

基于线程感知寄存器重命名的 SMT 处理器资源分配

杨 华¹⁾ 崔 刚²⁾ 刘宏伟²⁾ 杨孝宗²⁾

¹⁾(沈阳航空工业学院计算机学院 沈阳 110136)

²⁾(哈尔滨工业大学计算机学院 哈尔滨 150001)

摘 要 SMT 处理器的资源分配一般是通过调控各线程的取指过程间接实现的,这种间接调控有时会导致资源滥用和饥饿,从而严重浪费资源并降低整体性能。以往的改进措施往往实现代价较大,且不能消除资源分配的“不均衡性”,因此效果不太理想。文中提出一种新的 SMT 处理器资源调控机制——线程感知寄存器重命名 TSRR (Thread-Sensitive Register Renaming),消除了资源分配的“不均衡性”,其优点如下:(1)资源分配自动适应线程运行状态的变化,实现“按需分配”;(2)通过调控重命名寄存器文件(RRF)的分配来间接控制其它资源分配,实现代价较低;(3)兼顾资源分配的效率和公平,既防止了资源滥用和饥饿,又充分发掘各线程的性能潜力。此外,TSRR 还可以间接降低 RRF 的尺寸要求和取指逻辑的复杂度。

关键词 同时多线程;资源分配;寄存器重命名;处理器;高性能

中图法分类号 TP302

Regulating SMT Resource Allocation via Thread-Sensitive Register Renaming

YANG Hua¹⁾ CUI Gang²⁾ LIU Hong-Wei²⁾ YANG Xiao-Zong²⁾

¹⁾(School of Computer, Shenyang Institute of Aeronautical Engineering, Shenyang 110136)

²⁾(School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001)

Abstract SMT processors generally regulate the resource allocation indirectly by controlling the Instruction-Fetch(I-Fetch) process, which may lead to resource misuse and even starvation, incurring resource underutilization and performance depression. Various improving techniques have been proposed; however their effects are discounted due to either being too expensive to implement, or failing in eliminating the imbalance of resource allocation. This paper proposes a novel scheme, Thread-Sensitive Register Renaming (TSRR), which serves as a resource gating, remarkably eliminating the imbalance of resource allocation and improving the overall performance. TSRR features that: (1) it tracks the performance variations and dynamically tunes the resource amount available to each thread, realizing allocation-on-demand, (2) it is cost-effective because it tunes up all resources just by regulating the allocation of the rename-register-file (RRF), and (3) concerning both effectiveness and fairness, TSRR prevents both resource misuse and starvation, whereas fully exploits the performance potential of each thread. Meanwhile, TSRR can lessen the RRF size demands and I-Fetch hardware complexities.

Keywords SMT; resource allocation; register renaming; processor; high-performance

收稿日期:2006-04-27;最终修改稿收到日期:2007-12-29。本课题得到“十五”预研基金(41316.1.2)、国家自然科学基金(60503015)和教育
部博士点基金项目(20020213017)资助。杨 华,男,1974 年生,博士,副教授,主要研究方向为片上多线程体系结构、容错处理器体系
结构、可信计算。E-mail: yangh@syiae.edu.cn。崔 刚,男,1947 年生,教授,博士生导师,研究领域为容错计算、高性能计算机系统结构。
刘宏伟,男,1971 年生,博士,副教授,研究方向为可信计算、移动计算。杨孝宗,男,1939 年生,教授,博士生导师,研究领域为容错计算、可
穿戴计算。

1 引言

SMT (Simultaneously Multithreading) 处理器中多个线程同时执行、竞争并共享处理器内部的各种资源,有效提高了处理器的资源利用率和整体性能.如图 1 所示,SMT 的前端(front-end)包括取指(Fetch)、译码(Decode)、重命名(Rename)和入发射队列(Enqueue)等流水段,后端(back-end)包括执行(Execute)、写回(Writeback)和提交(Commit)等流水段. SMT 设计的一个重要目标是借助多线程对资源的“竞争式共享”,实现资源利用率的最大化和整体处理能力(指令吞吐率)的最大化^[1-4].从资源的角度看,指令的流动要占用带宽类和存储类资源^[5],二

者可分别看作指令流动的机会和结果.带宽类资源指 Fetch、Rename、Issue 和 Commit 等各段带宽,其占用和释放通常在一个周期内完成.存储类资源主要包括取指队列(IFQ)、重命名寄存器文件(RRF)、指令队列(IQ)、再排序缓冲(ROB)和 Load/Store 队列(LSQ)等,占用后一般要等待多个周期,直到指令进入下一流水段或执行完成后才能释放.因此,存储类资源竞争造成的线程间干扰比较大.一般认为,取指段作为整个流水线的入口,是实现多线程对资源合理分配的关键.因此,目前为止针对 SMT 资源分配的研究几乎都专注或涉及到取指策略——试图通过改善取指来为各线程提供充足正确的指令流,从而提高资源利用率,即资源分配是靠取指过程间接调控的^[2-3,6-10].

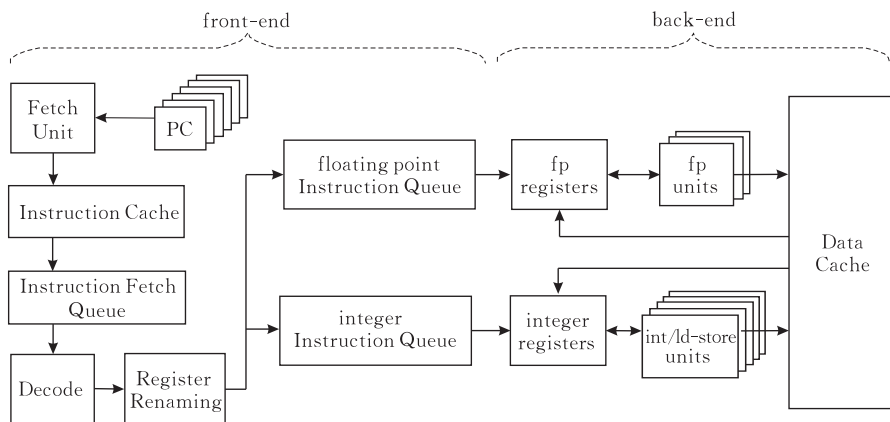


图 1 SMT 处理器基本结构

基本的取指策略如下^[2-3]：

(1) MISSCOUNT. 对 DCache 命中率较高的线程优先取指,减少 DCache-Miss 长时间等待造成的指令队列拥塞和 RRF 等资源的浪费.该策略适于消除 MEM 类线程(数据访问量较大且命中率较低)对其它线程的干扰.

(2) BRCOUNT. 对前端各段中分支指令较少的线程优先取指,减少误预测分支上的无效指令进入流水线.该策略适用于分支预测率较低的情况.

(3) ICOUNT. 对前端各段中指令数较少的线程优先取指,平衡各线程的前端指令数量,防止个别线程对资源的过度占用.

(4) Round-Robin. 最简单的策略,即各个线程轮流取指,也是防止个别线程对资源的过度占用.

研究表明,虽然上述策略各有优缺点,但总体上 ICOUNT 的效果最好^[2],被广泛采用.这是因为线程的前端指令数量通常与其运行速度密切相关,也就是说,如果一个线程的前端指令数量较少,通常意

味着该线程后端处理较快,即该线程的指令级并行(ILP)较大且运行速度较快.因此,ICOUNT 取指偏重于前端指令数量少的线程,一般能够达到根据各线程的运行状态(速度、资源需求)对资源分配进行合理调控的目的.

1.1 ICOUNT 的局限性

虽然前端指令数量一定程度地反映运行速度,但是由于长延迟指令(如 Load 指令发生 L2-DCache-Miss)的存在,前端指令数量少并不等价于线程运行速度快^[3,8,11],反之亦然.例如:下面的代码段中,某线程的 6 条指令进入流水段,其中 i_1 是一条 Load 指令,已进入 Execute 段; $i_2 \sim i_4$ 在指令队列中等待发射; $i_5 \sim i_6$ 在取指队列中等待译码.此时该线程的前端指令只有 5 条($i_2 \sim i_6$),与其他线程相比可能最少,根据 ICOUNT 策略应该优先取指.但是如果指令 i_1 发生 L2-DCache-Miss,该线程实际上处于停滞状态,等待 i_1 返回,至少需要上百个周期(取决于 Mem 的访问延迟).此时如果继续对该

线程进行取指,不但挤占了其它线程的取指带宽,而且新取得的指令进入流水段后也将停滞,并长时间占用宝贵的共享资源(RRF、IQ、ROB 等);这种情况经过若干周期的积累,会造成速度最慢(停滞或半停滞)的线程对资源的大量无效占用,严重影响其它线程的正常运行,我们称之为资源滥用。

...
i1:	ldq r3, 104(r18);	in Execute
i2:	s4addq r12, r3, r3;	in IQ
i3:	ldl r3, 0(r3);	in IQ
i4:	and r3, 1, r3;	in IQ
i5:	cpys f31, f31, f31;	in IFQ
i6:	cmovne r3, r4, r11;	in IFQ
...

下面以测试组合 `gzip_crafty_vortex_facerec` 和 `gzip_crafty_vortex_mcf` 为例(参见第 3 节),说明 ICOUNT 在防止资源滥用方面的局限性. 两个测试组的第 4 线程分别是 `facerec` 和 `mcf`,其余相同;以 IPC(Instruction-Per-Cycle)衡量性能,两个测试组合的总 IPC 分别是 4.54(即 $1.23 + 1.12 + 1.10 + 1.09$)和 2.63(即 $0.89 + 0.76 + 0.79 + 0.19$);造成二者巨大差别的原因是 `mcf` 发生 Load L2-DCache-Miss 的比率高达 90.7%,致使 `mcf` 大量指令拥塞在前端,严重妨碍了其余 3 个线程的运行.

1.2 各种改进策略及其分析评述

针对 ICOUNT 在资源分配方面的不足,前人相继提出了一些改进策略,目的是降低或防止资源滥用. 具代表性的有如下几种:

(1) STALL^[3]. 在 ICOUNT 基础上监测 L2-DCache-Miss 的发生,暂停其对应线程的取指、译码、重命名和入队列等前端处理,直到该指令返回.

(2) FLUSH^[3]. 在 STALL 的基础上,对发生的 L2-DCache-Miss 的线程不但暂停前端处理,而且清空该线程的所有前端指令,释放无效占用的资源,供其余线程使用.

(3) Data-Gating^[8]. 即由 L1-DCache-Miss 触发的 STALL 策略,改善原 STALL 策略中 L2-DCache-Miss 监测和触发不及时的问题.

(4) ADTS^[11]. 一种混合策略,根据各线程的运行情况在多个取指策略间动态切换(如在 ICOUNT、L1-MISSCOUNT 和 BRCOUNT 间切换);ADTS 需要附加一个专门的监测线程(DT),用以监测各线程的运行状态并触发不同策略间的切换.

(5) STATIC^[5]. 将资源静态划分给各个线程,防止线程间的恶性竞争.

(6) DCRA^[12]. 持续监测各线程的运行状态和

对各类资源的使用情况,定期将线程重新划为 Fast 组和 Slow 组,并跟踪标记各线程对每类资源的状态为 active 或 inactive,对 Slow-inactive 的线程优先分配资源. 该策略适用于各类资源都很充裕的“豪华型”处理器,强调的是各线程对资源占用的公平性,防止饥饿.

上述策略中,STALL、FLUSH、Data-Gating 和 ADTS 仍是从取指的角度间接改善资源分配,这一点与 ICOUNT 类似,是“隐式调控”. “隐式调控”的优点是调动各线程本身的“竞争力”,发挥 SMT 资源共享的优势;其缺点是不恰当的竞争会导致资源滥用,影响整体性能. 与 ICOUNT 相比,这四种策略有效减少了资源滥用,但仍存在如下问题:(1) STALL 和 FLUSH 靠监测 L2-DCache-Miss 触发,有一定的滞后;(2) STALL 触发时不清空前端指令,因此对已有的无效占用资源并不释放;(3) FLUSH 的实现需要复杂的硬件支持,例如要为每条指令设立检查点,用以在清除指令时恢复寄存器映射表等结构;另外,FLUSH 恢复后被清除的指令要再次进入流水线,这种重复的处理浪费功耗;(4) Data-Gating 的触发比较及时,但有时过于敏感,因为 L1-DCache-Miss 不代表一定发生 L2-DCache-Miss^[7]. (5) ADTS 需要监测和记录的信息太多(如各线程的运行速度、Cache 命中率、分支预测率),而且监测线程 DT 要挤占部分资源,因此该策略的实现复杂,代价很高. 前人还提出 FLUSH++^[6]和 DCache-Warn^[7]等改进措施. FLUSH++作为折衷在 STALL 和 FLUSH 间切换,然而线程行为的差异性^[13],加之 STALL 和 FLUSH 本身的不足,使得 FLUSH++的效果并不理想. DCache-Warn 试图通过预测 DCache-Miss 来改善 STALL、FLUSH 和 Data-Gating 的触发,然而增加的预测硬件不但代价较大,而且受限于多线程环境下 DCache 行为的难预测性^[14],效果难以保证.

STATIC 和 DCRA 直接干预资源分配,是“显式控制”. “显式控制”的优点是直接限制各线程的资源占用份额,减轻甚至杜绝资源抢占冲突;其缺点是压制了各线程本身的“竞争力”,限制不当则会削弱 SMT 资源共享的优势. 各线程的性能和资源需求处于动态变化中^[11,13],STATIC 显然对此难以适应——不恰当的划分影响整体性能,而平均分配难免浪费资源. DCRA 虽然能动态调整资源分配,但存在如下缺点:(1)要监测和记录的信息量太多,不但包括各线程的运行情况,还包括对各类资源的分配和占用情况,这要增加相当大的硬件复杂度;

(2) 由于各线程对资源占用的不均衡性, DCRA 对资源的分散式管理有时会出现类似“哲学家进餐问题”的冲突^①, 因此 DCRA 仅适用于各类资源都很充裕的情况; (3) DCRA 强调“防饥饿”, 因此在资源分配时偏向于 Slow-inactive 线程, 这会影响 Fast-active 线程的运行, 从而影响整体吞吐率。

综上所述, “隐式调控”(ICOUNT、STALL、FLUSH、Data-Gating、ADTS 等), 通过控制取指间接调控资源分配, 虽然灵活, 但调控滞后会造成存储类资源的滥用。“显式控制”(STATIC、DCRA 等) 强调资源的划分, 虽然减少了冲突, 但容易造成资源利用不充分。此外, 以上措施普遍代价较高、不利于实现。

1.3 TSRR 的提出

我们分析发现, 无论是“隐式调控”的滞后, 还是“显式控制”的不恰当约束, 归根到底是因为指令流动的“不均衡性”引起的。“不均衡性”来自两个方面: (1) 等待时间的不均衡, 从取指到提交, 有的指令仅需要几个周期, 有的则需要十几、几十、甚至上百个周期(如浮点除法指令、各级 DCache-Miss 及后继依赖指令); (2) 资源占用效率的不均衡, 对于存储类资源而言, 既有“随占用随释放”的高效情况, 又有长时间“占而不用”的浪费的情况(如 Load L2-DCache-Miss 及后继依赖指令)。

从“不均衡性”的角度看, 以往各种策略往往只注意到一个方面, 因此效果并不理想。例如, ICOUNT、STALL、Data-Gating 和 ADTS 关注提高指令流的质量来减少等待时间, 而忽视了已占用资源的效率; STATIC 旨在防止资源恶性竞争, 但缺少灵活性, 压制了高质量指令流(即线程运行较快、平均等待时间短)的性能潜力; 而 DCRA 除了实现复杂, 对高质量指令流也存在一定的制约。FLUSH 虽然兼顾了指令的等待时间和资源占用效率, 但其 L2-DCache-Miss 的监测延迟仍会造成一定程度的调节滞后。如何更有效地克服“不均衡性”, 成为进一步提高 SMT 资源分配效率的关键。

我们认为, 必须将两方面的“不均衡性”综合考虑才能找到更好的解决方案。为此, 我们提出了线程感知寄存器重命名技术 TSRR(Thread-Sensitive Register Renaming), TSRR 作用于 Rename 段, 通过调控 RRF 的分配对整个处理器的资源分配进行间接调控。TSRR 的提出基于以下几点:

(1) 理想的资源划分应该彻底消除无效占用, 这在线程行为无法完全预知的前提下几乎是不可能的, 但是如果尽早地发现并且消除无效占用, 同样可

以消除资源滥用对性能的影响。

(2) 带宽类资源适合共享, 让所有线程以某种方式竞争^[1-2,5], 如 Round-Robin、ICOUNT、OLD-EST-FIRST 等, 这些方式能够适应线程对带宽资源的需求变化, 合理分配带宽。

(3) 存储类资源更适合用“显式控制”的方式划分^[5,11], 关键是找到更理想的划分策略, 能适应各线程运行状态的持续变化。

(4) 动态超标量处理器中, 所有指令经过 Rename 段明确依赖关系后才能进入后继处理; 其中大多数指令(实验统计 70%~90%, 主要是 Load 和运算类指令)要在 Rename 段分配 RRF 成功后才能继续执行, 是关键路径上(即决定线程性能)的指令。因此, RRF 分配是指令进入流水线后的首次、也是最重要的一次存储类资源分配, 对于 SMT 处理器而言更是如此^[15]。实验表明, 控制了一个线程的 RRF 分配, 也就控制了该线程对后继流水段的资源分配。换句话说, 通过在 Rename 段调控各线程的 RRF 占用比例, 可以有效控制各线程对各类资源的分配。

TSRR 与 1.2 节列举的各种措施相比, 基本消除了“不均衡性”的影响, 其主要特点是: (1) 资源分配自动适应线程运行状态的变化, 实现“按需分配”; (2) 通过调控重命名寄存器文件(RRF)的分配来间接控制其它资源分配, 实现代价较低; (3) 兼顾资源分配的效率和公平, 既防止了资源滥用和饥饿, 又充分发掘了各线程的性能潜力。此外, TSRR 还可以间接降低 RRF 的尺寸要求和取指逻辑的复杂度。为了量化比较, 我们定义了新的衡量尺度——资源占用均衡度(BalDeg), 并通过模拟实验对 ICOUNT、STALL、FLUSH 和 TSRR 进行了比较。实验表明, TSRR 在性能和资源占用均衡度方面均明显优于其它策略。

2 TSRR 策略

TSRR 目标是让处理器的资源分配能够“感知”各线程运行状态的变化, 提供与之相符合的资源分配, 提高资源占用均衡度。下面论述 TSRR 的基本原理、具体运行方式、实现代价。

2.1 基本原理

TSRR 的基本原理, 是通过在 Rename 段调控各线程的 RRF 占用比例来间接控制各类资源的分

① 线程 T1 分得较多 A 资源和较少 B 资源, 线程 T2 与之相反。当 T1 遭遇 B 资源不足时, 即使 T2 有多余的 B 资源, T1 也无法占用; 同样情形, T2 也无法占用已分给 T1 的 A 资源。这样, “无法互通”使得 T1 和 T2 都因为缺乏对方的资源而影响性能。

配. 其设计目标是: (1) 减少甚至消除线程间的资源恶性竞争; (2) 资源分配感知并适应各线程运行状态的变化; (3) 杜绝饥饿. 设线程级并行度为 N (即 N -context SMT), RRF 的尺寸为 $RRFsize$, 我们将 $RRFsize$ 逻辑上分为 $localSum$ 和 $global$ 两部分, 并采用如下三个措施 (针对三个设计目标):

首先, 限定各线程对 RRF 的占用上限为 $local_i$ ($i=1, \dots, N$), $local_i$ 是根据各线程的运行状态划定的, 且 $\sum_{i=1}^N local_i = localSum$, 即划定各线程“专有”的 RRF 份额, 互不交叉, 借此消除线程间的恶性竞争.

其次, 动态调整 $local$ 的划分以适应各线程运行状态的持续变化. 各线程的运行状态 (相对性能和资源需求) 持续变化, 只有及时感知这种变化并据此调整 $local$ 的划分, 才能有效消除“不均衡性”. 这需要解决两个问题: (1) 如何评估各线程的运行状态 (资源需求) 的变化? (2) $local$ 的调整间隔如何? 即 $local$ 调整的触发条件是什么? 对第一个问题, 我们以各线程的 IPC 作为其运行状态的标示, 因为 IPC 既是重要的性能指标, 又能准确反映出资源需求 (即 IPC 越大, 以 RRF、IQ、ROB 等为代表的资源需求量和利用效率就越高). 鉴于 IPC 的含义, 在实际中可以用两次调整期间各线程提交的指令数来代替. 因此, 需要为每个线程设置一个计数器 $CInst_i$ ($i=1, \dots, N$), 记录两次调整期间各线程提交的指令数. 每次触发 $local$ 调整时, $local_i$ 按式 (1) 重新设定:

$$local_i = \frac{CInst_i}{\sum_{i=1}^N CInst_i} \times localSum \quad (1)$$

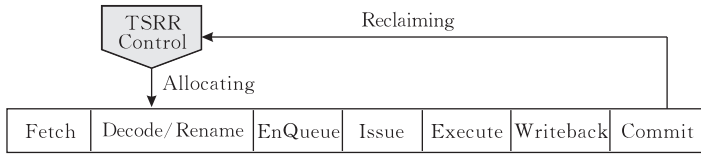


图 2 加入 TSRR 的流水线

图 3 给出了 TSRR 控制下的 RRF 的分配和回收流程, 图中灰色部分代表 TSRR 控制, 其中 $CTotal_i$ ($i=1, \dots, N$) 记录各线程占用的 RRF 单元数, $CGlobalLeft$ 记录可用的 $global$ 容量. Decode/Rename 段轮流 (Round-Robin) 从各线程的 IFQ 取指令, 以保证公平; 模拟实验表明, 轮流方式兼顾各个线程, 非常适合于像 Decode/Rename、Issue 等带宽类资源 (可视为指令前进的机会) 的分配. 从图 3 可以看出: TSRR 在分配 RRF 时优先占用各线程的 $local$ 份额, 只有无 $local$ 可用时 ($CTotal_i < local_i$ 为假) 才试图占用 $global$; 相反, TSRR 回收 RRF 时对

对第二个问题, 简单起见我们选用固定的周期性触发方式, 比如设定触发间隔为 100 万个周期.

第三, 保留少量的 RRF 供各线程竞争, 称为 $global$, 弥补 $local$ 划分的不足. $global$ 的具体作用: (1) 允许各线程竞争, 更好地挖掘和展现各线程的性能潜力, 使下一次 $local$ 调整更加准确; (2) 让各线程都有机会前进, 防止饥饿, 例如性能最差的线程的 $local$ 份额可能是 0, 但一样可以通过竞争 $global$ 来前进.

综上所述, 全部 RRF 的分配可以用式 (2) 来表示. 通过把 RRF 分为“专有的” $local$ 和“共享的” $global$, 并对 $local$ 进行“感知线程运行状态”的调整, TSRR 实现了资源占用方式的均衡——既保持了竞争的灵活和性能优势, 又消除了恶性竞争造成的资源滥用和饥饿.

$$RRFsize = global + localSum = global + \sum_{i=1}^N local_i \quad (2)$$

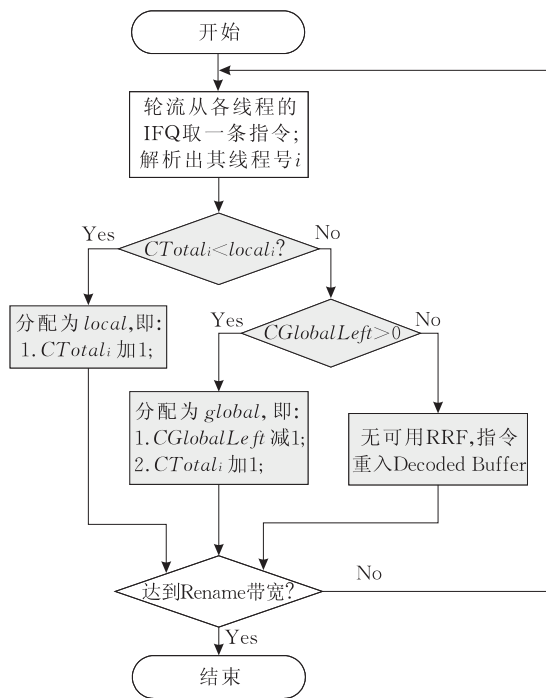
需要声明的是, RRF 的划分只是逻辑上的, 并不是物理划分, 即 $local$ 和 $global$ 是逻辑上的阈值, 不对应具体的 RRF 单元. 这样使 RRF 的分配和释放更加灵活和容易管理.

2.2 运行方式

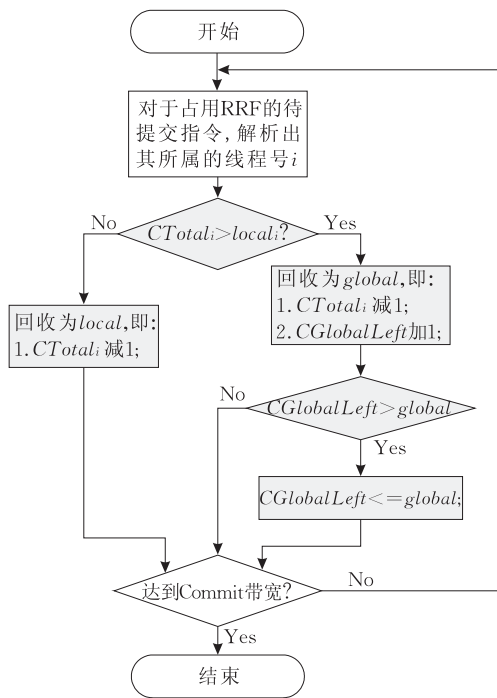
图 2 给出了引入 TSRR 后的流水线变化, 即在原流水线基础上加入 TSRR 控制, 总长度仍是 7 段. TSRR 作用于 Decode/Rename 段, 通过调控 RRF 的分配 (Allocating) 间接控制整个处理器的资源分配; 另外, 在 Commit 段回收 (Reclaiming) RRF 时需更新 TSRR 控制记录.

$global$ 优先, 即占用量超过 $local$ 份额时 ($CTotal_i > local_i$ 为真) RRF 单元作为 $global$ 回收. 需要注意的是 $CGlobalLeft$ 应该总小于或等于 $global$; 但在 $local$ 刚调整后的几个周期内, 回收 RRF 时可能出现 $CGlobalLeft$ 大于 $global$ 的情况, 这是由 $local$ 调整引起的, 只需将 $CGlobalLeft$ 设置为 $global$ 即可 (参见图 3(b)).

综上, 多数情况下各线程只占用自己的 $local$ 份额, 即使在少数情况因 $local$ 不够而占用了 $global$, 也会尽快释放. 这使得 TSRR 在保持 RRF 竞争的同时减少甚至消除了 RRF 占用冲突, 提高了 RRF 占用均衡度.



(a) RRF分配流程



(b) RRF回收流程

图 3 TSRR 控制下 RRF 的分配和回收流程

2.3 对 *local* 初始设置和线程切换的考虑

初始 RRF 为平均分配, 即每个线程的初始 *local* 为 $localSum/N$. 整个运行过程的分配合理性靠 *local* 定期调整来实现. 各线程的运行状态变化不能未卜先知, 因此任何资源调控机制都会有一定的调整和适应阶段, 模拟实验证明, TSRR 的初始值设置只对最初几个调整周期稍有影响, 对大规模连续运行时的影响可以几乎忽略. *local* 的持续动态调整、辅以 *global* 的竞争和释放机制, 使得 TSRR 能够及时反映出各线程的资源需求变化, 这也正是 TSRR 的优势所在. 例如, 在 4-线程、 $RRFsize = 60$ 、*local* 调整周期为 1M cycle 的 TSRR 模拟实验中, 发生 *local* 变化的次数占总调整次数的比例高达 65.4%, *local* 次序(各线程按 *local* 大小排序)的调整比例高达 28.1%. 这足以反映 *local* 调整的及时性、频繁性、必要性.

对于 *N*-context SMT 处理器来说, 最多支持 *N* 个线程同时运行, 因此在两次 *local* 调整之间有新线程被调入时, 可分为两种情况: (i) 有 *N* 个线程正在运行, 此时发生 SMT 线程切换, 即新线程将某个正在运行的线程替换出来^①, 新线程直接继承被替换线程的 *local*, 开始运行并等待下次调整; (ii) 正在运行的线程数小于 *N*, 此时将新线程调入处理器, 竞争 *global* 开始运行, 并等待下次调整.

实际上, 无论 (i) 和 (ii) 哪种情况, 新线程的调入

过程必然要引起一定的开销, 因此最直接的策略就是: 发生线程切换时立即触发一次 *local* 调整; 为了公平, 新调入线程的 *local* 值可设为初始平均值, 即 $localSum/N$, 其余线程仍按式(1)调整. TSRR 有很强的调整适应能力, 寄存器分配会很快适应线程切换的变化, 达到新的均衡. 如果线程切换频繁的话, 影响处理器性能的因素将主要来自切换开销(超出本文讨论范围).

2.4 实现代价

由 2.1 节和 2.2 节可知, TSRR 的实现比较简单, 只需对 RRF 的分配和回收情况加以简单的记录和控制即可(图 3 中的灰色部分). 在 *N*-context 的 SMT 中实现 TSRR 需增加以下的硬件:

- (1) 寄存器 $local_i$ ($i = 1 \sim N$): 设置各线程的 *local* 阈值, 定期调整;
- (2) 寄存器 *global*: 设置 *global* 阈值;
- (3) 计数器 $CTotal_i$ ($i = 1 \sim N$): 记录任一时刻各线程 RRF 的占用量;
- (4) 计数器 *CGlobalLeft*: 记录任一时刻可用的 *global* 数量;
- (5) 计数器 $CInst_i$ ($i = 1 \sim N$): 记录两次 *local* 调整期间各线程提交的指令数, *local* 调整后清 0;

^① SMT 发生线程切换时, 由 OS 或 SMT 的线程替换策略选择和决定被替换的线程, 超出本文的讨论范围.

(6) 简单的比较和控制逻辑:用于实现图 3 中的比较逻辑,控制 *local* 或 *global* 的分配和回收.

此外,每次触发 *local* 调整时需根据式(1)重新计算各线程的 *local*,可以用处理器中通用功能单元 (ALU、Mul/Div 等) 完成,需要几十或上百个周期. 然而,与上百万个周期的调整间隔相比,调整计算的代价微不足道,不会对性能造成影响. 各线程的 IPC 可用调整间隔内的提交指令数代替,现代处理器内部大都提供这类监控记录^[16]. 模拟实验还表明,即使配合简单的 Round-Robin 取指,TSRR 的效果仍然不错,降低了处理器对复杂取指逻辑的依赖.

综上,TSRR 对软件透明,增加的硬件代价很少,完全不影响流水线长度和时钟周期;与 1.2 节列举的措施相比,TSRR 具备明显的实现优势. 例如:FLUSH 的实现需要复杂的硬件支持,为每条指令设立检查点,用以在清除指令时恢复寄存器映射表等结构;另外,FLUSH 恢复后被清除的指令需要再次进入流水线,这种重复处理造成功耗浪费. 又例如:DCRA 需要监控和记录的信息太多(各线程的运行状态、各类资源的使用情况)、控制复杂(定期划分 Fast 组和 Slow 组、跟踪标记 active 或 inactive),适

用于“豪华型”处理器,强调防止饥饿. 然而在资源不是很充裕的情况下,过分强调公平(防饥饿)会伤害到效率,即对 Slow-inactive 的优先容易导致 fast-active 的性能潜力受到压制,反而降低整体性能,有悖于 SMT 节省资源、提高性能的宗旨. 从这个意义上讲,过分强调公平会导致另一种不公平.

3 模拟实验

以 Tullsen 描述^[1-2]为蓝本,我们对 Simple-Scalar^[17]/Alpha AXP ver3 进行扩充和修改,构建了 SMT 节拍级模拟器,支持 Round-Robin 和 ICOUNT 取指、OLDEST_FIRST 发射、Cache 和流水线资源的竞争和共享等 SMT 体系结构的基本特性. 该模拟器比较完备,已用于我们对多线程体系结构的研究当中. 在此基础上,本次实验添加了对 STALL、FLUSH 和 TSRR 策略的支持. 表 1 给出了模拟的 4 种策略及主要参数设置:4-线程 8-发射的 SMT 常见设置,RRF 尺寸(RRFsize)作为考察变量未列出,执行单元和 IQ 等硬件设置都较充裕,让 4 种策略充分发挥,得到公平的比较结果.

表 1 模拟的 4 种策略及主要参数设置

器件名称	ICOUNT	STALL	FLUSH	TSRR
	基本的 ICOUNT 策略, 监控 L2 D-Cache 和无改进	监控 L2 D-Cache 和 D-TLB miss, T10 触发 ^[3]	监控 L2 D-Cache 和 D-TLB miss, T10 触发 ^[3]	<i>global</i> = 4, <i>local</i> 调整间隔为 1M cycle
L1 I-Cache	128KB, 32-Byte block, 4-way 关联, 1-cycle 延迟, LRU 替换			
L1 D-Cache	128KB, 64-Byte block, 2-way 关联, 1-cycle 延迟, LRU 替换			
Unified L2 Cache	1MB, 64-Byte block, 4-way 关联, 10-cycle 延迟, LRU 替换			
Branch predictor	gshare (2K-entry), 32-entry RAS, 4-way 512-set BTB			
Main memory	Infinite capacity, 80-cycle 延迟			
I-TLB, D-TLB	4-way 512-entry I-TLB, 4-way 1024-entry D-TLB, 200-cycle miss 延迟, LRU 替换			
Fetch/Decode/Issue/Commit 宽度	8 inst/cycle, 4-context, ICOUNT1.8 取指, OLDEST_FIRST 发射, 256-entry IQ			
Out-of-order 执行单元	6 Int. ALU, 3 Int. Mul/Div, 6 FP ALU, 3 FP Mul/Div, 4 Read-port, 2 Write-port, 4×64-entry ROB, 4×32-entry LSQ			

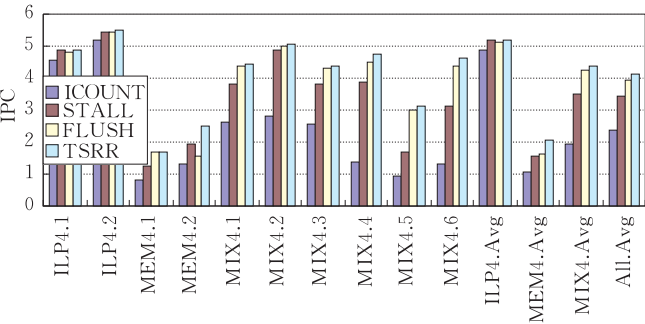
测试程序来自 SPEC 2000 并采用 Ref 输入,根据其运行特性可分为 ILP 类(并行度高且 D-Cache 访问少)和 MEM 类(D-Cache 访问多且命中率低). 我们选用 ILP 类和 MEM 类各 8 个程序,构造了 10 个类型各异的测试组合(test-suite),如表 2 所示,使评测结果更具代表性. 我们将 *RRFsize* 从 40 递增到 100 进行了一系列实验,评测 RRF 尺寸对各策略的影响. 每次运行时先略过(不统计性能)各线程的初始部分,然后开始性能模拟,运行 SimPoint^[13]生成的线程代表性片断,直到任一线程提交 100M 条指令为止.

表 2 4-线程测试组合(test-suite)

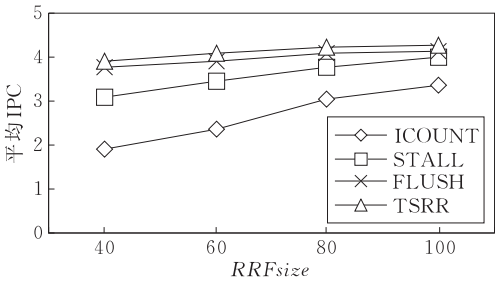
类别	代号	组成
ILP	ILP4.1	gzip_crafty_vortex_facerec
	ILP4.2	gcc_eon_mesa_sixtrack
MEM	MEM4.1	vpr_mcf_art_ampp
	MEM4.2	swim_applu_earthquake_lucas
MIX	MIX4.1 (3ILP+1MEM)	gzip_crafty_vortex_mcf
	MIX4.2 (3ILP+1MEM)	gcc_eon_mesa_earthquake
	MIX4.3 (2ILP+2MEM)	eon_facerec_vpr_swim
	MIX4.4 (2ILP+2MEM)	crafty_sixtrack_ampp_lucas
	MIX4.5 (1ILP+3MEM)	vortex_mcf_art_ampp
	MIX4.6 (1ILP+3MEM)	mesa_applu_earthquake_lucas

为了清晰,本节不仅给出了 $RRFsize=60$ 时各 test-suite 的详细结果(图 4~图 11 的(a)),还给出

了 $RRFsize$ 在 40~100 间递增时所有 test-suite 的平均结果(图 4~图 11 的(b)),以反映整体变化趋势.

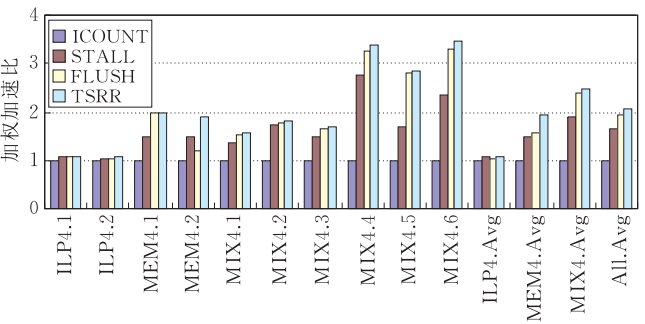


(a) $RRFsize=60$

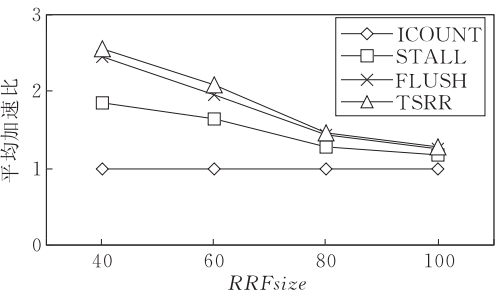


(b) $RRFsize=40\sim100$

图 4 IPC 对比(4-线程)

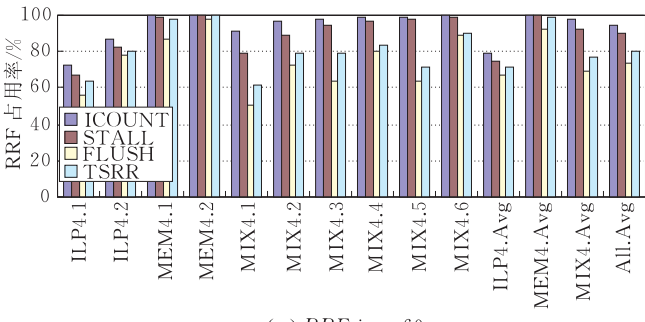


(a) $RRFsize=60$

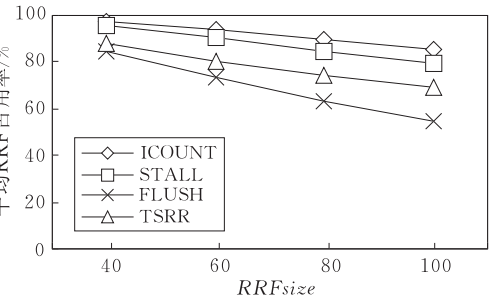


(b) $RRFsize=40\sim100$

图 5 加权加速比对比(4-线程)

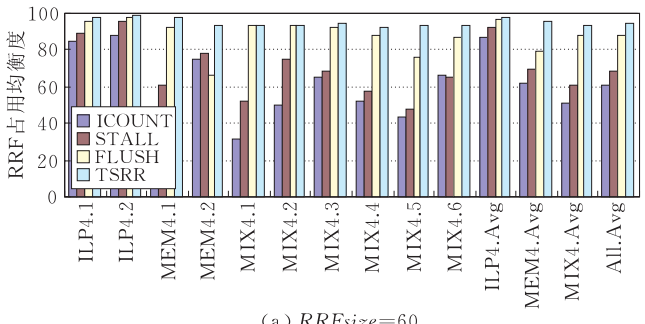


(a) $RRFsize=60$

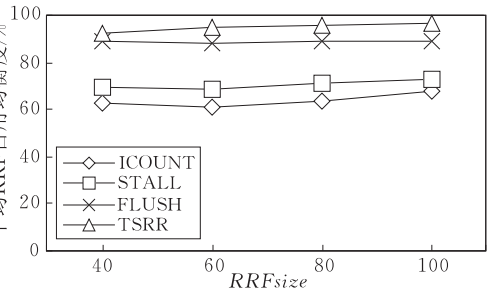


(b) $RRFsize=40\sim100$

图 6 RRF 占用率对比(4-线程)



(a) $RRFsize=60$



(b) $RRFsize=40\sim100$

图 7 RRF 占用均衡度对比(4-线程)

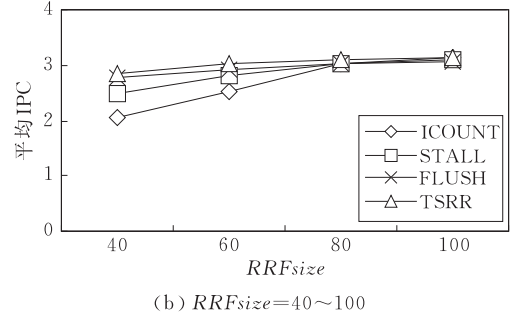
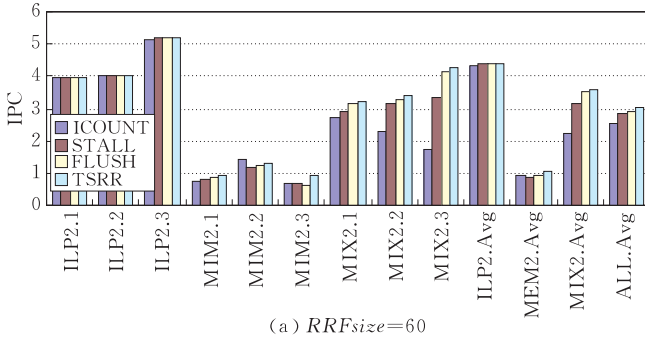


图 8 IPC 对比(2-线程)

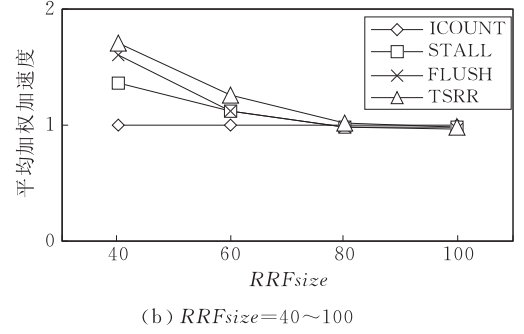
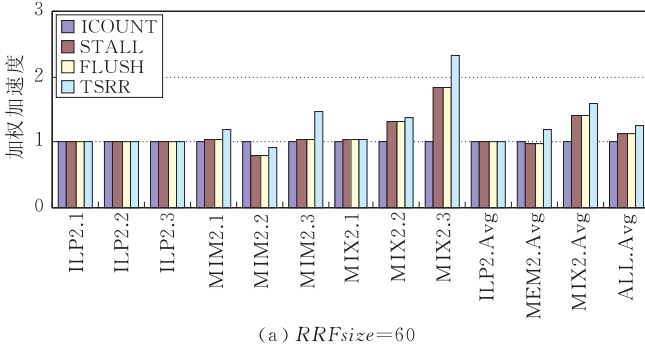


图 9 加权加速比对比(2-线程)

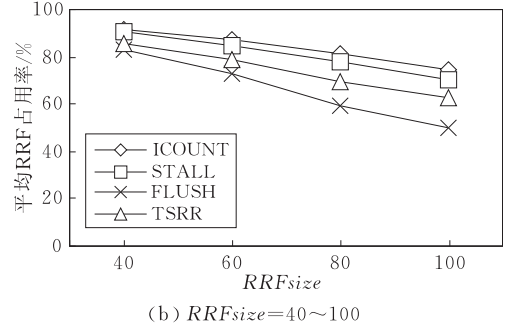
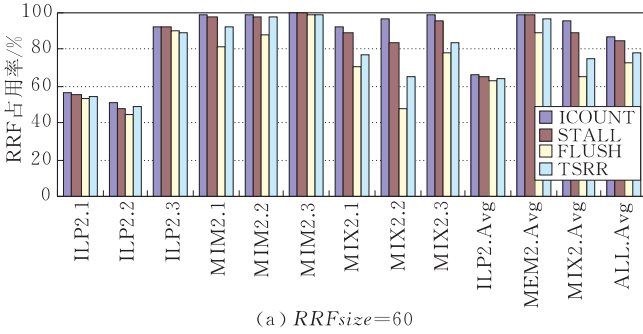


图 10 RRF 占用率对比(2-线程)

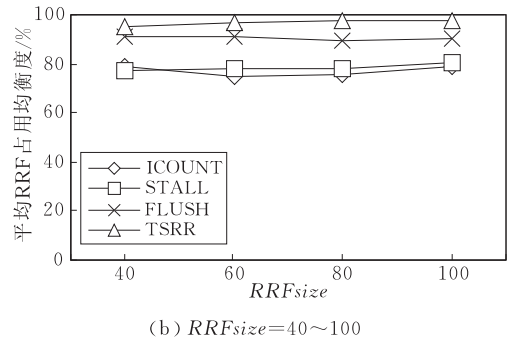
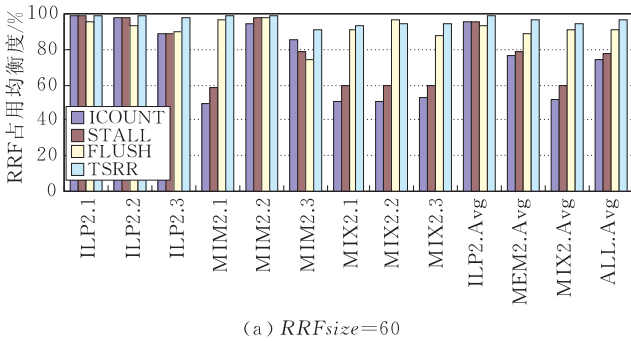


图 11 RRF 占用均衡度对比(2-线程)

3.1 性能对比

图 4 给出了不同策略的 IPC 对比. 为了更准确地评价性能, 图 5 给出了各策略相对 ICOUNT 的加权加速比, 即 $Weighted Speedup$ ^[3], 其定义如下:

$$Weighted Speedup = \frac{1}{N} \sum_{i=1}^N \frac{IPC_i^*}{IPC_i},$$

其中 IPC_i 和 IPC_i^* 分别是各线程在策略改进前后的 IPC. 该评价尺度避免了“只看 IPC 整体而忽略个

别”的偏差,是 SMT 不同策略间比较的重要尺度.从图 4 和图 5 可以看出:

(1) 对所有 test-suite, TSRR 的性能均高于其它策略,特别是显著高于 ICOUNT 和 STALL. TSRR 的性能比 FLUSH 只是略有提高,但其实现代价却远低于 FLUSH. 值得注意的是 ILP4.1 和 MEM4.2 的表现,二者的 FLUSH 性能均低于 STALL,经分析这是 FLUSH 中不恰当的指令丢弃引起的:前者是因为 ILP 类 test-suite 在资源充裕的时候丢弃指令是得不偿失的,后者是因为 MEM.2 中各线程的 L2-DCache-Miss 率都很高,导致大量指令被“中途丢弃”.这反映出 FLUSH 面对这两种情况时的不足,而 TSRR 能从容地适应各类线程组合.

(2) TSRR 对 MIX 类 test-suite 的性能提升最明显,对 ILP 类则作用较小.这是因为 MIX 线程组合最容易发生资源滥用而影响性能(例如 MIX4.1 与 ILP4.1 只是第 4 线程不同,但二者 ICOUNT 策略的 IPC 却相差很大,这是 MIX4.1 的第 4 线程资源滥用引起的,类似例子还有 MIX4.2 和 ILP4.2),而 TSRR 专门消除资源滥用.

(3) $RRFsize$ 从 40 递增至 100, TSRR 的性能始终高于其它策略.4 种策略的性能差距随 $RRFsize$ 的递增而减少,这是因为更充裕的资源(RRF)可供分配,降低了资源滥用造成的危害.

3.2 资源占用均衡度对比

如前所述,RRF 作为关键路径上的存储类资源,其分配成功与否决定着指令能否继续执行和对后继资源的占用.换句话说,RRF 的分配决定着整个处理器的资源分配.图 6 给出了不同策略下 RRF 的占用率对比,可以看出:

(1) 对所有 test-suite, TSRR 的资源占用率均低于 ICOUNT 和 STALL,但略高于 FLUSH.这是因为 FLUSH 触发时要清除部分指令并释放其占用的资源,有效减少了无效占用时间;然而其代价是被清除指令已完成的处理也要放弃,当 FLUSH 恢复时重新处理,因此浪费功耗.

(2) ILP 类 test-suite 的 RRF 占用率较低,说明其占用效率较高;相反 MEM 类的 RRF 占用率较高,说明其占用效率较低,有很多的无效占用;MIX 类介于二者之间,容易发生资源滥用.

(3) $RRFsize$ 从 40 递增至 100, TSRR 的资源占用率始终低于 ICOUNT 和 STALL,而略高于 FLUSH.

性能和资源占有率都只从一个侧面反映出资源

利用情况,只有将二者结合起来,并考虑到 SMT 线程间的相互影响,才能更准确地反映资源的实质利用效率,即资源的“按需分配”的程度.为此,我们定义一种新的评价尺度——资源占用均衡度($BalDeg$),式(3)以 RRF 为例说明 $BalDeg$ 的定义:

$$BalDeg = 1 - \frac{1}{2} \sum_{i=1}^N |IPCpct_i - RRFpct_i| \quad (3)$$

其中, $IPCpct_i$ 和 $RRFpct_i$ 分别是各线程的 IPC 和 RRF 所占比重,即相对于所有线程总和的百分比.从定义可以看出 $BalDeg$ 的范围是 0~1,值越大表示资源占用越均衡.图 7 给出了不同策略下的 RRF $BalDeg$ 对比:

(1) 对所有 test-suite, TSRR 的 RRF $BalDeg$ 均超过 90%,明显高于其它策略,其中 MEM 类和 MIX 类的提高显著.例如 $RRFsize = 60$ 时,TSRR 的 RRF $BalDeg$ 平均为 95%(最低 93.1%,最高 98.1%),这说明 TSRR 策略下资源占用的均衡度已经很高,基本消除了资源滥用和饥饿现象.

(2) $RRFsize$ 从 40 递增至 100, TSRR 始终高于其它策略.

(3) 对比图 6 和图 7 可以看出,虽然 TSRR 的资源占用率略高于 FLUSH,但其资源占用均衡度也明显高于 FLUSH.这说明与 FLUSH 相比,TSRR 更好地挖掘了资源潜力,达到更高效率的资源利用,图 4 和图 5 的性能对比证明了这一结论.

虽然这里只给出 RRF 的占用情况,但实验统计显示,各线程对其它资源(IQ、ROB、LSQ 等)的占用情况与此类似,因此可以认为 RRF 分配情况代表整个处理器资源的分配情况.综上,TSRR 通过调控 RRF 的分配基本克服了资源占用的“不均衡性”,实现了整个处理资源的按需分配;与以往策略(参见 1.2 节)相比,TSRR 具备明显的性能优势、实现优势.

3.3 local 调整间隔和 global 尺寸的影响

为了说明 local 调整间隔对 TSRR 的影响,我们将 local 调整间隔分别设为 0.1M 和 0.01M 个周期,保持其余设置不变,进行实验,发现整体性能变化很小(平均低于 1%).这说明 1M 周期量级的 local 调整就足以适应线程运行状态的变化,而且调整代价几乎可以忽略,因此没有必要进一步加大调整频率.如果 local 调整过频,不仅会增大调整代价,反而会因为过于敏感而无法准确反映线程状态变化,造成过犹不及的后果.

为了说明 global 尺寸对 TSRR 的影响,我们还

将 *global* 增大到 *RRFsize* 的十分之一,仍保持其余设置不变,进行实验,发现整体性能变化很小(平均低于 2%).这说明很小的 *global*(低于整体的十分之一)即可起到刺激竞争并防止饥饿的作用.如果进一步提高 *global* 比例,虽然可以进一步刺激竞争,但增大了资源滥用的机会;一个极端的例子就是将资源全部设为 *global*,即 TSRR 退化为 ICOUNT.

3.4 资源数量和线程数量的影响

对于给定的 *N*-context SMT 处理器,其硬件资源是一定的,通常设计能够满足 *N* 个线程同时运行即可. SMT 的特点是能够适应线程数量从 1~*N* 的变化,在 ILP 和 TLP 之间动态切换.因此,如果 TLP 小于 *N*(或资源数相对较多)时,每个线程得到更充裕的资源,线程间的资源竞争会减弱,恶性竞争就更少,各线程的 IPC 潜力更充分地发挥.实际上, TLP 小于 *N* 时, SMT 的性能更多地依赖于 ILP,包括 ICOUNT、STALL、FLUSH、TSRR 在内的各种资源调控机制的作用都将弱化.一个极端的例子在 SMT 处理器上运行单线程,此时 SMT 实际上已退化成传统的超标量处理器.

表 3 2-线程测试组合(test-suite)

类别	代号	组成
ILP	ILP2.1	gzip_crafty
	ILP2.2	eon_vortex
	ILP2.3	mesa_sixtrack
MEM	MEM2.1	vpr_mcf
	MEM2.2	swim_art
	MEM2.3	equake_lucas
MIX	MIX2.1	gzip_vpr
	MIX2.2	eon_swim
	MIX2.3	mesa_equake

为了说明 TSRR 策略在 TLP 小于 *N*(或资源数相对较多)时的表现,我们进行了 2-线程模拟实验,仍采用表 1 的配置,即在 4-context SMT 上运行 2-线程,表 3 给出所用的 2-线程组合.图 8~图 11 给出了 2-线程时各策略的 IPC、加权加速比、RRF 占用率、RRF 占用均衡度的对比情况:

(1)整体情况与 4-线程模拟实验(参见图 4~图 7)类似,TSRR 比其它策略有明显优势:IPC、加权加速比最高,RRF 占用率较低,RRF 占用均衡度最高;对 MIX 类 test-suite 的效果最好.

(2)随着 RRF 尺寸的增加,不同策略的性能差异逐渐减小,TSRR 的性能优势也逐渐消失.这是因为资源相对充裕,线程间相互竞争减弱,资源调控机制的作用也被弱化.

(3)与图 4~图 7 相比较,可以看出 2-线程时资源调控策略的弱化过程比 4-线程提前.这是因为线程数较少时,对资源的需求量也较少,更容易达到饱和.

4 结 论

传统的 SMT 资源分配是靠调控取指间接实现的(以 ICOUNT、BRCOUNT、MISSCOUNT 和 Round-Robin 为代表),这种隐式调控有时引发资源滥用和饥饿,严重影响整体性能.为了提高资源利用效率,前人提出了各种改进策略,主要包括隐式调控的改进(以 STALL、FLUSH、Data-Gating、ADTS 为代表)和显式控制(以 STATIC、DCRA 为代表).本文对这些策略进行了分析评述,发现分别存在如下问题:(1)无法兼顾资源分配的效率和公平,经常是顾此失彼;(2)控制死板,压制线程本身的性能潜力;(3)硬件代价较大,实现困难.因此,这些策略虽然有所改进,但仍无法消除资源滥用和饥饿现象.

本文分析发现“不均衡性”是影响资源分配效率的根本原因,也是以往策略难以克服的弱点,因此提出一种新的资源分配策略——基于线程感知寄存器重命名的资源分配(TSRR),并定义了一个衡量资源分配效率的尺度——资源占用均衡度(*BalDeg*).与以往策略相比,TSRR 的主要优点是:(1)资源分配自动适应线程运行状态的变化,真正实现“按需分配”;(2)通过调控 RRF 分配来间接控制其它资源分配,硬件实现复杂度较低;(3)兼顾资源分配的效率和公平,既杜绝了资源滥用和饥饿,又充分发掘各线程的性能潜力.此外,TSRR 的高效分配间接降低了 RRF 的尺寸要求和取指逻辑的复杂度.理论分析和模拟实验表明,TSRR 基本克服了资源分配的“不均衡”,其资源占用均衡度达到 95%左右,明显优于其它策略;TSRR 的性能优势也很明显.例如:RRF 尺寸为 60 时,ICOUNT、STALL、FLUSH 和 TSRR 的资源占用均衡度分别是 61%,69%,88%和 95%;STALL、FLUSH 和 TSRR 相对于 ICOUNT 的性能加速比分别是 1.64,1.95 和 2.08.

从分配方式的角度看,TSRR 是一种能“感知”线程运行需要的“隐显结合”的资源分配措施:(1)显式行为.直接划定各线程的 *local* 份额,消除线程间的恶性竞争,此外保留少量 *global* 份额供各线程竞争,弥补 *local* 划分的不足;(2)隐式行为.通过控制关键资源(RRF)的分配,间接控制其它资源分配;

(3)感知行为. 直接以性能监控作为反馈, 动态调整 *local* 划分以适应线程运行状态的变化.

总之, TSRR 克服了资源占用的“不均衡性”, 显著提高了 SMT 的资源分配效率和整体性能, 代表 SMT 资源分配策略向着“隐显结合”的方向发展. TSRR 对软件透明, 增加的硬件代价很小. 可以预见, 随着处理器与存储系统间速度差距的日益增大、以及线程级并行度的进一步增加, TSRR 在解决 SMT 处理器资源分配方面的优势将会更加明显.

参 考 文 献

[1] Tullsen D M, Eggers S J, Levy H M. Simultaneous multithreading: Maximizing on-chip parallelism//Proceedings of the 22nd Annual International Symposium on Computer Architecture. Santa Margherita Ligure, Italy, 1995: 392-403

[2] Tullsen D M, Eggers S J, Emert J S et al. Exploiting choice: Instruction Fetch and issue on an implementable simultaneous multithreading processor//Proceedings of the 23rd Annual International Symposium on Computer Architecture. Philadelphia, USA, 1996: 191-202

[3] Tullsen D M, Brown J A. Handling long-latency loads in a simultaneous multithreading processor//Proceedings of the 34th Annual International Symposium on Microarchitecture. Austin, USA, 2001: 318-327

[4] Robatmili B, Yazdani N, Sardashti S et al. Thread-sensitive instruction issue for SMT processors. IEEE Computer Architecture Letters, 2004, 3(1): 5

[5] Raasch S E, Reinhardt S K. The impact of resource partitioning on SMT processors //Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques. New Orleans, USA, 2003: 15-25

[6] Cazorla F J, Fernandez E, Ramirez A et al. Improving memory latency aware fetch policies for SMT processors//Proceedings of the 5th International Symposium on High Performance Computing. Orlando, USA, 2003: 70-85

[7] Cazorla F J, Ramirez A, Valero M et al. DCache warn: An I-Fetch policy to increase SMT efficiency//Proceedings of the 18th International Parallel and Distributed Processing Symposium. Santa Fe, USA, 2004: 1037-1046

[8] El-Moursy A, Albonesi D H. Front-end policies for improved issue efficiency in SMT processors//Proceedings of the 9th International Symposium on High-Performance Computer Architecture. Anaheim, USA, 2003: 31-40

[9] Falc3n A, Ramirez A, Valero M. A low-complexity, high-performance fetch unit for simultaneous multithreading processors//Proceedings of the 10th International Symposium on High Performance Computer Architecture. Madrid, Spain, 2004: 244-253

[10] Luo K, Gummaraju J, Franklin M. Balancing throughput and fairness in SMT processors//Proceedings of the International Symposium on Performance Analysis of Systems and Software. Tucson, USA, 2001: 164-171

[11] Shin C, Lee S W. Dynamic scheduling issues in SMT architectures//Proceedings of the 17th International Parallel and Distributed Processing Symposium. Nice, France, 2003: 77. 2

[12] Cazorla F J, Ramirez A, Valero M et al. Dynamically controlled resource allocation in SMT processors//Proceedings of the 37th International Symposium on Microarchitecture. Portland, USA, 2004: 171-182

[13] Perelman E, Hamerly G, Biesbrouck M V et al. Using SimPoint for accurate and efficient simulation. ACM SIGMETRICS Performance Evaluation Review, 2003, 31(1): 318-319

[14] Chandra D, Guo F, Kim S et al. Predicting inter-thread cache contention on a chip multi-processor architecture//Proceedings of the 11th International Symposium on High-Performance Computer Architecture. San Francisco, 2005: 340-351

[15] Yang Hua, Cui Gang, Liu Hong-Wei, Yang Xiao-Zong. Multi-usable rename register with 2-level renaming and allocating. Chinese Journal of Computers, 2006, 29(10): 1729-1739(in Chinese)
(杨华, 崔刚, 刘宏伟, 杨孝宗. 两级分配多可用重命名寄存器. 计算机学报, 2006, 29(10): 1729-1739)

[16] Sprunt B. Pentium 4 performance monitoring features. IEEE Micro, 2002, 22(4): 72-82

[17] Austin T, Larson E, Ernst D. SimpleScalar: An infrastructure for computer system modeling. IEEE Computer, 2002, 35(2): 59-67

附表 SMT 资源分配策略的比较和发展

分配策略	隐式/显式	侧重点(优点)	制约因素(缺点)	实现代价	负载相关(适用性)	总吞吐率	资源占用均衡度
STATIC	显	消除竞争	资源浪费严重	较低	各线程的运行特性类似时	较低	较差
ROUND-ROBIN	隐	机会平均	线程行为的差异性	很低	ILP 类负载, 各线程较均衡	较低	较差
BR-COUNT	隐	消除误预测分支的影响	片面, 只注重分支	较低	负载中有分支较多的线程	一般	较差

(续 表)

分配策略	隐式/显式	侧重点(优点)	制约因素(缺点)	实现代价	负载相关(适用性)	总吞吐率	资源占用均衡度
MISS-COUNT	隐	消除 D-Cache 失效的长时间等待	片面,只注重 D-Cache	较低	Mem 类或 Mix 类负载	一般	较差
ICOUNT	隐	平衡各线程的前端指令数量	长延迟指令的独占	较低	负载中长延迟指令较少时	较高	一般
STALL	隐	防止产生新的无效占用	触发滞后	一般	负载中长延迟指令较少时	较高	一般
FLUSH	隐	在 STALL 基础上,消除已有无效占用	触发滞后,指令中弃	较高	负载中长延迟指令较少时	很高	较好
DATA-GATING	隐	让 STALL 触发更及时	触发过于敏感	一般	负载中 D-Cache 性能较差时	较高	一般
ADTS	隐	根据监测,在多种策略间切换	代价大;需额外 DT 线程	很高	各类负载,资源很充裕时	较高	较好
DCRA	显	防饥饿	代价大;就低不就高	很高	各类负载,资源很充裕时	较高	较好
TSRR	隐显结合	实现按需分配	—	较低	各类负载	很高	很好



YANG Hua, born in 1974, Ph. D. , associate professor. His research interests mainly focus on chip multithreading architecture, fault-tolerant processor architecture, and dependable computing.

His current research interests include fault-tolerant computing, and high-performance computer architecture.

LIU Hong-Wei, born in 1971, Ph. D. , associate professor. His current research interests include reliable computing, and mobile computing.

YANG Xiao-Zong, born in 1939, professor, Ph. D. supervisor. His current research interests include fault-tolerant computing, and wearable computing.

CUI Gang, born in 1947, professor, Ph. D. supervisor.

Background

SMT plays a staple role in the forthcoming chip multithreading era, and the way how resources are allocated in the SMT processors consequently has a great impact on the overall performance. Having made an extensive survey and in-depth analysis, the authors find the imbalance of resource allocation is the fatal reason why resource underutilization and performance depression present themselves in various improving techniques. Accordingly, the authors propose a cost-effective scheme named TSRR, reaching allocation-on-demand and concerning both effectiveness and fairness.

This work is supported by the 10th 5-Year Pre-Research Project under grant No. 41316. 1. 2, the National Natural Science Foundation of China under grant No. 60503015, and the Ph. D. Supporting Foundation from the Ministry of Education of China under grant No. 20020213017. An important objective of the projects is to research and develop high-performance and reliable computer systems, which ultimately rely on high-performance and reliable processor architectures.