第34卷 第12期 计 算 机 工 程 2008年6月
Vol.34 No.12 Computer Engineering June 2008
·博士论文· 文章编号：1000—3428(2008)12—0013—03 文献标识码：A 中图分类号：TP303

IA-64的并行架构及其寄存器文件

邓晴莺，张民选，蒋 江
(国防科学技术大学计算机学院，长沙 410073)

摘 要：同时多线程能在同一时钟周期执行不同线程的指令，并且指令级并行和线程级并行。显式并行指令计算关注于编译器和硬件的相互协作。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。该文设计和实现一套并行环境，其中包括并行编译器OpenUH和基于IA-64的同时多线程体系结构EDSMT，实验表明，该并行架构适用于大多数并行应用，针对NAS的并行测试程序，该架构相对于SMTSIM平均有12.48%的性能提升。
关键词：同时多线程；显式并行指令计算；并行；寄存器文件

Parallel Infrastructure of IA-64 and Its Register File

DENG Qing-ying, ZHANG Min-xuan, JIANG Jiang
(School of Computer, National University of Defense Technology, Changsha 410073)

【Abstract】 Simultaneous Multithreading(SMT) processors execute instructions from different threads in the same cycle, which has the unique ability to exploit ILP and TLP simultaneously. Explicitly Parallel Instruction Computing(EPIC) emphasizes importance of the synergy between compiler and hardware. Register file design is very important in high performance processor design. Register stack and register stack engine are effective ways to improve performance. This paper presents efforts to design and implement a parallel environment, which includes an optimizing, portable parallel compiler OpenUH and SMT architecture EDSMT based on IA-64. Meanwhile, its register file mechanism is carefully designed. Experimental results show the infrastructure is suitable for parallel applications and the IPC increment over SMTSIM is 12.48% in average using the NAS parallel benchmarks.
【Key words】 Simultaneous Multithreading(SMT); Explicitly Parallel Instruction Computing(EPIC); parallel; register file

1 概述

同时多线程(Simultaneous Multithreading, SMT)和单芯片多处理(CMP)是利用片上资源开发线程级并行的2种体系结构手段。SMT允许来自多个线程的指令共享几个关键处理器资源，从而提高了资源的利用率，其优势在于它的较高的单位面积吞吐率(area-efficient throughput)；而CMP则通过在一个芯片上复制多个处理器核来提高系统吞吐率。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。

动态同时多线程(DSMT)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显式并行指令计算(Explicitly Parallel Instruction Computing, EPIC)[1]能用相对简单的硬件来有效开发指令级并行(ILP)。笔者在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。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，其设计十分关键。

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。笔者研究了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。重点研究了EDSMT的寄存器文件结构以及寄存器重命名机制。

2 基本并行架构

2.1 并行编译结构

随着处理器速度越来越快，HPC理论上的峰值速度越来越高，但是用户得到的性能却没有呈相应比例的增加，并行编译技术的目标就是要减小这个差距。多源、多目的的并行编译，多级(multilevel)并行编译(高级并行以及指令级并行)、多粒度(multigrid)并行编译成为当前并行编译器的发展趋势，而OpenUH正符合了这一趋势[2]。

OpenUH源于SGI的Pro64，基于IA-64平台，融合了Open64的2个主要分支(ORC和Pathscale)。它翻译了OpenMP 2.5指令，并与C, C++, FORTRAN77/90(一部分FORTRAN95)相结合。OpenUH包括了一整套优化模块：过程间分析(IPA)模块，循环嵌套优化(LNO)模块和全局优化模块等，并在多个层次采用了大量的最先进的分析和转换工具。它能直接生成针对Itanium平台的目标代码，而对于其他平台，OpenUH可以通过中间表示到源程序的转换工具，作为其源到源的编译器。

2.2 EDSMT微体系结构

动态同时多线程DSMT以及基于EPIC的设计理念是EDSMT研究的2个重要基础[3]。

基金项目：国家“863”计划基金资助项目(2002AA110020)；国家自然科学基金资助项目(60273069, 60376018)
作者简介：邓晴莺(1980－)，女，博士研究生，主研方向：高性能计算机体系结构；张民选，教授、博士；蒋 江，讲师、博士
收稿日期：2007-06-29 **E-mail：**freesunnybird@gmail.com

DSMT 在超标量处理器的基础上，对同时多线程的取指、现场切换、指令退出(Retire)提供相应的硬件支持，同时增加 TraceCache 等动态线程提取机制，以及前瞻(Speculation)多线程动态控制机制。线程共享其他执行资源，硬件设计人员可以将主要精力集中在构建一个快速的单线程超标量处理器上，然后在此基础上增加支持多个线程切换的能力和硬件动态提取线程的能力。这一方式的好处在于设计目标明确、附加硬件少、开发周期短，能有效利用芯片面积和资源共享。

由 HP 和 Intel 开发的 EPIC 体系结构允许编译器把程序的指令级并行性直接传达给硬件，以解决存储延时和误取开销。通过采用 EPIC，软件开发 ILP 的指令窗口会远远大于硬件开发 ILP 的指令窗口，可以开发更多并行性；由于编译器完成了并行性开发的主要工作，硬件的设计相对简单，能够将主要的芯片面积用于提供大量的计算资源，有利于提高微处理器的性能；微处理器硬件设计复杂度相对较低，有利于提高芯片的频率，缩短设计周期。

结合DSMT和EPIC技术，EDSMT尽量利用Itanium的现有资源[4]，通过较少的改动获得较大的性能提高。图 1 描述了EDSMT的微体系结构。来自多个不同线程的指令包(每个指令包有 3 条指令)，通过指令分派逻辑分发到由多个功能部件组成的功能池。多线程的调度由指令分派部件完成。调度的原则为尽量减少相关性，提高并行度。

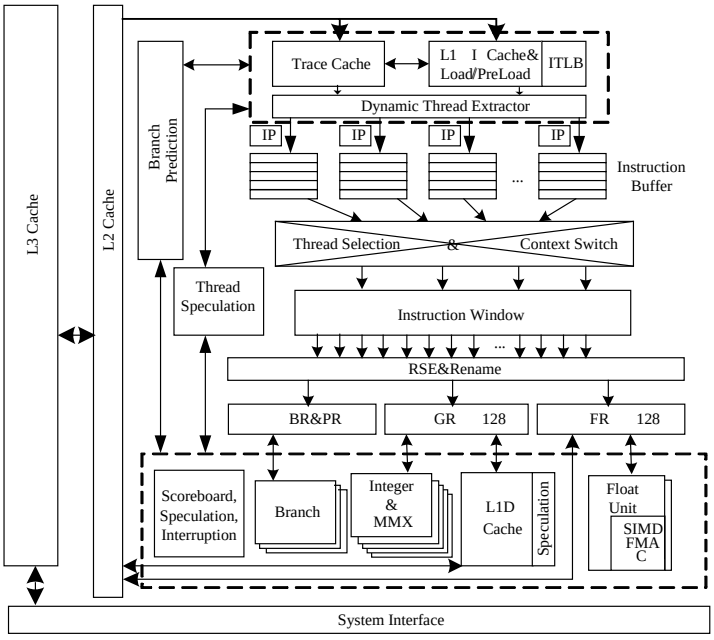

图 1　EDSMT 的微体系结构

2.3　EDSMTSIM 设计

模拟器在体系结构设计中有着重要的作用，EDSMT 作为一种新的体系结构，笔者为它开发了新的模拟器——EDSMTSIM。

EDSMTSIM 采用了踪迹驱动的微处理器模拟方式，参考了 SMTSIM 一些设计技术。所谓踪迹驱动，就是将指令的实际执行和流水线中的控制分离。流水线不负责指令实际译码执行，只负责处理相关检测、分支解决、异常处理等控制逻辑，而由专门的功能模块来模拟指令的实际执行。流水线在执行中调用该功能模块对指令进行模拟。这样可以将指令集的执行模拟和流水线的复杂控制分开，简化了软件模拟器的开发工作，使流水线将主要精力放在了相关检测、分支解决、异常处理等复杂逻辑上。

3　EDSMT 中的寄存器栈以及寄存器栈引擎

IA-64 中通用寄存器被分成 2 个子集，通用寄存器 0~31 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~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 0~96(从GR32 开始)之间变化[4]。

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 及 fill)。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地进行，而栈寄存器通过寄存器栈引擎(RSE)自动进行这些操作。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寄存器(backing store)。

3.1　基于多线程的寄存器管理机制 MTRM

处理器中的物理寄存器资源是有限而宝贵的，复杂的 Itanium 处理器也只使用了 128 个通用物理堆栈寄存器，通过 RSE 机制对这些隐式寄存器加以分配利用。RSE 技术的实现方案为了支持软件流水及寄存器旋转等关键技术，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。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文件实现方案——Mapping Table-based Register Management (MTRM)，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 2 为 MTRM 机制的逻辑示意图。

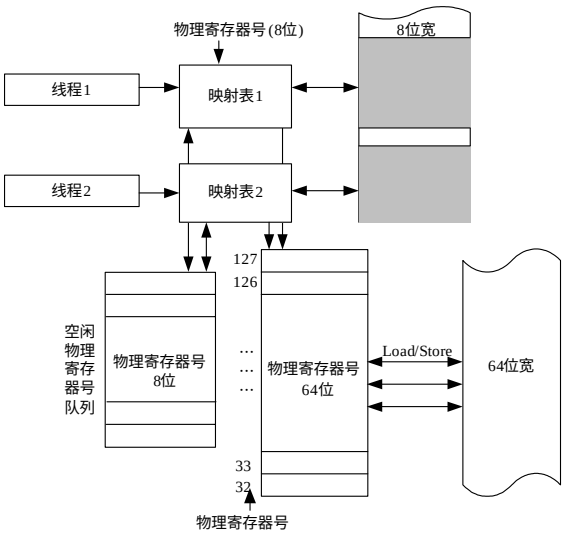

图 2　基于多线程的寄存器管理机制 MTRM

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。使用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，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。

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，具体的实现方案可以表述为：

(1)使用一个地址入口号从 32~127 的映射表(96 个地址号)来实现寄存器的重命名，分配给过程的逻辑栈帧中连续的逻辑寄存器号重命名为映射表栈帧中的连续地址号(即虚拟的连续物理寄存器号，Itanium 中是重命名到物理栈帧中连续的物理寄存器号)，映射表中填入从空闲物理寄存器号队列中取的物理寄存器号(可以存在不连续的情况)。物理寄存器号可以在需要的时候填入，不需要时删除，这样不仅充分地使用寄存器资源，而且也不影响分配给过程的映射表栈帧地址入口号的连续性，同样支持软件流水和寄存器的旋转技术。

(2)逻辑寄存器栈帧起始号是 32，其对应 Itanium 结构中物理寄存器栈帧起始号为 Rse.bof。若用映射表实现其重命名后，逻辑寄存器栈帧起始号 32 对应的依然是 Rse.bof，只是此时 Rse.bof 代表的是映射表地址入口号中的一个。映射表中填入和删除物理寄存器号的实现对软件透明。因此 Itanium 系列微处理器中 RSE 的实现方案同样适用于映射表实现方案，只是原相对物理寄存器号的操作现转变为对映射表地址入口号的操作。

不同于 Itanium 结构中“逻辑寄存器-物理寄存器-后备存储区”3 层结构的方案，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 实验结果

本文使用 NAS 的并行测试程序(NPB)作为测试标准，以 EDSMTSIM 和 SMTSIM 为模拟器，对比了基于 IA-64 的 EDSMT 结构和 Alpha 上的 SMT 的性能。

首先，对 OpenUH 编译器与同样是开源的 Omni 1.6 编译器进行了比较，实验以 NPB 3.2 的子集作为测试程序并使用了 Class A 的数据集。编译时使用–O2 的编译选项，2 个编译器编译的结果在 4 线程的 EDSMTSIM 上运行，图 3 显示了这些测试程序 OpenUH 相对于 Omni 的 IPC 的提升。尽管 OpenUH 设计的时候更多地考虑的是其可移植性，这样必然会在给定的平台上损失一定的性能，但它还是在大多数测试程序上获得了比 Omni 更高的性能(除了 EP)，平均加速为 2.055。实验结果表明 OpenUH 是一款适合于安腾平台并行研究的 OpenMP 编译器。

由于 OpenUH 编译器具有良好的可移植性和有效性，因此，使用它作为前端来为被测系统编译 NPB 2.3 OpenMP/C 测试程序，而目标机器的本地 GCC 则用来作为后端编译器以生成多线程的代码和进行链接。实验采用的是数据集 A，使用–O2 的编译选项，目标机器为 4 线程。图 4 显示了针对 7 个测试程序 EDSMT 相对 Alpha 的 SMT 的 IPC 加速比，其中 EDSMT 在多数情况下都优于 Alpha 的 SMT(除了 EP 和 SP)，其平均加速为 12.48%，可见 EDSMT 对于并行应用来说是一种十分有效的体系结构。

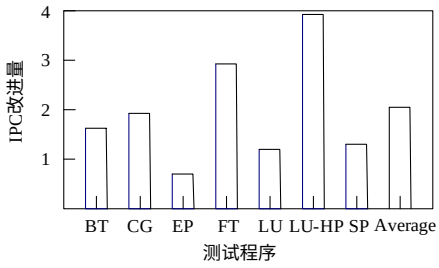

图 3　使用 NPB 3.2 OpenUH 相对 Omni 的 IPC 加速比

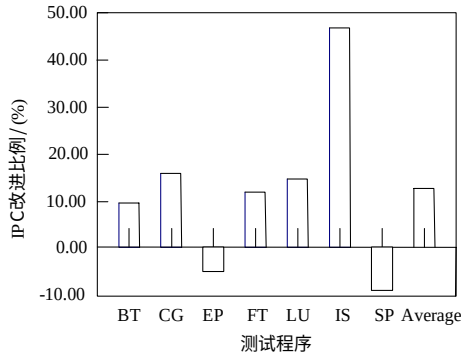

图 4　使用 EDSMT 相对 Alpha 上的 SMT 的 IPC 加速比

5 结束语

由于典型的服务器工作负载仅有有限的指令级并行和较多的访存延迟，对于它们而言，与超标量指令发射相关的大量的硬件实际上并没有起到相应的作用。大量的高吞吐率计算需求，使得现在 SMT 和 CMP 渐渐成为了市场的主流。在微处理器采用多核多线程的同时，为了充分发挥这些结构的优势，软件也需要过渡到并行编程上去。

本文提出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。实验结果表明，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。同时笔者研究了 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。它能使得单线程以及多线程程序的寄存器使用时更加有效。

今后，笔者将着重研究其他的一些优化技术：取指策略，资源组织和分配，缓存结构，线程调度等；同时致力于用精简的 EDSMT 核来构建同构以及异构的 CMP 系统。

参考文献

[1] Schlansker M S, Rau B R. EPIC: Explicitly Parallel Instruction Computing[J]. Computer, 2000, 32(2): 37-45.

[2] 尉红梅, 姚建华. 并行语言及编译技术现状和发展趋势[J]. 计算机工程, 2004, 30(z1): 97-98.

[3] 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.

[4] McNairy C, Soltis D. Itanium 2 Processor Microarchitecture[J]. IEEE Micro, 2003, 20(5): 44-55.