OpenSMT:一个同时多线程处理器模拟器的设计和实现*)

路 放[1] 安 虹[1,2] 梁 博[1] 任 建[1]
(中国科学技术大学计算机科学技术系 合肥 230027)[1]
(中国科学院计算技术研究所系统结构研究室 北京 100080)[2]

摘 要 同时多线程(SMT)技术是目前微处理器体系结构的研究热点之一。为了支持对SMT技术和基于SMT核的单芯片多处理器(CMP)体系结构技术的深入研究,我们在广泛使用的超标体系结构模拟器SimpleScalar的基础上,通过对SMT结构的关键特性进行适当的抽象,开发了一个SMT体系结构模拟器OpenSMT。本文介绍了该模拟器主要的设计思想和实现方法,包括多个线程上下文结构的表示、超标量流水线各个阶段的模拟,以及模拟器设计和实现时需要解决的几个关键问题等。初步的应用研究表明,与现有可免费获得的研究用SMT模拟器相比,该模拟器能够较好地平衡模拟性能、灵活性和精度三个基本设计目标,实现了执行驱动、易于扩展指令集结构、良好的用户接口、灵活的软件结构、适宜评估更广泛的SMT体系结构设计空间等设计要求。

关键词 处理器模拟器,同时多线程,软件模型,结构模型,性能评价

OpenSMT: The Design and Implementation of a Simulator for Simultaneous Multithreading Processor Architecture

LU Fang[1] AN Hong[1,2] LIANG Bo[1] REN Jian[1]
(Department of Computer Science and Technology, University of Science and Technology of China, Hefei 230026)[1]
(Computer Architecture Laboratory, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100086)[2]

Abstract Simultaneous multithreading (SMT) becomes one of the major trends in the future micro-processor design. In order to provide a platform to support the research about SMT processor or CMP architecture with SMT cores, the SMT simulator(OpenSMT) is built on the SimpleSalar Tool Set by abstracting the key characters of SMT architecture properly. This paper describes the main features of this simulator including structure of multithreading context, the detailed designs about each pipeline stages, and the approaches to simulate the multithreaded micro-processor. Compared with other free SMT simulators, OpenSMT balances the tradeoff among the performance, flexibility and accuracy. In addition, OpenSMT uses the execution-driven technique, has the flexible software model and allows the configuration of a large set of architectural parameters to evaluate the SMT architecture.

Keywords Processor simulator, Simultaneous multithreading(SMT), Software model, Architecture model, Performance evaluation

1 引言

当前,随着工艺和应用模式的发展,无论是高端还是低端、通用还是嵌入式,传统的采用指令级并行的微处理器体系结构正在由单线程向多线程、单核向多核发展。例如,Intel已在其Pentium 4中采用了同时多线程(Simultaneous MultiThreading,SMT)技术[1],命名为超线程(Hyperthreading)技术[2],并宣称进一步将采用CMP技术。IBM的Power 4芯片采用的是单芯片多处理器(Chip MultiProcessors,CMP)结构[3],在单个芯片上集成了两个4发射超标量处理器;Power5进一步引入了SMT技术,基于SMT核构建CMP结构[4]。此外,已经中止开发的Alpha 21464(EV8)芯片,采用的也是SMT结构[5]。

随着线程级并行处理器体系结构的发展,传统的单线程单处理体系结构模型正逐步转向多线程多处理体系结构模型,广为流行的超标量结构模拟器SimpleScalar[6,7]已经不能胜任新的研究任务。已有一些专用的多线程模拟器被开发出来,例如从SimpleScalar扩展而来的多处理器模拟器SIMCA[8]、支持SMT技术的SMTSIM[9]。不幸的是,这些模拟器只针对固定的系统结构和指令集,模拟器的软件模型不够灵活,在其基础上很难开展其他独立的研究。因此,开发一个通用的多线程处理器的结构模型,制定一套多线程处理器的性能评测模型就变得十分重要。本文主要介绍我们基于SimpleScalar工具集开发的一个SMT结构模拟器OpenSMT的主要设计思想和实现方法。OpenSMT继承了SimpleScalar的许多优点,包括软件结构模型和可扩展性等,可以在各种体系结构配置下获得有效的性能统计数据,用以评估各种SMT体系结构的设计方案和关键技术。

*)本文工作得到以下项目的资助:国家自然科学基金资助项目(60373043);安徽省自然科学基金资助项目(050420206);国家863高科技发展计划资助项目(2001AA111100)和(2002AA110010);中国科学院知识创新工程重大项目(KGCX2-109)。路 放 硕士生,主要研究方向为多线程处理器体系结构、处理器模拟器;安 虹 博士,副教授,主要研究方向为计算机体系结构、并行计算;梁 博 博士生,主要研究方向为性能评估模型、网络处理器;任 建 硕士生,主要研究方向为分支预测。

本文其余部分的组织如下：第 2 节介绍 OpenSMT 的设计目标、开发策略以及 SimpleScalar 模拟器的结构；第 3 节详细叙述 SMT 体系结构的模型和 OpenSMT 模拟器中流水线的相关设计；第 4 节重点讨论 OpenSMT 设计过程中解决的几个关键问题；第 5 节通过对 SMT 处理器一级指令 cache 的性能评估来检验 OpenSMT 的设计目标；最后给出全文的总结。

2 设计目标和开发策略

OpenSMT 是在 SimpleScalar3.0d 基础上开发出来的，开发过程分为以下几步：①确立设计目标和开发策略。②分析一般的处理器模拟器的软件组织结构。③评测几个现有的 SMT 模拟器，并重点分析 SimpleScalar 模拟器。④建立要模拟的 SMT 处理器体系结构模型并确定需要模拟的关键结构特性。⑤解决模拟器设计中的几个关键问题。⑥编程实现。⑦实验评估和试用改进。下面具体介绍上述开发过程中所做的工作。

2.1 设计目标

处理器模拟器软件模型的设计必须综合考虑模拟的性能、灵活性、精度(或复杂性)三个主要设计目标。性能是指在用于模拟的机器资源已经给定的情况下，模型所能承受的模拟工作量。灵活性是指对模型设计进行修改时的难易程度，以及为多种甚至完全不同的设计方案建模的难易程度。精度则定义了模型对处理器体系结构的抽象程度，一个高度复杂的模型应该能够如实地模拟出处理器运行的各个方面。

但是，实际上要同时满足这三个目标是很困难的，所以大部分的模型在实现时只优化其中的一两个目标，这就是为什么现在有这么多的软件模型存在，甚至对一个简单的处理器设计都是这样。用于研究的模型更倾向于牺牲一些模拟精度来换取更好的模拟性能和灵活性，从而有利于计算机体系研究人员减少工作代价并获得更多研究成果，这也是我们设计 OpenSMT 的一个期望。

OpenSMT 模拟器的设计目标为：①不仅为单纯的 SMT 处理器结构研究提供一个通用的模拟实验平台，而且希望在此基础上能方便地进一步构造基于 SMT 核 CMP 结构的模拟器，用于支持对多核体系结构的比较性研究。②采用执行驱动(Execution Driven)的模拟技术，能真实地再现模拟程序执行的全过程，包括推测执行、程序执行时发生的冲突等。③易于扩展的指令集结构，目前的实现要支持 MIPS 指令集体系结构。④良好的用户接口便于使用，灵活的软件结构便于修改。⑤提供更多的结构配置方案，便于评估更广泛的微体系结构设计空间。⑥作为一个研究用工具而非工程设计用工具，因此不针对具体的处理器结构设计。⑦将模拟性能控制在可接受的范围内。

2.2 开发策略

由于现代微处理器结构的复杂性，重新开发一个完整的模拟器，工作量非常大，并且需要较长时间的使用才能验证其正确性，因此选择一个经过使用检验的模拟器作为开发基础是更加行之有效的方案。我们曾试图通过改进现有的 SMT 模拟器来达到上述设计目标，所以首先分析和评测了两个可以免费获得的 SMT 模拟器。一个是 SMTSIM [9]，它是华盛顿大学 Susan J. Eggers 领导的研究小组早期为研究和评估 SMT 技术而开发的模拟器，它支持多种取指令的算法和分支预测的方案，模拟准确性高。但是它只提供了对 Alpha 指令集的模拟，流水线的结构也主要针对 Alpha 处理器，可考察的设计空间小，无法满足一般的比较性研究工作的需要。SS-SMT[10]是另一个基于 SimpleScalar 开发出来的 SMT 模拟器，它大量地复制了 SimpleScalar 中的结构，使得其模拟效率不高，其特殊的 Cache 设计模型也无法客观地反映标准的超标量体系结构模型。并且，我们在评测时发现，该模拟器中存在着许多错误，例如无法处理多个线程负载的顺序退出等。

SimpleScalar 工具集是作为 Wisconsin 大学的 Multiscalar 项目[11]的一部分，在 Gurindar Sohi 的指导下于 1992 年开发出来的。通过不断的版本更新发展到今天，它为不同的体系结构建模和模拟提供了一个基础平台，支持用户通过二次开发，扩展其中的工具包来完成更多的体系结构建模任务，现已成为研究界使用最多、最为成功的超标量结构模拟器。该工具包支持对多种流行的指令集体系结构的模拟，包括 Alpha、Power PC、x86、以及 ARM 等，提供了多个微体系结构模型：从简单的非流水线结构到复杂的具有多级存储层次的动态调度微体系结构，并能在保证一定模拟性能的同时，支持向更多体系结构和微体系结构模型的扩展，以适应多种多样的处理器体系结构研究任务。例如，sim-safe 仅模拟指令集体系结构，而 sim-outorder 则能模拟复杂的动态调度、推测执行、多级存储系统等微体系结构。该模拟器在早期的设计时就考虑了模拟性能、灵活性和精度三方面设计目标的协调，提供了较完整的指令级并行处理器的性能评估模型。有关该模拟器的详细介绍可参见文[7]。以 SimpleScalar 为基础开发一个 SMT 模拟器，可以减少开发许多常用的建模组件，节约大量时间。例如，微体系结构建模组件：分支预测、指令队列以及 Cache 等；辅助建模组件：指令集的仿真、I/O 系统仿真、离散事件的管理等。同时，还可以借用原有的软件框架，继承该模拟器原有的许多优点，减少完全重新设计可能引入的大量错误，使得我们的开发工作更专注于向 SMT 结构的改造。因此，最后我们决定基于 SimpleScalar 模拟器自主开发一个 SMT 模拟器。

2.3 处理器体系结构模拟器的软件结构模型

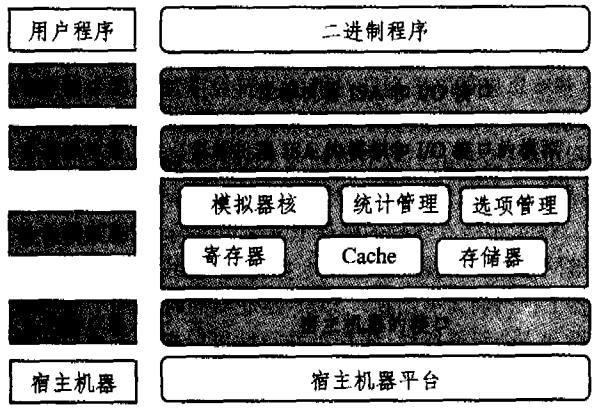

图 1 一般的处理器体系结构模拟器软件结构模型

一般的处理器模拟器(以下简称模拟器)介于被执行的代码和主机系统之间，软件结构大致可以分为四层：程序接口层、功能模拟层、性能模拟层、模拟器和主机的接口层(见图 1)。其中最重要部分是功能模拟层，实现指令仿真、I/O 仿真和系统调用；性能模拟层，包括模拟器内核、标准结构组件模块(如寄存器、Cache、存储器等)；以及一些辅助模块，包括程序装载器、选项管理器、统计参数管理器等。

2.4 sim-outorder 模拟器的流水线结构

OpenSMT 通过修改 SimpleScalar 工具集中的 sim-outorder 模拟器得到。为了便于与修改后的结构进行对比，我们先简单介绍一下 sim-outorder 模拟器的流水线结构，详细描述参见文[12]。sim-outorder 模拟了一个标准的超标量结构，包括指令和数据 Cache，分支预测，寄存器重命名，乱序执行等。这个模拟器利用 RUU(Register Update Unit)[13] 来控制指令的重命名和依赖关系，RUU 使用一个重定序缓存(Reorder Buffer)来自动地进行寄存器的重命名以及保存没有经过提交阶段的指令的运算结果。sim-outorder 的流水线如图 2 所示，分为 5 个阶段：取指令、调度、发射和执行、写回、提交。

在取指令阶段，处理器根据当前程序计数器(PC)和分支预测器从指令 Cache 中取出相应 Cache 行的指令，放到指令队列(IFQ)中。如果发生指令 Cache 不命中，处理器进入阻塞状态，直到将所需要的指令读入指令 Cache。

指令调度阶段同时负责进行指令译码和寄存器重命名。这个阶段的任务是从指令队列中取出指令译码，重命名寄存器，分配 RUU 和 LSQ(load/store 指令队列)，将所有源操作数已准备好的指令放入就绪队列(RQ)中。同时将内存操作分成两个独立的指令，分别用来进行有效地址计算和真正的内存操作。

指令的发射和执行被合并在一个阶段完成，主要任务是为处在就绪队列中的指令分配执行所需资源并执行指令。在每一周期内，只要功能部件空闲，且还没有达到发射带宽，则从就绪队列中发射尽可能多的指令到功能部件。最后，根据各个功能部件时间延迟对写回事件进行调度。

写回阶段的任务是解决数据依赖和误预测状态。在每一个周期，它都扫描是否存在已经完成的指令。如果存在，则根据指令的输出来查询依赖关系链，并将那些正在依赖该输出的指令进行标记。如果某条指令被标记，所有操作数都就绪，那么该指令可以发射到就绪队列。如果发生误预测，流水线将被刷新。

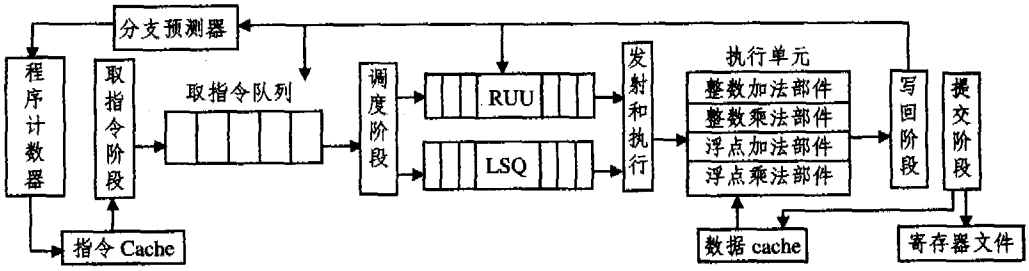

图 2 sim-outorder 模拟器流水线的结构图

虽然指令可以乱序地发射和执行，但是必须顺序提交。在提交阶段，要更新数据 Cache、内存，同时将这些已经执行完、等待提交的指令从 RUU 或 LSQ 中删除。

下面我们将详细介绍如何对 sim-outorder 中流水线各个阶段进行改造得到 OpenSMT。

3 SMT 处理器体系结构模型及其模拟

3.1 基本原理

SMT 是多线程与超标量自然结合产生的微体系结构模型，基本思想是以指令发射槽的并行粒度加速多个线程的执行，最大可能地实现宽发射和乱序执行，提高处理器运算部件的利用率，缓和由于数据依赖或 Cache 不命中带来的访存延迟，所付出的代价是略微增加了发射和控制逻辑的复杂性。

SMT 中所有线程的硬件上下文可以同时激活，每个周期能够同时从多个线程中取指令执行，多个线程可以在单个处理器上执行而无需切换。SMT 处理器同时利用了线程级并行性和指令级并行性，线程级并行性可以来自多道程序设计系统负载中的多个独立的程序，也可以来自并行程序设计；单个线程中利用的是指令级并行性。来自不同线程的指令是相互独立的，可以被同时发射执行，理论上可以利用全部的发射带宽。单个线程执行过程中出现的延迟可以通过切换到处理器中其它线程来隐藏，从而将无用的发射槽限制在一个周期以内。由于一个 SMT 处理器同时利用了粗粒度(线程级)和细粒度(指令级)的并行性，因而更有效地使用了资源。SMT 每个周期竞争所有可得到的资源，更加合理地使用更宽和更深的流水线。对多线程(或多道程序设计)工作负载，比单线程的超标量处理器达到了更好的吞吐量和加速比。

根据一般的 SMT 结构特征和我们打算支持的关键技术研究，我们对要模拟的 SMT 结构特性进行了以下适当的抽取和模拟。

3.2 多个线程上下文结构的表示

多线程处理器要求硬件支持在多个线程的上下文间切换执行，为了提供这种机制，处理器必须复制每个线程运行的状态。相对于 SimpleScalar 中的单线程结构，OpenSMT 中每个线程需要有自己的上下文，并在线程切换的时候保留和恢复对应线程的上下文。一个线程的上下文主要包括：(1)一组独立的寄存器文件；(2)一个独立的程序计数器(PC)；(3)线程的编号 Thread_ID；(4)线程执行的状态位；(5)独立的统计数据。

我们将所有线程的上下文存储在指针数组 context* the-contexts[MAXTHREADS]中，通过线程的编号 Thread_ID 索引到对应线程上下文的指针。上下文相当于是处理器状态的一份副本，当线程需要切换的时候，将处理器状态保存到当前线程的上下文中，修改其执行状态为非活跃；并用新选中的切换线程的上下文将处理器恢复到上次执行时处理器的状态，修改其执行状态为活跃。

除了上下文的组织和切换，OpenSMT 模拟的多线程处理器控制程序结束的方法也与 SimpleScalar 不同。在 SimpleScalar 中，一旦遇到要求退出模拟的 SS_SYS_exit 系统调用，就执行 longjmp()函数跳回到 main()函数中，打印统计数据并结束模拟。OpenSMT 由于要支持多线程，当一个线程遇到 SS_SYS_exit 系统调用时，仅表示该线程结束，我们将其结束状态位设为“真”，在每次选择线程时也要检查结束状态位，跳过已经结束的线程。并且，在执行 SS_SYS_exit 系统调用时，要将一个全局的线程计数器减 1(初始值为运行的线程数)，当其为 0 时，程序才返回到 main()函数，结束统计

数据并退出。

3.3 基本流水线的模拟

OpenSMT 对 sim-outorder 模拟器流水线的改造包括：①对流水线中取指令、写回、提交等几个阶段的改造。②对硬件资源根据多个线程共享或独占的不同方式进行相应的扩展。③对寄存器重命名和误预测时流水线的刷新也要做相应的改变。图 3 描述了当线程数为 4 时的 OpenSMT 的结构图（简记为 OpenSMT-4），灰色部分为改造后的与流水线相关的部件。下面具体介绍几个重要流水阶段的模拟问题。

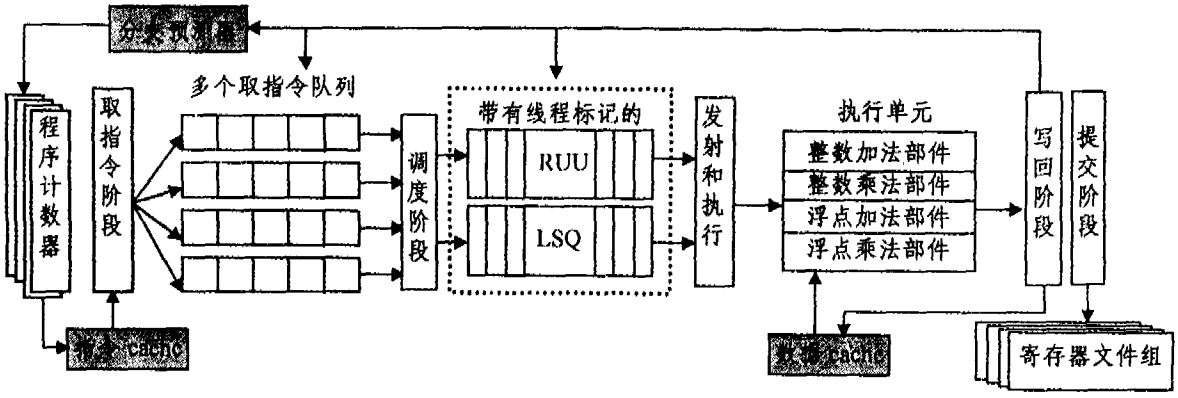

图 3 OpenSMT-4 模拟器的结构图

3.4 取指令阶段

在 OpenSMT 抽象出的 SMT 结构定义中，多个线程的指令在流水线中混合执行，取指令单元与 sim-outorder 有两个重要的区别：①每个时钟周期取指令单元必须从多个线程中取指令。②需要一个调度算法来选择每次被取指令的线程。

如图 3 所示，除了在多线程上下文中要为每个线程提供独立的程序计数器以外，在 sim-outorder 中要为每个线程复制一份取指令队列，取指令队列按照 Thread_ID 索引，每个线程取出的指令对应填充到自己的取指令队列。这种独占方式的好处将在下面的寄存器重命名中体现出来。在我们目前实现的模拟器中，线程的调度算法采用略加修改的时间片轮转（Round-Robin）算法：每个时钟周期从备选的线程中顺序选择两个线程，先从第一个被选择的线程取指令，尽可能填充取指令带宽。如果未填满，再从第二个线程中取指令，下个周期依次循环进行。根据华盛顿大学的研究，该算法能够在硬件设计的复杂度和性能之间达到很好的折衷[14]。

sim-outorder 在调度阶段从指令队列中取出指令译码，将 load/store 指令放入 LSQ 队列，其他指令放入 RUU 队列。相比之下，OpneSMT 中对 SMT 的寄存器重命名是这个阶段最重要的变化。SMT 的每个线程都拥有独立的寄存器文件，只有同一个线程的指令间才有真正的依赖关系，寄存器的重命名不能简单地只判断输入输出的寄存器名。采用线程独立的取指令队列正是为了简化寄存器重命名的设计，这样在调度阶段，访问每个取指令队列就意味着访问的是同一个线程的指令，每个取指令队列的寄存器重命名只需根据输入、输出关系建立依赖链，将所有源操作数已准备好的指令放入就绪队列 RQ 中。需要注意的是必须在依赖链和 RQ 中增加 Thread_ID 标志。最后，循环遍历每一个取指令队列，直到全部完成。

3.5 发射阶段和执行阶段

在发射和执行阶段，因为调度阶段已经建立了指令间的依赖关系，所以混合多个线程的指令和数据通路，不会造成线程间的混淆。OpenSMT 的 SMT 结构可以利用 sim-outorder 已经有的许多硬件机制，根据建立好的依赖关系按照动态调度的方式乱序发射和执行。

3.6 写回阶段

在 sim-outorder 中，写回阶段如果发现误预测，即表示整个流水线走到了一条错误的执行路径上，需要刷新 RUU 中所有的指令并把程序计数器恢复到预测失败的地方。然而，当流水线中同时执行多个线程发生误预测时，并不是所有的线程都需要刷新。模拟器必须有能力仅仅刷新发生误预测的那个线程在 RUU 中的指令。为了完成这一点，OpenSMT 在调度阶段在 RUU 中利用了 Thread_ID，通过遍历 RUU 队列，仅将发生误预测的当前线程的指令刷新，而不改变其他线程指令的状态。

3.7 提交阶段

在调度阶段，对取指令队列的寄存器重命名可能会破坏取指令的初始顺序，但单个线程的取指令顺序是不会被破坏的，因为每个线程运行的是相互独立的工作负载。这样，只要在提交阶段使每个线程按顺序提交，就可以保持每个程序执行的正确性。此外，每个线程在提交时必须按照 Thread_ID 来更新各自的上下文。

4 设计中几个需要注意的关键问题

OpenSMT 模拟器除了改造 sim-outorder 流水线结构以外，为了支持多线程结构，其他模块也有很多需要重新设计的地方，其中主要包括多个线程的内存地址空间、多线程结构中 Cache 的模拟，性能统计数据的收集等。

4.1 内存地址空间对多个多线程的扩展方法

OpenSMT 模拟器面临的第一个关键问题是如何模拟多个线程的地址空间。SimpleScalar 系统的内存寻址空间为 31 位（2GB），采用段页式的寻址方式，没有模拟内存下一级的存储介质（如硬盘），工作负载在模拟初期就要装载入内存，不考虑运行时内存的缺页异常。在 OpenSMT 中，为了使每个线程拥有独立的虚拟地址空间，即每个线程的逻辑地址仍然保持 31 位，必须能够控制每个线程从虚拟地址到实际物理地址的映射。

如图 4 所示，当工作负载程序要访问模拟的存储器时，在进行虚拟地址转换之前，模拟器根据当前线程的编号从存储器的索引数组中找到对应的存储器指针，每个存储器指针指向线程独立的一块本地机的内存空间，这段内存空间被用来模拟对应线程虚拟的存储器。代码段从 0x00400000 地址开

始，用来存放被模拟程序的二进制代码，数据段从0x10000000地址开始，堆栈段用来保存函数的参数和环境变量，从0x7fffc000开始从高地址向低地址增长。这样，可以在保持单个线程原有的虚拟地址空间的同时，又通过存储器的索引数组实现了多个线程实际地址空间的分离。并且以指针索引每个线程的虚拟存储器，保证了模拟器在切换线程时的效率。

因为没有模拟每个进程真正独立的页表，模拟的多线程存储地址空间模型与实际的处理器设计并不一致。但由于每个线程模拟的工作负载是彼此独立的应用，线程之间没有通讯和访问同一个物理地址的干扰。因此，在这种设计下可以正确地模拟存储器的功能和性能情况。

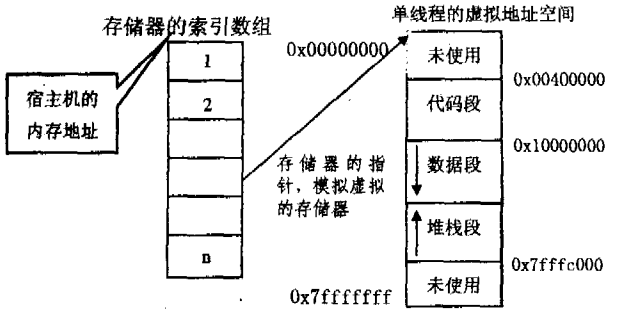

图4 存储器地址空间

4.2 多线程结构中Cache的模拟

OpenSMT继承了SimpleScalar对单个线程段地址的约定，对SimpleScalar的程序装载器(loader)的变化相对较小，只需要针对每一个工作负载程序，开辟一段新的内存空间并记录在存储器的索引数组中，多次调用loader完成所有程序的装载。

由于OpenSMT中单个线程的地址空间设计并没有改变SimpleScalar中原来单个工作负载程序逻辑地址的分布，并且Cache模块只是模拟Cache访问延迟的特性，并不返回真正访问的数据，因此这种设计使得OpenSMT对Cache的修改工作量大大减少。首先在原有的Cache_blk_t结构中(Cache_blk_t结构描述Cache块的信息，包括存储的数据、tag位等)增加线程标识符，并在Cache初始化时置为-1，即表示目前没有任何一个线程占有Cache行。在读Cache时，除了比较tag位，同时也要判断线程标识符是否和当前线程的Thread_ID相同，来确定Cache是否命中。同样，写Cache行时，要把线程标识符设置成当前线程的Thread_ID。

4.3 性能统计数据的收集

每时钟周期完成的指令数(IPC)是传统单线程处理器性能模型中重要的评估指标，在其基础上扩展的全局IPC可以用来衡量SMT等多线程处理器的性能，计算公式如下：

$$\text{全局 IPC}=\left(\sum_{i=1}^{n}\text{第 } i \text{ 个线程的指令数}\right)/\text{机器的执行周期} \quad (1)$$

多个线程性能统计数据的收集方法，主要结合SimpleScalar中统计模块的优点，在每个线程上下文中保存一个stat_sdb_t类型的统计项链表，用来统计单个线程的独立信息(例如该线程工作负载的执行周期和指令数)，从而得到此线程独立的IPC。同时，模拟器要维护一个全局的统计数据库，统计整个模拟器的执行周期，根据式(1)计算出处理器的全局IPC。

全局IPC描述了处理器整体的运算速度和指令吞吐量，客观地反映整个处理器的性能。单个线程的IPC则描述每个线程的运行特性，可以反映多线程执行对每个线程的影响。

5 一个简单的应用实例

本节我们通过一个初步的应用实例，说明如何用OpenSMT模拟器对SMT处理器的一级指令Cache做性能评估。

SMT与传统的超标量结构一个重要的不同在于取指令阶段，由于要从多个线程中同时取指令，访问内存地址的冲突会造成指令Cache效率的下降，因此对一级指令Cache的研究就变得十分重要。本实验从SPEC CPU2000精简测试程序[15]中选取2个整型程序gcc和perlbmk、2个浮点程序equake和ammp。模拟器的配置参数如下：①流水线的取指令和发射宽度为4。②7个功能部件：3个整数运算单元、3个浮点运算单元、1个访存单元。③二级指令和数据的统一Cache，大小为256kB，相联度为4，访问延迟设为6个机器周期。④一级Cache的访问周期为1个机器周期。从图5中可以看出，随着一级指令Cache的增大，Cache的不命中率会逐渐减小。指令Cache取8kB，当同时线程数在1，2，3和4之间变化时，OpenSMT-4的不命中率明显高于其他三种，相当于OpenSMT-2的1.6倍。当指令Cache增大到16kB时，OpenSMT-4的不命中率便有明显的减少，仅高于OpenSMT-3。当指令Cache大小再进一步增大到32kB时，OpenSMT-4的一级Cache达到最小的不命中率。通过这个模拟实验，我们可以发现SMT结构中一级指令Cache的特殊行为，即在一级指令Cache不够大的情况下，由于多个线程之间取指令容易造成地址冲突，处理器不停地把Cache行换进换出，发生了严重的抖动，造成很高的不命中率。从上面对线程数目的分析，结合Cache大小的硬件复杂度，可以认为32kB是一级指令Cache大小的当指令折衷配置方案。

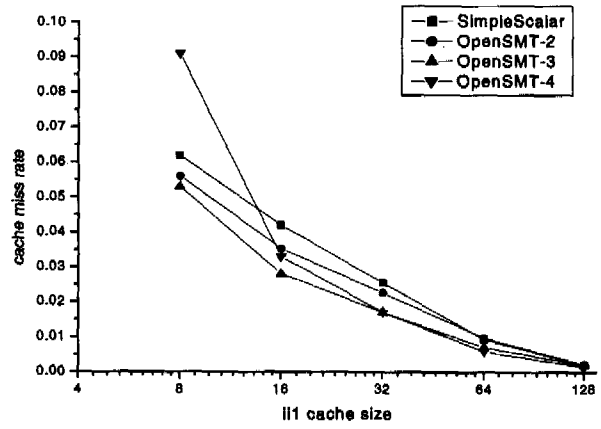

图5 一级指令Cache的大小对Cache不命中率的影响
(OpneSMT-n表示同时运行n个线程)

我们课题组目前正在使用OpenSMT模拟器进行SMT体系结构其它方面的研究工作。例如，通过改造该模拟器的分支预测器，针对几种典型的应用和几种著名的分支预测方案，研究在SMT中存在单条分支和多条分支的情况下，不同的分支预测方案对分支预测精度和处理器整体性能的影响，为设计SMT处理器时采用何种分支处理方案提供参考。相关的研究结果将于近期发表。

总结 我们设计和开发OpenSMT模拟器的主要目的是

为SMT处理器体系结构的研究提供一个通用的模拟实验平台。在开发过程中，我们从广泛使用的SimpleScalar工具集入手，使工作重点可以集中在SMT结构的模拟设计，有效地节约了开发的周期。同时，工作中也暴露出一些问题，例如SimpleScalar的全局变量和宏造成模拟器程序难于调试；伴随着结构的复杂导致模拟器性能的下降(同时运行四个工作负载时大概是SimpleScalar运行时间的4倍到5倍)。目前，我们已经通过了一些简单的应用实例，验证了OpenSMT模拟器的可用性、准确性和灵活性，基本达到了设计目标要求。通过小组内使用其开展SMT体系结构相关研究，将进一步检验和改进其设计和实现，待其基本稳定后，将作为开源程序公开发布。

未来进一步的开发工作将主要从以下三方面展开：第一，在OpenSMT基础上进一步构造基于SMT核的CMP结构的模拟器，用于支持对多线程多核体系结构的比较性研究。第二，支持对复杂的工作环境的模拟，例如对操作系统、网络环境的模拟，这样才能更加真实准确地反映桌面和网络环境的系统特性。第三，建立更加完备的多线程性能评估模型。传统超标量结构的性能评估模型已经不能适应新兴的多线程处理器结构的特征，必须有新的评价指标来对多线程处理器的性能进行有效评估。

参 考 文 献

1 Tullsen D M, Eggers S J, Levy H M. Simultaneous multithreading: Maximizing on-chip parallelism. In: 22nd Annual International Symposium on Computer Architecture, 1995. 392～403
2 Marr D T, Binns F, Hill D L, et al. Hyper-threading technology architecture and microarchitecture. Intel Technology Journal, 2002, 6(1): 4～15
3 Diefendorff K. Power4 Focuses on Memory Bandwidth. Microprocessor Report, 1999, 13(13): 11～17
4 Clabes J, et al. Design and implementatin of the power5 microprocessor. In: ISSCC Digest of Technical Papers, 2004. 56～57
5 Diefendorff K. Compaq Choose SMT for Alpha. Microprocessor Report, 1999, 13(16)
6 Austin T, Larson E, Ernst D. SimpleScalar: An Infrastructure for Computer System Modeling. IEEE Computer, 2002, 35(2): 59～67
7 Burger D, Austin T M. The SimpleScalar tool set version 2.0: [Technical Report1342]. Computer Sciences Department, University of Wisconsin, 1997
8 Huang J, Lilja D J. An Efficient Strategy for Developing a Simulator for a Novel Concurrent Multithreaded Processor Architecture. In: Proceedings of the6th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 1998
9 Tullsen D M, Lo J, Sirer G, et al The SMTSIM Simulator. http://www.cs.ucsd.edu/users/tullsen/smtsim.html
10 Goncalves R, Ayguade E, et al. A Simulator for SMT Architectures: Evaluating Instruction Cache Topologies. In: 12th Symposium on Computer Architecture and High Performance Computing, 2000. 279～286
11 Sohi G S, Breach S, Vijaykumar T N. Multiscalar Processors. In: 22nd International Symposium on Computer Architecture (ISCA-22), 1995
12 Austin T, Ernst D, et al. SimpleScalar Tutorial(for release 4.0). http://simplescalar.com/docs/simple_tutorial_v4.pdf
13 Smith J E, Sohi G S. The Microarchitecture of SuperScalar Processors. Proceedings of the IEEE, 1995, 83(12): 1609～1624
14 Tullsen D M, Eggers S J, Emer J S, et al. Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor. In: Proc. Int'l. Symp Computer Architecture, ACM, 1996. 191～202
15 Standard Performance Evaluation Corporation. SPEC CPU2000 Benchmarks. http://www.spec.org/osg/cpu2000/, Dec. 2001

(上接第146页)

也验证了，分解后的子网通过同步合成和共享合成均可得到原先的网系统，从系统合成的角度而言，这也是一条可行之路。综合文[7,12]和本文的工作，可得到一种Petri网系统合成与分解的体系，如图4所示。

图4 Petri网的合成与分解技术

参 考 文 献

1 袁崇义. Petri网原理. 电子工业出版社，1998
2 Peterson J. Petri net theory and the modeling of systems. 吴哲辉译，徐州：中国矿业大学出版社，1989
3 Hyunglee K. Generalized Petri Net Reduction Method. IEEE Transaction on Systems, Man and Cybernetics, 1987, SMC17(2)
4 Suzuki I. A Method for Stepwise Refinement and Abstraction of Petri Nets. J. of Computer and System Sciences, 1983, 27: 51～76
5 王培良，等. P/T系统的和分解. 计算机科学，1999，26：147～151
6 王培良，等. Petri网的并分解. 控制理论与应用，2001，18(1)：116～118
7 曾庆田，吴哲辉. 基于库所指标的Petri网分解方法. 计算机科学，2002，29(4)：15～17
8 曾庆田，吴哲辉. 类S-图的语言性质分析. 计算机科学，2002，29(5)：120～123
9 曾庆田，吴哲辉. Petri网语言的同步交运算. 小型微型计算机系统，2004年录用，待发表
10 曾庆田，吴哲辉. 基于分解的结构复杂Petri网的行为描述. 系统工程学报，2003年录用，待发表
11 曾庆田，吴哲辉. Petri网分解的保性条件分析. 小型微型计算机系统，2003年录用，待发表
12 蒋昌俊. Petri网的动态不变性. 中国科学(A辑)，1997，27(6)：567～573
13 蒋昌俊. 离散事件动态系统的PN机理论. 北京：科学出版社，2000