指令吞吐率.文中简单介绍了SMT技术,讨论了常用的取指策略,比较了各策略在提高性能方面的优劣.给出特定负载下理论上的最优取指策略,在此基础上提出一种基于负载瞬时IPC性能的动态取指策略IPCBFP.实验表明,该策略可以有效地提高负载的性能,平均加速比对于两线程负载可以达到17%,对于四线程负载可以达到8%.该策略还具有平均占用指令队列项少,指令队列冲突率低的特点,而且,对降低SMT的Cache失效率和TLB失效率方面也有一定的作用.

关键词 同时多线程处理器;取指策略;指令队列;IPC;指令吞吐率

中图法分类号 TP302

An Instantaneous IPC Based Instruction Fetch Policy for SMT Processors

HE Li-Qiang[1),2)] LIU Zhi-Yong[2)]

[1)](*College of Computer Science, Inner Mongolia University, Huhhot* 010021)

[2)](*Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100080)

Abstract Simultaneous Multithreaded Processors improve the instruction throughput by allowing fetching and executing instructions from several running threads simultaneously in each clock cycle. In this paper, first, the authors introduce simply several instruction fetch policies of SMT processors, and compare their performance on improving IPC of single running workload. Next, an ideal fetch model of instruction fetch policy is given, and a realistic policy named IPCBFP, based on the ideal model, is proposed and analyzed. This policy fetches instructions for a thread according to its instantaneous IPC value and its current instruction number in the instruction queue. Simulation results show that IPCBFP policy can improve the performances of the workloads dramatically. In two-thread and four-thread mix workload experiments, the speedups are 17% and 8% on average respectively. In addition, the sizes and the conflict rates of IQ on average in the experiments with the policy are small than that with ICOUNT.2.8 policy, which is the best fetch policy up to date to our knowledge. And the authors' policy also has some advantages on the degradation of cache miss rates and TLB miss rates.

Keywords simultaneous multithreaded processor; fetch policy; instruction queue; IPC; instruction throughput

收稿日期:2004-05-18;修改稿收到日期:2006-11-08.本课题得到国家自然科学基金(60325205,60303033,60373043)资助.**何立强**,男,1973年生,博士,讲师,主要研究方向为计算机体系结构.E-mail:hlq@ict.ac.cn;helq@imu.edu.cn.**刘志勇**,男,1946年生,博士,研究员,博士生导师,主要研究领域为高性能计算机算法、体系结构、并行处理、互联网络.

1 引 言

同时多线程处理器(Simultaneous Multithreaded Processor,SMT)[1-2]通过在每个时钟周期内从多个同时运行的线程读取指令执行而极大地提高了处理器的指令吞吐率.然而,SMT 在提高指令吞吐率的同时也增加了线程间对共享硬件资源的竞争.合理分配硬件资源,在提高多线程总体性能的同时尽量不影响单个程序负载的性能,其关键的因素在于取指策略.另外,由于允许在每个时钟周期从多个线程读取指令执行,取指部件和访存部件成为制约 SMT 性能的瓶颈.合理取指,减少不必要的提前或无效取指操作,提高取指效率和取指质量,其关键的因素也是取指策略.

本文首先简单介绍 SMT 的体系结构,并回顾几个常用的取指策略,在此基础上,给出一个理想的取指策略模型,该模型描述了取指策略所能达到的性能上界;同时,根据该模型的思想提出了一个有效的动态取指策略——IPCBFP[9],并分析了它的特点以及具体的实现途径.模拟实验的结果表明,当取指带宽为 8 条指令时,本文所提出的动态取指策略在绝大多数情况下都比目前 ICOUNT.2.8 策略的性能好,其最坏情况也能获得与其相同的性能指标.对于两线程同时运行的程序负载,其平均加速比可以达到 17%;对于四线程的负载可以达到 8%.而且,该策略具有公平性,在提高程序负载集合整体性能的同时均匀提高单个参与运行线程的性能.同时,该策略还具有平均占用指令队列项少,指令队列冲突率低的特点;另外,该策略对于降低同时多线程处理器的 Cache 失效率和 TLB 失效率方面也有一定的作用.

本文第 2 节简单介绍 SMT;第 3 节回顾几个常用的取指策略;第 4 节描述取指策略的理想模型和一个新的动态取指策略 IPCBFP;第 5 节给出模拟实验的结果;最后,在第 6 节中给出结论.

2 同时多线程技术

SMT 的系统结构如图 1 所示.为了同时运行多个线程,处理器需要为每个线程设置一套独立的用于保存其运行状态的硬件机构.在每个时钟周期,取指部件根据分支预测器的结果从指令 Cache 中读取多个线程的指令送到取指队列,取指队列中的指令按"先进先出"的次序每周期将数量等于译码带宽的指令送到译码部件进行译码,指令经译码和重命名后进入到定点或浮点发射队列等待发射,当条件满足(操作数准备好且部件空闲)时被发射到相应的功能部件乱序执行,执行结束后再经过指令重排序部件依次写回到寄存器中.

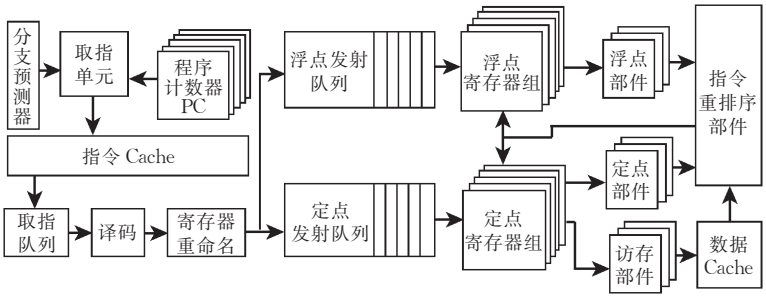

图 1 同时多线程处理器系统结构

3 常用的取指策略

最简单的取指策略为"随机法",即取指部件随机地从备选线程取指令执行.另外一种简单的取指策略为"轮询法",即取指部件根据取指带宽轮流从全部或部分活动线程中读取固定或可变数目的指令到取指队列中供执行.两种策略的性能接近,轮询法不考虑线程的推进速度和优先级等因素,有可能出现某些线程的指令因延迟执行而长期占据指令队列项导致其它线程无法继续取指和推进的现象,从而影响了系统整体的指令吞吐率.

在已有的取指策略中,ICOUNT[2]策略的取指效率和取指质量最高,依据该策略进行取指的程序性能也最好,文本所提出的取指策略将以该策略作为参照进行性能上的比较.在 ICOUNT 取指策略中,取指部件优先选择占据指令队列项数最少的几个线程进行取指,因此,若某线程运行速度快,则其指令在队列中的延迟时间短,可以被很快发射到功能部件执行,表现在指令队列上即占据较少的队列

项，从而在下次取指时可以获得较高的优先权；而运行速度慢的线程，由于其指令延迟执行而长期占据队列项，形成指令堆积，因而在下次取指时将具有较低的优先权. 在此，ICOUNT 策略倾向于使运行速度快的线程优先运行，同时又能保证一定的取指公平性. ICOUNT 策略有多种取指参数组合，其中，ICOUNT. 2. 8 的性能最优，其具体策略为：每次选两个线程取指，共取 8 条指令；首先优先为队列项最少的线程取指，当其发生指令 Cache 失效或下一条指令需要跨越指令 Cache 行时即停止取指，然后，将剩余的取指带宽分配给第二个选中的线程. 在此，需要强调的是，指令队列是指包含同时多线程处理器内取指队列、译码队列、重命名队列和定点/浮点发射队列在内全部队列的统称，其项数则是这些队列项数的总和，而线程所占指令队列的项数则是该线程在这些队列中所占项数的总和，在以下的叙述中与该说明相同.

在 ICOUNT 基础上，结合线程优先级的因素有 PICOUNT 和 PICOUNT2 取指策略. 前者在取指时计算备选线程的优先级和占用指令队列项数两个参数的加权和，其值最小的几个线程获得取指的优先权；后者则在计算加权和时偏重于线程优先级所占的权值. 在此，考虑优先级的取指策略可以保证高优先级线程的取指和执行，但却可能导致低优先级线程长期得不到运行或性能发生下降，类似的情况也出现在文献[4]中的取指策略中.

针对不同的性能指标，也有一些其它的取指策略. 如文献[5]提出的贪心算法在取指时可以选择具有最小 Cache 或 TLB 失效率的线程进行取指；或选择具有最小平均访存时间的线程进行取指；也可以选择具有最高 IPC(Instructions Per Cycle)值的线程进行取指；另外，也可以根据各线程对硬件资源的使用互补情况进行取指选择，如：侧重于使各线程总的数据 TLB 失效率最小，或侧重于使总的平均访存时间最小，或侧重于使总的 IPC 值最大等. 文献[3]则从提高指令发射队列使用效率，尽量减少指令在队列中等待时间的角度提出了几种门控取指策略，如：当线程的延迟执行指令数达到某个限额时即停止从该线程取指，或在线程实际或预测的数据 Cache 失效率达到某个限额时即停止取指等.

目前，国内对 SMT 的研究成果为数不多[11]，对其取指策略的研究更是不多见.

4　IPCBFP 取指策略

本节首先给出一个取指策略的理想模型，该模型说明了最优取指策略的行为特征及所能达到的性能上界，接着，在此基础上提出一个有效的取指策略——IPCBFP，描述了该策略的基本思想及具体的实现方法.

4.1　取指策略的理想模型

在 SMT 中，指令的取指和发射通常包含两个步骤：(1)取指部件从多个选中的线程读取多条指令送入取指队列；(2)取指队列中的指令经译码和重命名后送到发射队列，随后，指令以不同的速度被发射到功能部件中执行，其发射的速度由指令的类型和指令间的相关情况及可用的硬件资源数具体决定. 在这两个步骤中，取指策略所涉及的是第一个步骤.

在取指策略决定线程的选择和所选线程的取指数量时，其关键的因素是要保证线程在指令队列中有足够的指令可供发射到功能部件中执行. 若线程的指令不足，则会因为指令缺乏导致程序执行速度的减慢，反之，若指令过多，则多余的指令会长期占据指令队列项，从而在队列项资源紧张时导致其它线程的指令不能进入到指令队列，进而影响这些程序的执行. 因此，最优的取指策略应能恰当地为所有运行的线程取指令，既保证线程有足够的指令执行，同时又不为线程过多地取指令，从而有效地利用有限的指令队列项资源，最大限度地提高程序的性能，即达到程序的性能不受取指策略制约的目的.

上述最优取指策略是同时多线程处理器取指策略的一个理想模型，该模型的最优化目标为指令吞吐率，在实际系统中无法实现，因为，取指策略在程序运行前无法了解在某次取指时程序未来的执行速度，也就无法根据其指令发射和执行的速度补充足够的指令，但是，该模型描述了最优取指策略所能达到的指令吞吐率性能上界，即：在确定的硬件配置条件下，程序在有足够指令可供执行时所能达到的最快执行速度，该速度仅由程序自身的特征和系统的硬件条件决定. 也就是说，该最优化模型表示除了程序自身的特点和硬件配置这些外部条件外取指策略本身所能达到的最优的指令吞吐率性能.

4.2　IPCBFP 策略

在上述取指策略的理想模型中，程序的执行速度可以用其瞬时 IPC 值加以表示. 在实际系统中，

可以将该值作为取指策略进行下次取指时的参考依据.根据理想模型中的取指思想,我们提出一种新的同时多线程处理器取指策略,称为 IPCBFP(IPC Based Fetch Policy),即基于线程瞬时 IPC 值的动态取指策略.

IPCBFP 策略包含两个过程.首先,线程的选择,即取指部件在每个时钟周期进行取指时,选择几个线程和哪几个线程进行取指.在此,我们将文献[2]中的 SMT 作为我们的研究对象,该结构有 8 套用于保存线程运行状态的硬件,允许最多 8 个线程同时运行,取指带宽为每周期 8 条指令.在这样的硬件条件下,IPCBFP 策略在取指时选择 2 个线程进行取指,总共取 8 条指令,所选的 2 个线程在所有同时运行的线程中占据的指令队列项最少.IPCBFP 策略与 ICOUNT.2.8 策略在线程选择方式上完全相同,这主要是基于一个共同的目的,即:避免因从多个线程取指而把取指带宽划分得过细,从而使某些线程因 Cache 失效等原因无法进行取指的现象发生.

IPCBFP 策略的第二个过程为取指过程.在该过程中,取指部件根据所选线程的瞬时 IPC 值和该线程在指令队列中已有的指令数来决定在本周期中为其读取的新的指令数.若其在指令队列中已有足够的指令可供执行,则本周期中将不再为其继续取指,否则,取指部件根据其需要的指令数进行取指,但至多只能取全部的指令带宽所容纳的指令数.在这个过程中,计算一个线程所需要的指令数可以用式(1)来表示.

$$I = IPC - I' \tag{1}$$

在式(1)中,I 为线程在某次取指时所需的指令数,IPC 为该线程的瞬时 IPC 值,用其表示线程执行的速度,I'为线程在指令队列中已有的指令数.若 $I \leq 0$,则表示该线程在指令队列中有足够的指令可供执行,本周期中不需要对其进行取指操作.

式(1)是一种较理想的取指情况,它具有理论上的指导意义,描述了线程运行时的指令供求关系.但是,在一个实际系统中,由于 Cache 失效和分支误预测以及指令间因相关而导致的指令等待等原因的存在,使得实际取得的线程指令数应大于由式(1)计算所得的指令数,这样才能保证超标量执行部件有足够的指令执行,进而最大限度地提高线程的指令吞吐率.在此,综合上述因素后,一个合理的操作是给瞬时 IPC 值乘以一个系数,以得到近似准确的可保证线程执行速度的指令数,这样,式(1)改写为

$$I = IPC \times P - I' \tag{2}$$

式(2)中,系数 P 的选取直接影响到 IPCBFP 取指策略的性能,为此,我们进行了一组实验,实验中的取指带宽为 8 条指令,根据实验结果选择了一个性能最优的 P 值,为 16,具体的实验结果将在第 5 节介绍.根据 P 值的设定,式(2)可进一步改写为式(3).在此,需要说明的是,系数 P 的值是一个经验值,它的选取可通过启发性过程或模拟实验获得,且是与具体的硬件配置、系统所支持的最高线程数以及指令队列的长度等因素紧密相关的,并不是一个通用的常数,因此,需要针对不同的硬件环境进行相应的修改.

$$I = IPC \times 16 - I' \tag{3}$$

利用式(3)计算线程所需的指令数时,若瞬时 IPC 值等于 0,则不进行取指.但在系统初启时或线程发生分支误预测而导致指令取消时,其瞬时 IPC 值往往为 0,此时,线程对取指部件表现出来的执行速度为 0,在这种情况下如果不进行取指,则会影响到线程的继续执行,这在系统初启时表现得尤为突出,因此,我们将 IPC 值加 1 后改写为式(4)以避免出现这种情况.

$$I = (IPC + 1) \times 16 - I' \tag{4}$$

在我们研究的体系结构中,由于取指带宽限制为每周期 8 条指令,因此,一个线程每次能够取得的指令数最多也为 8 条,式(4)应改写为式(5).

$$I = \min((IPC + 1) \times 16 - I', 8) \tag{5}$$

至此,IPCBFP 策略可总结为:取指部件在每次取指时选择同时运行线程中占据指令队列项最少的两个线程进行取指,并根据式(5)计算占用队列数最少的线程所需的指令数,根据计算的结果为该线程取指令,至多取 8 条指令,剩余的取指带宽为所选的第二个线程进行取指.在为每个线程取指时,当线程发生指令 Cache 失效或所需读取的指令需要跨越 Cache 行的边界时即停止取指.

IPCBFP 策略与目前性能最优的 ICOUNT.2.8 策略的不同之处在于两者为所选线程所取的指令数量上.ICOUNT 策略尽量为所选的第一个线程取指,直到停止取指的条件发生或达到最大的取指带宽时为止,在取指带宽有剩余时才为第二个线程取剩余数量的指令;而 IPCBFP 策略则在与 ICOUNT 相同的条件下为每个线程计算了一个取指数量的上界,而不是像 ICOUNT 策略那样将其设置为取指带宽,从而更加均衡地利用了取指带宽,避免出现第一个线程被取了过多的指令而第二个线程没有足够的

指令被取到的现象发生，从而使线程的总体性能在 ICOUNT 上得到进一步的提高. 本文在附录 A 中给出了这两种策略在取指带宽为 4 条指令时为线程所设置的取指上限的实例.

在具体实现 IPCBFP 策略时，为计算线程的瞬时 *IPC* 值需要付出较多的硬件代价. 一种较好的折衷方案是将线程在指令队列中的指令数取逻辑“反”并对一个常数(如 16)取模操作后得到的新值作为式(5)中 min 操作的左半部分，从而极大地简化了硬件的实现，使得 IPCBFP 策略可以在常量级的时间复杂度内完成，该思路可用式(6)表示，其具体的硬件实现见文献[10].

$$I=\min(\overline{I'} \bmod 16,8) \tag{6}$$

5　实验结果

我们采用 SMTSIM[6] 进行实验. 该模拟器为执行驱动的模拟器，可以执行由 Alpha 指令集构成的应用程序，并模拟 SMT 各功能部件的使用情况，给出程序运行后的各种统计数据，该模拟器的体系结构参数如表 1 所示. 我们在该模拟器的基础上实现了 IPCBFP 取指策略，通过在 IPCBFP 策略和 ICOUNT. 2. 8 策略下运行相同的测试程序来进行实验，并通过比较实验结果得到 IPCBFP 策略的性能优势. 在此，我们将 SMT 在运行负载程序后得到的总体指令吞吐率作为衡量性能的评价标准.

表 1　同时多线程模拟器体系结构参数列表

项 目	值
功能单元	浮点：3，定点：6（包含访存部件：4）
流水线	9 段
指令队列	取指队列 64 项，译码和重命名队列 8 项，浮点/定点队列 32 项
功能部件延迟	与 Alpha 21264 相同
I-Cache	64KB，2 路，64Byte/line
D-Cache	64KB，2 路，64Byte/line
L2-Cache	512KB，2 路，64Byte/line，片上
L3-Cache	4MB，2 路，64Byte/line，片外
ITLB	48 项，失效开销为 160 时钟周期
DTLB	148 项，失效开销为 160 时钟周期
到 CPU 的延迟	L2-Cache：6，L3-Cache：18，Mem：80
分支预测器	2KB Gshare，误预测开销：7

实验中所用的程序负载为 SPEC CPU 2000[7] 测试集. 我们选择了 6 个定点和 6 个浮点测试程序，为了使程序负载具有更大的普遍性，在生成负载集时不是将单线程简单地进行复制，而是将不同线程随机组合为 6 个两线程的负载集，3 个四线程及 1 个八线程的负载集. 这些程序用 GCC 进行编译，编译时选择-O4 优化选项，并静态链接为可执行的应用程序. 在实验中，各 SPEC 程序采用 Ref 输入集，其中，bzip2 采用 train 输入集；各程序负载的运行指令数根据文献[8]中的方法进行设定，其具体值如表 2 所示.

表 2　各程序负载所运行的指令数

程序负载	指令数(亿)
equake，mesa	30
crafty，mcf	15
art，perlbmk	3
bzip2，lucas	53
mgrid，ammp	40
parser，twolf	53
art，perlbmk，crafty，mcf	15
equake，mesa，mgrid，ammp	40
bzip2，lucas，parser，twolf	53
art，perlbmk，crafty，mcf，equake，mesa，mgrid，ammp	40

图 2 给出了式(2)中不同 P 值对部分程序负载性能影响的实验结果. 从图中可以看到，当 P 值由 8 逐渐递增到 16 时，所有的程序负载的性能都不断提高；当 P 值在 16～24 之间变化时，各程序负载的性能几乎不变；而当 P 由 24 递增到 32 时，多数程序负载的性能都趋于下降. 因此，从实际的性能和取指策略的易实现方面考虑，P 值选 16 较为合适，正如式(3)所采用的.

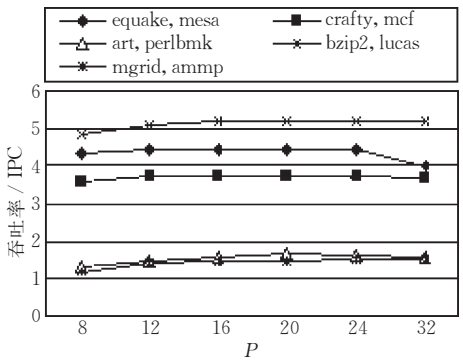

图 2　程序负载的性能随参数 P 值变化的情况

图 3 给出了两程序负载运行时，IPCBFP 策略和 ICOUNT. 2. 8 策略的指令吞吐率性能. 在 6 个程序负载中，mgrid-ammp 的性能略有下降，其它负载的性能都有较大的提高，其中，parser-twolf 的性能提高幅度最大，达到了 ICOUNT. 2. 8 策略下程序负载性能的 12 倍，除此特例外，各程序负载平均的性能加速比为 17. 192%.

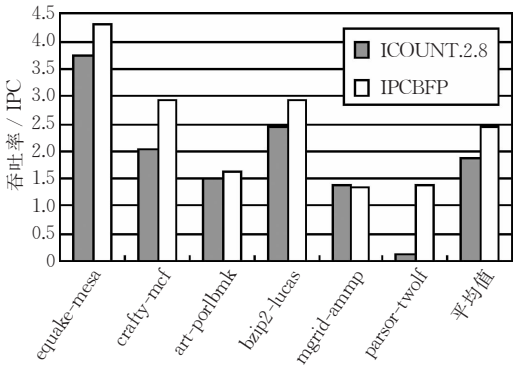

图 3 两线程负载运行时 IPCBFP 和 ICOUNT. 2. 8 的指令吞吐率性能

IPCBFP 策略在提高程序负载总体性能的同时均衡提高构成负载的各线程的性能，即：该策略具有公平性. 图 4 给出了两线程负载运行时各线程独立的 IPC 性能值，从图中可以看到，除 art 和 bzip2 的性能下降外，其它 10 个线程的性能都有较大的提高，其中，parser，equake，crafty，perlbmk 和 mgrid 的性能提高最为明显.

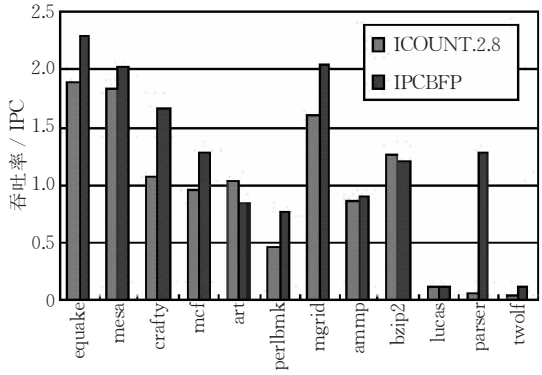

图 4 两线程负载运行时各线程独立的 IPC 性能

造成这种系统总体性能和单线程性能同时提高的原因是由于 IPCBFP 策略更加均衡地在线程间分配取指带宽，它在绝大多数情况下与 ICOUNT 策略相比较是减少而不是增加了占据指令队列项最少的线程(也是最优先取指的线程)所取的指令条数，从而在不影响其执行速度的情况下相应地减少了该线程的分支误预测开销和未准备好执行的指令所占的指令队列项开销，这样，就能够将剩余的取指带宽分配给其它占据队列项次少的线程，增加其有用指令的条数，进而提高处理器的总体性能和单线程独立的性能.

对于 parser-twolf 在性能上的大幅度提高，其原因是由于在 ICOUNT. 2. 8 策略下为 parser 取到过多的指令，从而导致与 twolf 产生大量的 DTLB 访问冲突和定点指令队列冲突，其中 D-TLB 冲突导致了大量的 D-TLB 访问失效，而定点指令队列冲突率也高达 97. 37%，同时使系统的平均指令队列长度也高达 63. 39 项；这种 D-TLB 访问的冲突和队列的冲突大大降低了 parser 线程的性能，使其 IPC 值由单线程运行时的 1. 758 下降到 0. 073，而 twolf 的性能与单线程运行时相比则由于线程共享导致每周期的取指数量减少而从 0. 116 上升到 0. 037，同时，由于 twolf 本身存在的大量指令相关使其 IPC 值很低，因而 D-TLB 访问的冲突和指令队列冲突的影响已经被这种相关所掩盖了；在两种因素的作用下，该负载程序的总体 IPC 性能仅为 0. 11；而在 IPCBFP 策略下，通过调整取指带宽的分配，减少了对 parser 的取指数量，使其 D-TLB 的冲突率减小，失效率下降到原来的 1/10，同时使队列冲突率降为 0，平均指令队列长度也减少为 27. 69 项，从而极大地提高了其 IPC 性能，达到了 1. 276，接近其在单线程运行时的性能，而 towlf 程序则由于取指数量的进一步增加而提高了分支误预测的开销，导致其性能反而下降到 0. 126，最后，系统总的 IPC 为 1. 402，比 ICOUNT 策略提高了约 12 倍.

除负载的 IPC 性能提高外，IPCBFP 策略对减少线程平均占用指令队列项及降低指令队列冲突率方面也有所帮助，这是由于 IPC 性能的提高意味着指令在进入队列后可以被更快地发射到功能部件中执行，从而更快地释放掉其占用的指令队列项，因而平均指令队列长度和队列冲突率都得到减少. 在此，平均占用指令队列项的项数为每时钟周期线程占据指令队列项数的算术平均值. 在两线程负载的实验中，有 4 个负载集(crafty-mcf，art-perlbmk，bzip2-lucas，parser-twolf)的平均占用指令队列项的项数比在 ICOUNT. 2. 8 策略下的项数减少了一半，其余两个负载集的项数则与 ICOUNT. 2. 8 策略下的项数接近相同，如图 5 所示.

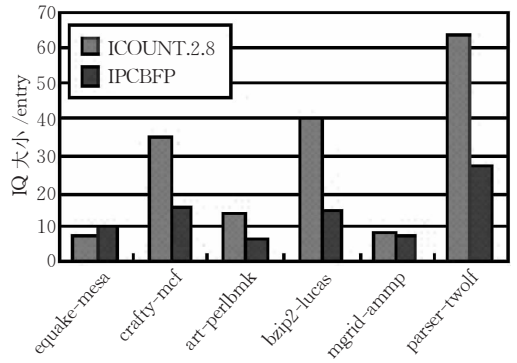

图 5 两线程实验中 IPCBFP 和 ICOUNT. 2. 8 在平均占用指令队列项方面的比较

在降低队列冲突率方面，IPCBFP 策略在两线程负载实验中的冲突率全部为 0，而 ICOUNT.2.8 策略则有两个负载(crafty-mcf，parser-twolf)的冲突率较高，分别达到了 20.3% 和 97.4%，其余与 IPCBFP 策略相同，在此，队列冲突率是指负载所占整数队列冲突率、访存队列冲突率和浮点队列冲突率的算术平均值.

由于在 D-TLB 上的性能提高，并进而改善了其在 Cache、指令队列等的性能，最终使其指令吞吐率提高了近 12 倍.

除此之外，IPCBFP 策略对于降低程序负载的 Cache 失效率和 TLB 失效率方面也有一定的作用，如图 6. 在两线程负载实验中，指令 Cache、数据 Cache、二级 Cache 失效率的平均降幅分别为 34.4%、33.2% 和 16.7%. 在此，Cache 失效率的改进主要来自于线程访存次数的减少，由于 IPCBFP 策略通过调整取指带宽的分配降低了线程分支误预测的开销，使得那些在误预测路径上的访存指令被减少了，从而在总体上减少了处理器的访存次数，提高了 Cache 的命中率. 对于 D-TLB 失效率，IPCBFP 策略在两线程实验中对所有的负载都有所改善，其中，crafty-mcf 和 parser-twolf 的改善最大，分别为 ICOUNT 策略下的 50% 和 1/11，其余的改进幅度则相对较小. 在此看出，IPCBFP 策略对于 parser-twolf 的性能改进主要来自图 8.

图 7 给出了四线程和八线程负载运行时 IPCBFP 和 ICOUNT.2.8 的指令吞吐率性能，其平均加速比的结果不如两线程负载运行时的结果，为 8%. 图 8 给出了在四线程负载运行时构成各负载的线程的独立 IPC 性能值，与图 5 的结论相同，除 art 的性能有所降低外，其它线程的性能均得到了提高. 在此，程序负载指令吞吐率性能提高的原因与两线程负载时

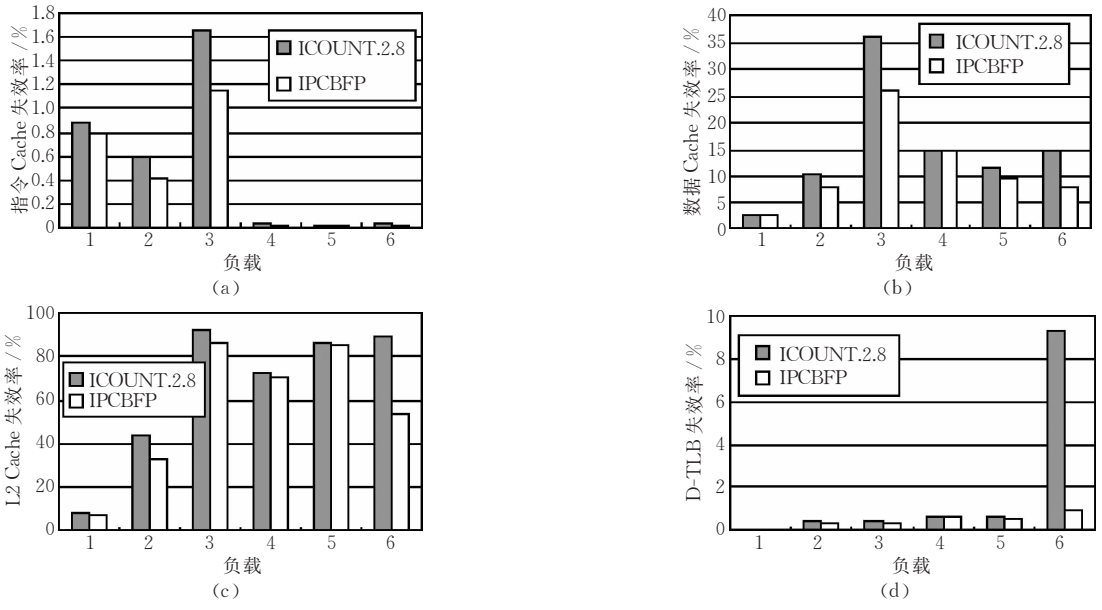

图 6　两线程实验中 Cache 和 D-TLB 命中率的比较

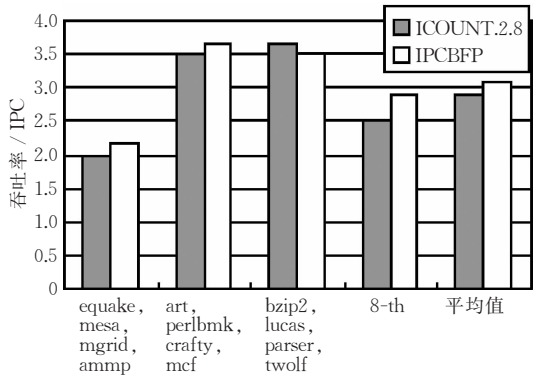

图 7　四线程负载运行时 IPCBFP 和 ICOUNT.2.8 的指令吞吐率性能

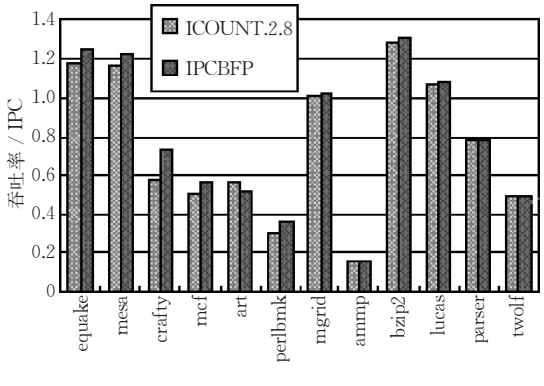

图 8　四线程负载运行时各线程独立的 IPC 性能

完全一样,但提高幅度较小的原因是由于当同时运行线程数增加时取指带宽已经在各线程间进行了分割,从而对单线程而言不会出现像两线程负载时所取指令数相差很大的情况,即取指带宽的分配已接近于均衡,因此给 IPCBFP 策略所留的改进余地不大,性能提高的幅度也相对较小.

6 结 论

本文在简单介绍了 SMT 和现有的取指策略后,给出了一个 SMT 取指策略的理想模型,该模型描述了取指策略所能达到的程序性能的上界,其后,在理想模型的基础上提出了一种基于线程瞬时 IPC 的取指策略 IPCBFP,论述了其基本原理和具体的实现技术.

IPCBFP 策略的性能比目前最好的 ICOUNT.2.8 取指策略的性能有较大的提高,对于两线程同时运行的程序负载,其平均性能加速比为 17%,对四线程程序负载的平均性能加速比为 8%;同时,该策略对于减少程序负载的平均占用指令队列项和降低指令队列冲突率方面也有所帮助,而且,对于降低程序运行时的 Cache 失效率和 TLB 失效率方面也有一定的作用.

IPCBFP 策略的实现非常简单,其硬件逻辑仅包含逻辑"反"和"选择"操作,可在常量级的时间复杂度内完成,其实现复杂度与 ICOUNT.2.8 策略相同.

致 谢 感谢中国科学院计算技术研究所胡伟武研究员对本文工作的悉心指导,特别感谢匿名评审的老师对本文原稿所提出的中肯而高质量的评审意见!

参 考 文 献

[1] Tullsen D, Eggers S et al. Simultaneous multithreading: Maximizing on-chip parallelism//Proceedings of the 22nd ISCA. Italy, 1995: 392-403

[2] Tullsen D. Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor//Proceedings of the 23rd ISCA. Philadelphia, 1996: 191-202

[3] Ali EI-Moursy, David H A. Front-end policies for improved issue efficiency in SMT processors//Proceedings of the 9th HPCA. Anaheim, 2003: 31-40

[4] Raasch S, Reinhardt S. Applications of thread prioritization in SMT processors//Proceedings of the Multithreaded Execution and Execution Workshop (MTEAC). Orlando, 1999

[5] Parekh S, Eggers S et al. Thread-sensitive scheduling for SMT processors. Department of Computer Science, University of Washington: Technical Report, 2000

[6] Tullsen D. Simulation and modeling of a simultaneous multithreading processor//Proceedings of the 22nd Annul Computer Measurement Group Conference. San Diego, CA, USA, 1996: 819-828

[7] Henning J L. SPECCPU 2000: Measuring CPU performance in the new millennium. IEEE Computer, 2000, 33(7): 28-35

[8] Sherwood T, Perelman E, Hamerly G, Calder B. Automatically characterizing large scale program behavior//Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems. San Jose, California, 2002: 45-57

[9] He L, Liu Z. An effective instruction fetch policy for simultaneous multithreaded processors//Proceedings of the 7th HPCAsia. Tokyo, 2004: 162-168

[10] He Li-Qiang, Liu Zhi-Yong, Hu Wei-Wu. An instruction fetching controller for SMT processor. China Patent No. 200410009288.5(in Chinese)
(何立强,刘志勇,胡伟武.一种应用于同时多线程处理器的取指控制装置及其方法.中国专利,专利受理号:200410009288.5)

[11] Zhao Rong-Cai, Tang Zhi-Min, Zhang Zhao-Qing, Gao Guang R. Research on compiler indicated low power techniques for multithread programs, Journal of Computer Research and Development, 2002, 39(12): 1572-1579(in Chinese)
(赵荣彩,唐志敏,张兆庆, Gao Guang R.编译指导的多线程低功耗技术研究.计算机研究与发展, 2002, 39(12): 1572-1579)

附录 A. IPCBFP 与 ICOUNT 取指策略在每时钟周期内取指令条数的区别.

在此,以一个实例来说明 IPCBFP 取指策略的实施过程.为简单起见,假设同时多线程处理器同时运行 2 个线程,保留站队列共 32 项,用于存放经重命名后的定点和浮点指令,取指、译码、重命名队列(或窗口)分别为 4 项,处理器为 4 发射,每拍最多取 4 条指令.在上面的假设下,队列项总数为 $32+4\times3=44$.需要说明的是,当处理器为 4 发射时,式(6)中"取模"的数为 8.

对当前两线程拥有所有指令条数分布的情况列表如下.为减少篇幅,仅考虑线程 1($T1$)的指令数少于线程 2($T2$)的情况,反之相同.在附表中,4′表示其取指上限为 4 条指令,但必须在线程 1 取指后的剩余取指带宽中进行取指.由附表可以看出,与 ICOUNT 策略相比,IPCBFP 策略实际上是减少了优先级最高的线程($T1$)的取指令条数,从而可以将剩余的取指带宽分配给其它的线程($T2$),增加了其有用的指令数,进而提高了系统的整体指令吞吐率.

附表　IPCBFP 与 ICOUNT. 2. 8 在每时钟周期内取指令条数的区别

当前线程指令数		IPCBFP 的取指上限		ICOUNT 的取指上限		当前线程指令数		IPCBFP 的取指上限		ICOUNT 的取指上限	
$T1$	$T2$	$T1$	$T2$	$T1$	$T2$	$T1$	$T2$	$T1$	$T2$	$T1$	$T2$
1	$1<T2\leq43$	4	4′	4	4′	12	$12<T2\leq32$	3	4′	4	4′
2	$2<T2\leq42$	4	4′	4	4′	13	$13<T2\leq31$	2	4′	4	4′
3	$3<T2\leq41$	4	4′	4	4′	14	$14<T2\leq30$	1	4′	4	4′
4	$4<T2\leq40$	3	4′	4	4′	15	$15<T2\leq29$	0	4′	4	4′
5	$5<T2\leq39$	2	4′	4	4′	16	$16<T2\leq28$	4	4′	4	4′
6	$6<T2\leq38$	1	4′	4	4′	17	$17<T2\leq27$	4	4′	4	4′
7	$7<T2\leq37$	0	4′	4	4′	18	$18<T2\leq26$	4	4′	4	4′
8	$8<T2\leq36$	4	4′	4	4′	19	$19<T2\leq25$	4	4′	4	4′
9	$9<T2\leq35$	4	4′	4	4′	20	$20<T2\leq24$	3	4′	4	4′
10	$10<T2\leq34$	4	4′	4	4′	21	$21<T2\leq23$	2	4′	4	4′
11	$11<T2\leq33$	4	4′	4	4′	22	$22<T2\leq22$	1	4′	4	4′

HE Li-Qiang, born in 1973, Ph. D., lecturer. His current research interest is computer architecture.

LIU Zhi-Yong born in 1946, Ph. D., professor. His mail research interests include computer architecture, high performance computer algorithm and parallel computing.

Background

As a multi-threaded processor architecture, SMT has been realized for several years in the industry. While for the performance improving, there are still many issues needed to be researched in the academic literature. Among of them, the front-end, especially the fetch policy, is an important one. Till now, there are not enough researches that addresses on the issues of SMT in our country. This research is supported by the Chinese Natural Science Foundation as a part of the Godson project. The object is to find a useful and effective fetch policy for the SMT front-end which maybe used in the next generation of Godson processor. Through this research, the authors′ propose a new fetch policy based on the instantaneous IPC of the running thread. The simulation experiment shows that this policy can improve the average instruction throughput up to 17%.