基于映射表的寄存器文件设计以及编译器优化

邓晴莺, 张民选
(国防科学技术大学计算机学院, PDL, 湖南长沙 410073)

摘 要: 寄存器文件的设计在高性能处理器设计中十分重要, 寄存器栈和寄存器栈引擎是提高其性能的重要手段. 编译优化常常基于特定的体系机构以及目标机器. 本文针对 EDSMT 微体系结构(基于 IA-64 的同时多线程体系结构) 提出了一种新颖的基于映射表的寄存器机制——MTRM(Mapping Table-based Register Management), 它通过映射表将连续的虚拟寄存器物理号映射到不连续的实际物理寄存器, 并研究了编译器支持下的及时去配, 实验结果表明该方案能有效提高性能.

关键词: 寄存器文件; 同时多线程; EPIC; 并行; 编译优化
中图分类号: TP333 文献标识码: A 文章编号: 0372-2112 (2008) 02-0392-05

Mapping Table Based Register File Design and Compiler Optimization

DENG Qing-ying, ZHANG Min-xuan
(*PDL, College of Computer, National University of Defense Technology, Changsha, Hunan 410073, China*)

Abstract: Register file design is very important in high performance processor design. Register Stack and Register Stack Engine are effective ways to improve performance. Compiler optimizations are often driven by specific assumptions about the underlying architecture and implementation of the target machine. In this paper, we present our efforts to design and implement register file management mechanism——MTRM (Mapping Table-based Register Management) on EDSMT, which is a kind of SMT architecture based on IA-64. MTRM assigns a Mapping Table for each thread to mapping their logic registers to physic registers, which adds a middle level into Itanium's original rename mechanism. MTRM focused on supporting the effective sharing of registers in an EDSMT processor, using register renaming to permit multiple threads to share a single global register file. Existing hardware is effective at allocating physical registers; it has only limited ability to identify register deallocation points. Compile optimization is considered to deallocate dead registers, while Special Bit and Special Instruction are used as two effective ways. Simulation results indicate that these mechanisms can reduce register deallocation ineffciencies; in particular, on small register files, the best of the schemes attains speedups of up to 2.2 for some applications, and 1.8 on average.

Key words: register file; simultaneous multithreading(SMT); explicitly parallel instruction computing(EPIC); parallel; compile optimization

1 引言

同时多线程(SMT)[1]和单芯片多处理(CMP)[2]是利用片上资源开发线程级并行的两种体系结构手段. 它们都面向于多线程负载应用, 现在已有较多的针对它们的性能、功耗等问题的研究[3~5]. 动态同时多线程(DSMT)[7]把动态线程提取和线程切换技术融合到 SMT 体系结构中, 能更好地开发线程级并行(TLP). 基于软硬件协同工作的 EPIC(显式并行指令计算)[8]能用相对简单的硬件来有效开发指令级并行(ILP). 我们在 EPIC 的基础上扩展了同时多线程的执行能力, 从而建立了一个新的体系结构——EDSMT[11].

SMT 处理器为了支持多个硬件现场需要有大的寄存器文件. 通过对物理寄存器的线程间共享以及良好的管理, SMT 处理器能减少对寄存器的需求, 且能在给定寄存器文件大小下提高性能. 有寄存器重命名机制的处理器能较准确地知道新寄存器分配的时机, 却难以准确知道何时去配寄存器, 这会造成性能下降.

为了深入研究并行编译、SMT 和 CMP 体系结构以及软硬件协同工作, 建立一个基本的研究平台十分重要. 我们研究了 EDSMT 的体系结构, 并建立了一个踪迹驱动的模拟器——EDSMTSIM; 同时采用经过修改的休斯顿大学开发的 OpenUH(基于 Pro64 和 OpenMP) 来作为并行编译器. 本文此基础上研究了 EDSMT 的寄存器文件结构及其寄存器重命名机制, 并以编译器来辅助[9]进行寄存器去配.

2 IA-64 中的寄存器设计技术

IA-64 中通用寄存器被分成 2 个子集, 通用寄存器 0 至 31 为静态通用寄存器, 它们对所有过程都是可见的. 通用寄存器 GR32 到 GR127 为栈式通用寄存器, 它们作为每个进程的局部区域, 大小在 0 到 96(从 GR32 开始)

收稿日期: 2007-03-12; 修回日期: 2007-09-10
基金项目: 国家 863 高技术研究发展计划(No. 2002AA110020); 国家自然科学基金(No. 60273069, No. 60376018)

之间变化[10].

通过设置寄存器栈, 避免了在过程调用及返回时频繁的寄存器存储与恢复(spill and fill). 栈机制是通过寄存器重命名来实现的, 重命名基址由过程调用和返回来控制, 并且这些操作对于程序不可见. 栈寄存器中对某一过程可见的部分称为寄存器栈帧(Register Stack Frame).

在过程调用和返回时, 静态寄存器的保存和恢复必须通过软件显式的进行, 而栈寄存器由寄存器栈引擎(RSE)自动实现. 每当程序生成一个新的线程, 应用程序实时库或者操作系统就会为其留出一块内存空间作为该线程的寄存器栈换出的后备存储器(backing store).

寄存器旋转的重命名技术支持软件流水, 能使得几个循环遍重叠执行而无须展开循环[10]. RSE 为了支持软件流水及寄存器旋转, 给每个过程的逻辑栈帧分配了连续的逻辑寄存器号, 经过重命名后相应的物理栈帧中的物理寄存器号也是连续的.

3　EDSMT 中的 MTRM 机制

3.1　基于多线程的寄存器管理机制 MTRM

寄存器文件是多线程处理器设计中的瓶颈之一. 要进一步提高寄存器资源的使用效率, 并考虑到软件流水和寄存器旋转等关键技术对物理寄存器号的连续性要求, EDSMT 微体系结构提出了一种新颖的基于映射表的寄存器管理机制——MTRM(Mapping Table-based Register Management), 它通过映射表将连续的虚拟寄存器物理号映射到不连续的实际物理寄存器. 此机制兼容 Itanium 体系结构, 能够直接支持寄存器旋转和软件流水等关键技术. 图 1 为 MTRM 机制的逻辑示意图.

MTRM 机制为每个线程分配一个映射表, 每个线程使用其映射表中映射的物理寄存器, 映射表中的物理寄存器号码是从空闲物理寄存器号队列中取来分配给线程的, 需要时分配, 不需要时则应及时去配. 这种机制使得线程使用的物理寄存器号码可以不连续, 且各个线程使用各自的物理寄存器, 彼此互不干涉. 对于某个线程, 若为其分配的映射表入口号不够使用时, 需要把此映射表的一部分寄存器号以一定的格式存储到后备存储区去, 直到需要时再恢复. 当所有的物理寄存器使用完毕时, 即空闲物理寄存器号队列为空时, 则需要把一定数量的物理寄存器数据保存到后备存储区, 释放出一定数量的物理寄存器以供使用.

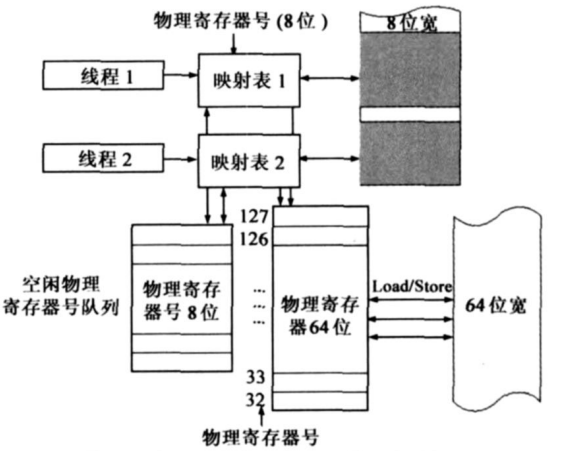

图 1　基于多线程的寄存器管理机制 MTRM

3.2　MTRM 详细设计方案

MTRM 机制用映射表替代 Itanium 系列微处理器中物理寄存器的位置, 堆栈结构原是针对物理寄存器堆, 现针对的是映射表(空闲物理寄存器号以队列结构供映射表使用), 原来逻辑寄存器号与物理寄存器号的对应关系成为逻辑寄存器号与映射表入口地址号的对应关系. 其具体实现方案结构见图 2.

MTRM 机制主要包括两大部分: 第一部分是映射表实现的 RSE 部分; 第二部分是物理寄存器的转移使用, 只有在物理寄存器全部使用完的情况下才涉及到这一部分. 第一部分以映射表栈代替原来的物理寄存器栈

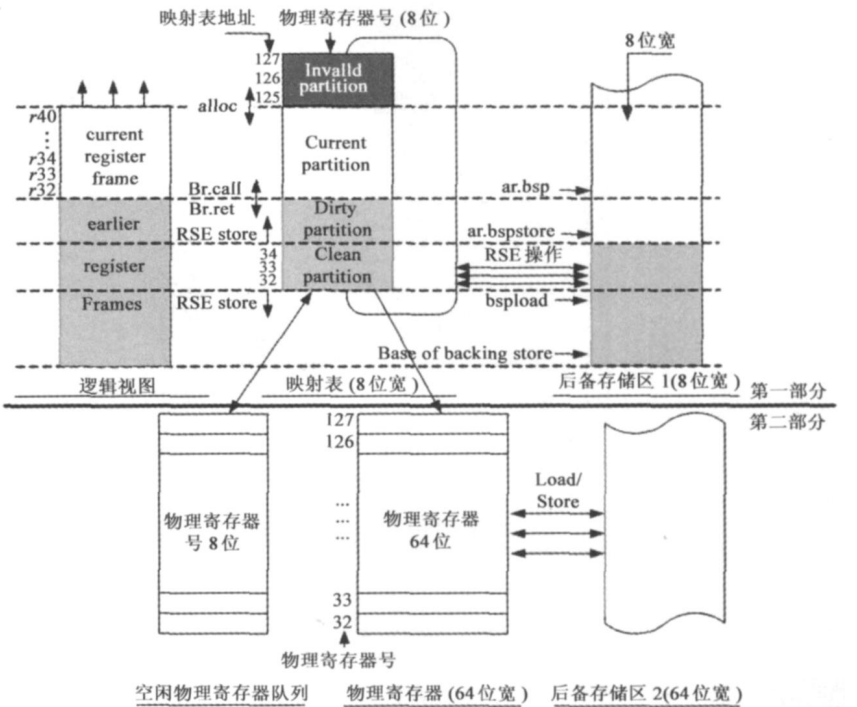

图 2　MTRM 实现方案

帧,除中间部分原是物理寄存器的堆栈结构(物理寄存器号从 32 到 127)现成为映射表堆栈结构(映射表地址号从 32 到 127),及进行 RSE 操作的不是 65 位物理寄存器数据而是 8 位数据(物理寄存器号),因此后备存储区 1 是 8 位宽,其它操作同 Itanium 的 RSE 方案一致. 映射表中填写的物理寄存器号(8 位)从第二部分的空闲物理寄存器号队列中取得. 若空闲物理寄存器号队列为空,表示 96 个物理寄存器全部被占用,没有物理寄存器可提供了,此时就要考虑物理寄存器的转移使用,如图 2 第二部分的 Load/ Store 操作. 过程调用或线程切换时,将物理寄存器数据存到后备存储区,则此物理寄存器可以被再次使用. 当过程返回时,根据需要把存储在后备存储区的物理寄存器数据取到相应的物理寄存器里,恢复其数据.

3.3 MTRM 实现技术

类似于 Itanium 中的操作,过程之间的参数传递也是通过各个栈帧之间的重叠区来传递. 过程调用时,若映射表栈帧经过 Clean 分区到达 Dirty 分区底部时,引起 RSE 的 Spill 操作,不过 Spill 到后备存储区的是物理寄存器号(8 位),而不是物理寄存器的值(65 位). 过程返回时,若映射表栈帧经过 Dirty 区,到达 Clean 区的底部时,则需要把后备存储区中的物理寄存器号(8 位) Fill 回到映射表栈帧中. 由于映射表相对于软件是透明的,因此其寄存器重命名逻辑,Rse. bof、BspStore 等参数,以及与 RSE 相关的指令都不需要改变.

96 个物理寄存器的资源是有限的,当空闲物理寄存器号队列所能提供的寄存器数目不够过程调用分配映射表栈帧所需要的寄存器数目时,就需要准备足够的物理寄存器以供使用,被转移的物理寄存器数据存放到后备存储区 2 中,如图 3 表示.

物理寄存器的转移使用的主要过程如下:

(1) 数据槽号的填写. 数据槽号是后备数据区的标号表示. 对存放物理寄存器数据的后备存储区数据槽从 32 开始定义,依次向上累加,出现 RNat 槽不计. 32 号数据槽是后备存储区 2 的初始位置,94 号与 95 号数据槽之间存在一个 RNat 槽,每帧有 63 个数据槽. 数据槽号的表示方法有机地将寄存器空间和后备存储区域联系起来,是实现 MTRM 机制的基础.

(2) 定义计数指针指向已经存储了物理寄存器数据的上一个数据槽号,如图 3 中计数指针为 99,则 98 号以下(32 号到 98 号)数据槽都存储了物理寄存器数据(除开链表中的);对于后备存储区 2 中位置低于计数指针的没有被使用的数据槽,则用一个“链表”结构把这些数据槽号链接起来.

通过 MTRM 机制对物理寄存器的转移使用方式有效地实现了寄存器的高效管理与透明使用,是一种适合多线程处理器的有效机制.

不同于 Itanium 结构中“逻辑寄存器-物理寄存器-后备存储区”三层结构的寄存器实现方案,EDSMT 提出的 MTRM 机制是一种用映射表(空闲物理寄存器号以队列结构供映射表使用,物理寄存器文件与映射表之间有映射关系)替代中间物理寄存器层,即“逻辑寄存器-映射表-后备存储区”新三层结构实现 RSE 的方案,此方案中物理寄存器的堆栈结构变成了映射表的堆栈结构. 通过 MTRM 机制,利用软硬件协同工作的方式,更好地在多线程环境下实现对寄存器的有效管理.

4 EDSMT 中寄存器机制的编译优化

对于线程中寄存器的使用,传统的方法是为每个过程分配一定数量的固定的物理寄存器,分配给某个线程的物理寄存器不论其是否空闲都不能被其它线程使用,这种机制虽然设计简单,但是不利于高效的寄存器管理,同样容易造成寄存器使用的浪费. 要进一步提高寄存器资源的使用效率,则要及时去配已经使用完毕的寄存器,以供其它线程或者过程使用.

MTRM 能有效共享 EDSMT 处理器中的寄存器资源,并使用寄存器重命名允许多个线程共享一个全局

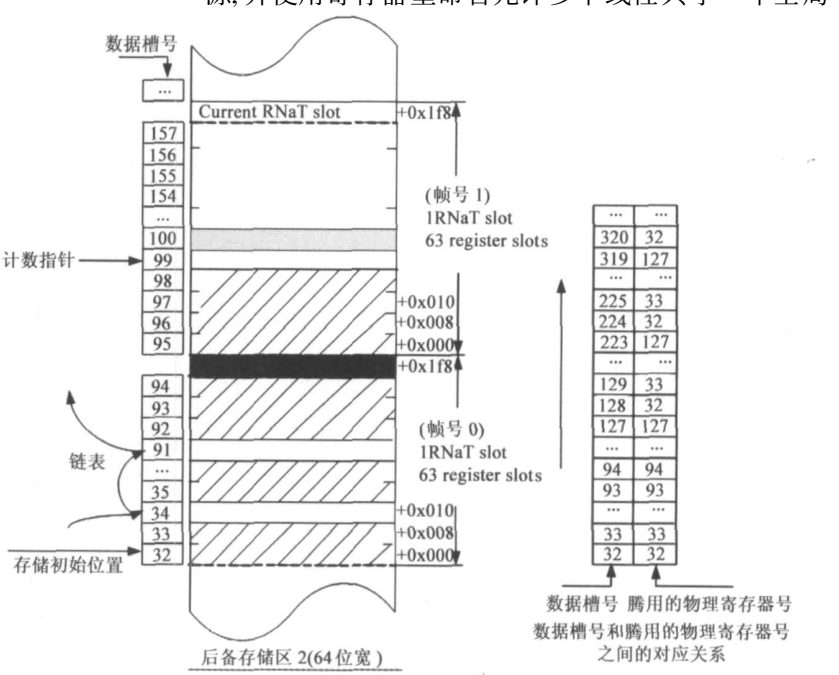

图 3 物理寄存器的转移使用方式

寄存器文件. 这样, 当一个线程有很大的寄存器需求而另一个线程的寄存器需求很小时, 该线程的寄存器压力能得到有效缓解. 但是, 现有的重命名技术并不能完全开发共享寄存器文件的潜力, 尤其是现有的硬件只能有效地分配物理寄存器, 而不能很好地识别寄存器的去配点, 它只能以保守的方法释放寄存器, 这样很可能会浪费掉本可以更好利用的寄存器资源.

. 一般存在两种类型的死寄存器: (1) 分配给空闲的硬件现场的寄存器(2) 活跃现场中最后一次使用中已经提交的寄存器[6]. 在本文中, 我们的编译优化主要针对的是后者. 为此, 我们研究了两种编译器能把最后一次使用的信息传递给处理器硬件的机制; 这样, 重命名硬件可以更加有效地进行寄存器的去配. 而如果没有这些信息, 硬件则只能保守地在这些寄存器被重定义时才进行去配. (*a*) 特殊位: 它通过专用的指令位(采用的是 IA-64 指令系统中的保留位) 来把最后使用信息传递给硬件. 它能立即识别最后使用信息同时也没有多余指令开销, 能达到采用编译器静态最后使用信息的性能提升的上限. (*b*) 特殊指令: 这是一种特殊位的更现实的实现方法. 不同于用指令本身来表达最后使用信息, 它采用独立的指令来描述释放一个或两个寄存器. 在指令包含有寄存器的最后使用时(且寄存器也没有被该指令重定义) 编译器在其后产生一条释放寄存器指令(利用 IA-64 指令集中未被使用的操作码). 这种方法同前一种一样能尽快释放寄存器, 但是会有额外的动态指令开销.

现有的重命名硬件提供了寄存器去配的机制(例如把物理寄存器放回空闲物理寄存器号表), 且在同一时钟周期能进行多个去配.

5　实验结果

我们设计了一个实验来评价 EDSMT 体系结构下的去配机制. 实验使用 SPECint2000 作为测试程, 而模拟器则采用 EDSMTSIM, 它的基本配置如表 1 所示. 编译器采用 OpenUH, 它能够提供很大的优化灵活性, 并能很好地利用 EPIC 的特质. 在应用的初始化阶段, 我们使用了快速模拟模式来预热缓存, 而在主要的计算阶段进行详细模拟.

表 1　模拟器的基本配置

参数	EDSMTSIM
取指带宽	2 bundles , 6 instructions per cycle
基本取指策略	ICOUNT2.2
指令队列	16 for each separated Queues
功能单元	2 IU, 4 MU, 2 FU, 3 BU
物理寄存器	128 GR, 128 FR, 64 PR, 8 BR per thread
L1I cache, L1D cache	16KB, 4-way, 64-bytes lines, 1 cycle access
L2 cache	256KB, 8-way, 64-bytes lines, 10 cycles latency
L3 cache	3MB, 12-way, 64-bytes lines, 50 cycles latency
主存延时	100 cycles

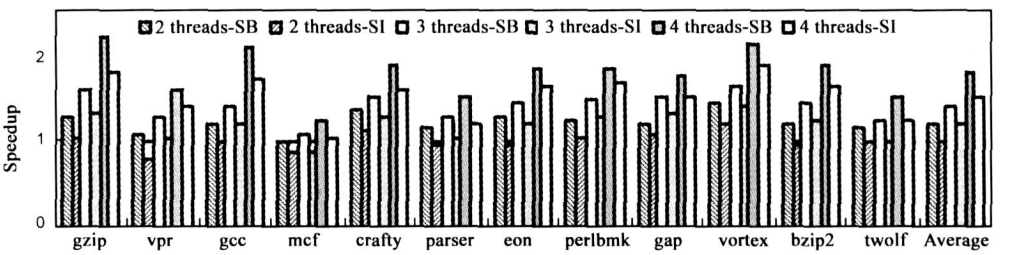

图 4　128 个完全共享寄存器下, 特殊位方法和特殊指令方法分别针对没有显式寄存器去配的加速比

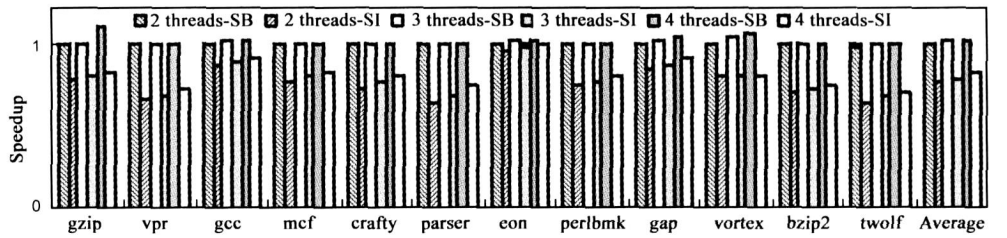

图 5　256 个完全共享寄存器下, 特殊位方法和特殊指令方法分别针对没有显式寄存器去配的加速比

图 4 和图 5 将两种去配机制进行了对比, 描述了在不同的线程配置下它们分别针对不采用显式寄存器去配的加速比. 特殊位方法针对小的寄存器文件能显著提高性能(平均 80%, 最高能达到 120%). 该方法强调了寄存器去配的优势: 通过提高寄存器的利用率, 一个有较小寄存器文件的 EDSMT 的处理器能达到有较大寄存器文件的性能, 使它在多个寄存器现场下亦不需要倍增其寄存器. 实验结果表明 4 个现场的 EDSMT, 在有效的 RSE 以及编译帮助下能减轻物理寄存器的压力. 由于能在寄存器最后使用时及时去配, 特殊位方法对于减少死寄存器更加有效.

当在有较小寄存器文件且寄存器成为性能瓶颈

时,特殊指令方法有较好的结果.而当寄存器文件较大且寄存器使用率不够高时,因为其额外指令开销,该方法的性能不佳.

为了度量寄存器文件管理技术针对寄存器文件大小的敏感程度,我们同样实验了不同大小的寄存器文件(从 128 到 256).超过 256 个时,其它处理器资源如指令队列等成为了性能瓶颈;对于 4 个现场则至少需要 128 个寄存器.较小的寄存器文件对于处理器设计十分重要[6]:(1)它的访问时间短,这样可减小时钟周期(若寄存器访问在关键路径上时),或减少寄存器读写的额外的级数(stage);(2)占用芯片面积小;(3)消耗功耗少.

在这一节,我们描述并评价了编译器传递最后使用信息给硬件的两种机制,并通过实验表明它们能提高采用 MTRM 的寄存器文件管理机制的 EDSMT 处理器的寄存器利用率.这两种方法或是用指令中的某些位或者采用新的指令来指导重命名硬件来释放物理寄存器.模拟结果表明这两种机制能提高寄存器去配的效率;尤其是对于采用小的寄存器文件时,对于一些应用最好能获得 2.2 的加速比(平均 1.8 的加速比).这些机制也同样符合 EPIC 的设计哲学.

6 结论及将来工作

SMT 和 CMP 已经逐渐成为了市场的主流[12].为此我们建立了一个并行研究平台,它包括可移植的 OpenUH 编译器以及基于 IA-64 同时多线程的体系结构 EDSMT.该并行平台对于并行编译和体系结构的研究,尤其是对于把编译器和体系结构紧密联系的 EPIC 结构的研究十分有利.本文在此基础上研究了 EDSMT 结构下能提高寄存器利用率的相关技术,提出了满足多线程需求下的小且速度快的寄存器文件设计.通过编译器以及 MTRM 硬件的相互协作,能使得单线程以及多线程程序的寄存器使用更加有效.实验结果表明对于给定数量的硬件现场该方案能提高性能,同时对于给定数量的寄存器也能处理更多的现场.

今后,我们将细化该寄存器文件的设计和优化,并加入功耗等特性的考量;同时致力于研究该平台下的其他特性,研究采用精简的 EDSMT 核来构建同构以及异构的 CMP 系统.

参考文献:

[1] D Tullsen, S Eggers, H Levy. Simultaneous multithreading: maximizing on-chip parallelism[A]. The 22rd Annual International Symposium on Computer Architecture (ISCA)[C]. New York: ACM Press, 1995. 392-403.

[2] K Olukotun, B A Nayfeh, L Hammond, K Wilson, K Chang. The case for a single-chip multiprocessor[J]. Computer Architecture News, 1996, 24(SI): 2-11.

[3] Y Li, D Brooks, Z Hu, K Skadron, P Bose. Understanding the energy efficiency of simultaneous multithreading[A]. The 2004 International Symposium on Low Power Electronics and Design [C]. Newport Beach, California: ACM Press, 2004. 44-49.

[4] R Sasanka, S V Adve, Y K Chen, E Debes. The energy efficiency of CMP vs. SMT for multimedia workloads[A]. The 18th Annual International Conference on Supercomputing[C]. Saint-Malo, France: ACM Press, 2004. 196-206.

[5] Y Li, K Skadron, Z Hu, D Brooks. Performance, energy, and thermal considerations for SMT and CMP architectures[A]. The Eleventh IEEE International Symposium on High Performance Computer Architecture (HPCA)[C]. San Francisco, California: IEEE Computer Society Press, 2005. 71-82.

[6] Jack L Lo, Sujay S. Panrekh, Susan J Eggers, Henry M Levy, Dean M Tullsen. Software-directed register deallocation for simultaneous multithreaded processors[J]. IEEE Tronsactions on Parallel and Distributed Systems, 1999, 10(9): 922-933.

[7] Haitham Akkary, Michael A Driscoll. A dynamic multithreading processor[A]. The 31st annual ACM/IEEE international symposium on Microarchitecture[C]. Dallas Texas: ACM Press, 1998. 226-236.

[8] M S Schlansker, B R Rau. EPIC: explicitly parallel instruction computing[J]. IEEE Computer, 2000, 32(2): 37-45.

[9] Jack L Lo, Susan J Eggers, Henry M Levy, Sujay S Parekh, Dean M Tullsen, Tuning. Compiler optimizations for simultaneous multithreading[A]. The 30th annual ACM/IEEE international symposium on Microarchitecture[C]. Los Alamitos: IEEE Computer Society, 1997. 114-124.

[10] Cameron McNairy, Don Soltis. Itanium 2 processor microarchitecture[J]. IEEE Micro, 2003, 20(5): 44-55.

[11] 蒋江,邢座程,张民选. EDSMT 微体系结构研究[J]. 计算机工程与科学, 2005, 27(4): 88-91.
Jiang Jiang, Xing Zuocheng, Zhang Minxuan. Research on the microarchitecture of EDSMT[J]. Computer Engineering & Science, 2005, 27(4): 88-91. (in Chinese)

[12] kunle olukotun and lance Hammond. The future of microprocessors[J]. QUEUE, 2005, 3(7): 26-29.

作者简介:

邓晴莺 女,1980 年生于天津,国防科学技术大学博士生,主要研究方向为高性能计算机系统结构以及微电子.
E-mail: freesunnybird@gmail.com