基于同时多线程的IFSBSMT取指策略研究

李静梅 关海洋
(哈尔滨工程大学计算机科学与技术学院 哈尔滨 150001)

摘 要 取指策略直接影响处理器的指令吞吐率。针对传统处理器取指策略存在取指带宽利用不均衡、指令队列冲突率高的缺点,提出基于同时多线程处理器的取指策略IFSBSMT。该策略以线程的IPC值为基础,选取优先级高的线程进行取指,并利用预取指令条数预算的方式分配取指带宽,采取线程IPC值和L2 Cache缺失率的双优先级动态资源分配机制分配处理器的系统资源。研究结果表明,IFSBSMT策略有效地解决了取指带宽、指令队列冲突及资源浪费问题,进一步提高了指令吞吐率,且具有较好的取指公平性。

关键词 同时多线程,取指策略,IFSBSMT,取指带宽,指令队列冲突,双优先级动态资源分配
中图法分类号 TP302 **文献标识码** A

Fetch Policy with IFSBSMT Based on Simultaneous Multithreaded Processors

LI Jing-mei GUAN Hai-yang
(School of Computer Science and Technology, Harbin Engineering University, Harbin 150001, China)

Abstract Fetch policy influences instuction throughput rate of computer processor directly. In the view of the shortages of unbalaned utilization of fetch bandwidth and high conflict rate of instruction queue, a new fetch policy named instruction flow speed based on simultaneous multithreading(IFSBSMT for short) was proposed. This strategy fetch takes the value of instrunctions per clock(IPC) to select the high priority thread and the method of prefetch instructions number to allocate fetch width. Meanwhile, IPC value for thread and L2 Cache miss rate are taken as the dual priority dynamic resource allocation mechanism to allocate the system resource for processors. The result shows IFSBMT can effectively improve the instruction fetch bandwidth and overcome the problems of instruction queue and resources waste, which can improve instruction throughput and abtain a better fetch fairness.

Keywords Simultaneous multithreading, Fetching strategy, IFSBSMT, Fetch bandwidth, Instruction queue conflict, Dual priority dynamic resource allocation

随着计算机体系结构的发展,为满足人们对高性能处理器的迫切需求,多线程(Simultaneous Multithreading, SMT)处理器应运而生,并成为目前主流的微处理器体系结构。同时多线程处理器摒弃传统的物理设计技术,在不增加芯片面积的前提下,通过将物理核映射为多个逻辑核的方式,充分挖掘和利用指令级和线程级的并行性,使得处理器性能大幅度提高[1]。同时多线程处理器为处理器性能的提升提供了良好的底层硬件架构,但是传统处理器的取指策略存在取指带宽利用不均、指令冲突率过大等缺点,已无法满足现行的高密度计算任务的需求,严重影响了同时多线程处理器性能的充分发挥。本文在对同时多线程技术和现有取指策略深入研究的基础上,针对传统处理器取指策略存在的不足,通过综合指令流计算、取指优先级自适应和动态资源分配等技术的优点,提出了一种基于指令流速的双优先级SMT取指策略。最后通过设计合理的性能测试方案,验证了新型取指策略的高效性和优越性。

1 同时多线程处理器

同时多线程处理器是一种微处理器体系结构,它结合了超标量处理器和多线程处理器的优点,通过将每个物理核映射为多个逻辑核的方式,使得每个线程在逻辑核上的运行相当于在自身的物理核上运行一样,保证了在一个时钟周期内可以有多个线程的多条指令同时运行,充分挖掘了指令级和线程级的并行性,极大地提高了处理器的指令吞吐率[2]。

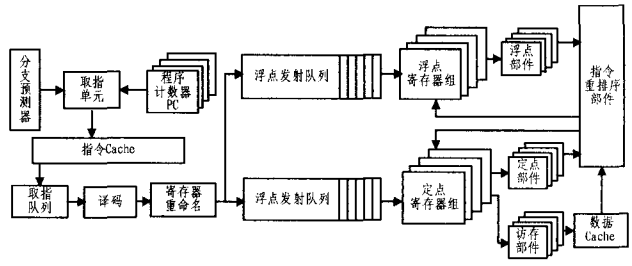

图1 同时多线程处理器的硬件结构图

同时多线程处理器的硬件结构如图1所示,该结构是基于指令乱序执行的超标量处理器实现的。为了保证在同一时钟周内,多个线程的多条指令能够在处理器同时发射和执行,同时多线程处理器需要为每个线程设置线程ID号以识别不

到稿日期:2011-10-24 返修日期:2012-02-02 本文受国家自然科学基金(60873138,61003036),黑龙江省自然科学基金(F201124)资助。
李静梅(1964—),女,博士,教授,主要研究方向为计算机系统结构、嵌入式系统、并行计算;关海洋(1987—),男,硕士,主要研究方向为多线程技术、并行计算,E-mail:haiyang_guan@163.com。

同的线程；同时，处理器还为每个线程设置单独的硬件保存线程执行过程的状态信息，这种单独的硬件称为“线程上下文(Thread Context)”，它包括程序计数器 PC、返回地址栈(Return Address Stack)和寄存器文件(Register Files)等[3]。本文的研究都是基于这种硬件结构展开的。

2 现有取指策略分析

在同时多线程处理器的取指策略中，实现最为简单的是随机取指和轮询取指。随机取指是指取指部件在每个时钟周期内从正在执行的线程随机地读取指令执行，而轮询取指是取指部件在每个时钟周期内从运行的线程中读取固定或可变数目的指令执行。二者的取指性能基本接近，但其均未考虑线程的取指优先级和指令的执行速度，因此可能会出现指令堆积和线程资源独占的现象，严重地影响后继指令的顺利执行，限制处理器取指性能的提升。

在现有的取指策略中，ICOUNT 策略的取指性能是最好的，它是一种基于指令吞吐率的取指策略，倾向于执行速度快的线程优先取指，具有一定的取指公平性[4]。其实现的基本原理为：每个时钟周期内，取指部件会选择运行速度快的线程授予较高的优先级进行取指操作，具体表现为在指令队列中占用的项数较少。反之，运行过慢的线程会被授予较低的优先级，长期堆积在指令队列中。由此可以看出，ICOUNT 策略优先选择高优先级线程进行取指操作，直到达到最大的取指带宽或出现 Cache 行越界才停止取指；在取指带宽有剩余的情况下，低优先级的线程才进行取指，导致高优先级的线程取得过多的指令造成取指带宽的浪费，而低优先级的线程因取得的指令过少，严重影响了线程的执行速度；而且，它们在指令队列中所占项数差别较大，导致取指过程中指令队列的冲突率过大，影响取指操作的正确性和取出指令的可用性。

ICOUNT 取指策略按照所执行线程数和取指带宽参数的不同还可以分为很多种。目前取指性能最好的当属 ICOUNT2.8，这里的 2 表示每个时钟周期内有两个线程进行取指，8 表示取指带宽为 8，即每个时钟周期线程最多可以取得 8 条指令，其实现的基本原理如上所述。本文取指策略也是在 ICOUNT2.8 的基础上进行改进的，将在第 3 节进行具体阐述。

指令吞吐率方面的取指策略还有 BRCOUNT、IQPOSN。BRCOUNT 策略优先选择包含分支指令较少的线程进行取指，减小分支指令对处理器性能的影响；IQPOSN 策略授予指令队列尾部的线程以较高的优先级，避免长期的指令队列堆积影响关键线程的执行，其取指性能和 ICOUNT 策略相当[5]。

资源分配方面的取指策略有 MISSCOUNT、MPF 等。MISSCOUNT 策略是优先运行未发生 Cache 失效的线程，减少 L2 Cache 失效造成指令堆积和系统资源独占现象的发生，对存储器访问密集型的工作负载效果尤为显著[6]。MPF 策略是对 MISSCOUNT 的进一步改进，通过对取指优先级的重新细化，有效地降低了 Cache 失效对取指性能的影响[7]。

此外，还有一些通过门限策略和分支分类的方式控制处理器的取指操作[8,9]，在此不再赘述。

3 IFSBSMT 取指策略设计

本节在对上述取指策略深入分析的基础上，综合考虑同时多线程处理器结构的特点，提出一种基于指令流速的同时多线程取指策略，即 IFSBSMT(Instruction Flow Speed Base On Simultaneous Multithreading)。该策略通过指令数预算的方式控制线程的取指，并采用 IPC(Instruction Per Clock)和 L2 Cache 失效率的双优先级机制合理分配处理器的系统资源，以缓解线程对共享系统资源的竞争，有效地提高了处理器的指令吞吐率和系统资源的利用率。

3.1 IFSBSMT 取指策略实现原理

IFSBSMT 策略的整个实施过程包括线程选择、取指带宽划分和系统资源分配 3 个阶段。

所谓的线程选择是指在每个时钟周期内，取指部件选择多少线程和对哪些线程取指。在这里，IFSBSMT 策略采取 ICOUNT.2.8 的线程选择方式，即每个时钟周期内选择两个线程进行取指，每次最多读取 8 条指令，有效地避免了对取指带宽划分过细而导致的某些线程因指令 Cache 失效等原因无法进行取指现象的发生。

IFSBSMT 策略的第二个阶段是取指带宽的划分，也是整个策略实施的关键阶段。在这一阶段，取指部件根据线程指令的流速和线程在指令队列中的指令数计算本周期内需要读取的指令数。若指令队列中有足够的指令执行，则不进行指令的读取，否则按照需求读取一定数目的指令，读取指令的最大数目为最初设定的取指带宽，本文的取指带宽为 8。线程在某一时钟周期内执行的指令数约为线程在指令队列中指令数的平方根[10]，那么所需指令数的计算公式应为式(1)。

$$I=I_{fs}-\sqrt{I'} \tag{1}$$

式中，I 为线程在某个时钟周期内需读取的指令数，I_{fs} 为线程在运行过程中的指令流速，其值应为线程 IPC 与某一系数的乘积，I' 为线程在指令队列中的指令数。将指令流速计算方式带入式(1)后改写为式(2)。

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

在处理器的实际运行过程中，由于 Cache 失效和分支误预测等因素的存在，系统实际获取的线程 IPC 值往往低于预估值，因此需乘以一系数 P 来纠正这一误差，即上面提到的指令流速计算方式。其中，系数 P 值与处理器的体系结构密切相关，P 值的具体设定将在实验验证环节进行具体介绍。

在系统初始化、Cache 失效和分支误预测的情况下，取指部件并不进行指令的读取操作，因此，此时的线程 IPC 值均为 0，线程的指令流速也相应地为 0，这严重影响了线程的执行速度。为避免此种现象的出现，将线程的 IPC 做加 1 处理，那么式(2)将改写为式(3)。

$$I=(IPC+1)\times P-\sqrt{I'} \tag{3}$$

在 IFSBSMT 策略具体的硬件实现过程中，线程 IPC 值的计算不仅需要额外的硬件开销，而且需通过线程的预执行和采样来修正 IPC 值，这严重影响了指令的执行速度。因此，通过采取参数 I 按位取反并对因子 P 进行取模的方式来化简繁琐的线程 IPC 值计算，从而有效地减少了 IFSBSMT 策略所需的硬件开销。新的公式表示为式(4)。

$$I=\overline{\sqrt{I'}} \bmod P \tag{4}$$

式中，需对线程在指令队列中的指令数 I' 进行开根号操作，其硬件实现的复杂度过高。本文通过采用二阶泰勒公式对根号进行处理，将式(4)改写为式(5)。

$$I=\overline{1/2(I'+1)} \bmod P \tag{5}$$

尽管利用二阶泰勒公式对式(5)进行优化所得的结果为近似值，但这并不影响取指的正确性，况且和实际取出的指令数相比，这点误差可以忽略不计。

在每个时钟周期内，某一线程所取指令数的最大值不应超过预设的取指带宽 N。因此，线程需读取的指令数应为式(5)的计算结果与取指带宽的较小值，且参数 P 的最优值应取 16，预设的取指带宽为 8。那么，最终每周期线程所取指令数的计算如式(6)所示。

$$I=\mathrm{MIN}(\overline{1/2(I'+1)} \bmod 16,8) \tag{6}$$

第二阶段只对取指带宽的划分进行设计。但由于在线程取指过程中 L2 Cache 失效现象时有发生，导致共享资源可能会被某一线程独占，影响其他后继线程的顺利执行，并限制 SMT 处理器总体性能的提升，因此，还需对线程的共享资源进行合理分配以解决这一问题。

IFSBSMT 策略的最后阶段是对系统资源的分配。本文采用线程 IPC 和 L2 Cache 失效率的双优先级方式对线程的共享资源进行动态分配，其实现的基本原理为：根据线程 IPC 值设定的资源分配优先级为主优先级；根据 L2 Cache 失效率设定的优先级为次优先级，次优先级由高到低依次划分为 CLevel 1、CLevel 2 和 CLevel 3，其评定的标准为：线程未发生 L1 data cache 和 L2 cache 失效为 CLevel 1，线程发生 L1 data Cache 失效、L2 Cache 未失效为 CLevel 2，线程发生 L2 Cache 失效为 CLevel 3。在主优先级不同、次优先级相同的情况下，以主优先级作为资源分配的依据，主优先级高的线程具有较高的资源分配权限。在主优先级相同、次优先级不同的情况下，则以次优先级作为资源分配的依据，次优先级高的线程具有较高的资源分配权限。资源分配的具体公式如式(7)所示。

$$N_i=\frac{P_{T_i}}{P_{T_i}+P_{T_j}}\times R \tag{7}$$

式中，P_{T_i} 和 P_{T_j} 分别表示线程 T_i 和 T_j 的资源分配优先级，N_i 表示分配给线程 T_i 的资源数目，R 表示系统资源的总数。

在主优先级和次优先级均不相同的情况下，以线程的 IPC 值与次优先级数的比值作为资源分配评定的依据。资源分配的具体公式如式(8)所示。

$$N_i=\frac{TL_i/CL_i}{TL_i/CL_i+TL_j/CL_j}\times R \tag{8}$$

式中，TL_i 和 TL_j 分别表示线程 T_i 和 T_j 的主优先级，CL_i 和 CL_j 分别表示线程 T_i 和 T_j 的次优先级数，其值可取 1、2、3，N_i 表示分配给线程 T_i 的资源数目，R 表示系统资源的总数。

表 1　双优先级动态资源分配机制实例化分析表

所选线程		静态资源分配机制		双优先级动态资源分配机制	
线程 T_1	线程 T_2	RT_1	RT_2	RT_1	RT_2
TL_1CL_1	TL_1CL_2	32	32	50	13
TL_1CL_1	TL_1CL_3	32	—	44	—
TL_1CL_2	TL_1CL_3	32	32	38	12
TL_1CL_1	TL_2CL_1	32	32	43	21
TL_1CL_2	TL_2CL_2	32	32	36	19
TL_1CL_3	TL_2CL_3	—	32	—	26
TL_1CL_1	TL_2CL_2	32	32	51	12
TL_1CL_1	TL_2CL_3	32	32	58	11
TL_1CL_2	TL_2CL_1	32	32	34	28
TL_1CL_2	TL_2CL_3	32	32	40	9
TL_1CL_3	TL_2CL_1	—	32	—	48
TL_1CL_3	TL_2CL_2	32	32	16	46

下面以双线程共享 64 项系统资源为例，对此机制进行实例化分析，并与传统的静态资源分配机制进行比较。具体分析结果如表 1 所列。其中，TL_1 和 TL_2 表示线程的主优先级($TL_1>TL_2$)，CL_1、CL_2 和 CL_3 表示线程的次优先级($CL_1>CL_2>CL_3$)，RT_1 和 RT_2 分别表示分配给线程 T_1 和线程 T_2 的系统资源数，“—”表示线程资源充足。

由表 1 分析可知，在传统的静态资源分配机制中，发生 L2 Cache 失效的线程也可获取一半的线程共享资源，这将造成资源的极大浪费。双优先级动态资源分配机制以优先级为参数，通过动态计算线程所需的共享资源，使得系统资源的分配更为合理，促进了系统资源利用率的提高。

3.2　IFSBSMT 取指策略硬件实现

IFSBSMT 策略的取指带宽划分阶段在物理实现方面也较为简单，其具体的硬件结构如图 2 所示。硬件的具体实现流程：在每个时钟周期内，T 选 2 多路选择器选择指令队列项数计数器值最小的两个线程进行输出，假设线程 1 的优先级高于线程 2。首先，线程 1 的计数值先经加法器和乘法器执行多项表达式的运算，之后将结果值依次进行一次按位取反和模 16 运算操作，将输出值通过 2 选 1 选择器与取指带宽进行比较，取其较小值；除读取指令的计算外，线程 2 的执行过程与 1 相同。对于线程 2，读取的指令数应为线程 1 的取指数与取指带宽的差值。最后，两个线程的输出结果送入取指部件寄存器完成取指带宽的划分。

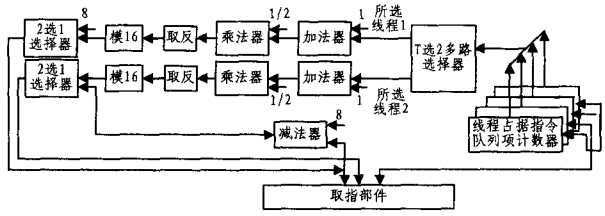

图 2　IFSBSMT 策略的硬件实现结构图

4　IFSBSMT 取指策略性能测试

本节通过在同时多线程处理器结构和 IFSBSMT 策略的特点深入研究的基础上，设计合理的性能测试方案，并深入分析 IFSBSMT 策略的性能测试结果，最终在测试结果数据分析的基础上，验证了 IFSBSMT 策略在取指性能上的高效性和合理性。

4.1　性能测试方案设计

IFSBSMT 策略的性能测试方案主要包括以下 4 个方面：模拟平台搭建、基准测试程序的选择、性能参照对象的选择和性能测试参数设计。本节分别从这 4 个方面对策略的性能测试方案进行具体介绍。

4.1.1　模拟平台搭建

本文采用西部实验室 Dean. M. Tullsen 等人研发的 SMTSIM 模拟器进行实验研究。SMTSIM 模拟器是基于 James Lames 编写的 SPIM 模拟器进行开发的，可同时运行 8 个线程，每个线程运行的指令可以达到 300M。同时，SMTSIM 模拟器还支持 Alpha 可执行程序的运行，且运行速度也是目前 SMT 模拟器中最快的[11]。模拟器参数基本配置如表 2 所列。

表 2　SMTSIM 模拟器参数的基本配置表

参数	值
功能单元	浮点:3 定点:6(包含 4 个访存部件)
流水线	9 级
取指/译码带宽	8 条指令/时钟周期
指令队列	定点:64 浮点:64
重命名寄存器	定点:32 浮点:32
L1 I-Cache	64kB 2 路组相连 64 字节/行
L1 D-Cache	64kB 2 路组相连 64 字节/行
L2 Cache	512kB 2 路组相连 64 字节/行
L3 Cache	4MB 2 路组相连 64 字节/行
ITLB	48 项
DTLB	128 项
CPU 访问延迟	L1I-Cache:1、L1D-Cache:1、L2 Cache:10、L3 Cache:20 主存:100
分支预测器	2k gshare 误预测开销为 3 个时钟周期
BTB	256kB 2 路组相连

4.1.2　基准测试程序选择

实验选取 SPEC 2000 测试集中的 7 个定点程序和 5 个浮点程序，并将其随机组合为 6 个双线程负载集进行性能测评。同时，由于在实验中完整的模拟测试程序需要花费大量的时间，有时甚至不可能完成，因此针对不同测试程序的运行指令数也进行具体的配置。具体的测试程序参数及运行指令数配置如表 3 所列，运行指令数的单位为十亿。

表 3　IFSBSMT 策略性能测试基准程序参数配置表

测试程序	数据类型	负载类型	运行指令数
gzip、swim	int、float	ILP、Mem	1.5
bzip2、lucas	int、float	Mem、Mem	3.3
mcf、crafty	int、int	Mem、ILP	1
parser、twolf	int、int	Mem、Mem	3.3
perlbmk、art	int、float	ILP、Mem	0.3
appl、sixtrack	float、float	Mem、Mem	2

4.1.3　性能参照对象

性能参照采用目前取指性能最好的 ICOUNT2.8 策略与 IFSBSMT 策略进行性能比对。通过与高性能的取指策略进行性能对比，更能显现出 IFSBSMT 策略在取指性能上的优越性和可用性。

4.1.4　性能测试参数

性能测试实验采用的评估参数包括：处理器和线程 IPC、指令队列长度及队列冲突率。

处理器的 IPC 值是指处理器在每个时钟周期内执行的指令数，它是衡量处理器指令吞吐率和加速比的重要性能指标。

IFSBSMT 策略在取指带宽划分上进行了优化，对基准测试程序占用指令队列长度和队列冲突率两项参数产生了一定影响，因此需对其进行具体的性能衡定。指令队列长度是指基准测试程序占用定点队列、浮点队列和访存队列的长度之和。指令队列冲突率是指基准测试程序所占定点队列冲突率、浮点队列冲突率和访存队列冲突率的算术平均值。

4.2　参数 P 值的选取

IFSBSMT 策略采用指令流速为参数，计算线程执行所需的实际指令数，指令的流速应为线程的瞬时 IPC 值，但是线程实际的 IPC 值往往低于预估值，因此需乘以纠正因子 P 弥补这一误差。由式(6)可知，纠正因子 P 的取值直接影响线程取指的合理性，进而影响处理器指令吞吐率性能的提升。但是，因子 P 并非常数，它的取值与实际的硬件环境密切相关，如处理器体系结构、同一时钟周期运行的线程数、取指带宽、发射带宽和指令队列长度等。因此，本文通过实验方式对不同基准测试程序在不同 P 值下的取指性能进行测试，选取所有测试程序统一的性能拐点作为因子 P 的取值。

实验采用表 3 中的基准测试程序，即每个时钟周期同时运行两个线程，每个线程执行不同的工作负载，这使得实验模拟更为接近现实的程序运行状况，在一定程度上增强了实验结果的正确性和可用性。在实验中，线程的取指采用 IFSBSMT 策略进行。通过分析不同线程组合在不同 P 值下的指令吞吐率性能，选取最佳的 P 值导入式(6)，完善线程取指的量化处理。具体的实验结果如图 3 所示。

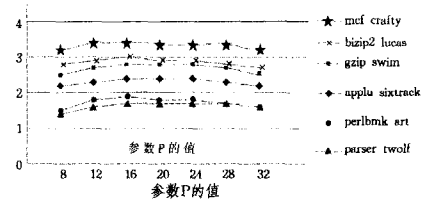

图 3　IFSBSMT 策略下不同参数 P 值的程序负载性能

由图 3 可以看出，当 P 值在(8,16)的区间变化时，所有基准测试程序的性能不断上升。而当 P 值由 16 逐步递增至 24 时，除了 perlbmk、art 程序组合的取指性能有所下降外，其他工作负载程序的取指性能均保持不变。当 P 值继续递增直至 32 时，所有程序的负载都呈现下降趋势。显然，16 为所有工作负载程序的性能拐点，因此式(6)中的参数 P 值取为 16。

4.3　性能测试结果分析

本节通过采取与 ICOUNT2.8 取指策略进行性能比对的方式，分别从处理器 IPC、线程 IPC、指令队列长度 3 个方面对 IFSBSMT 策略进行性能测试，并将获得的结果数据进行统计形成性能结果对比图，对策略在各方面的性能进行分析和评估。

处理器的 IPC 值是衡量处理器指令吞吐率和取指策略性能好坏的重要指标。对于处理器指令吞吐率的测试实验，本文采用 12 个工作负载程序两两随机组合的方式形成 6 个复合型测试程序进行性能测评，这可以使得测试环境更为接近实际程序运行状态。同时还对测试结果的平均值进行了统计，以便分析策略性能提升的整体性。具体的性能测试结果如图 4 所示。

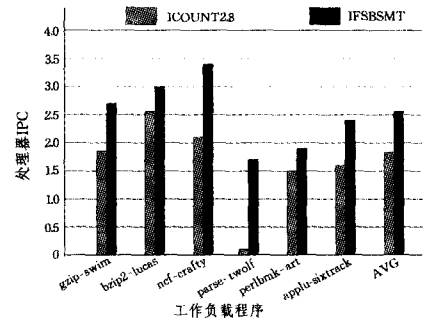

图 4　处理器 IPC 性能测试对比

由图 4 分析可知，相对于 ICOUNT2.8 策略，在 IFSBSMT 策略下处理器的指令吞吐率大幅度提升。其中，parser-twolf 负载程序性能的提升最为显著，IPC 由原有的 0.12 上升至 1.76，在两线程负载运行的条件下处理器的 IPC 性能达到 2.64，而 ICOUNT2.8 策略下处理器 IPC 的性能仅为 1.72，工作负载程序性能的平均加速比达到 42.9%。处理器指令

吞吐率性能的大幅提升主要是由于 IFSBSMT 策略更加合理地利用了取指带宽，即在每个时钟周期内，高优先级线程只读取线程执行所需的指令数，之后将剩余的取指带宽分配给其他线程，这样既满足程序自身运用的需求，又促进后继线程的顺利取指，还推进了线程执行速度。同时，采用双优先级的动态资源分配机制，避免了高优先级线程的资源独占和线程拥塞现象的发生，每个线程运行所需的系统资源根据其优先级被自适应地分配，系统资源的利用率进一步提高，间接地促进线程的顺利执行。在两种因素的共同推动下，处理器指令吞吐率性能提升显著。

IFSBSMT 策略在实现的过程中，除了侧重指令吞吐率性能的提升，还具有一定的取指公平性，所谓的取指公平性是指处理器对取指带宽的利用更为均衡，使得低优先级线程在每个时钟周期也可以获得一定的剩余带宽完成取指，并且双优先级资源分配机制也会根据线程优先级的变化动态分配给线程适量的系统资源。在不影响高优先级线程顺利执行的前提下，使得低优先级线程能够正常运行。由此可见，IFSBSMT 策略在提高处理器整体指令吞吐率性能的同时，还促进了单线程指令吞吐率性能的提升。下面针对处理器 IPC 测试实验 6 个两线程工作负载中的 12 个单线程的 IPC 性能进行测试，具体的性能测试结果如图 5 所示。

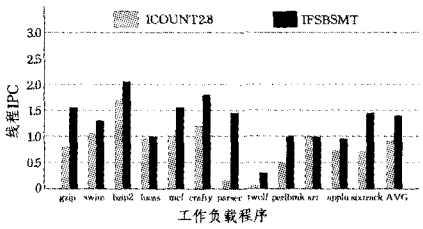

图 5　单线程 IPC 性能测试对比

由图 5 分析可知，与 ICOUNT2.8 策略相比，在 IFSBSMT 策略下 12 个工作负载程序的取指性能均有一定的提升。性能测试结果经过统计后显示，IFSBSMT 策略下单线程的平均 IPC 值为 1.32，而 ICOUNT2.8 策略仅为 0.84。可见，IFSBSMT 策略对于单线程指令吞吐率性能提升也有一定的作用。

IFSBSMT 策略在不影响处理器指令吞吐率的前提下，通过减少高优先级线程的取指数，使得线程占用指令队列的长度和指令队列冲突率相应地减小，极大地提高了系统资源的利用率。具体测试结果如图 6 所示。

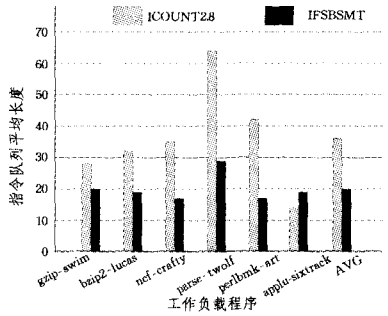

图 6　指令队列平均长度性能测试对比

由图 6 测试结果数据的统计分析可知，除 applu-sixtrack 程序负载外，其他程序负载占用指令队列的长度均所有减小，主要是由于 applu-sixtrack 程序负载读取可用指令数量的增多，使得其占用指令队列的长度有所增加，但最终表现为工作负载 IPC 性能的提升。总体而言，ICOUNT2.8 策略下程序负载的平均占用指令队列长度为 35.83，而 IFSBSMT 策略仅为 20.17。

IFSBSMT 策略通过取指带宽的合理划分和双优先级动态资源分配机制的应用，从根本上解决了线程占用指令队列冲突现象的发生，在两线程工作负载的情况下，处理器指令队列的冲突率为 0。而 ICOUNT2.8 策略由于取指带宽的利用不均衡，因此依然存在指令队列冲突问题。

结束语　基于同时多线程的取指策略 IFSBSMT 通过线程的合理选择，赋予运行速度快的线程以较高的取指优先级，推进了线程的执行速度。采取预取指令数预算的方式进行取指，使得取指带宽的利用更为均衡，在不影响高优先级线程执行的前提下，加快了低优先级线程的执行速度，有效地提高了处理器的指令吞吐率。采取负载 IPC 和 L2 Cache 缺失率的双优先级动态资源分配机制，合理分配处理器的系统资源，促进了处理器取指效率和取指质量的提升。

实验结果表明，IFSBSMT 取指策略有效地提高了处理器的指令吞吐率性能，且具有一定的取指公平性。同时，双优先级动态资源分配机制促进了处理器系统资源利用率的提升。由此可见，IFSBSMT 取指策略的应用对同时多线程处理器的研究具有重要的指导意义和研究价值。

参 考 文 献

[1] 李祖松，许先强，胡伟武. 龙芯 2 号处理器的同时多线程设计[J]. 计算机学报，2009，32(11)：2265-2273

[2] 路放，安虹，梁博，等. OpenSMT：一个同时多线程处理器模拟器的设计和实现[J]. 计算机科学，2006，33(1)：158-163

[3] Tullsen D M，Eggers S J，Emer J. Exploiting Choice：Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor[C]// 23nd Annual International Symposium on Computer Architecture. Philadelphia，American，1996：191-202

[4] 孙彩霞，张民选. 使用取指策略控制同时多线程处理器中个体线程的性能[J]. 计算机学报，2008，31(2)：309-317

[5] 何立强，刘志勇. 一种有效的同时多线程处理器取指控制机制[J]. 计算机学报，2006，29(4)：535-543

[6] Licas. Branch Classification to Control Instruction Fetch in Simultaneous Multithreaded Architectures[C]// Proceedings of the International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems(IWIA'02). UPC，Spain，2002：1-12

[7] 孙彩霞，张民选. 基于多个取指优先级的同时多线程处理器取指策略[J]. 电子学报，2006，34(5)：790-795

[8] Ali E-M，David H A. Front-End Policies for Improved Issue Efficiency in SMT processors[C]// The 9th International Symposium on High-Performance Computer Architecture. Washington，DC，2003：31-40

[9] Evers M，Yeh T-Y. Understanding branches and designing branch predictors for high-performance microprocessors[C]// Proceedings of the IEEE. American，2001：1610-1620

[10] Michaud P，Seznec A，Jourdan S. An Exploration of Instruction Fetch Requirement in Out-of-order Superscalar Processors[C]// International Journal of Parallel Programming. American，2001：311-318

[11] 潘治. 同时多线程模拟研究现状[C]// 高性能计算应用大会. ShangHai，China，2005：68-72