公平运行同时多线程处理器中的线程

孙彩霞, 张民选
(国防科学技术大学计算机学院, 湖南长沙 410073)

摘 要: 同时多线程(SMT, Simultaneous Multithreading)处理器中, 取指策略隐式的决定了共享资源在线程之间的分配, 进而决定了吞吐量和公平性. 然而, 前面对取指策略的研究大都集中在吞吐量优化上, 对公平性的研究极少. 本文把公平性作为优化目标, 提出了一种新颖的取指策略 FAIR. 实验结果表明: 对于所有类型的负载, FAIR 都能获得很好的公平性, RPRrange 都不超过 5%. 而且, FAIR 并没有以牺牲吞吐量来获取公平性. 与 ICOUNT 这种典型的以优化吞吐量为目标的取指策略相比, FAIR 的吞吐量平均只降低了 3.8%.

关键词: 同时多线程; 取指策略; 吞吐量; 公平性
中图分类号: TP303 文献标识码: A 文章编号: 0372-2112 (2008) 02-0224-06

Co-Scheduling Threads in SMT Processors Fairly

SUN Cai-xia, ZHANG Min-xuan
(*College of Computer, National University of Defense Technology, Changsha, Hunan 410073, China*)

Abstract: In Simultaneous Multithreading (SMT) processors, the instruction fetch policy determines the way of allocating shared resources among co-scheduled threads implicitly, and affects throughput and fairness consequently. However, prior work on fetch policies almost focuses on throughput optimization. The issue of fairness between threads in progress rates is studied rarely. In this paper, we purchase fairness as the main optimization goal and propose a novel fetch policy called FAIR. Results show that for all types of workloads, FAIR can achieve good fairness and RPRrange (a fairness metric defined in this paper) is less than 5%. Furthermore, FAIR does not sacrifice throughput severely. Compared to fetch policies orienting towards throughput maximization such as ICOUNT, degradation of FAIR in throughput is 3.8% on average.

Key words: SMT; instruction fetch policy; throughput; fairness

1 引言

在同时多线程(SMT, Simultaneous Multithreading)处理器[1~3]中, 同时运行的线程共享处理器中的一些资源, 主要包括发射队列, 物理寄存器, 执行单元和再定序缓冲(ROB, Reorder Buffer)等. 共享资源在线程之间的分配方式对 SMT 处理器的吞吐量和公平性具有决定性的影响. 吞吐量衡量的是每个时钟周期处理器可以提交的指令总数, 而不管这些指令来自哪个线程. 公平性衡量的则是由于资源共享导致的每个线程执行速度降低幅度是否一致.

目前, SMT 处理器中共享资源的分配主要由取指策略隐式的决定. 然而, 前面对取指策略的研究几乎都是针对如何提高吞吐量的, 对公平性的研究极少. 实际上, 在多用户系统中, 公平性比吞吐量更加重要, 因为没有哪个用户愿意等待更长的时间. 如果多用户系统采用的是单线程处理器, 那么操作系统线程调度器通过合理的分配时间片就可以保证所有线程公平的运行. 如果多用户系统采用的是 SMT 处理器, 每个时间片内, 所有被调度运行的线程共享处理器资源, 由于各个线程竞争资源的能力不同, 导致每个线程执行速度的降低幅度也不一样. 因此, 为了保证所有线程公平的运行, 单纯依靠操作系统线程调度器的调度是不够的, 硬件必须提供相应的支持.

本文把公平性作为优化目标, 提出了一种新颖的取指策略 FAIR. FAIR 使用线程的相对执行速度(相对于线程单独运行在 SMT 处理器上的执行速度)决定线程的取指优先级, 一个线程的相对执行速度越低, 它的取指优先级越高, 以此加速该线程的执行. 通过直接控制线程的执行速度, FAIR 实现了每个时间片内被调度线程的公平运行.

收稿日期: 2006-06-16; 修回日期: 2007-12-03
基金项目: 国家 863 高技术研究发展计划重大项目(No. 2005AA110020)

2 相关工作

在同时多线程处理器中, 共享资源在线程之间的分配要么是静态的, 要么是动态的. 静态资源分配[4]把共享资源等分给所有的线程, 因此同等的对待所有线程. 但是这种"同等"对待并不意味着公平对待. 从 Sherwood 等在[5]中的分析我们知道, 不同的程序具有不同的资源需求, 即使是同一个程序, 在不同执行阶段资源需求也不一样. 因此, 资源等分并不能够保证所有线程公平的运行.

在动态资源分配中, 所有线程可以自由的竞争共享资源. 取指策略就是一种动态资源分配方式, 它隐式的决定了共享资源在线程之间的分配.

轮转法(RR, Round-Robin)[2]是最基本的一种取指策略. 这种方法按照指定的顺序轮流从每个线程取指. 和静态资源分配一样, RR 同等的对待每一个线程, 所以静态资源分配中存在的问题在 RR 中同样存在.

ICOUNT[2]是一种典型的、以优化吞吐量为目标的取指策略, 被广泛应用于 SMT 处理器中. ICOUNT 优先从处于译码段、重命名段和发射队列中指令数目最少的线程取指, 也就是说, 哪个线程的指令能够快速的通过指令队列, 那么就优先从哪个线程取指, 完全忽略线程之间的公平性.

基于 ICOUNT, 还提出了许多取指策略, 如 STALL, FLUSH[6], DG 和 PDG[7]等, 这些策略同样以优化吞吐量为目标. 为了提高吞吐量, 这些策略通常会优先执行固有 IPC 较高的线程, 这样必然会牺牲公平性.

在文[8]中, Luo 等提出利用流水线提供的反馈信息修改线程的取指优先级, 以同时优化吞吐量和公平性. 虽然他们提出的策略能够在吞吐量和公平性之间进行一个很好的折中, 但是所获得的公平性还不足以满足操作系统线程调度器的需求.

最近, Cazorla 等[9]提出使用 DCRA(Dynamically Controlled Resource Allocation) 策略显式的控制资源分配. DCRA 策略首先根据线程的资源需求对线程进行分类, 然后根据分类信息决定如何在线程之间分配共享资源. 同前面介绍的取指策略和静态资源分配策略相比, DCRA 能够获得更好的公平性. 但是, DCRA 策略的最终目标仍是获取更高的吞吐量, 它所得到的公平性同样不能满足操作系统线程调度器的需求.

我们提出的 FAIR 和现有取指策略一样, 也是隐式的决定共享资源在线程之间的分配. 不同的是, FAIR 以优化公平性为主要目标, 并在保证公平性的同时, 尽可能减少吞吐量的损失.

3 实验设置

我们采用 SMTSIM[10]进行实验模拟, SMTSIM 模拟了所有类型的延时, 包括 cache 访问延时, 分支误预测延时, TLB 失效处理延时等. 同时, SMTSIM 可以收集和统计实验中有关的信息, 如 cache 失效率, 分支预测精度以及 IPC 等, 在本文的实验中, 我们只关心线程的 IPC 值. 表 1 给出了模拟器的基本配置.

表 1　模拟器的基本配置

参数	值
取指带宽	8 条指令/周期
指令对列	64 项整型队列, 64 项浮点队列
功能单元	6 个整型(4 个 load/store), 3 个浮点
重命名寄存器	100 个整数, 100 个浮点
分支预测器	2K 项的 gshare
分支目标缓冲	256 项, 4 路组相联
L1I cache	64KB, 2 路组相联, 每行 64 字节, 1 个周期访问延时
L1D cache	64KB, 2 路组相联, 每行 64 字节, 1 个周期访问延时
L2 cache	512KB, 2 路组相联, 每行 64 字节, 10 个周期访问延时
L3 cache	4MB, 2 路组相联, 每行 64 字节, 20 个周期访问延时
主存	100 个周期访问延时

表 2 给出了实验中所使用的所有测试程序, 这些程序都来自 SPEC2000 测试集[11], 并且都使用 reference 输入数据集. 在实验中, 完整的模拟测试程序要花费大量的时间, 因此采用文[12]中所提到的方法, 只模拟每个程序中最有代表性的 3 亿条指令片段. 表 2 的第 4 列给出了每个线程快速前进的指令数目. 根据 cache 行为可以把使用的测试程序分成两组: 一组是存储器访问密集的程序, 这些程序的特点是 cache 失效很多; 另一组程序的 cache 失效率相对较低, 指令级并行性(ILP, Instruction Level Parallelism)较高. 表 3 给出了实验中使用

表 2　测试程序

类　型	测试程序	输入集	快进指令数目(亿)	L2 cache 失效个数/指令
存储器访问密集型	parser	ref	16	0.011
	Twolf	ref	31	0.015
	Lucas	ref	35	0.021
	Art	c756hel.in(ref)	67	0.072
	Swim	ref	5	0.041
	Applu	ref	7	0.019
高 ILP 型	Gzip	input.graphic(ref)	3	1.3e-4
	Eon	cook(ref)	3	1.0e-4
	gap	ref	3	5.0e-4
	crafty	ref	0	9.0e-4
	fma3d	ref	1	2.4e-6
	mesa	ref	3	5.3e-4

的多线程负载. MEM 型负载中的测试程序全部来自于表 2 中的第一组, ILP 型负载中的测试程序全部来自于表 2 中的第二组, MIX 型的负载则由两组中的程序按相同比例组合而成. 为了避免实验结果倾向于某个测试程序组合, 我们为每种类型的负载随机抽取了多组, 最终结果取多组测试程序运行结果的平均值.

表 3 多线程负载

线程数	类型	包含的测试程序
2	ILP	{gzip, crafty}, {gzip, fma3d}, {gap, mesa}, {fma3d, mesa}
	MIX	{gzip, parser}, {gzip, lucas}, {fma3d, twolf}, {fma3d, swim} {parser, eon}, {parser, mesa}, {art, fma3d}, {applu, mesa}
	MEM	{parser, twolf}, {parser, lucas}, {twolf, applu}, {art, swim}
4	ILP	{gzip, eon, gap, crafty}, {gzip, crafty, fma3d, mesa}
	MIX	{gzip, gap, parser, twolf}, {gzip, crafty, art, swim} {parser, twolf, fma3d, mesa}, {applu, lucas, fma3d, mesa}
	MEM	{lucas, art, swim, applu}, {parser, twolf, lucas, art}
6	ILP	{gzip, eon, gap, crafty, fma3d, mesa}
	MIX	{gzip, gap, mesa, parser, twolf, lucas}, {gap, famed, mesa, parser, twolf, lucas}
	MEM	{parser, twolf, lucas, art, swim, applu}
8	ILP	{gzip, eon, gap, crafty, fma3d, mesa, gzip, crafty}
	MIX	{gzip, gap, fma3d, mesa, parser, twolf, art, applu} {gzip, eon, gap, crafty, lucas, art, swim, applu}
	MEM	{parser, twolf, lucas, art, swim, applu, art, parser }

使用 IPC 作为吞吐量度量标准, 并定义一个新的公平性度量标准 *RPrange*. 假设线程 i 在 SMT 处理器上单独运行时的 IPC 是 *IPCalonei*, 而和其他线程同时运行时的 IPC 是 $IPCsmt_i$. 那么, 线程 i 的相对执行速度(Relative Progress Rate) RPR_i 由等式(1)定义.

$$RPR_i = \frac{IPCsmt_i}{IPCalone_i} \times 100\% \quad (1)$$

假设 SMT 处理器上共有 n 个线程同时运行, 我们把所有线程相对执行速度的极差(Range)作为公平性度量标准, 表示为 *RPRrange*, 由等式(2)给出.

$$RPRrange = MAX(RPR_1, RPR_2, \cdots, RPR_n) - MIN(RPR_1, RPR_2, \cdots, RPR_n) \quad (2)$$

RPRrange 的值总是介于 0 和 1 之间. *RPRrange* 越小公平性就越好.

4 FAIR 取指策略

4.1 基本思想

FAIR 取指策略的基本思想是使用线程的相对执行速度决定线程的取指优先级, 一个线程的相对执行速度越低, 它的取指优先级就越高. 因此, 需要知道每个线程的相对执行速度, 即需要知道每个线程的 *IPCsmt* 和 *IPCalone*. 为了动态获取每个线程的 *IPCalone*, 我们在 *SMT* 处理器上单独运行每个线程. 因此采用两个阶段: 采样阶段(sample phase)和调整阶段(tune phase)[13].

在采样阶段, 处理器运行在单线程模式下, 每个线程单独在处理器上运行一段时间. 每个线程单独运行结束后, 通过计算可以获得该线程的 *IPCalone*.

在调整阶段, 处理器运行在 SMT 模式下. 每个周期, 计算每个线程的 *IPCsmt*, 从而获得每个线程的相对执行速度, 并根据相对执行速度对所有线程的取指优先级重新排序. 相对执行速度越低, 取指优先级越高.

有一点必须加以考虑: 同一个程序在不同执行阶段的 IPC 值会有很大的差别[5]. 因此, 为了在采样阶段获得的 *IPCalone* 能够代表该线程在调整阶段的运行速度, 我们交替执行采样阶段和调整阶段, 也就是, 所有的线程同时运行一段时间(一个调整阶段)后重新采样, 用新的采样结果指导下一个调整阶段的执行. 采样阶段和调整阶段的交替执行过程如图 1 所示.

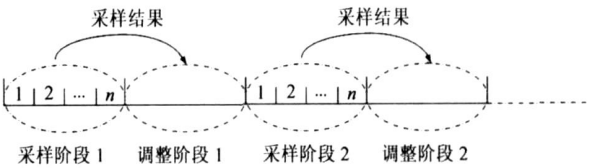


图 1 采样阶段和调整阶段的交替执行过程

为了充分利用取指带宽, 和其他取指策略一样, FAIR 也是每个周期从 2 个线程取指. 在我们的模拟中, 取指带宽是 8, 所以 FAIR 策略可以表示成 FAIR2.8. 然而, 当只有两个线程同时运行时, FAIR2.8 可能不能获得很好的公平性. 这是因为, 虽然相对执行速度较低的线程会优先取指, 但是如果它不能填满取指带宽, 那么就会从另一个线程, 也就是从相对执行速度较高的线程取指. 而由于分支指令的影响, 很难做到每个周期都从一个线程取出 8 条指令, 因此, 改变取指优先级有时无法调整两个线程的执行速度. 为了解决这个问题, 对只有两个线程同时运行的情况需要做特殊处理. 一种直观的做法就是把每个周期可以取指的线程数目由 2 变成 1, 也就是使用 FAIR1.8. 这种方法虽然可以解决公平性问题, 但是由于浪费了取指带宽, 所以对吞吐量的影响会比较大. 一种比较好的解决办法就是在 FAIR2.8 和 FAIR1.8 之间进行切换: 把两个线程相对执行速度的差值作为切换条件, 如果这个差值超过某个阈值, 那么就使用 FAIR1.8 策略阻止从相对执行速度较高的线程取指; 否则使用 FAIR2.8 策略, 使得取指带宽可以得到充分利用.

从上面的描述可知, FAIR 策略共有三个参数: $Length_{sample}$, $Length_{tune}$ 和 $Threshdd_{switch}$.

Length$_{sample}$: 在每个采样阶段, 每个线程单独运行

的时间. 假设共有 n 个线程, 那么每个采样阶段的长度将是 $Length_{sample}$ 的 n 倍.

Length$_{tune}$*:* 调整阶段的长度.

Threshold$_{switch}$*:* 当只有两个线程同时运行时, FAIR1. 8 和 FAIR2. 8 的切换阈值.

所以一个完整的 FAIR 可表示为 $FAIR\text{-}Length_{sample}\text{-}Length_{tune}\text{-}Threshold_{switch}$, 其中 $Length_{sample}$ 和 $Length_{tune}$ 的时间单位都是处理器时钟周期.

4. 2　实现

FAIR 策略最重要的是要交替执行采样阶段和调整阶段, 也就意味着要在单线程模式和多线程模式之间转换. 我们用一个有限状态机(FSM, Finite State Machine) 来实现执行模式的转换. 该状态机需要一个寄存器和两个计数器.

寄存器保存每个状态应该持续的时间, 用时钟周期数表示. 每个状态结束后, 下一个状态将要持续的时钟周期数将被写入该寄存器.

一个计数器用于识别状态机的状态. 假设共有 n 个线程同时运行, 那么状态机有 $n+1$ 个状态, 其中前 n 个状态处理器都处在采样阶段, 分别单独运行每个线程, 第 $n+1$ 个状态表示处理器处在调整阶段, n 个线程同时运行. 第 $n+1$ 个状态结束后, 状态机又回到初始状态, 重新开始采样阶段. 当线程 i 进行采样时, 其他线程是不能取指的, 为了做到这一点, 只需要修改处理器中三个参数的值: *(1)* 每个周期可以取指的线程数目置为 1; *(2)* 同时运行在处理器上的线程数目置为 1; *(3)* 把最高取指优先级赋给线程 i. 当采样阶段结束进入调整阶段时, 上面的三个参数的值做出如下修改: *(1)* 每个周期可以取指的线程数目置为 2; *(2)* 同时运行在处理器上的线程数目置为 n; *(3)* 线程的取指优先级由 FAIR 策略决定.

另一个计数器用于记录每个状态已经持续的时钟周期数目, 每个时钟周期该计数器的值加 1. 当该计数器的值达到寄存器中保存的值时, 表示一个状态结束了, 该计数器的值将被清零, 同时处理器进入下一个状态.

5　实验结果

在本节, 首先分析 FAIR 中的三个参数对公平性和吞吐量的影响, 并为每个参数选择适当的值. 然后, 使用选择好的参数值, 把 FAIR 策略和 RR 以及 ICOUNT 两种取指策略进行比较. 之所以选择这两种策略, 是因为: RR 是最基本的一种取指策略, 该策略同等的对待所有线程, 通过 RR 策略获得的公平性结果可以证明我们在第二节给出的结论: 同等对待并不意味着公平对待; ICOUNT 策略是以优化吞吐量为目标的取指策略的典型代表, 通过和 ICOUNT 比较吞吐量结果, 可以知道 FAIR 策略在获取公平性时对吞吐量造成多大的影响.

5. 1　参数选择

关于 FAIR 中的三个参数对吞吐量和公平性的影响, 很容易得出一些直观的结论. 假设一个参数发生变化时, 其他两个参数不变. 另外一个假设是 $Length_{sample}$ 远远小于 $Length_{tune}$. 我们不难得出下面的结论.

(1) $Length_{sample}$ 越大, 采样得到的 *IPCalone* 越具有代表性, 从而公平性会越好. 然而, 由于采样阶段处理器运行在单线程模式下, 所以对吞吐量的影响也会变大.

(2) $Length_{tune}$ 越大, 每次采样得到的 *IPCalone* 代表的执行区间越大, *IPCalone* 精确的代表线程在调整阶段执行速度的可能性就越小, 从而影响公平性. 不过 $Length_{tune}$ 变大使得处理器运行在单线程模式下的频率降低, 这样对吞吐量的影响也会变小.

(3) $Threshold_{switch}$ 越小, 公平性越好. 但是如果 $Threshold_{switch}$ 太小, 那么很难切换到 FAIR2. 8, 取指带宽的浪费势必会对吞吐量造成一定的影响.

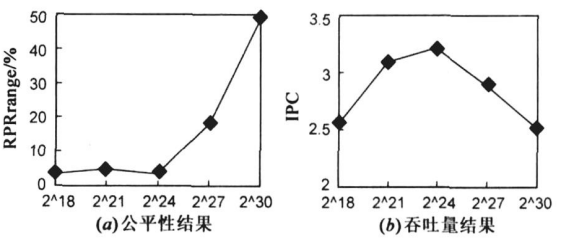
程序的 IPC 在不同执行阶段的差别较大, 所以上面的结论在某些时候可能并不完全正确. 但可以肯定的是, 三个参数对吞吐量和公平性的影响是相反的, 有利于吞吐量时公平性就会不好, 而为了获取好的公平性就要会对吞吐量造成一定的损失. 我们在参数值的选择上遵循以下原则: 在保证公平性满足一定要求的情况下, 尽可能减小吞吐量的损失.

5. 1. 1　$Length_{sample}$ 的选择

从吞吐量的角度考虑, $Length_{sample}$ 越小越好. 然而, FAIR 策略的主要目标是优化公平性, 因此 $Length_{sample}$ 不能太小, 否则采样得到的 *IPCalone* 无法代表调整阶段相应线程的执行速度. 由于 $Length_{sample}$ 对吞吐量和公平性的影响与 $Length_{tune}$ 的值有很大的关系, 而 $Length_{tune}$ 的值还没有确定, 所以我们没有通过模拟, 而是直接选择 2^{16} 作为 $Length_{sample}$ 的值[13].

5. 1. 2　$Length_{tune}$ 的选择

图 2 给出了 $Length_{tune}$ 对吞吐量和公平性的影响, *(a)* 给出的是公平性结果, *(b)* 给出的是吞吐量结果.



(a) 公平性结果　　(b) 吞吐量结果

图 2　$Length_{tune}$ 对吞吐量和公平性的影响

直观上分析, $Length_{tune}$ 越大, 吞吐量结果应该越好.

因为 $Length_{tune}$越大, 处理器运行在单线程模式下的频率就越低, 而由于 $Length_{sample}$已经确定, 所以处理器每次运行在单线程模式下的时间是固定的, 这样处理器运行在单线程模式下的总时间就随着 $Length_{tune}$的增大而减小, 所以对吞吐量的影响也会变小. 可是从图 2(*b*)我们发现这个结论并不完全正确. 在 $Length_{tune}$小于 2^{24}之前, 吞吐量随着 $Length_{tune}$的增大而增大, 可是当 $Length_{tune}$进一步增大时, 吞吐量反而降低了. 这是因为, 当 $Length_{tune}$非常大时, 采样得到的 *IPCalone* 无法准确代表调整阶段相应线程的执行速度. 如果某个线程采样得到的 *IPCalone* 偏小, 那么该线程对总体吞吐量的贡献就会变小; 可是当某个线程采样得到的 *IPCalone* 偏大时, 它对总体吞吐量的贡献未必就会变大, 因为该线程在调整阶段的最大 IPC 并没有达到 *IPCalone*. 虽然 $Length_{tune}$变大会使采样次数减少, 可是当 $Length_{tune}$很大时采样次数相差并不多, 所以单线程运行时间变短对吞吐量的提高已经显得微不足道, 这样我们就得到了图 2(*b*)所示的吞吐量结果. 从公平性角度考虑, 图 2(*a*)显示当 $Length_{tune}$等于 2^{24}时, 公平性结果也很好. 因此我们选择 2^{24}作为 $Length_{tune}$的值.

5.1.3 $Threshold_{switch}$的选择

当只有两个线程同时运行时, 我们需要确定 $Threshold_{switch}$的值. 图 3 给出了 $Threshold_{switch}$对吞吐量和公平性的影响, (*a*)给出的是公平性结果, (*b*)给出的是吞吐量结果. 图 3(*b*)的纵坐标给出的是 IPC 的减小量, 所有结果都是与 $Threshold_{switch}$无穷大时的 IPC 比较得到的. 实际上, $Threshold_{switch}$无穷大意味着任何时候都是采用 FAIR2. 8 策略, 而 $Threshold_{switch}$为 0 意味着只有 FAIR1. 8 会被采用. 从图 3(*a*)可以看出, 为了获得好的公平性, $Threshold_{switch}$越小越好; 而从吞吐量的角度考虑, 图 3(*b*)显示 $Threshold_{switch}$越大, 吞吐量的损失就越小. 综合考虑图 3(*a*)和(*b*), 选择 0. 001 作为 $Threshold_{switch}$的值.

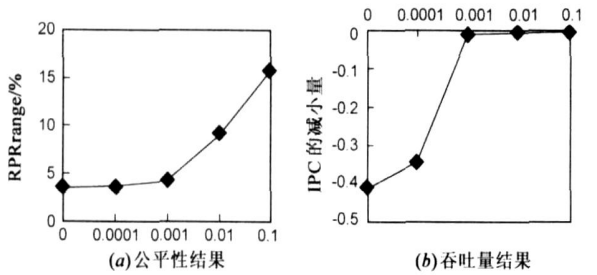


图 3 $Threshold_{switch}$对吞吐量和公平性的影响

现在, FAIR 策略可以完整的表示为 FAIR- 2^{16}- 2^{24} - 0. 001. 在 5. 2 节将比较 FAIR- 2^{16}- 2^{24}- 0. 001、RR 和 ICOUNT. 为了简单起见, 在下面的陈述中仍旧使用 FAIR.

5.2 比较 FAIR 和 RR、ICOUNT

图 4 给出了 FAIR、RR 和 ICOUNT 的公平性结果. 可以看出, RR 和 ICOUNT 策略获取的公平性非常不好, 平均 RPRrange 分别为 20. 3% 和 16. 1%, 这意味着最"快"线程和最"慢"线程的相对执行速度的差值达到了 20%. 对于 FAIR, ILP 型、MIX 型和 MEM 型负载的平均 RPRrange 分别为 3. 9%, 3. 4% 和 3. 7%, 公平性得到了极大的改进. 从图 4 还可以看出, 对于模拟的所有负载, FAIR 得到的 RPRrange 都不超过 5%, 所以 FAIR 策略获得的公平性非常稳定.

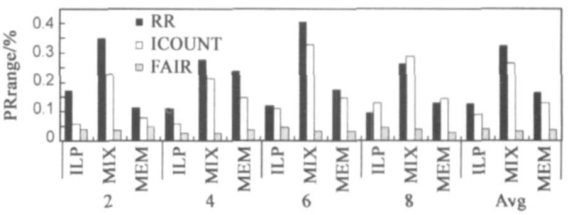


图 4 FAIR、RR 和 ICOUNT 的公平性结果

图 5 给出了 FAIR、RR 和 ICOUNT 的吞吐量结果, (*a*)给出的是绝对 IPC 值, 而(*b*)给出的是 FAIR 策略相对于 ICOUNT 的吞吐量损失. 从图 5(*a*)可以看出, 对于所有类型的负载, FAIR 都优于 RR. 与 ICOUNT 相比, FAIR 的吞吐量损失并不大, 图 5(*b*)显示平均吞吐量损失只有 3. 8%, 最坏情况下的损失是 6. 8%.

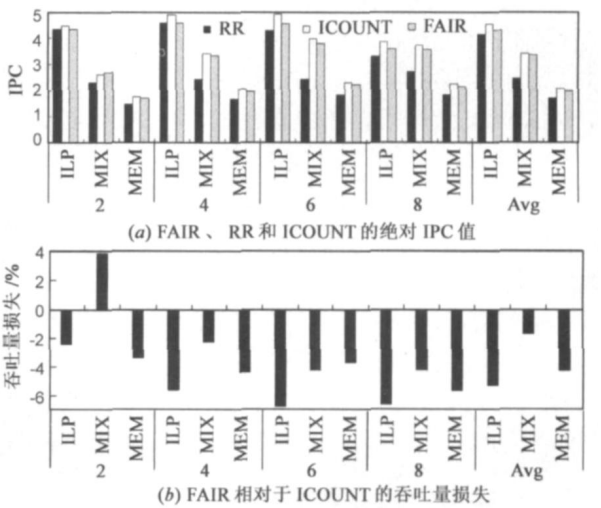


图 5 FAIR、RR 和 ICOUNT 的吞吐量结果

从图 5(*a*)和(*b*)还可以看出, 对两线程的 MIX 型负载, FAIR 在吞吐量方面要优于 ICOUNT, 提高了 3. 9%. 主要原因是模拟的 ICOUNT 策略每个周期从两个线程取指, 也就是模拟的 ICOUNT2. 8. 而两线程 MIX 型负载包含一个 MEM 型测试程序和一个 ILP 型测试程序, 当 MEM 型线程发生 cache 失效时, 如果 ILP 型线程无法填满整个取指带宽, 那么 MEM 型线程会继续取指占用共享资源. 这种情况持续一段时间后, 可能发生共享资源的阻塞(clogging), 导致 ILP 型线程没有足够的资

源继续执行. 与 ICOUNT 不同的是, 当只有两个线程同时运行时, FAIR 策略会在 FAIR2.8 和 FAIR1.8 之间进行切换, 而 FAIR1.8 能够有效的阻止 MEM 型线程占用大量的共享资源, 保证 ILP 型线程和 MEM 型线程能够根据各自的 *IPCalone* 公平的占用共享资源. ILP 型线程的有效执行最终提高了总体吞吐量.

6 结论

在 SMT 处理器中, 同时运行的线程共享处理器资源, 共享资源在线程之间的分配方式对处理器的吞吐量和公平性具有决定性的影响. 目前, 共享资源的分配主要由取指策略决定. 然而, 现有的取指策略几乎都是针对如何提高吞吐量的, 这些策略为了获取高吞吐量往往会牺牲公平性. 本文把优化公平性作为主要目标, 提出了 FAIR 取指策略. FAIR 使用线程的相对执行速度决定取指优先级, 一个线程的相对执行速度越低, 它的取指优先级就越高, 以此加速该线程的执行, 进而实现所有线程公平的运行.

我们把 FAIR 和两种典型的取指策略 RR 和 ICOUNT 进行了比较. 结果表明:

(1) 在公平性方面: 对于所有类型的负载, FAIR 都能获得很好的公平性, RPRrange 都小于 5%. 而 RR 和 ICOUNT 的平均 RPRrange 分别为 20.3% 和 16.1%.

(2) 在吞吐量方面: 对于所有类型的负载, FAIR 都优于 RR; 与 ICOUNT 相比, FAIR 所获得的吞吐量平均降低了 3.8%, 最坏情况下不超过 7%. 特别的, 对于两线程 MIX 型负载, FAIR 要优于 ICOUNT, 吞吐量提高了 3.9%.

参考文献:

[1] D Tullsen. Simultaneous multithreading: Maximizing on-chip parallelism[A]. S. Eggers, et al. Proceedings of the 22nd Annual International Symposium on Computer Architecture[C]. Santa Margherita Ligure, Italy: ACM Press, 1995. 392-403.

[2] D Tullsen. Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor[A]. S. Eggers, et al. Proceedings of the 23rd Annual International Symposium on Computer Architecture[C]. PA, USA: ACM Press, 1996. 191-202.

[3] S J Eggers, J Emer, H M Levy, et al. Simultaneous Multithreading: a Platform for next-generation processors[J]. IEEE Micro, IEEE Computer Society Press, Sept.-Oct. 1997, 17(5): 12-19.

[4] S E Raasch. The impact of resource allocation on SMT processors [A]. S. K. Reinhardt. Proceedings of 12th International Conference on Parallel Architectures and Compilation Techniques[C]. New Orleans, Louisiana, USA: IEEE Computer Society Press, 2003. 15-25.

[5] T Sherwood, B Calder: Time varying behavior of programs [R]. Technique Report CS99-630, University of California, San Diego, USA, Aug. 1999.

[6] D Tullsen. Handling long-latency loads in a simultaneous multithreaded processor[A]. J. Brown. Proceedings of the 34th Annual ACM/IEEE International Symposium on Microarchitecture [C]. Texas, USA: IEEE Computer Society Press, 2001. 318-327.

[7] A. El-Moursy. Front-end policies for improved issue efficiency in SMT processors[A]. D. Albonesi. Proceedings of the 9th International Conference on High Performance Compute Architecture[C]. Califomia, USA: IEEE Computer Society Press, 2003. 31-42.

[8] K Luo. Balancing throughput and fairness in SMT processors [A]. J Gummaraju, et al. Proceedings of the Intl. Symposium on Performance Analysis of Systems and Software[C]. Arizorra, USA: IEEE Computer Society Press, 2001. 164-171.

[9] F J Cazorla. Dynamically controlled resource allocation in SMT processors[A]. A Ramirez, et al. Proceedings of the 34th Annual ACM/IEEE International Symposium on Microarchitecture [C]. Portland, Oregon: IEEE Computer Society Press, 2004. 171-182.

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

[11] The standard performance evaluation corporation[Z]. http://www.specbench.org.

[12] T Sherwood. Basic block distribution analysis to find periodic behavior and simulation points in applications[A]. E. Perelman, et al. Proceedings of the 10th Intl. Conference on Parallel Architectures and Compilation Techniques[C]. Barcelona, Spain: IEEE Computer Society Press, 2001. 3-14

[13] F Cazorla, P Knijnenburg, et al.: Predictable performance in SMT processors[J]. IEEE Transactions on Computers, IEEE Computer Society Press, July 2006, 55(7): 785-799.

作者简介:

孙彩霞　女, 1979 年 10 月生, 博士, 讲师. 主要研究方向为微处理器体系结构、同时多线程、片上多处理器以及大规模集成电路设计等. E-mail: cxsun1979@163.com

张民选　男, 1954 年 3 月生, 教授, 博士生导师. 主要研究方向为高性能计算机系统结构、微处理器设计、低功耗设计及 ASIC 技术等.