OpenCMP:一个支持事务存储模型的多核处理器模拟器*)

何裕南[1] 安 虹[1,2] 郭 锐[1] 梁 博[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部 北京 100080)[2]

摘 要 CPU设计正在由仅开发指令级并行性的单线程单核结构转向利用线程级并行性的多线程多核结构,但至今还没有一个可移植性好并被广泛使用的开源多核处理器模拟器,限制了在这样的结构上开展高质量的研究工作。我们开发了一个多核处理器体系结构模拟器OpenCMP,用于支持当前和未来对多线程多核处理器体系结构关键技术的研究。该模拟器适当地抽象了多核处理器结构,为主流的多核处理器结构研究提供一个可扩展、灵活的模拟工具框架,包括支持对乱序、顺序的处理器核和同时多线程处理器核的模拟,以便对更大的多核设计空间进行比较性研究。本文以支持事务存储模型的多核处理器结构模拟器为例,详细描述了如何通过抽象多核结构和事务存储模型的最基本特性和组成部分,扩展单核处理器模拟器SimpleScalar,设计与实现一个多核处理器模拟器。初步研究表明,与现有的多核处理器模拟器相比,该模拟器能够较好地支持对事务存储模型和基于事务存储模型的多核处理器体系结构的研究。

关键词 处理器模拟器,单芯片多处理器,事务存储模型,软件模型

OpenCMP: A Simulator for CMP with Transactional Memory Model

HE Yu-Nan[1] AN Hong[1,2] GUO Rui[1] LIANG Bo[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 For CPU design, a shift of focus from exploiting instruction-level parallelism with single-thread and single-core to exploiting thread-level parallelism with techniques like Chip Multi-Processing(CMP) and Multi-Threading(MT) have been witnessed. Up to now, the lack of portable and widely disseminated simulators for CMP systems has doubtless contributed to a lower quantity of research occurring in those areas. We develope a CMP processor architectures simulator called OpenCMP as evaluation infrastructure to support current and future investigations on CMP architecture. The tools set provides a scalable and flexible simulating framework for mainstream CMP machines, in which each core is flexible to simulate out-of-order cores and simple in-order cores, and have Simultaneous MultiThreading(SMT) support as well. This will allow comparative studies. This paper describes the design of OpenCMP with transactional memory model, shows how to abstract the key characteristics and components of CMP and transactional memory model, and how to build the simulator based on the SimpleSalar Tool Set. Compared with other existing CMP simulators, OpenCMP has the flexible software model and allows the configuration of a large set of architectural parameters to evaluate the CMP architecture with transactional memory model.

Keywords Processor simulator, Chip multi-processor, Transactional memory, Software model

1 引言

上个世纪80年代,CPU设计的主要转变是从CISC指令集体系结构转向RISC指令集体系结构,使得CPU单核的设计更加简单、高效。90年代的CPU设计主要致力于努力提高CPU单核的指令级并行性(instruction-level parallelism, ILP)和主频。最终导致单核的复杂性和功耗急剧上升,线延迟问题开始限制这种集中控制的单核结构继续发展,设计和验证的复杂度越来越成为改变结构的主导因素。

近年来,我们目睹了CPU设计正在由仅开发ILP的单线程单核结构转向利用线程级并行性(Thread-Level Parallelism, TLP)的单芯片多处理器(Chip Multi-Processing or Chip Multi-Processor, CMP)和多线程(Multi-Threading)结构。例如,Intel已在其Pentium4中采用了同时多线程(Simultaneous MultiThreading, SMT)技术[1],命名为超线程(Hyper-threading)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]。CMP/MP结构的基本思想是将多个较简单的

*)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);Intel高等教育项目(PO#4507176412)。何裕南 硕士生,主要研究方向为多线程多核处理器体系结构;安 虹 博士,副教授,主要研究方向为并行计算机体系结构、微处理器芯片体系结构;郭 锐 硕士生,主要研究方向为多线程多核处理器体系结构;梁 博 博士生,主要研究方向为多线程多核处理器体系结构。

核集成到单个硅片上，在每个核上执行一到多个线程应用(包括多道程序设计系统中的多个进程，由并行程序设计产生的并发线程，以及硬件产生的线程等各种粒度的线程)，通过增加核数和线程数，使得芯片的总体性能随处理器核的增加具有更好的可扩展性，隐藏各种 ILP 技术无法克服的长延迟事件，例如访存和分支处理，有效地控制功耗，降低芯片系统设计和验证的复杂度。

CMP/MP 结构的出现使得传统的单线程单核处理器体系结构模拟器，如超标量结构模拟器 SimpleScalar[5,6] 已经不能胜任新的研究任务。已有一些 CMP 模拟器被开发出来，例如 Wisconsin 大学的 GEMS 模拟器[7]，Urbana-Champaign 的 SESC[8]，Michigan 大学的 M5[9]等。但是至今还没有一个可移植性和可扩展性好并被广泛使用的开源 CMP/MP 模拟器，限制了在 CMP/MP 结构上开展高质量的研究工作。我们结合我们小组正在开展的多线程多核处理器体系结构关键技术研究工作的需要，参考现有的模拟器设计，包括 SimpleScalar[5]、SimOS[10]、SESC[8]和 M5[9]，渐进地开发了一个多线程多核处理器模拟器 OpenCMP，用于支持当前和未来对多线程多核处理器体系结构关键技术的研究。OpenCMP 将设计成一个工具套件，当前实现的设计目标是能够评估具有 2，4，8，16 个线程处理单元的结构。随着工艺的发展，可以进一步扩展到用于评估具有 32，64，128，256 个线程处理单元的结构。我们期望该工具集能够像 SimpleScalar 适当地抽象单核超标量处理器结构那样，适当地抽象多核处理器结构，为主流 CMP/MP 结构的模拟提供一个灵活的框架，包括支持乱序和顺序的处理器核，单线程和同时多线程(Simultaneous MultiThreading，SMT)的处理器核等等，以便对更大的 CMP/MP 设计空间进行比较性研究。

本文以开发一个支持事务存储模型的多核处理器结构模拟器为例，详细描述了支持事务存储模型的多核结构模拟器 OpenCMP 的主要设计思想和实现方法。在 CMP 结构上采用由用户显式制导的并行程序设计模型，使用锁和同步变量来实现同步的方法存在很大的局限性。粗粒度锁不易开发并行性；细粒度锁会带来较大的额外系统开销；并且锁机制会导致死锁、优先权倒置、编程困难等种种问题[13]。事务存储(Transactional Memory，TM)模型提供了一种在 CMP 结构上程序并行执行和同步的方法，能够解决由锁机制带来的种种问题，提高程序的并发性。已有大量的研究工作致力于硬件支持的事务存储模型的研究[13]，充分显示出该存储模型可能被未来主流 CMP 结构采纳的巨大潜力，但目前还没有一个可扩展性好且开源的 CMP 模拟器能够提供硬件支持的事务存储模型的 CMP 结构模拟。开发一个支持事务存储模型的 CMP 结构模拟器，建立相应的性能评估模型，对于开展 CMP 结构上的事务存储模型的研究十分重要。

本文的其余部分组织如下：第 2 节先介绍了 OpenCMP 的设计方法；第 3 节讨论了设计和实现中的关键问题；第 4 节详细描述了支持事务存储的 CMP 模拟器 OpenCMP 的设计与实现；第 5 节给出了在 OpenCMP 上进行事务存储模型研究的一个实例；最后总结了该模拟器目前的研制现状，并简单介绍了我们正在进行的下一步开发工作。

2 设计目标和实现策略

为了在 CMP/MP 体系结构上开展事务存储模型的研究，需要设计并实现一个支持事务存储模型的 CMP/MP 体系结构模拟器。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，所以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。SimpleScalar 是一种被计算机体系结构研究者广泛使用的计算机系统模拟和建模工具，具有源代码开放，便于修改和移植的特点。SimpleScalar 工具集[6]是作为 Wisconsin 大学的 Multiscalar 项目[12]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，现已成为在研究界使用最多、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、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[11]。

以 SimpleScalar 为基础开发一个模拟器，可以减少开发许多常用的建模组件，节约大量时间。OpenCMP 是基于 SimpleScalar3.0d 开发的支持事务存储 CMP 模拟器，通过扩展单核处理器到多核处理器，并加入事务存储模型的支持而实现。

3 设计实现中的关键问题

我们通过修改 SimpleScalar 工具集中的 sim-outorder 模拟器，得到支持硬件事务存储模型的 CMP 体系结构模拟器 OpenCMP。为了模拟 CMP 结构和事务存储模型，sim-outorder 模拟器中的很多模块需要重新设计和实现。OpenCMP 模拟器实现的最基本问题是如何从单核处理器向多核处理器扩展。SimpleScalar 作为单核处理器模拟器，不能胜任多核处理器的模拟研究工作，将单核模拟器中的资源部件扩展是设计实现的最基本要求。本文主要策重于介绍如何加入事务存储模型。

3.1 事务存储模型简介

在事务存储模型中，事务(Transaction)是并行处理和调度的基本单位。一个事务就是一段连续的指令流，这段指令流中所有读写共享存储空间的操作作为一个整体被原子性地提交到共享存储空间中[13]。事务存储模型使得程序在发生数据冲突时才串行化执行，最大限度地挖掘程序中潜在的并行性。

事务存储模型中的事务有如下特性：(1)原子性：一个事务要么被完成且提交了对存储器的所有修改；要么被取消，并且丢弃该事务执行期间对存储器所做的所有修改。(2)可串行性：虽然事务的执行可能是并发的，但是事务好像被一个接着一个那样串行执行。事务的执行不会交叠，事务的提交顺序对于所有的处理器单元是相同的序。多个事务的并发执行与按某一次序串行地执行它们时的结果相同。

事务的概念最早出现在数据库领域，通过事务来保证对数据库并发访问的正确性。事务存储模型发展了数据库中事务的概念，文[14]将事务的概念引入了并行计算领域。硬件事务存储模型(Hardware transactional memory，HTM)[13]提出使用硬件的支持来实现事务的原子性和可串行性，利用 Cache 及其一致性来实现无锁(lock-free)的同步。HTM 是由 Herlihy 和 Mossi[13]在 1993 年最早实现的。Stanford 大学的

Transactional memory coherence and consistency(TCC)结构[15,16],以及 Unbounded TM(UTM)[17], Thread-level TM (TTM)[18], Virtualized TM(VTM)[19]也都是使用硬件支持事务存储模型的实现方案。

在事务存储模型中,一个事务的生命周期分为4个阶段(如图1所示):(1)初始化阶段。一个事务在执行前需要初始化,包括初始化事务执行所需要的上下文等信息。(2)执行阶段。一个事务执行的时候,需要缓存访问共享存储空间的读写集合,并侦听其余处理单元上事务提交的写集合的信息。如果一个事务在执行的过程中监听到发生了数据依赖冲突,则该事务就要回退,重新执行。(3)等待提交阶段。事务经过执行阶段后就进入等待提交阶段,此阶段要判断事务是否可以提交。如果事务可以提交,则进入提交阶段;如果在等待提交的过程中发生了数据依赖冲突,则取消本次执行,回退重新执行。(4)提交阶段。事务的写集合被广播到其余事务,并且提交到共享存储空间。至此,一个事务完成了它的生命周期。

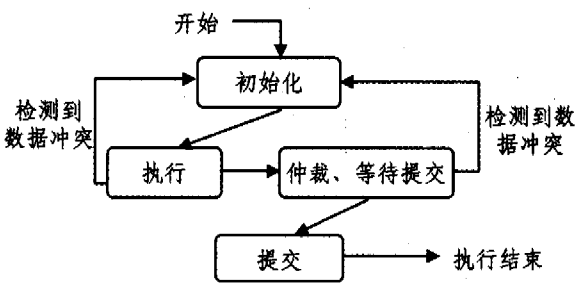


图1 事务执行的生命周期模型

3.2 如何加入对事务存储模型的支持

为了建立事务存储CMP结构软件模型,需要对事务存储结构进行抽象得到事务存储模型最基本的组成成分。

事务存储CMP结构必须满足事务存储模型的原子性和可串行性。事务存储模型的原子性可以通过缓存所有读写集合和统一提交来实现,事务的可串行性可以通过执行时的数据依赖检测和回退执行来实现。

通过抽象,事务存储模型可以分为以下四个模块:(1)缓存访存操作模块。对于共享空间的修改必须被缓存。(2)检测数据依赖冲突模块。在并行执行过程中,如果事务的读集合被其它已提交的事务修改,就需要提供数据依赖冲突检测来发现RAW冲突,以保证程序执行的正确性。(3)登记初始注册点,取消执行和回退执行模块。如果检测到了数据依赖冲突,就必须取消正在执行的事务,并把处理器的状态回退到执行此事务初始注册点(最开始的上下文)。(4)统一提交事务模块。执行结束并且在执行的过程中没有发生数据依赖冲突,就准备提交事务,这个阶段分为两个过程,一个是把事务对共享地址空间的修改提交到共享存储空间,另一个是提交的事务将所有对共享地址空间的修改通知其余处理单元,由其余处理单元执行冲突检测。

OpenCMP模拟器扩展了传统的CMP结构,加入了事务存储模型这种特殊结构的支持,如何加入事务存储的主要模块是模拟器设计的关键问题。如果可以通过扩展CMP模拟器中各功能部件来实现事务模型的原子性和可串行性,则修改变得简单可行。

OpenCMP模拟器通过软硬件来实现事务执行的控制,软件控制通过中断来触发。SimpleScalar模拟器缺乏对于中断产生和处理过程的模拟,为了提供事务执行的控制行为,需要添加事务中断的模拟功能。

SimpleScalar中的Cache模块只是模拟Cache访存延迟的特性,没有真正访问数据,也没有缓存数据。SimpleScalar没有模拟内部总线,所有访存延迟预先算好。这些特征不易于模拟多核结构,多核结构访存是随机行为,需要通过总线仲裁机制来实现,所以访存延迟是随机的,因此需要通过添加随机访存特性的模拟或者根据具体目标应用集合的访存模型来模拟。此外,由于事务存储模型写操作需要缓存数据,所以需要添加存储数据的功能。

3.3 如何模拟多核处理器的地址空间

模拟器设计面临的另一个问题是如何模拟多核处理器的地址空间,SimpleScalar系统的内存寻址空间为31位,采用段页式寻址方式,没有模拟内存的下一级的存储介质,工作负载在初期就要装入内存。在OpenCMP中如何为每个核上运行的线程提供虚拟地址空间是设计需要解决的问题之一。

3.4 如何加入事务存储的性能评价模型

事务存储CMP结构作为一种特殊的体系结构,性能评价模型添加了对于事务存储模型特性的参数统计功能。结合SimpleScalar中统计模块的优点进行存储模型性能统计数据的收集是模拟器设计和性能评价的关键之处。

后面将结合本文所设计的事务存储CMP结构来说明在模拟器设计中上述问题的解决办法和实现方法。

4 设计与实现

本文所设计的事务存储CMP结构通过扩展现有的基于总线侦听技术的CMP结构来实现。OpenCMP模拟器对于SimpleScalar扩展主要包括:单处理器结构到多核CMP处理器结构的扩展,添加了事务控制协处理单元,改变了Cache的访存特性,中断处理的扩展等。

OpenCMP对于从单处理器结构到多核CMP处理器结构的扩展采用了Stanfrod大学的Hydra CMP[20]结构设计,本文限于篇幅主要策重于介绍如何加入事务存储模型。下面将做详细介绍。

4.1 OpenCMP的基本结构

OpenCMP中每个处理单元通过一级数据Cache、写操作地址队列缓存和行换出地址缓存处理读写集合的缓存,在提交阶段提交到二级Cache。数据依赖的检测采用运行时的动态检测,处理单元侦听总线,与私有的一级数据Cache比较,当发生数据依赖的时候产生中断,通知处理单元产生了RAW数据依赖。回退执行通过设立初始注册点技术(log)来实现,处理单元执行当前事务时初始上下文被保存,回退执行时,重新载入事务初始时的上下文。

图2所示是OpenCMP中单个处理节点,每个处理节点都包含一个处理单元,处理单元可以是单发射或者多发射超标量结构(可配置)。每个处理单元都有一个协处理单元,协处理单元负责处理与事务相关的中断以及优先级控制。协处理单元中的优先级寄存器保存了事务的优先级级别。

处理节点之间通过总线相连。为了利用事务提交时大量数据写回的特性,总线设计采用独立的写回总线以提供写回操作的带宽。控制总线负责处理节点之间通讯,OpenCMP不采用处理器之间寄存器直接通讯,节点之间通过控制命令进行通讯,控制总线不需要仲裁,只有最高优先级的事务才具有发送控制命令的权力。总线仲裁采用分布式的仲裁策略。

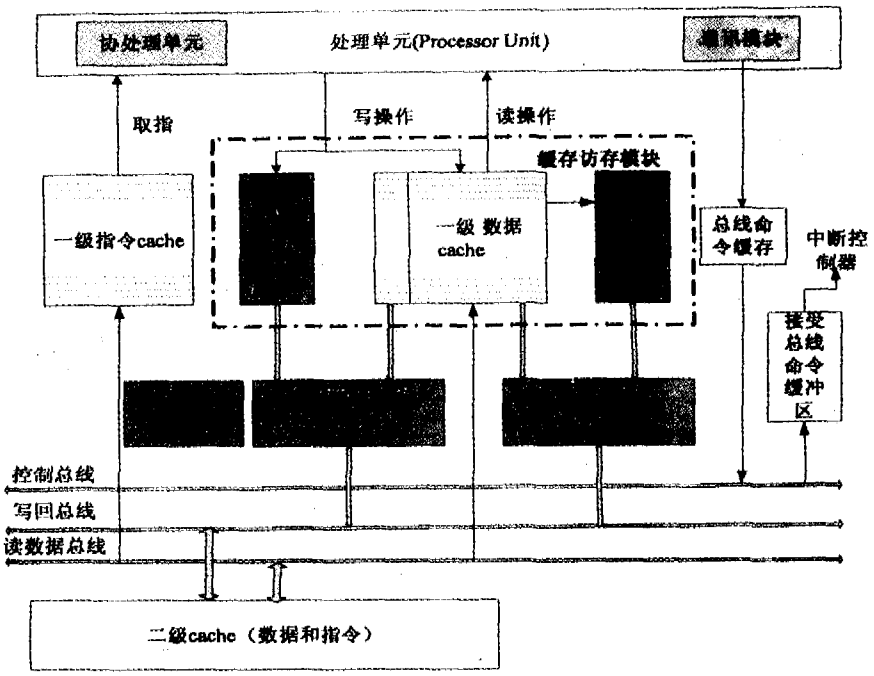


图 2　OpenCMP 单个节点结构

处理节点拥有私有的一级数据 Cache，共享二级 Cache。一级 Cache 采用写回(write-back)、写分配(write-allocate)的策略。一级数据 Cache 负责缓存数据和侦听总线的功能。读写操作都要缓存在一级数据 Cache 中，事务提交前不允许将一级数据 Cache 行丢弃或者写回。在提交阶段将修改的数据全部提交到二级 Cache 中。数据写回通过提交控制器逻辑实现。为了保证一致性，数据 Cache 采用监听写回总线来解决数据依赖冲突。一级指令 Cache 可以是每个节点私有的，也可以共享同一个一级指令 Cache。如果主要针对循环体做并行化，则在并发执行时，每个处理单元上执行的指令基本相同，用共享的指令 Cache 可以减少访存的缺失。而私有的指令 Cache 更易于结构的可扩展性。目前我们的结构实现的是共享指令 Cache。每个处理单元拥有一个写操作地址队列缓存和一个行换出地址队列缓存，用来保存事务读写的踪迹。

4.2　事务处理指令集

SimpleScalar 支持多种指令集，OpenCMP 扩展了类 MIPS 的 pisa 指令集版本，加入对于事务存储模型的支持。OpenCMP 中事务执行状态控制的指令可分为 3 类：(1)通讯指令，(2)缓存控制指令，(3)协处理器指令。

通讯指令负责处理单元之间的通讯，由通讯模块实现。处理单元之间通过命令总线进行通讯。处理单元需要与其余处理单元进行通讯的时候，会发出总线指令，通过总线通讯模块将相应的总线指令翻译成总线命令。通讯指令包括 startloop，commit，kill3 条。startloop 产生 X_start 总线命令，告诉其余的处理单元开始执行。commit 指令产生 X_commit 总线命令，通知其余处理单元已经执行完了当前的事务。kill 指令产生 X_kill 总线命令，取消所有处理单元的执行。

缓存控制命令包括 clearcache，clearbuffer 和 commitbuffer。clearcache，clearbuffer 分别将 cache 和写操作地址队列缓存、行换出地址队列缓存中所有的内容选择性清除，清除工作由缓存模块处理。而 commitbuffer 则将对于共享地址空间的写集合进行提交。

协处理器指令是用来在事务寄存器和通用寄存器之间交换数据的，包括 mfc2 和 mtc2 两条。

通过修改 pisa. def 中的指令宏定义和 pisa. h 中 MD_SET_OPCODE 宏定义可以实现对于添加指令的扩展功能。

4.3　事务处理寄存器

SimpleScalar 的 pisa 版本中带有 32 个整数和 32 个浮点寄存器，另外，pisa 中还有其他一些常用寄存器，如 pc 寄存器用来保存程序计数器，hi、lo 寄存器可以分别保存乘除运算结果的高位和低位等其他寄存器。

为了扩展事务存储模型的支持，OpenCMP 加入了事务寄存器。这些寄存器可分为 2 类：(1)事务状态寄存器，(2)事务中断寄存器。

事务状态寄存器保存了事务执行的状态，这些寄存器包括：处理器状态寄存器 state_reg，设置为 0 表示处于 nop 状态，设置为 1 处于非 nop 状态；优先级最高寄存器 head_reg，当处理单元为最高优先级时设置该寄存器；推测优先级寄存器 spec_priv_reg 设置了处理单元的优先级，及其他的寄存器等。

事务中断寄存器保存了由于事务控制产生中断而使用的寄存器，包括：事务中断源寄存器 spec_cause 用来指明中断源；事务中断处理程序地址寄存器 spec_handler 存放有关事务中断的处理程序地址。以及 spec_epc 保存中断返回的地址等。

4.4　基于侦听的 Cache 一致性和存储同一性

OpenCMP 使用了文[15]中提出的一致性协议。事务在执行过程中缓存所有的写集合，当事务执行完毕，系统检查事务是否能够提交，如果能够提交，就将事务的写集合提交到总线上。其余处理单元侦听总线来保证一致性，一致性协议能够保证事务之间的同步，简化了存储同一性模型。

OpenCMP 能够保证 Cache 一致性。事务在执行过程中缓存所有的写集合，在执行的过程中不需要传统的 MESI 或者 Dragon 一致性协议的支持，一级私有数据 Cache 中可以包含推测修改过的数据，也可以包含别的处理单元正在修改的

数据，其中的数据是该节点的一个私有拷贝。当事务执行完毕，利用总线侦听技术，将提交事务的写集合广播到其余处理节点，同时，其余处理单元检测自己使用的共享数据，如果发现事务所读的共享数据被提交事务修改，则需要取消事务的执行，装入新的值，重新执行。硬件提供了数据依赖的检测。同时，由于事务提交的原子性，WAR 和 WAW 以来被掩盖。该 Cache 一致性协议实现简单。

OpenCMP 能够保证存储同一性。OpenCMP 中事务存在优先级的属性。执行模型中，优先级高的事务应该先于优先级低的事务提交，且事务提交写集合是原子性的。所以，一个优先级高的事务所有的写操作一定会在程序上先于优先级低的事务的写操作。而由于采用了回退机制，所有发生依赖的读写操作都会被回退机制强行确定先后次序。

4.5 缓存访存模块

OpenCMP 使用一级数据 Cache 来保存写操作的结果。每个处理节点都包含私有的一级数据 Cache，Cache 中每个块有 3 个状态位 V、R、W，分别表示有效、已读、已写。为了减少写操作对于 Cache 行状态的影响，OpenCMP 中数据 Cache 中每个字可被设置一个 W 标志位。OpenCMP 修改了一级数据 Cache 行为，所有的写操作结果只能在提交的阶段被写回二级 Cache。对于 Cache 行为的改变，通过修改 mem_access 函数行为来实现。

OpenCMP 结构中每个处理节点增加了写操作地址队列缓存结构 write_address_t 和行换出地址队列缓存结构 read_address_t。read_address_t 包含地址 address、tag 等，为了实现简单 write_address_t 除了包含有 address、tag 等，还包含了修改的数据 data。

写操作地址队列缓存通过首尾相连 write_address_t 结构构成链表结构保存了所有写指令地址。加入写操作地址队列缓存能够提供事务存储模型一次性提交数据的功能，OpenCMP 在事务提交后，需要将事务对于共享存储空间的修改提交到二级 Cache 中，将所有的写操作地址缓存起来便于将写集合回写，由于 SimpleScalar 中是在 ruu_dispatch 阶段进行指令的执行，而在 cache 模块只进行延迟的模拟，所以，将 write_address_t 中保存了数据 data，当进行 read 操作时，就从 write_address_t 中的 data 中取数据，如果缺失就去内存中取，所有的写指令将数据写入 write_address_t 的 data 数据结构中，简化了实现。

同时 OpenCMP 提供了一个行换出地址队列缓存，由 read_address_t 结构组成链表，用来缓存所有将被换出的且只被执行了读操作的 Cache 行，进行查找只需要遍历链表。

对于缓存溢出情况，OpenCMP 系统采用停等方式和动态拆分事务的方式来处理缓存益处。当一个事务执行时发生缓存已满的情况，则当前事务停止执行，等待成为优先级最高的事务。当事务优先级别处于最高时，硬件就会将当前事务的读写集合提交，但是不发送 X_commit 总线命令，系统优先级不发生改变，当所有的读写集合提交完成，事务继续执行，直到事务执行完毕，则事务作为一个普通事务进行提交，发送 X_commit 总线命令。

4.6 检测数据依赖模块

事务存储模型的事务之间可能存在数据依赖。事务执行过程中，所有的事务需要在提交阶段之前等待优先级更高的事务去提交，提交过程是串行的。所以，事务间 WAR 冲突和 WAW 冲突能够被这种顺序提交机制消除，不需要额外的支持。对于事务间真数据依赖 RAW，就必须提供运行时数据依赖冲突检测的机制。

OpenCMP 结构提供运行时的数据依赖检测来保证程序运行的正确性。为了能够提供数据依赖的检测，数据 Cache 采用侦听写回总线的方法。OpenCMP 结构中每个处理节点都包含私有的一级数据 Cache。当数据被载入时则 cache_blk_t 结构中的 tag 被置位 READ，这个状态位将一直伴随着直到事务提交，写操作不会改变读操作位的状态。

当一个事务执行完毕以后，事务缓存的所有写集合将被提交到写回总线上，其余事务所在的处理节点侦听写回总线的地址总线。侦听工作由侦听部件来完成，OpenCMP 中 snoopy 模块负责依赖检测，写回总线的地址总线上的每个地址都将被 snoopy 模块缓存，然后取出与该处理节点中一级数据 Cache 中 tag 位为 READ 的块地址作比较，同时，也要与行换出的所有读集合的地址进行比较，如果发现地址相同，则产生中断，通过中断控制器来通知处理单元。snoopy 函数在每个周期的 ruu_fetch 后执行，通过保存在总线结构中的源数据来实现冲突检测。

当检测到数据依赖，则会产生中断 VIOLATION_INT，需要取消事务的执行，装入初始执行的上下文，重新执行。

4.7 控制提交模块

当一个事务执行完毕，且优先级最高，则该事务可以提交，这些工作由 trans_commit 模块完成。OpenCMP 中 commit 指令触发 trans_commit 函数执行，该函数将遍历 write_back_t 组成的写回地址队列中的地址和数据，然后提交到总线结构的缓存中。事务的提交需要把写集合依次提交到总线上，将短时间占用很大的总线带宽，OpenCMP 结构采用独立的回写总线来提供带宽需求。

Trans_commit 当所有的写操作提交完成后，将设置该处理节点需要执行的下一次迭代的上下文。

4.8 中断模拟模块和系统调用模拟

OpenCMP 中与事务相关的中断包括：处理节点侦听总线命令 X_start、X_commit、X_kill 所产生的中断 COMMIT_INT、STLOOP_INT 、KILL_INT，以及由一级数据 Cache 侦听到发生 RAW 数据依赖冲突的中断 VIOLATION_INT。与事务相关的中断处理程序由每个处理节点在当前事务执行的上下文中运行，由于中断处理程序本身可能产生数据依赖，所以中断处理程序的执行是位于 uncached 的地址空间中，中断处理程序访存不通过 Cache。

OpenCMP 中与事务相关的中断由事务中断模块来实现。interrupt []数组保存了处理单元的待处理中断，数组由 PID 确定所属的处理单元。函数 interrupt_handle 是中断硬件处理的主要函数，负责屏蔽中断，精确定位中断点，产生中断处理向量。在函数实现的过程中为了简化设计，事务中断将排空流水线。interrupt_handle 函数在每个周期的 snoopy 侦听函数后执行，每个处理单元每次循环都需要执行。

OpenCMP 中系统调用交给当前的优先级最高的线程运行，如果当前发出系统调用的节点还不是最高优先级，则等待当前事务处理成为优先级最高才能处理系统调用。OpenCMP 对于系统调用采用了串行化的模拟方法，只有当前事务有提交权限才能做处理，保证处理不会被取消。

4.9 Cache 及总线的模拟

SimpleScalar 中 Cache 模块只是模拟 Cache 访存延迟的特性，没有真正访问数据，也没有缓存数据。SimpleScalar 没

有模拟内部总线，所有访存延迟预先算好。这些特征不易于模拟多核结构。

多核结构中，访存延迟受到多核争夺总线影响，是一个随机的行为。为了模拟访存延迟，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 arb_status 状态位。arb_status 状态位包括 IDLE，WAIT，ARB，DONE 四个状态。Cache 访存函数由 Cache_access 扩展为数据 Cache 读操作 dCache_read、数据 Cache 写操作 dCache_write、以及指令 Cache 的 iCache_access。其次需要加入总线模拟，添加总线数据结构 bus_t，结构包括 read_bus_free 数据，指示 bus 需要多久才空闲。

dCache_read 和 dCache_write 需要仲裁总线，为了模拟总线仲裁，需要提前计算 Cache 访存的延迟。当一个访存操作发出，如果 Cache 发生缺失，当前 Cache 行就进入 IDLE 状态。如果此时总线是空闲的，则计算该次延迟进入并 ARB 状态等待。如果总线不空闲，则计算还需要多久才能空闲，返回需要等待值并且跳转到 WAIT 状态，等待时间等于此等待值后才又进行访存试探。当条件满足，则进入 ARB。ARB 状态中，各个处理单元会将计算出来的延迟进行比较，得到总线的处理单元进入 DONE 状态，处理后并返回 IDLE，否则节点在 ARB 状态继续等待。所以状态的比较基于 Cache 的行进行。

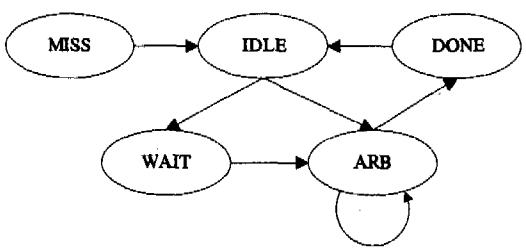


图 3　Cache 块状态转换图

4.10　内存地址空间扩展

OpenCMP 中所有的事务共享数据段、代码段以及堆空间，但是每个事务都有独立的栈空间。SimpleScalar 中每个存储器指针指向一块本机的内存空间用来模拟对应模拟空间的存储器。被模拟的应用程序的代码段在模拟空间中从 0x00400000 地址开始存放，数据段从 0x10000000 开始存放，栈从 07fffc000 开始从高地址向低地址扩展。为了支持多个处理单元，我们分隔栈空间，每个处理单元占用 0x10000 的栈空间。

OpenCMP 中将中断处理程序和被模拟的应用程序统一编译，编址到统一的内存空间中，不需要修改程序装载器 loader。

4.11　性能统计数据的收集

OpenCMP 性能统计数据的收集方法，主要结合 SimpleScalar 中统计模块的优点，每个核保存一个 stat_sdb_t 类型的统计项链表，用来统计单个核的独立信息。

OpenCMP 扩展的 SinmpleScalar 性能参数可分为 3 大类：(1)指令数和执行时间相关的性能参数；(2) 事务存储模型添加模块的统计信息，包括写操作地址队列缓存、行换出地址队列缓存等的基本信息，以及如数据依赖发生的次数、事务提交次数、中断产生次数、中断处理的延迟；(3)总线性能统计，包括事务每次提交所需要的周期数等。

5　一个性能分析的实例

本文选择了 JavaGrande 中的 FFT kernel 程序的 C 语言版本，通过手工编译将其编成事务模型的转换源代码，进行源到源的变换。产生的代码通过交叉编译后，在我们的模拟器 OpenCMP 上模拟，本次试验模拟器配置如表 1 所示(模拟器是动态可配置的)。本文用该模拟器模拟了 2～16 个线程的情况。通过比较在不同处理器个数上，OpenCMP 的 IPC 的增幅来判断其性能。

表 1　模拟器配置参数

处理单元	单发射或者多发射，(可配置)
一级数据 Cache	私有，16k，写回，4 路组相联，128 字节的块大小，两个读端口，一个写端口，延迟为 1，(可配置)
一级指令 Cache	16k，直接映射，128 字节的块大小，两个读端口，延迟为 1，(可配置)
二级 Cache	256k，写回，4 路组相联，512 字节的块大小，两个读端口，两个写端口，缺失延迟 100 拍，命中延迟 6。(可配置)
写总线带宽：128 字节	128 字节。(可配置)
写操作地址队列缓存	32k 全相联，1 字块大小，延迟为 1。(可配置)
行换出地址缓存	32k，1 字块大小，延迟为 1。(可配置)
分支预测器	无，(可配置)

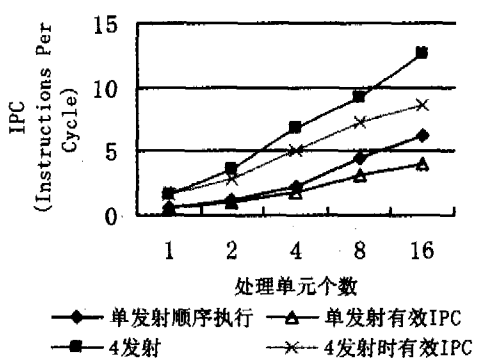


图 4　处理单元个数对于 IPC 的影响

图 4 给出了 OpenCMP 在相同的处理单元配置情况下，不同处理单元个数对于系统 IPC 的影响。总的 IPC 包括了事务执行过程中处于忙等待和软件控制事务执行的额外开销产生的 IPC。我们模拟了单发射和 4 发射的情况。从图中可以看出系统的 IPC 随着处理单元的增加而增加。由于引入了软件事务控制，所以真正用于执行 benchmark 的有效 IPC 会比总的 IPC 的增长速度要慢，当处理单元个数越多，则用于控制的计算时间也就越多，IPC 的增长速度也就变慢。有效 IPC 的增加受限于软件开销和处理器个数以及应用本身的并行性，是一个综合的影响。

我们课题组目前正在使用 OpenCMP 模拟器进行 CMP 上事务存储模型的研究工作。例如：通过修改写操作地址队列缓存和行换出地址缓存大小，研究缓存模块对于处理器性能的影响，以及总线带宽对于性能的影响等。限于篇幅本文不在此赘述，相关的研究结果将于近期发表。

总结和未来工作　我们开发了一个多线程多核处理器模拟器工具集 OpenCMP，用于支持当前和未来对多线程多核处理器体系结构关键技术的研究。本文详细描述了支持事务存储 CMP 结构的一个设计方案以及模拟器设计实现。通过

抽象事务存储模型的最基本特点和组成部分，并扩展单核处理器模拟器 SimpleScalar，我们已经完成了基于循环并行化的支持事务存储 OpenCMP 模拟器的方案设计和模拟器的实现。基于 OpenCMP 模拟器，我们用 fft 做了一个简单的性能评测的例子来验证模拟器的设计。实验表明，OpenCMP 已能够较好地支持 CMP 事务存储模型的研究。

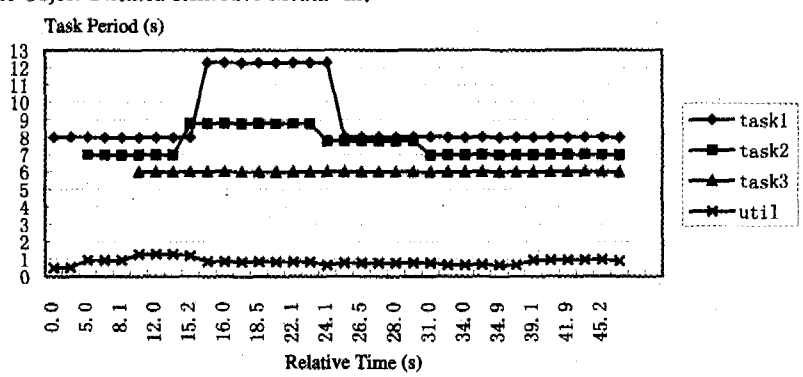
我们对 OpenMP 的进一步改进主要从以下几个方面展开：(1)加入对于子程序调用和对于传统的程序并行化方法的支持。(2)进一步优化事务存储模型的几个抽象模块的设计，以提高性能。目前，对于总线的模拟，由于要进行多次的访问，模拟性能不高。(3)支持对于全系统的模拟，包括对操作系统等的模拟。(4)完善模拟器的性能评估模型。(5)由于目前采用手工编译插入指令，对于工具集中编译器没有进行扩展，不能进行大规模程序的并行化。为了进行更大规模的评测，需要修改工具集中的编译器提供对更多更大的 benchmark 程序的模拟分析。此外，目前的控制软件开销过大，也是需要改进的地方。

参 考 文 献

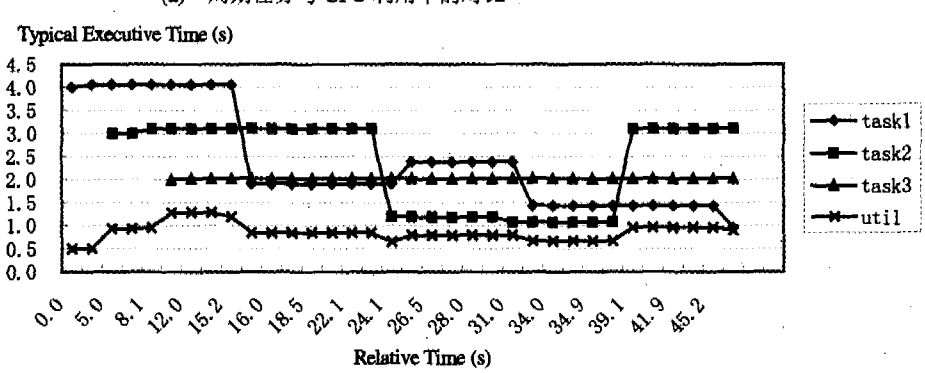
1 Tullsen D M, Eggers S J, Levy H M. Simultaneous multithreading: Maximizing on-chip parallelism. 22nd Annual International Symposium on Computer Architecture, June 1995
2 Marr D T, Binns F, Hill D L, et al. Hyper-threading technology architecture and microarchitecture. Intel Technology Journal, Feb. 2002
3 Diefendorff K. Power4 Focuses on Memory Bandwidth. Microprocessor Report, Oct. 1999
4 Clabes J, et al. Design and implementatin of the power5 microprocessor. In ISSCC Digest of Technical Papers, Feb. 2004
5 Burger D, Austin T M. The SimpleScalar tool set version 2.0:[Technical Report]. 1342. Computer Sciences Department, University of Wisconsin, June 1997
6 Austin T, Ernst D. SimpleScalar Tutorial (for release 4.0). http://SimpleScalar.com/docs/simple_tutorial_v4.pdf
7 Martin M M K, Sorin D J, Beckmann B M, et al. Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset. Computer Architecture News (CAN), September 2005
8 SESC: http://sesc.sourceforge.net/index.html
9 Nathan L B, Erik G H, Steven K R. Network-Oriented Full-System Simulation using M5. The Sixth Workshop on Computer Architecture Evaluation Using Commercial Workloads, Feb. 2003
10 Mendel R, Edouard B, Scott D, et al. Using the SimOS Machine Simulator to study Complex Computer Systems. ACM TOMACS special issue on Computer Simulation, 1997
11 路放，安虹，梁博，等. OpenSMT：一个同时多线程处理器模拟器的设计和实现. 计算机科学，2006(1)
12 Sohi G, Breach S, Vijaykumar T. Multiscalar Processors. Proceedings of the 22nd Annual International Symposium on Computer Architecture (ISCA-22), June 1995
13 Herlihy M, Moss J B. Transactional Memory: Architectural Support for Lock-Free Data Structures. Proceedings of the 20th Annual International Symposium on Computer Architecture, May 1993
14 Knight T. An architecture for mostly functional languages. Proceedings of the 1986 ACM Conference on LISP and Functional Programming (LFP '86), August 1986
15 Hammond L, Wong V, Chen Mike et al. Transactional Memory Coherence and Consistency. Proceedings of the 31st Annual International Symposium on Computer Architecture, June 2004
16 Hammond L, Carlstrom B D, Wong V, et al. Programming with Transactional Coherence and Consistency (TCC) ASPLOS04, October 2004
17 Ananian C S, Asanovic K, Kuszmaul B C, et al. Unbounded Transactional Memory. In: Proceedings of the Eleventh International Symposium on High-Performance Computer Architecture. Feb. 2005
18 Moore K E. Thread-Level Transactional Memory. Wisconsin Industrial Affiliates Meeting. Wisconsin Industrial Affiliates Meeting, Oct. 2004
19 Rajwar R, Herlihy M, Lai Konrad. Virtualizing Transactional Memory. Proceedings of the 32nd Annual International Symposium on Computer Architecture. Jun. 2005
20 Hammond L, Hubbert B, Siu M, et al. The Stanford Hydra CMP. IEEE MICRO Magazine, March-April 2000

(上接第 222 页)

4 Object Management Group. The CORBA Architecture and Specification, Reversion 3.0. OMG Inc, March 2004
5 Cazzola W. Evaluation of Object-Oriented Reflective Model. In: Proceeding of ECOOP Workshop on Reflective Object-Oriented programming and Systems(EWROOPS'98), Brussels, Belgium, July 1998



(a) 周期任务与 CPU 利用率的对比



(b) 典型执行时间与 CPU 利用率的对比

图 5