缓解同时多线程结构中线程对关键资源的竞争

印 杰 江建慧
(同济大学计算机科学与技术系 上海 201804)

摘 要 同时多线程处理器同时执行来自不同线程的指令,兼顾了线程内和线程间的指令并行,使处理器的性能得以大幅提升。然而这种对资源的共享方式,可能带来对关键资源(包括重命名寄存器、指令队列等)的恶性竞争,从而出现"饿死"现象,甚至影响处理器的吞吐率。这主要是由于某些线程遇到长延迟指令,并长期占据关键资源,从而导致其他线程对资源的需求无法得到满足,同时这也降低了资源的利用率。降低竞争带来的负面影响,主要有3种方法:线程调度——在取指段,决定从哪些线程取指令;指令调度——决定哪些指令进入关键资源;关键资源划分——为每个线程分配独立的关键资源。主要对这些调度策略进行综述。

关键词 同时多线程,线程调度,指令调度,资源划分

Easing the Competition for Key Resource among Threads in SMT

YIN Jie JIANG Jian-hui
(Department of Computer Science and Technology, Tongji University, Shanghai 201804, China)

Abstract Simultaneous Multithreading Processors boost performance by executing instructions from different threads simultaneously, which explore both inter-thread and intra-thread parallelism. Sharing critical resources (including rename register file, instruction queue and so on) among different threads may also bring vicious competition, which may result in starvation, even degrade the performance. This is mainly due to long delays encountered by some thread, and the threads take a lot of key resources for long time, while the demand of the other threads for key resources cannot be met. This may reduce the utilization of resources. There are three methods to reduce the negative impact of competition: thread scheduling decides which threads to fetch instructions from in the fetch stage; instruction scheduling determine which instructions to enter the key resource in the dispatch stage; Partitions of the key resources allocate the key resources among threads. These scheduling strategies were reviewed in the paper.

Keywords SMT, Thread scheduling, Instruction scheduling, Resources partitions

1 引言

过去的十年,处理器的性能得到了急剧提升。一方面由于工艺的进步,使得片上能集成更多的晶体管,芯片能达到更高的频率。另一方面由于体系结构的发展,乱序执行、分支预测、值预测等技术对处理器性能的提升起了很大的加速作用。同时多线程(SMT:Simultaneous Multithreading)就是这样一种技术。

同时多线程结构能够同时执行来自不同线程的多条指令,通过挖掘单线程内部的指令级并行和线程间的线程级并行来降低水平浪费和垂直浪费。该结构是所有体系结构中资源利用率最高的。Intel,HP等商家对此进行了大量研究,并纷纷推出了相应的产品[1,2]。

同时多线程结构结合了超标量结构多发射的能力和传统的多线程结构隐藏延迟的能力。不同的是,传统的多线程结构[3-6]通过上下文的快速切换来共享处理器的执行资源,而同时多线程结构使多个上下文保持同时活跃来竞争执行资源。这种动态的共享方式既解决了长延迟问题,又解决了单线程的并行度有限的问题。

同时多线程结构一方面通过线程对资源的共享来提高资源利用率。另一方面,线程间对资源的竞争亦可能导致处理器性能的下降(图1[7]表明当线程数目为8时处理器的性能反而不如线程数目为6时)。主要原因在于当某个线程遇到Cache缺失或者分支预测错误时,该线程后续指令所占的指令队列空间会被浪费。这种情况下,会导致其他线程无法得到足够的资源(出现"饿死"现象),并且会降低资源的利用率,使得处理器性能下降。

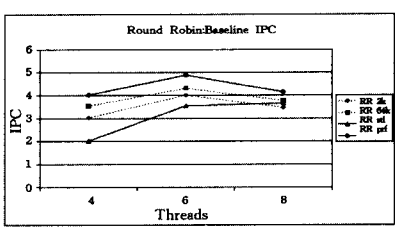

图1 SMT处理器性能 Vs. 线程数目

到稿日期:2009-08-28 返修日期:2009-12-01 本文受国家科技部"九七三"计划项目(2005CB321604)资助。

印 杰(1981-),男,博士生,CCF会员,研究方向为容错计算、处理器体系结构,E-mail:jieyin2008@gmail.com;江建慧(1964-),男,博士生导师,研究方向为容错计算、软件可靠性工程、处理器体系结构、计算机辅助设计/测试/评估。

降低这种竞争带来的负面影响，是处理器体系结构研究者所奋斗的目标。从1995年同时多线程概念被提出以来[8]，短短十四五年，学术界对此进行了大量研究。所有方法可以分为3类：线程调度——在取指段，决定从哪些线程取指令；指令调度——决定哪些指令进入关键资源；关键资源划分——为每个线程分配独立的关键资源。本文按照这3种方法进行组织。图2为各种技术的详细分类。

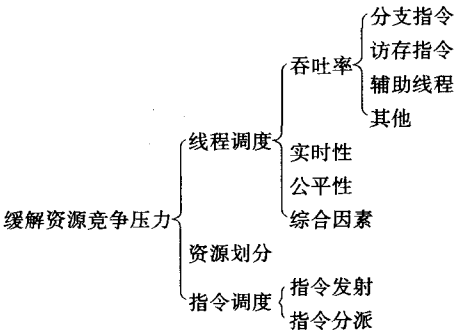

图2 各种缓解资源竞争压力技术分类

2 线程调度

同时多线程结构，在取指段可以从多个线程进行取指。线程调度指在处理器的取指段，通过选择从哪些线程取指来控制线程对资源的共享。被选中的线程，其指令会在后面的一段时间内占据部分关键资源。针对不同的目标，提出了不同的线程调度策略。有的策略偏重于处理器的吞吐率，有的偏重于线程间的公平性，有的则用于满足某些线程的实时性，也有策略综合考虑这些目标。该部分根据这些目标进行组织。

2.1 吞吐率

ICOUNT[9]是第一个被提出来提高处理器吞吐率的线程调度策略，也是使用最为广泛的一个，该策略根据流水线前端各个线程指令数目的多少，对线程进行取指优先级排序。数目较少的赋予较高的优先级。在流水线前端，指令最少的那个线程往往也是在流水线中流动速度最快的那个，因此对资源的利用率更高，所以给予更高的取指优先级。

但是ICOUNT并不能完全避免竞争带来的负面影响。长延迟指令及其后续指令仍有可能进入流水线。因此提出了许多改进的调度策略，这些策略一般从如下几个方面着手：减少进入错误路径的指令比例，即根据线程遇到的分支指令进行线程调度；避免遇到长延迟的线程长时间占据关键资源，一般针对线程的访存指令遇到cache缺失，通过辅助线程来加快线程的执行。

2.1.1 分支指令

分支指令引起的控制相关是影响处理器性能的一个关键因素。最初的设计中，遇到分支指令，流水线往往会停顿下来，等待分支指令结果。后来，随着分支预测器的出现，控制相关导致的性能下降有所减缓。但是，分支误预测带来的性能开销不能忽略。同时多线程结构给了分支指令问题另外的解决途径。图3表明，当线程数目增加时[7]，由于分支误预测而进入错误路径的指令比例随之减少。但是分支误预测依旧存在，如何通过线程的调度来进一步减小进入错误路径指令的比例得到了广泛的研究。

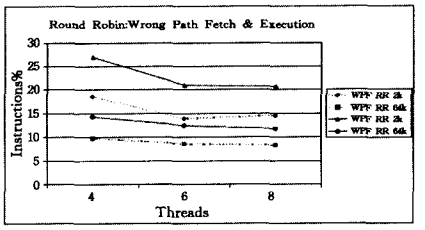

图3 进入错误路径指令比例 Vs. 线程数目

文献[9]提出了BRCOUNT策略。该方法赋予可能出现错误路径的线程较低的优先级。作者认为，译码阶段、重命名阶段和指令队列中分支指令越少，该线程出现分支误预测的可能性越低。因此，可以通过统计流水线前端各个线程所包含的分支指令数目来对线程进行取指优先级排序。

BRCOUNT策略在统计分支指令数目的过程中，将所有分支指令同等对待。实现方法比较简单，但是效果并不明显。文献[7]提出了agstall策略，该策略基于程序执行过程中的一个重要特征：大约60%的动态分支指令，都有较强的偏转倾向，跳转或不跳转的概率在95%以上[10]。该策略的主要思想是将分支指令分为偏转倾向明显和不明显两类，偏转倾向明显的，则易于预测。对于不易预测的分支指令所在线程，则停止取指，直到计算出分支结果。试验表明，该方法减少了86%的分支误预测。但是，该方法中，当遇到低可信分支即停止该线程取指的话，在线程较少的时候，会由于缺乏足够可并行指令而导致资源利用不充分。

与agstall思想类似，文献[11,12]提出了SAFE-T调度策略。首先，通过ICOUNT对各个线程进行优先级排序，然后统计流水线中每个线程各包含多少低可信条件分支。当该数目超过一定阈值时，该线程被停止取指。Agstall本质上即为该阈值设置成1的情况。模拟实验结果表明，执行的错误指令减少了41.6%，性能提高了14.5%。

在同时多线程结构中，为了应付分支指令带来的性能开销，一方面通过选择合适的线程进行取指，另一方面，可以通过对分支预测器进行改进来提高性能。文献[13]研究发现同时多线程结构中，分支预测器访问延迟和访问端口数量对处理器的性能有着重要的作用，作者提出了基于分支预测器流水化的思想来降低访问延迟对性能的影响。为便于看清各种策略的异同，表1罗列了各种策略的核心思想。

表1 三种策略的核心思想

调度策略	策略描述
BRCOUNT	统计分支指令数目，并以此确定线程取指优先级
Agstall	停止从遇到低可信分支的线程取指
SAFE-T	低可信分支数目超过阈值，则停止从该线程取指

2.1.2 访存指令

在同时多线程结构中，访存指令cache缺失带来的长延迟是影响处理器性能的另一个关键因素。在粗粒度多线程处理器中，当某个线程遇到cache缺失时，则将该线程彻底地从流水线中切换出去。但是在SMT中，当某个线程遇到cache缺失时，该线程一方面已经占用了很多宝贵的资源(这些资源本可以被其他线程利用)，另一方面，有可能还在继续往流水线里面取指令。

文献[14]认为，一个cache命中率很低的线程对同时运行的另一个命中率较高的线程有很大的抑制作用。主要有两

方面的原因：一是前者可能将后者位于 cache 中的数据替换出去，从而降低后者的 cache 命中率；二是前者可能会占用一些本可以被分配给后者的关键资源。研究发现，当流水线中至少有一个 L2 cache 缺失时，指令队列的占有率为 97%，而其他时候则为 62%。

因此，对于那些 cache 命中率比较差的线程，有必要通过控制其取指来避免其在流水线中浪费一些资源。在很多情况下，释放那些遇到 cache 缺失的线程所占用的资源比保留其资源以等待访存值所带来的效果要好些[14]。

最初意识到这个问题，并提出解决之道的是 SMT 结构的提出者——Deam M. Tullsen[9]。作者提出了 MISCOUNT 策略，即通过计算各个线程所遇到的 cache 缺失的数量，赋予 cache 缺失数量较少的线程较高的优先级。

但是，指令到了执行段时，才能发现是否 cache 缺失，这个时候该线程的后续指令可能已经被取进来，一定程度上阻塞了相关资源。因此，文献[15]借鉴 Alpha 21264 中的 load 预测机制，提出了 PDG 调度策略。在流水线前端就预测该 load 指令是否会发生 cache 缺失，并依此进行线程调度。当线程的 cache 缺失超过一定数目时，则停止该线程取指。但是该方法中，发生多少 cache 缺失时停顿线程，是一个尚需深究的问题。

文献[15]中，作者同时还提出了 DG 策略，不同的是，PDG 是在流水线前端预测访存指令是否会发生 cache 缺失，然后采取相应措施，而 DG 策略是在等 cache 缺失发生了才采取措施。

文献[16]比较具体研究了 cache 缺失对处理器的影响。作者研究了两个问题：一是什么时候判断 cache 缺失(DM：Detection Moment)。二是在 cache 缺失的时候，采取何种措施(RA：Response Action)。

什么时候判断 cache 缺失，作者提出了两种方法：一是等确实发生 L2 cache 缺失时，进行断定；二是，当该指令在指令队列中超过一定时钟周期，则认定其发生 cache 缺失。

对于判断出 cache 缺失的线程，作者提出了两种应对措施：一种是停止从该线程取指(即 STALL 策略)，直到 cache 缺失解决；另一种是冲刷其在流水线中的相关指令(即 FLUSH 策略)。或者是二者结合起来，首先进行停顿，当发现某个资源被耗尽时，则进行冲刷。

STALL 策略往往会导致大量指令被浪费，而 FLUSH 策略中，如果每个 L1 cache 缺失都停顿，在线程数目较少时，就会缺乏足够的并行性。因此，文献[16]提出了一个比较适中的策略——Dwarn：首先，对线程按照线程有无 L1 cache 缺失进行分类，取指时首先从正常的一组中进行取指，当正常组中线程不够时，则从另一组中取指。这既避免了那种一旦遇到 L1 cache 缺失就停顿的严厉，又避免了直到 L2 cache 缺失才有所动作的迟钝。

Dwarn 策略中，另一组的线程有可能遇到 L2 cache 缺失，也可能不遇到。由于仅仅降低了其取指优先级，因此遇到 L2 cache 缺失的那个线程仍然可能进入流水线，导致性能下降。文献[17]对 Dwarn 进行了改进，提出了 Dwarn+策略，对于遇到 L1 cache 缺失的线程，按照是否发生 L2 cache 缺失，进行了进一步的分组，并禁止发生 L2 cache 缺失的那个线程进入流水线，直到其 cache 缺失解决。

借鉴文献[16]的分类方法，即按照 DM 和 RA 对各种调度策略进行分类，如表 2 所列。

表 2　各种调度策略分类

DMRA	取指段	L1 cache 缺失	指令发射 X 周期后	L2 cache 缺失
停止取指	PDG	DG	STALL	
冲刷流水线			FLUSH	
降低优先级		MISCOUNT/ Dwarn/Dwarn+		Dwarn+

所有策略中，MISCOUNT，Dwarn 和 Dwarn+都是在检测到 L1 cache 缺失之后采取调整线程取指优先级的方法。不同的是，MISCOUNT 按照 L1 cache 缺失数目对线程进行取指优先级排序，而 Dwarn，Dwarn+则按照有无发生 L1 cache 缺失对线程进行分组，组内线程则采用 ICOUNT 策略，较于 Dwarn，Dwarn+则更进一步，将发生 L1 cache 缺失的那一组按照是否发生 L2 cache 缺失进一步分组。

前面所提调度策略，均通过控制线程调度来解决访存指令带来的长延迟。除此之外，亦有研究通过其他方法来解决此类问题。文献[18]通过研究每个负载的 cache 命中率随着 cache 大小而变化的函数，来确定在给定 cache 大小情况下同时执行的负载该如何组合。

文献[19]在系统资源空闲期间对某些线程的某些部分进行提前执行(在不影响其他线程正常执行的情况下)，以此来提前解决一些该线程可能遇到的 cache 缺失带来的线程停顿。

前面的研究中，当遇到 cache 缺失时，或停顿或冲刷该线程，这就导致每个 cache 缺失的指令都是串行化执行的。而实际情况中，很多 cache 缺失的 load 指令是靠在一起的。在乱序执行的处理器中，靠在一起的 load 指令是可以并行进行的，这样就能隐藏一部分 load 带来的 cache 缺失的开销。文献[20]充分利用了该特点，对多线程进行调度。当检测到或者预测到某个指令为长延迟 load 指令(可能发生 cache 缺失)时，预测 n 条指令中可并行的长延迟 load 指令数目，如果为 0，则停止取指或者冲刷该流水线，如果不为 0，则取出相应的指令进行并行执行。

2.1.3　辅助线程

在众多的线程调度算法中，每种算法都有一定的适用范围。不同的调度算法，对于不同的线程组合，以及运行的线程总数，都有着不同的效果。因此，有研究者试图去研究一种调度算法，适用于任何范围。文献[21]提出了 ADTS 策略，通过运行一个特殊线程来动态地收集每个活跃线程的运行特征，并以此为根据，采用启发式算法，动态地调整接下来的线程调度算法。

2.1.4　其他

在所有的资源中，指令队列处于非常关键的地位。指令队列越长，能找到的可并行指令就会越多。为了充分利用指令队列，文献[15]提出了 UCG 策略和 FP_UCG 策略(分别针对整型队列和浮点队列)，即根据每个线程在指令队列中未就绪指令的数量进行线程调度。当线程在指令队列中未就绪指令数量超过一定阈值时，则停止从该线程取指。这就避免了某些线程大量占用指令队列，从而使处理器性能下降。

2.2　实时性

同时多线程结构作为一种已经得到广泛应用的体系结

构，不光要关注处理器的吞吐率，还需要关注某些线程的实时性需求，尤其是在交互式系统中。

文献[22]最先提出了一个较简单的方法来保证实时性。对于那些有实时要求的线程，始终保持其最高的取值优先级。其余线程按照 ICOUNT 或者 RR 进行取指。

文献[22]仅仅能保证一个线程的实时性能。文献[23]给几个有实时需求的线程都分配了较高的优先级，从而保证了几个线程的实时性能。

文献[24]通过实时监控指定线程的性能，并将其与期望的性能进行比较，以此作为依据来决定取指数量。取指顺序则由每个线程事先指定的优先级来决定。实验表明，该方案能较好地控制指定线程的性能。

在多个线程共享 cache 时，有实时要求的线程非常有可能被其他线程将数据从 cache 中替换出去，从而无法保证该线程的实时性能。文献[25]通过寄存器来保存实时线程所占据的 cache 行号，并保证其他线程无法将其替换出 cache，从而有效保证其性能。

由于程序执行中，小循环往往占据着较长的时间。因此，文献[25]提出了通过两个寄存器来保存这些小循环在 cache 中的位置，并保证其他线程无法将其替换出去。实验表明，该方法不光对实时线程的性能有所提高，对处理器的整体吞吐率也有提高。

为了保证 SMT 中某个特殊线程的性能，文献[26]提出了帮助线程(Helper Thread)的方法。通过对有实时需求的线程提取出帮助线程，该帮助线程能提前执行，从而提前解决 cache 缺失等问题。

2.3 公平性

在多用户系统中，公平性是一个重要的衡量指标。在操作系统层面，通过时间片的方法能保证一定的公平性。从体系结构的角度来看，保证每个线程都有均等的机会进入流水线，是保证公平性的重要方法。本质上讲，RR[9]就是这样一种方法，RR 通过轮流选择线程取指，从而保证一定的公平性。

文献[22]中，公平性的保证，是通过按照时间片方法，轮流保持每个线程最高的取值优先级，其余线程则是按照 ICOUNT 或者 RR 进行取值。但是，该方法使处理器的吞吐率明显下降。

2.4 综合因素

前面提到的调度策略，一般只考虑其中的一种因素，解决一类问题。或者为了减少错误执行的指令数目，或者降低长延迟的影响。而在处理器的设计中，有时需要兼顾各种因素，比如吞吐率和公平性等。这种情况下，往往采用一种分层考虑的方法来进行线程调度。

文献[27]提出了一些组合调度策略，比如 RR 调度策略；该策略在某个线程资源占用率超过一定值时，停止其取值，以求降低竞争。在 ICOUNT 策略中，加入对分支指令可信度的判断，进一步对线程的优先级进行细分，减少程序进入错误路径的概率。

文献[28,29]综合考虑了线程所包含的低可信度分支指令数目和 L2 cache 缺失两种因素，对所有的线程进行优先级排序，并根据优先级顺序进行相应的取值。

文献[30]首先根据每个线程在 IQ 中的数量来确定线程的优先级。选取数量最少的两个线程进行取指。优先级最高的那个线程的取指数目根据该线程的瞬时 IPC 和该线程在 IQ 中指令的数目来确定。剩余的取值带宽由另一个选中的线程利用。实验表明，该策略较好地解决了指令带宽的分配，因此较于 ICOUNT 策略，性能有了较大的提高。

3 指令调度

线程调度算法解决的是选择哪些线程取指问题。当这些指令进入流水线后，则面临另一个问题，即选择哪些指令进入指令队列；当指令队列中有多个指令就绪时，选择哪些指令发射到执行单元。

3.1 指令发射

在超标量结构中，选择哪些就绪指令发射到处理单元，往往采用先来先发射的策略。在同时多线程处理器中，由于多个线程的同时存在，并且每个线程对资源的需求往往不一样，每个线程对处理器总的吞吐率的贡献大小也不一。因此，选择哪些指令进行发射亦是一个需要深究的问题。

文献[9]作为最早研究该问题的文献，提出了一些发射策略：OPT_LAST 和 BRANCH_FIRST 策略。对于投机指令，赋予较低的优先级。对于分支指令，赋予较高的优先级。但是，实验表明，这些策略并没有取得比先来先发射策略更好的效果。

3.2 指令分派

线程调度往往有一定的延迟性。当检测到线程遇到长延迟指令时，该线程的后续指令可能已经进入了指令队列。较于线程调度策略，通过选择哪些指令进入指令队列，对资源竞争的控制更直接。当检测到长延迟时，可以直接控制该线程进入指令队列的指令数目。

文献[32]提出了 2OP_BLOCK 指令分派算法。实验发现，当指令分派到指令队列时，那些两个源操作数均未准备好的指令在指令队列中往往占据很长时间，降低了指令队列的可用率。因此，作者提出当指令两个源操作数均未就绪时，停止该线程，直到该指令至少有一个操作数就绪，方将该指令分派到指令队列，降低了引起指令队列堵塞的风险。

在 2OP_BLOCK 中，当所有线程同时遇到两个操作数未就绪时，则会让处理器停顿下来，尤其是在负载较少的情况下。该方法的本质是通过牺牲单个线程的 ILP 来开发多个线程间的 TLP。当负载足够时，能取得较好的效果，但是，当负载数目不够的时候，则会对结果有所影响。文献[33]对 2OP_BLOCK 方法进行了改进，当某条指令遭遇两个源操作数均未就绪时，该线程并未停顿。其后续的指令，当至少有一个源操作数就绪的时候，依然可以分派到指令队列中。该方法是通过乱序分派的方法来减轻 2OP_BLOCK 带来的 ILP 受到的抑制。

4 资源划分

同时多线程结构为了提高性能，往往需要尽量提高线程间对资源的共享。但是，这也往往会增加处理器的控制复杂度‘延迟，从而降低处理器的频率。学术界研究通过更多的动态共享来取得更高的吞吐率。但是商用处理器中，共享程度并没有这么高(the Intel Xeon[1] and Compaq EV8[2])。商用处理器中，取指队列、ROB 等更多地采用每个线程私有的

方式。

共享即意味着竞争。前面的算法是通过线程调度或者指令调度来避免竞争带来的饥饿现象和性能急剧下降。也有研究通过对资源的划分来降低竞争的负面影响。较于线程调度方式，资源静态划分方法能有效地降低硬件的复杂度，并减少延迟。文献[34]对同时多线程中的资源划分进行了较充分的研究，作者将共享资源分为存储类资源和带宽类资源。存储类资源包括取指队列、指令队列、再定序缓冲、存储队列等。带宽类资源包括取指带宽、发射带宽、分派带宽等。

研究发现，静态分配存储类资源对性能的提高较明显，而对带宽类资影响则相对较小。这主要是因为存储类资源由于某些指令会占据较长时间，从而更容易导致饥饿现象。而带宽类资源每个周期都重新分配，因此静态分配并不能减少饥饿现象，反而会降低线程间的共享。作者进一步研究发现，在RR策略中，如果采用静态划分的方法，所带来的效果接近于ICOUNT策略。

静态分配分为3种类型：每个线程分得$1/n$的资源(共n个线程)；每个线程都可以获得100%的资源，也就是没有划分，是一种完全共享的情况；最后一种介于二者之间，每个线程有单独的保留资源，还有一块公共资源供所有线程进行竞争。

文献[31]中，每个时钟周期根据指令队列中每个线程所包含的就绪指令的数目来确定每个线程指令发射数目，从而对发射带宽进行有效划分。每个线程内部则采用先来先发射策略。实验表明，该策略较先来先发射策略，在性能上取得了30%左右的提高。

每个线程在不同时期对不同资源的需求往往是不同的。文献[35]将这些因素进一步考虑到资源划分中，提出了资源动态划分策略DCRA。每个时钟周期，对于不同的资源，根据线程对这些资源的需求进行划分。这就避免了静态划分带来的资源没有被充分利用，同时也控制了某个线程对资源的独占。实验表明，该方法无论在吞吐率，还是公平性上，较主流的线程调度方法和资源静态划分方法，均有明显提高。

文献[33]对资源的划分是基于线程对该资源的需要，比如对ROB的划分，仅仅基于每个线程对ROB的需要，但是，处理器的性能是各种资源竞争相互作用的结果。文献[36]基于处理器的IPC进行更进一步的资源划分(Hill-Climbing)。处理器每隔一个时间段，对资源分配方案进行调整，并统计该时间段的IPC，最后选择IPC最高的那个时间段对应的分配方案。并在后续的时间段中，采用该资源分配方案。作者的实验表明，该方法较DCRA方法，处理器的性能有明显提高。

文献[37,38]对重命名寄存器进行了动态的划分，根据每个线程的瞬时IPC来确定每个线程所能分配的重命名寄存器数目(TSRR)。对于剩余的重命名寄存器，则在几个线程间进行共享。实验表明，该方法有效地降低了某些线程长时间占据重命名寄存器带来的饥饿现象和处理器性能的急剧下降。

结束语 随着同时多线程处理器的普及，如何有效地降低线程间竞争带来的负面影响，将会受到持续的关注。线程调度策略得到的研究最为充分。近年来，指令调度和资源划分方法也开始得到大量研究。3种方法所达到的IPC远远低于处理器的理论峰值。处理器的性能仍然有提升的空间。如何通过线程调度、指令调度和资源划分等方法来进一步提高同时多线程处理器的性能，将会成为一个持续的研究热点。

参考文献

[1] Marr D T, Binns F, Hill D L, et al. Hyper-Threading Technology Architecture and Microarchitecture [J]. Intel Technology Journal, 2002, 6(1): 4-15

[2] Preston R P, et al. Design of an 8-Wide Superscalar RISC Microprocessor with Simultaneous Multithreading[C]// Proc. of IEEE International Solid-State Circuits Conference. San Francisco, USA, February 2002: 334-335

[3] Agarwal A, Lim B H, Kranz D, et al. APRIL: A Processor Architecture for Multiprocessing[C]// 17th Annual International Symposium on Computer Architecture. Seattle, WA, USA, June 1990: 104-114

[4] Alverson R, Callahan D, Cummings D, et al. The Tera Computer System[C]// Proc. of the 4th International Conference on Supercomputing. Amsterdam, June 1990: 1-6

[5] Laudon J, Gupta A, Horowitz M. Interleaving: A Multithreading Technique Targeting Multiprocessors and Workstations[C]// Proc. of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems. San Jose, California, USA, October 1994: 308-318

[6] Smith B J. Architecture and Applications of the HEP Multiprocessor Computer Systems [J]. Real Time Signal Processing Ⅳ, 1981, 298: 241-248

[7] Knijnenburg P M W, Ramirez A, Latorre F, et al. 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. Kohala Coast, Big Island, Hawaii, January 2002: 67-76

[8] Tullsen D M, Eggers S J, Levy H M. Simultaneous Multithreading: Maximizing On-Chip Parallelism[C]// Proc. of 22nd Annual International Symposium on Computer Architecture. Santa Marguerite Liguria, Italy, 1995: 392-403

[9] Tullsen D M, Eggers S J, Levy H M, et al. Exploiting Choice: Instructions Fetch and Issue on an Implementable Simultaneous Multithreading Processor[C]// 23rd Annual International Symposium on Computer Architecture. Philadelphia, PA, USA, May 1996: 191-202

[10] Evers M, Yeh T-Y. Understanding Branches and Designing Branch Predictors for High-Performance Microprocessors[J]. Proceedings of the IEEE, 2001, 89(11): 1610-1620

[11] Kang D, Gaudiot J-L. Speculation-aware Thread Scheduling for Simultaneous Multithreading [J]. IEE Electronics Letters, 2004, 40(5): 296-298

[12] Kang D, Gaudiot J-L. Speculation Control for Simultaneous Multithreading[C]// Proceedings of the 18th International Parallel and Distributed Processing Symposium. Santa Fe, New Mexico, April 2004: 76-85

[13] Falcon A, Santana O J, Ramirez A, et al. Tolerating Branch Predictor Latency on SMT[C]// Proceedings of the 5thInternational Symposium on High Performance Computing. Tokyo, Japan, October: 86-98

[14] Tullsen D M, Brown J A. Handling Long-latency Loads in a Simultaneous Multithreading Processor[C]// Proc. of the 34th IEEE International Symposium on Microarchitecture. Austin, USA, Dec 2001: 318-327

[15] Ei-Moursy A, Albonesi D H. Front-End Policies for Improved Issue Efficiency in SMT Processors[C]// Proc. of the 9th International Symposium on High-Performance Computer Architecture. Anaheim, California, USA, February: 31-40

[16] Cazorla F J, Ramirez A, Valetor M, et al. Dcache Warn: an I-Fetch Policy to Increase SMT Efficiency[C]// Proc. of the 18th International Symposium on Parallel and Distributed Processing. Santa Fe, New Mexico, USA, April 2004: 74-83

[17] 孙彩霞，张民选. Dwarn+：一种改进的同时多线程取指策略[J]. 小型微型计算机系统，2007，28(7)：1720-1723

[18] Suh G E, Devadas S, Rudolph L. A New Memory Monitoring Scheme for Memory-Aware Scheduling and Partitioning[C]// Proc. of the 8th International Symposium on High-Performance Computer Architecture. Boston, Massachusetts, USA, February 2002: 117-128

[19] Ramirez T, Pajuelo A, Santana O J, et al. Runahead Threads: Reducing Resource Contention in SMT Processors[C]// 16th International Conference on Parallel Architecture and Compilation Techniques. Brasov, Romania, September 2007: 423-423

[20] Eyerman S, Eeckhout L. A Memory-Level Parallelism Aware Fetch Policy for SMT Processors[C]// Proc. of the IEEE 13th International Symposium on High Performance Computer Architecture. Phoenix, Arizona, February 2007: 240-249

[21] Shin C, Lee Seong-Won. Dynamic Scheduling Issue in SMT Architectures[C]// Proc. of the 17th International Parallel and Distributed Processing Symposium. Nice, France, April 2003

[22] Raasch S E, Reinhardt S K. Applications of Thread Prioritization in SMT Processors[C]// Proc. of the Workshop on Multithreaded Execution and Compilation. Orlando, Florida, January 1999

[23] Yamasaki N, Magaki I, Itou T. Prioritized SMT Architecture with IPC Control Method for Real-Time Processing[C]// Proc. of 13th IEEE Real Time and Embedded Technology and Applications Symposium. Bellevue, Washington, USA, April 2007: 12-21

[24] 何立强，刘志勇. 一种具有 QoS 特性的同时多线程处理器取指策略[J]. 计算机研究与发展，2006，43(11)：1980-1984

[25] Ma Pengyong, Chen Shuming, Hu xiao. Two methods to enhance the master thread's performance in SMT Chip[C]// 2007 IFIP International Conference on Network and Parallel Computing. Dalian, China, September 2007: 578-583

[26] Aamodt T M, Chow P, Hammarlund P, et al. Hardware Support for Prescient Instruction Prefetch[M]. Madrid, Spain, February 2004: 84-95

[27] Luo Kun, Ju J G, Franklin M. Balancing Throughput and Fairness in SMT Processors[C]// ISPASS. 2001

[28] 张盛兵，王晶. 同时多线程结构的线程预构[J]. 西北工业大学学报，2007，25(2)：159-163

[29] Wang Jing, Zhang Shengbing, Zhang Meng, et al. A Modified Instruction Fetch Control Mechanism for SMT Architecture[C]// 2007 IEEE Region 10 Conference. Taipei, Taiwan, October-November 2007: 1-4

[30] He Liqiang, Liu Zhiyong. An Effective Instruction Fetch Policy for Simultaneous Multithreaded Processors[C]// Proc. of the 7th International Conference on High Performance and Grid in Asia Pacific Region. Japan, July 2004: 162-168

[31] Robatmili B, Yazdani N, Sardashti S, et al. Thread-Sensitive Instruction Issue for SMT Processors [J]. IEEE Computer Architecture Letters, 2004, 3: 5

[32] Sharkey J J, Ponomarev D V. Efficient Instruction Schedulers for SMT Processors[C]// Proc. of the 12th International Symposium on High-Performance Computer Architecture. Austin, Texas, Feb. 2006: 288-298

[33] Sharkey J J, Ponomarev D V. Balancing ILP and TLP in SMT Architectures through Out-of-Order Instruction Dispatch[C]// Proc. of the 2006 International Conference on Parallel Processing. Columbus, Ohio, August 2006: 329-336

[34] Raasch S E, Reinhardt S K. The Impact of Resource Partitioning on SMT Processors[C]// Proc. of the 12th International Conference on Parallel Architecture and Compilation Techniques. New Orleans, Louisiana, Sept. - Oct. 2003: 15-25

[35] Cazorla F J, Ramirez A, Valero M, et al. Dynamically Controlled Resource Allocation in SMT Processors[C]// Proc. of the 37th International Symposium on Microarchitecture. Portland, December 2004: 171-182

[36] Choi S, Yeung D. Learning-Based SMT Processor Resource Distributing via Hill-Climbing[C]// Proc. of the 33rd International Symposium on Computer Architecture. Boston, MA, USA, June: 239-251

[37] Yang Hua, Cui Gang, Yang Xiaozong. Eliminating Inter-Thread Interference in Register File for SMT Processors[C]// Proc. of the 6th International Conference on Parallel and Distributed Computing, Applications and Technologies. Dalian, China, Dec. 2005: 40-45

[38] 杨华，崔刚，刘宏伟，等. 基于线程感知寄存器重命名的 SMT 处理器资源分配[J]. 2008，31(5)：845-857

（上接第 190 页）

[4] Hashemian S V, Mavaddat F. A Graph-Based Approach to Web Services Composition[C]// IEEE/IPSJ International Symposium on Applications and the Internet. 2005: 183-189

[5] Xie X Q, Chen K Y, Li J Z. A Composition Oriented and Graph-Based Service Search Method[C]// Asian Semantic Web Conference. 2006: 530-536

[6] Liang Qianhui, Stanley Y W S. AND/OR Graph and Search Algorithm for Discovering Composite Web Services [J]. International Journal of Web Services Research, 2005: 48-68

[7] Paolucci M, Wagner M. Grounding OWL-S in WSDL-S [C]// Proceedings of IEEE International Conference on Web Services. 2006

[8] Zeng L, Benatallah B, Dumas M, et al. Quality driven Web services composition[C]// Proc. of the World Wide Web Conference. 2003: 411-421

[9] 邓水光，吴健，李莹，等. 基于回溯树的 Web 服务自动组合[J]. 软件学报，2007，18(8)：1896-1910

[10] 邝砾，邓水光，李莹，等. 使用倒排索引优化面向组合的语义服务发现[J]. 软件学报，2007，18(8)：1911-1921

[11] 岳昆，王晓玲，周傲英. Web 服务核心支撑技术：研究综述[J]. 软件学报，2004，15(3)：428-442

[12] 石静，丁长明，赵泽宇，等. Web 服务合成研究综述[J]. 计算机科学，2004，31(6)：54-58

[13] 高亚春，张为群. 基于 QoS 本体的 Web 服务描述和选择机制[J]. 计算机科学，2008，35(12)：273-276

[14] 孙亮，任小康. 基于本体的图像语义检索模型[J]. 重庆工学院学报：自然科学版，2009，23(1)：127-131