面向SMT体系结构的片上资源分配策略研究*)

张 骏[1] 樊晓桠[1] 刘松鹤[2]

(西北工业大学计算机学院 西安 710072)[1] (长安大学信息工程学院 西安 710064)[2]

摘 要 SMT处理器通过同时执行来自多个线程中的指令来提高性能,所有线程通过竞争共享的方式来最大化片上资源的利用率。然而,SMT处理器的集中控制结构所固有的线延迟约束和多个线程对片上资源持有的不均衡性使得设计者不得不考虑在线程间进行资源分配,来减少通信延迟和可能出现的线程饥饿。本文介绍了针对SMT体系结构片上资源分配的基本原理、研究内容;分析了片上资源分配对SMT体系结构造成的影响;从显式和隐式两个角度讨论了SMT体系结构片上资源分配策略的运行机制和设计方法;举例分析了POWER5处理器的动态资源平衡策略;最后,展望了SMT处理器片上资源分配的未来发展趋势。

关键词 同时多线程,处理器,资源划分

Research of On-chip Resource Distribution Strategies for Simultaneous Multithreaded Architecture

ZHANG Jun[1] FAN Xiao-ya[1] LIU Song-he[2]

(College of Computer Science, Northwestern Polytechnical University, Xi'an 710072, China)[1]

(Department of Information Engineering, Changan University, Xi'an 710064, China)[2]

Abstract SMT processor increases performance through executing instructions from several threads simultaneously. All threads compete each other for on-chip resource to maximize the resource utilization. But, unfortunately, for a SMT processor, the inherent wire delay restriction due to centralized control structure and on-chip resource holding imbalance between multiple threads make designer have to apply resource distribution policy to reduce the communication delay between threads and probability of thread starvation. This paper introduces the basic principle and research content of resource distribution for SMT architecture, analyzes the impact of on-chip resource distribution to SMT architecture, discusses the running mechanism and design technique for resource distribution policy, and explains the dynamic resource allocated strategy in POWER5 processor. At last, the future development trend of on-chip resource distribution for SMT processor is prospected.

Keywords Simultaneous multithreading, Microprocessor, Resource distribution

1 引言

超标量处理器通过挖掘单个线程中的指令级并行性来提高性能。然而,数据和控制相关降低了进一步提高指令级并行的可能性。结果是,当指令级并行性不够高的时候,处理器的许多资源处在空闲状态,对性能的提高没有任何贡献。同时,多线程处理器同时执行来自多个线程的指令[7],这种指令级并行和线程级并行结合起来的方式可以更好地利用处理器的资源,从而提高性能。然而,多个线程并不是简单地共享资源,而是对资源进行竞争。

多线程处理器在任何时刻只允许单个线程执行,因此尽管没有浪费指令周期,但多个线程还是无法在一个时钟周期内同时使用指令发射槽,水平浪费仍然不可避免。SMT处理器通过从多个活跃线程中动态选择与执行指令流,在一个时钟周期内可发射多个线程的多条指令,能够更好地利用处理器资源,有效降低水平浪费和垂直浪费[5-8]。

由于SMT处理器能同时执行多个线程的指令,那么理论上片上的执行资源和存储资源应该被多个线程自由竞争使用。然而,同时多线程是一种集中控制结构,线延迟的约束使得设计者不得不考虑在线程间进行资源划分,来减少通信延迟,从而达到更高的时钟频率。另外,多个不同线程对片上资源使用行为的差异可能导致某些线程长时间持有资源,从而产生资源占用的不均衡,甚至导致线程饥饿,严重影响处理器的整体性能和吞吐率。所以,目前所说的SMT处理器资源共享和竞争是在一定的分配方案内,采用某种方式被多个线程所同时占有的,从而在资源既定的情况下尽量达到更高的性能。这种资源被多个不同线程共同占有和竞争使用的方式就是SMT处理器的资源分配策略。

2 SMT体系结构资源分配概述

SMT处理器通过同时执行来自多个线程中的指令来提高性能,这些线程通过共享的方式来更好地使用处理器资源。线程间关键资源的分配方式不但决定了处理器的最终性能,而且决定了单个线程的性能。如果单个线程垄断了大部分资源,这个线程就能以全速运行,但是别的线程就要因为资源缺乏导致运行速度变慢。一个SMT处理器的设计目标决定了资源应该被以何种方式共享。如果IPC是唯一的目标,那么资源就应该被分配该运行最快的线程,而不管对其他线程性能的影响。然而,现有的SMT处理器被操作系统确认为多个独立的处理器,结果是操作系统将线程调度到它认为可以并行的单元上去。如果某些线程相对于其他的线程来说有较高的优先级,那么操作系统的工作调度就有可能受到严重的影响。因此,确保每个线程都被公平地对待是SMT处理器设计中一个应有的目标。

*)国家自然科学基金项目(60573107)资助;应用材料基金(XA-AM-200509)。张 骏 博士研究生,主要从事计算机体系结构、专用集成电路方面的研究;樊晓桠 教授,博士生导师,主要从事专用集成电路、计算机系统结构方面的研究;刘松鹤 硕士研究生,主要从事计算机软件、计算机网络方面的研究。

SMT处理器资源的分配方式分为两种类型:一类是通过取指策略的选择来决定的隐式资源分配策略,因为取指策略决定了哪些指令能进入处理器,参与资源的竞争;另一类是直接控制片上资源在多个线程间进行划分的显式资源分配策略。现有的隐式资源分配策略包括ROUND-ROBIN[6],ICOUNT[6],STALL[9],FLUSH[9],DG[10],PDG[10]等,显式资源分配策略包括静态、动态、动态和静态相结合三种类型。

现有的隐式资源分配取指策略没有对资源如何在线程间进行分配进行直接的控制,只使用了线程可能存在资源滥用的间接标示信息,比如说二级CACHE的缺失。因为没有对资源直接进行控制,某个线程垄断资源的情况还是可能出现的。同时,更糟糕的是一个拥有大量资源的线程可能长时间不释放资源的情况是很平常的。研究者提出了几种取指方案来发现这种情况,以便于及时将这个线程阻塞或者挂起,将资源释放给其它线程使用。这些策略的主要问题是在阻止资源垄断的同时带来了资源的不充分使用,因为它们阻止了一个线程使用一些别的线程也用不上的资源。但这种隐式策略确实提供了一种相对廉价的资源分配手段。

显式的资源分配方案直接对所有的共享资源进行控制。这种直接的控制能更好地对资源进行使用,减少资源的不充分使用。每个应用程序在执行过程中对资源的要求是不同的。如果能够更好地识别出这些要求并根据需求进行相应的资源分配,那么SMT处理器就能达到更高的性能。一方面,在任何一个给定的时刻,资源使用量大的线程被强迫只能使用一定数量的资源,以防止可能产生的资源垄断。另一方面,为了更好地挖掘资源使用量大的线程的指令级并行性,应该尽可能多地使这些线程使用那些别的线程不会使用到的资源。在每个周期内,显式的资源分配方案直接监控资源的使用情况,不完全依靠间接的标示信息。因此,能很快地发现某个线程占用的资源超过了给它所分配的资源的数量,并阻塞这个线程,直到线程所占用的资源不再超过预期数量,从而实现资源的实时重分配。

3 资源分配对SMT处理器的影响

从性能方面来看,资源分配减少了线程间的通信延迟,从而达到更高的频率。这种划分提供的强制公平性在很大程度上避免了取指策略的复杂化和线程饥饿的发生,同时保证了处理器的整体性能和吞吐量保持在较高水平。SMT处理器性能提高是建立在多个线程动态地共享硬件资源的基础上的,动态资源划分的灵活性相对于静态资源划分来说提供了潜在的更高的资源利用率。而静态资源划分的公平性相对于动态资源划分来说提供了更大程度的并行性。

有资料[2]表明,SMT在线程间动态获取资源的能力给性能的提高带来的好处并不大。在线程间静态地对存储资源进行分配,比如说指令队列和重排序缓冲区是很适度的,往往对性能有积极的影响;但是对带宽资源的分配,比如发射或者提交,往往会降低SMT处理器的性能。

造成这种差异的原因部分来自于存储资源对于饥饿的产生更加敏感,因为只要少数线程占用了一种存储资源的大部分表项,那么大部分线程只能竞争剩余的小部分存储表项,使得饥饿更可能发生。相比之下,带宽资源在每个周期都会重新分配,这很轻易地就避免了线程饥饿的发生。

另外,从设计角度来看,为了实现资源分配,需要为每个线程增加一些计数器,分别用来对每个线程的资源使用情况进行资源计数。比如说使用物理寄存器的数量、占用的指令发射队列和重排序缓冲区的表项数目、CACHE空间的使用量及其缺失数等。另外,还需要配套的仲裁逻辑,以便根据上述硬件监视器的实时状态决定何时选择怎样的方式进行资源重分配。这些逻辑虽然会增加一些硬件复杂度,但是它们都不位于SMT结构的关键路径上,不会对时钟频率和性能造成太大影响。

4 隐式资源分配策略

取指策略决定了哪些指令能进入处理器,参与资源的竞争。而SMT处理器以多个线程为对象的取指策略就是SMT处理器的调度策略,后者是资源分配策略的落脚点。换句话说,SMT处理器的资源使用情况是取指策略发生变化的原因,而不同取指策略的结果是SMT处理器资源在不同的线程间发生了重新分配。

SMT处理器的调度必须保证一定的公平性,必须保证每个活跃线程都能向下执行。同样,必须提供线程优先级策略,防止无用指令消耗执行资源。另外,还要能够提高优先级,保证性能很关键的线程能够尽量地提高吞吐率。然而,一个纯粹的SMT设计没有线程切换的概念[1],实现上述机制的策略就有所不同:SMT通过在发射指令的选择上设置优先级,从而能够在更细的粒度上管理执行资源的分配,而不是换出一个低优先级的线程或者换入一个高优先级的线程。通过控制线程的取指,可以防止不同优先级上的线程占有处理器执行窗口超过某个固定数目。处理器中任何动态分配的资源都有类似的限制。例如,限制每个线程对LOAD/STORE队列的占有,从而限制了线程对存储器的压力;还可以限制每个线程在分支或者值预测结构中的表项,从而为更高优先级的线程留出更多的资源。

ROUND-ROBIN[6]是最基本的取指策略,它交替地从所有线程中取指令,不考虑每个线程对资源的使用情况。

ICOUNT[6]优先给占用最少数量指令槽数目的线程较高的优先级,并且在线程中存在较高指令级并行性的情况下可以获得更好的性能。然而,SMT结构在处理二级CACHE缺失严重的线程上有很大困难。当这种情况发生时,ICOUNT不能意识到要阻塞某个线程,并且会很多周期地停止处理,结果造成了共享资源被垄断了很长时间 。

STALL[9]方案建立在ICOUNT的基础上,对线程有高CACHE缺失率的情况作了改正。它能发现一个线程遇到了二级CACHE缺失,并停止这个线程继续取指令,从而避免了资源的滥用。然而,这个二级CACHE缺失的发现已经太晚了,已经造成了对大量可用资源的占用。

DG[10]方案在一个线程的CACHE缺失数超过一定门限后触发取指锁定。PDG[10]使用一个CACHE缺失预测器来触发取指锁定。取指锁定的一个问题是资源的阻塞仍然会发生,因为这些阻塞条件的发现有可能太晚或者不可靠。

另一种方案不用对资源阻塞进行预测和取指锁定,而是在发生资源阻塞后立即通过冲刷阻塞的指令进行恢复,这就是FLUSH。FLUSH[9]方案是对STALL的扩展,解决了太晚发现一个线程的二级CACHE缺失而不能及时释放它所占用的资源给别的线程使用的问题。然而,仍然存在缺失线程在没有别的线程需要使用它所占用的资源的情况下仍然受到了惩罚。另外,通过冲刷所有来自缺失线程的指令,大量的指令又要重新预取,大量的功耗需要花费在重复的工作上。

STALL 和 FLUSH 相结合的方式能最小化被冲刷掉的指令。

隐式的资源分配策略，其缺点就是只能对性能作间接的优化。进一步说，就是这些取指策略是根据针对每个同时运行线程的资源使用硬件监视器来决定未来的取指方案，比如说指令发射队列的占用情况和 CACHE 的缺失情况等。然而硬件监视器所反映出的已经出现的性能瓶颈并不能通过改变取指策略立即得到改善，即使能得到改善也要付出阻塞、冲刷、重新取指等代价。也就是说，这种间接的优化方式必然会错过最佳的资源重分配时机，所以需要构造显式的、直接的资源分配策略，在尽量减少所花费代价的情况下使所有线程平稳执行。

5 显式资源分配策略

5.1 静态资源分配策略

静态的显式资源分配策略平均地在线程间将资源进行分割。这种方法保证没有哪个线程能垄断资源，每个线程被公平地对待。但这种方案会遇到和超标量处理器相同的问题：如果任何一个线程没有充分地使用分配的资源，这些资源就会被浪费掉。而实际上 SMT 处理器很少使用全静态资源分配的情况，因为这相当于将多个彼此之间几乎无关的窄发射超标量处理器集成到一块芯片上，这背离了 SMT 处理器多个线程通过竞争的方式共享资源进而提高整体吞吐率的设计初衷。

5.2 动态和静态相结合的资源分配策略

动态和静态相结合的显式资源分配策略将部分资源平均地在线程间进行分割，其余资源被所有线程自由动态的竞争共享。虽然文献[2]中指出了在线程间静态地对存储资源进行分配，比如说指令队列和重排序缓冲区，往往对性能有积极的影响，但是对带宽资源的分配，比如发射或者提交，往往会降低 SMT 处理器的性能。但是该文中所谓的动态是自由的竞争，而不是一种带有策略性的自适应资源分配方案。另外，在 SMT 结构中所有类型的资源相互依赖、彼此协作，构成一个统一的整体，无论哪一种资源出现阻塞都会严重影响处理器性能。不同的应用组合在 SMT 结构下表现出的混合行为千差万别，要为这些不计其数的应用组合设计一种通用的动态和静态相结合的显式资源分配策略，使得无论哪一种资源都不会出现阻塞现象，是完全不现实的。因此，一种动态和静态相结合的显式资源分配策略只适用于某些类型的应用组合。

5.3 全动态资源分配策略

顾名思义，全动态的意思就是所有片上被多个线程同时使用的资源都不会在线程间采用一种固定的静态划分方式，而是根据硬件监视器信息实时地、动态地在多个线程间进行调整，从而减少资源的不充分使用，在某个时刻最大化资源利用率。

文献[3]提出了一种智能的全动态资源分配策略，这种策略认为不同程序在执行过程中对资源的要求是不同的。如果能更好地识别出这些要求并根据需求进行相应的动态资源分配，那么 SMT 处理器就能达到更高的性能。线程在执行过程中都会有不同的行为，它们会在高指令级并性阶段和只有少量并行性的存储器限制阶段交换，所以它们对资源的需求是动态变化的，必须将它们行为的变化考虑进来，以便给线程分配它们最能有效利用的资源。图 1 是这种全动态资源分配的模块图。

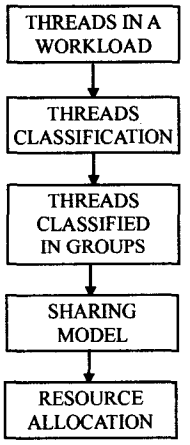

图 1 基于资源需求的全动态资源分配模块图

这种动态资源分配的主要思想建立在快速线程对资源的持有时间相对较短，存储器缺失和阻塞相对较少，适当地压缩快速线程的资源使用量并不会对其性能造成很大影响，而慢速的线程相对于快速线程来说需要更多的资源来挖掘指令级的并行性的基础上的。根据每个线程实际需要哪些资源和不需要哪些资源来进行分类，将线程划分为快速组和慢速组。将需要少量资源就能获得高性能的线程化为一类，将需要大量资源的线程划分为二类，因此在资源重分配时一类线程就能为二类线程暂时提供额外的资源。将某时刻最终的线程分类信息输入资源共享模型，共享模型将根据当前各个线程的资源使用情况、快速线程数量、慢速线程数量、待分配的额外资源数量等信息做出最终的资源重分配决策，并发出相应的控制信号实施资源重分配。这种线程的分类本身也是动态的，根据线程行为的变化进行不断的重新评估和重新分类。因此，它能动态地根据运行线程的具体状态来分配资源，具有较好的实时性，对整体性能曲线的最大化也有较好的收敛性。

6 POWER5 的动态资源分配策略

POWER5 处理器[11,14,15]包含了两个内核，每个内核运行在两路 SMT 模式下。在多线程模式下，一个线程有可能使用了大量的系统资源，甚至有可能阻塞另一个线程的执行。为了防止发生这种情况，POWER5 具有动态资源平衡控制逻辑，从而保证两个有不同需求的线程能平稳地执行。

POWER5 可以对在流水线中执行的指令进行跟踪。发射过的指令的控制信息都保存在一个叫做全局完成表的部件中，这个动作在发射指令的时候进行。动态资源平衡逻辑能监视资源的使用情况，比如说全局完成表，从而可以决定某一个线程对资源的使用情况是否已经超过的一个门限。如果超过门限，就挂起这个线程，允许另一个线程在没有拥塞的情况下执行。比如，如果一个线程遇到了多个二级 CACHE 缺失，相关的指令就要在发射队列中等待。这将妨碍指令的发射，导致另外一个线程执行缓慢。资源平衡逻辑在这一点上发现一个线程到达了二级 CACHE 的取数缺失门限。当这种情况发生，这个线程就要被挂起。类似地，资源平衡逻辑监视全局完成表来决定每个线程使用的入口数量。如果资源平衡逻辑发现一个线程开始使用太多的全局完成表入口，有阻塞另一个线程执行的可能，就会挂起过量使用全局完成表表项的线程。

POWER5 使用三种策略来挂起线程。第一，当一个线程使用了超过预定数量的表项时，降低线程的优先级是主要的

方法。每个线程有8个等级的优先级。这个优先级通过软件来设置，硬件可以对这个优先级进行调整，从而挂起线程。第二，禁止对线程的译码，直到拥塞情况消失。在一个线程发生了超过预定数量的二级CACHE缺失时，这种方法是主要的。第三，刷新所有正在等待分派的指令，并停止相应线程指令的译码，直到拥塞消失。这种方法在遇到长执行延迟的指令或者同步指令时是主要挂起线程的方法。

7 SMT处理器资源分配的未来发展趋势

首先，从设计角度来看，无论何种资源分配策略，其目标都是提高处理器的整体性能。而目前的隐式和显式的资源分配策略除了将多种不同类型的硬件资源监视器信息作为资源分配的根据外，还有一个重要的参数被忽视了，那就是资源分配方案所要优化的性能指标。假设一个SMT的设计中全局的IPC是最终的性能指标，那么资源分配方案将那些片上的硬件资源监视器的值保持在一个自认为较好的水平上，并不能说明全局IPC的值就一定比较高，多个线程的混合行为使得根据这些资源监视器所预期的IPC与真实IPC之间出现了误差。因此，应该使用一种更加直接的方式，使得真实IPC作为参数直接参与资源分配策略。

其次，由于片上的多种资源是所有应用程序运行的载体，属于不同类型的线程在同一种资源上也会表现出不同的行为。从线程角度来看，就是不同线程对资源的敏感程度不同。设计者应该在更深层次上挖掘典型应用对不同类型资源的敏感程度，从而为更便捷的管理同时运行多个线程，并为它们合理分配资源提供更多依据。比如说，可以将SMT处理器中的资源划分为带宽资源和存储资源。带宽资源包括取指带宽、指令发射带宽，但不包括指令提交带宽，存储资源包括指令队列、通用寄存器、重排序缓冲区、取数队列等。将快速线程定义为能快速地占用和释放资源，也就是说线程不会长时间地占用分配给它的资源，从而造成其他线程允许速度变慢。而只要有连续提供快速线程指令的能力，快速线程就能完成。也就是说，快速线程对资源的需求更倾向于带宽资源，而不是存储资源，这些信息就是资源分配的最好依据。

另外，目前大多数的资源分配策略如果发现某个线程运行速度变慢，影响到了其他线程的执行，比如说某个线程在缺失队列中占有很大的比例，那么就挂起这个线程，直到缺失解决。挂起的方案有禁止取指令，减少译码周期，冲刷相关指令。禁止取慢速线程指令和减少译码周期的方法，只能避免更多的慢速线程指令进入流水线，而没有快速改变资源分配的状态。慢速的线程仍然占用着大量的执行资源，并且在很长时间内不会释放这些资源。冲刷慢速线程相关指令的方法虽然能使慢速线程指令释放所占用的资源，但浪费了大量时钟周期。现代处理器大都采用深度流水线，同时在流水线中执行的指令超过100条，这种冲刷对执行周期的浪费是巨大的。这些方案只是从不同角度扼杀了慢速线程的执行，而回避了要解决的关键问题：线程已经处在慢速的状态下，如何加速它们的执行，使它们尽早执行完毕，退出流水线，从而释放它们占用的资源给别的线程使用。从存储资源上来看，可以通过尽量多地分配给慢速线程更多的执行资源，加快慢速线程的执行速度，以达到尽快缓解拥塞的目的。多余的资源就来自于快速线程的冗余资源。从带宽资源的角度来看，慢速线程已经有大量的指令处在流水线中，占用了大量资源，并且执行效率很低。如果继续对慢速线程的指令进行取指或者继续发射慢速线程的指令进入内核执行，必然会进一步加剧拥塞程度，影响快速线程指令的执行。而快速线程指令执行效率高，适当加大快速线程对带宽资源的占用，能平衡借出冗余资源对性能的影响。由此看来，当拥塞发生时，快速线程应该分出一部分冗余资源供慢速线程使用，同时加大对带宽资源的占用比例。相比之下，应该压缩慢速线程对带宽资源的使用，同时借用快速线程的冗余资源尽快地解决慢速线程的拥塞问题。

总之，SMT处理器的资源分配应该遵循一个原则：不能简单地阻止或者扼杀一个线程的执行，而要通过多个线程间互相协助、平稳共存的方式来达到缓解拥塞、多个线程顺畅执行的目的，避免重复操作和低资源利用率的情况发生。

结束语 SMT体系结构通过同时发射并执行来自多个线程中的指令成功地挖掘了多个线程中的指令及并行性，这不但提升了性能，还最大化了片上资源的利用率。但是集中式控制结构所固有的线延迟约束和不同线程对资源使用行为的差异所造成的不均衡性使得多线程间的资源分配策略成为一种必然。本文介绍了SMT体系结构片上资源分配的基本原理、研究内容；分析了片上资源分配对SMT体系结构造成的影响；从显式和隐式两个角度讨论了SMT体系结构片上资源分配策略设计思想及其优劣，认为全动态的片上资源分配方案是未来的发展方向；举例分析了POWER5处理器的动态资源平衡策略；最后，认为性能指标直接参与分配过程、不同线程在资源使用行为上的规律以及相互关联、多线程的协作共存模式等方面应该是SMT体系结构资源分配策略在未来的研究重点。

参考文献

[1] Snavely A, Tullsen D M. Symbiotic jobscheduling for a simultaneous multithreading processor // Proc. Ninth Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS IX). Nov. 2000: 234-244

[2] 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 (PACT'03). 2003

[3] Cazorla F J, Ramirez A, Valero M. Dynamically Controlled Resource Allocation in SMT Processors // Proceedings of the 37th International Symposium on Microarchitecture (MICRO-37). 2004

[4] Choi S, Yeung D. Learning-based SMT Processor Resource Distribution via Hill-Climbing // Proceedings of the 33rd International Symposium on Computer Architecture (ISCA'06). 2006

[5] Hirata H, Kimura K, Nagamine S, et al. An elementary processor architecture with simultaneous instruction issuing from multiple threads // Proceedings of the 19th Annual International Symposium on Computer Architecture. May 1992: 136-145

[6] Tullsen D, Eggers S, Emer J, et al. Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor // Proceedings of the 23th Annual International Symposium on Computer Architecture. Apr. 1996: 191-202

[7] Tullsen D, Eggers S, Levy H M. Simultaneous multithreading: Maximizing on-chip parallelism // Proceedings of the 22th Annual International Symposium on Computer Architecture. 1995

[8] Yamamoto W, Nemirovsky M. Increasing superscalar performance through multistreaming // Proceedings of the 1st International Conference on High Performance Computer Architecture. June 1995: 49-58

[9] Tullsen D, Brown J. Handling long-latency loads in a simultaneous multithreaded processor // Proceedings of the 34th Annual ACM/IEEE International Symposium on Microarchitecture. Dec. 2001

[10] El-Moursy A, Albonesi D. Front-end policies for improved issue efficiency in SMT processors // Proceedings of the 9th International Conference on High Performance Computer Architecture. Feb. 2003

[11] Kalla R, Sinharoy B, Tendler J. SMT implementation in POWER 5. Hot Chips, Aug. 2003

[12] Keltcher C N. The AMD opteron processor for multiprocessor servers. IEEE Micro, March-April 2003: 66-76

[13] Kongetira P. Niagara A 32-Way Multithreaded Sparc Processor. IEEE Micro, March-April 2005: 21-29

[14] Tendler J M. Power4 System Microarchitecture. IBM J RES & DEV, 2002, 46(1): 5-25

[15] Sinharoy B. Power 5 System Microarchitecture. IBM J RES & DEV, 2005, 49(4/5): 505-521