摘 要

高速缓存一致性问题不仅关系着系统的正确性，还对系统的性能有着重要影响。多核处理器的高速缓存一致性协议设计更为复杂和验证更加困难。使用多核处理器构建大规模并行计算系统已经成为主流。在此环境下，高速缓存一致性协议需要处理的一致性事务更多，涉及到处理器芯片内多个高速缓存之间、处理器芯片内高速缓存与片外高速缓存之间、处理器芯片之间的一致性问题等。所以研究多核处理器的高速缓存一致性问题具有重要的学术意义和应用背景。首先，本文对多核处理器中的高速缓存一致性协议进行了研究，重点研究了扩放性较好、能适应多核处理器本身特点的 MOESI 协议及其实现，并对该协议做出了优化；其次，本文研究了在由多核处理器构建的并行计算系统环境下的高速缓存一致性协议，实验证明本文工作能够有效减少片内高速缓存失效次数（13%到 30%）和提高系统性能（运行时间最多能减少 30%左右）；最后，本文研究了片上高速缓存的包含与非包含策略，提出了一个基于不包含策略的片上高速缓存系统设计，从而提高了片上高速缓存容量的利用率和提升了多核处理器的性能。

高性能计算机是一个国家的重要战略资源，其国产化水平是一个国家综合国力的集中体现。目前采用我国具有完全自主知识产权的龙芯多核处理器构建高性能计算机已经被一些研究单位或机构纳入日程。首先，本文特别针对科学计算领域应用，对龙芯体系结构的多核处理器的片上缓存系统的性能进行了评测，指出了龙芯多核处理器在科学计算领域中的一些性能特点。其次，依此进行了一些设计空间上的探索。实验证明，在某些情况下可以使片上二级高速缓存命中率提高 50%以上，等。

高性能计算机的网络，对于机器的整体性能具有至关重要的作用。首先，本文研究了一种先进的新型网络拓朴结构：MPU，研究内容包括其数学模型、网络拓扑、路由算法等。其次，本文从理论上对 MPU 与当前其他先进高性能计算机网络进行了对比分析。最后，本文还介绍了为 MPU 所开发的一个大型并行模拟器 MPUS 的原理、架构、工作流程，等。实验证明，MPU 的设计正确，且具有良好的可扩放性。

KD-50-I 万亿次计算机是首台基于龙芯通用高性能处理器的国产万亿次计算机。首先，本文围绕 KD-50-I 的体系结构设计，研究实现了 KD-50-I 的无盘启动技术、构建了 KD-50-I 的高效操作系统和文件系统、优化了 KD-50-I 通信库，从

而提高了系统的性能和可用性，有利于 KD-50-I 的推广应用。其次，本文研究了实际物理学研究中常用到的扫描电子显微成像模拟程序在 KD-50-I 上的应用，并对其进行了优化。本项工作提高了应用程序运行效率，为 KD-50-I 在不同领域的应用提供了示例。

关键词：高速缓存一致性　多核处理器　高性能计算机互连网络　高性能计算机

Abstract

The Cache Coherence Protocol is a key component towards the correctness and effciency of the computer system. It is more difficult to design and verify the cache coherence protocol for Chip Multiprocessor(CMP), as the protocol should deal with the interaction between CMPs, inter- and intra-CMPs, etc. High performance computers are important resources to our country. We have studied the cache coherence techniques for CMP. Taking into account the scalarbility and performance, our work focuses on the MOESI protocol and its implementation. We have studied the cache coherence protocol in multi-CMP (M-CMP) systems. Experiments show that our work can improve system performance up to 1.5X, and reduce the times of on-chip cache misses(13% up to 30%). To utilize on-chip cache efficiently is key to CMP's performance. We have studied the performance of inclusive and non-inclusive on-chip cache and proposed an on-chip cache architecture that is based on exclusive policy.

Some companies and research institution have dicided to comstruct high performance computers that based on LoongSon multi-core CPU. To comprehend the feature of LoongSon CPU under scientific applications more precisely, we profiled its performance. Based on the research, we proposed some idea to optimise its architecture. Experiments showed that our work can reduce the L2 Cache miss rate up to 50%.

The network of high performance computer is key to HPC's performance. We have studied a new network --- MPU, including its mathematic model, topology, routing algorithm, etc. We proved that MPU has better performance that some other modern networks theoretically. After that, we have studied a large scale parallel simulator for MPU --- MPUS, including its architecture, working procedures, etc. Experiments thowed that the design of MPU is right, and is good at scalability.

KD-50-I is the first totally made-in-China Tera-Flops high performance computer that based on LoongSon CPU. We have studied its architecture and

optimization to it. Our wrok contributes to the development and use of totally made-in-China high performance computer.

Key Words: Cache Coherence, Chip Multiprocessor, Parallel Computing, High Performance Computer

插图目录

表格目录

中国科学技术大学学位论文原创性声明

本人声明所呈交的学位论文，是本人在导师指导下进行研究工作所取得的成果。除已特别加以标注和致谢的地方外，论文中不包含任何他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的贡献均已在论文中作了明确的说明。

作者签名：李晓 签字日期：2009.6.17

中国科学技术大学学位论文授权使用声明

作为申请学位的条件之一，学位论文著作权拥有者授权中国科学技术大学拥有学位论文的部分使用权，即：学校有权按有关规定向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅，可以将学位论文编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。本人提交的电子文档的内容和纸质论文的内容相一致。

保密的学位论文在解密后也遵守此规定。

☑公开 □保密（___年）

作者签名：李晓 导师签名：王国良

签字日期：2009.6.17. 签字日期：2009.6.17

第 1 章 绪论

本章摘要 大规模应用的强大计算需求，推动了并行计算的快速发展。并行计算本身是一个复杂的系统问题，涉及并行计算机硬件体系结构、并行程序设计模型和支持并行程序设计的软件系统等诸多方面的问题。本章首先介绍了本文研究工作的背景和研究现状，包括并行计算机体系结构、多核处理器体系结构、并行程序模型、当前多核处理器设计所面临的问题，等。随后指出使用多核处理器构建大规模并行计算系统所面临的一些问题，并在此基础上，提出本文研究工作的研究内容和思路。本章最后列出常用的文献资源，并给出本文的组织结构。

1.1 并行计算和多核处理器

1.1.1 概述

近几年来，世界上和我国高性能并行计算机的发展取得长足进展，每秒数百亿次、数千亿次乃至数万亿次计算能力的高端并行机已相继研制成功并投入使用，使得以前许多无法靠手工计算解决的，或者对计算能力要求非常高的问题现在已经可以通过高性能计算机来计算解决。随着计算技术和计算方法的飞速发展进步，定量化和精确化已经成为几乎所有学科的发展趋势，并由此而产生了许多交叉性学科，如计算物理、计算化学、计算地质学、计算生物学、计算气象学等新兴学科，在世界上逐渐形成了所谓的计算科学与工程的计算性学科分支。计算，使人们人事科学研究的能力得到增强，使人们洞察自然的视野得到扩展，使科技转化为生产的过程得到加速。计算深刻地改变着人类认识世界和改造世界的方法与途径。

如今科学家们普遍认为，与依赖理论推导的理论科学和依赖实验的实验科学相并列，依赖于计算的计算科学已经成为第三门科学。这三大门类科学彼此促进，相辅相成，为人类科技发展和社会进步起着重大的推动作用。

人类的生产生活越来越复杂，所需解决的问题的规模也越来越大，对高性能计算能力的需求永无止境。这极大地推动了并行计算的发展。关于并行计算的定义，简单而言，是指在多台处理机上同时进行的计算。这种计算也可被称作高性能计算或超级计算。[Almasi 1989]。并行计算本身而言是一个非常复杂的系统问题，主要涉及到并行计算机系统结构、并行程序模型和支持并行程序设计和运行

的软件系统等诸多方面的问题[陈国良 2003]。

1.1.2 并行计算机体系结构

大型并行机系统一般可按照程序的控制流、数据流以及存储组织方式等简单分成六类[Hwang 1998]：单指令多数据流（SIMD，Single Instruction Multiple Data）、并行向量处理机（PVP，Parallel Vector Processor）、对称多处理机（SMP，Symmetric Multiprocessor）、大规模并行处理机（MPP，Massively Parallel Processor）、工作站集群（COW，Cluster of Workstations），和分布式共享存储多处理机 DSM（DSM，Distributed Shared Memory）。SIMD 机器多为专用，其余五种属于多指令多数据流（MIMD，Multiple Instruction Multiple Data）计算机。

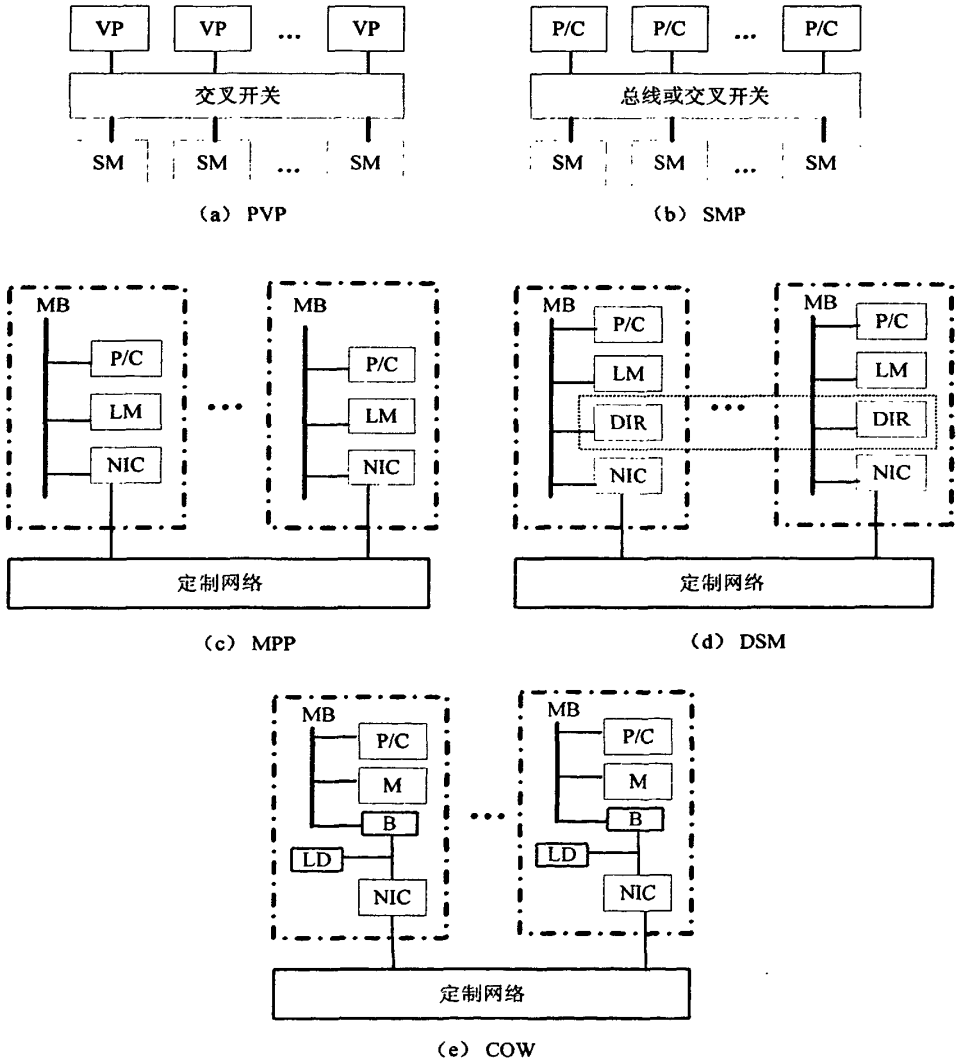


图 1.1 MIMD 并行计算机体系结构模型

图 1.1 给出了上述五种 MIMD 机器的体系结构模型。图中注释意义分别是：B(Bridge)是存储器总线和 I/O 总线之间的桥接；DIR（Cache Directory）指高速缓存目录，IOB（I/O Bus）指 I/O 总线，LD（Local Distk）指本地磁盘，MB（Memory Bus）指存储器总线，NIC（Network Interface Circuity）表示网络接口电路，P/C（Microprocessor and Cache）指微处理器和高速缓存，SM（Shared Memory）指共享存储器。

1.1.3 多核处理器体系结构

由于半导体技术的飞速发展，在过去几十年里固定面积的芯片上可以集成的晶体管数目越来越多，集成度越来越高，且价格越来越便宜。相应的，处理器的性能不断提高。其原因除了物理技术的进步之外（如主频不断得到提高，等），因应不同物理技术的计算机系统结构技术也走到很大作用，如：加大流水线的深度和指令发射宽度、乱序执行和推测多线程，等。今天的单核处理器的性能几乎是十五年前的 100 倍左右[Neil 2005]。

与处理器性能提升相伴的是，功耗的不断增加几乎到了无法忍受的地步，同时硬件复杂度的加大也为处理器的设计造成困难。接下来继续通过提高主频、增加流水线级数等手段来挖掘指令级并行性已经很难更进一步提高处理器性能。因此，工业界和学术界开始转向片上多核处理器（Chip Multiprocessor，CMP）结构。

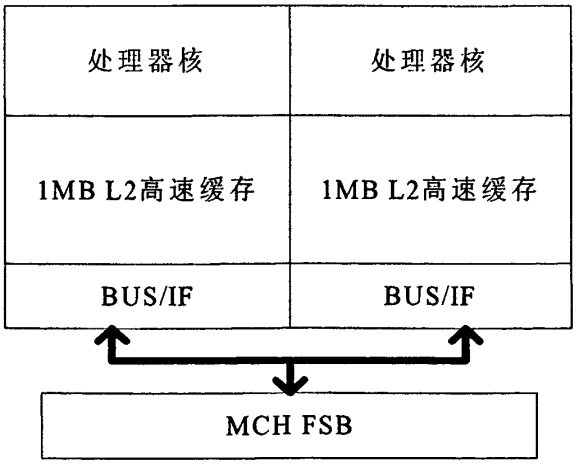
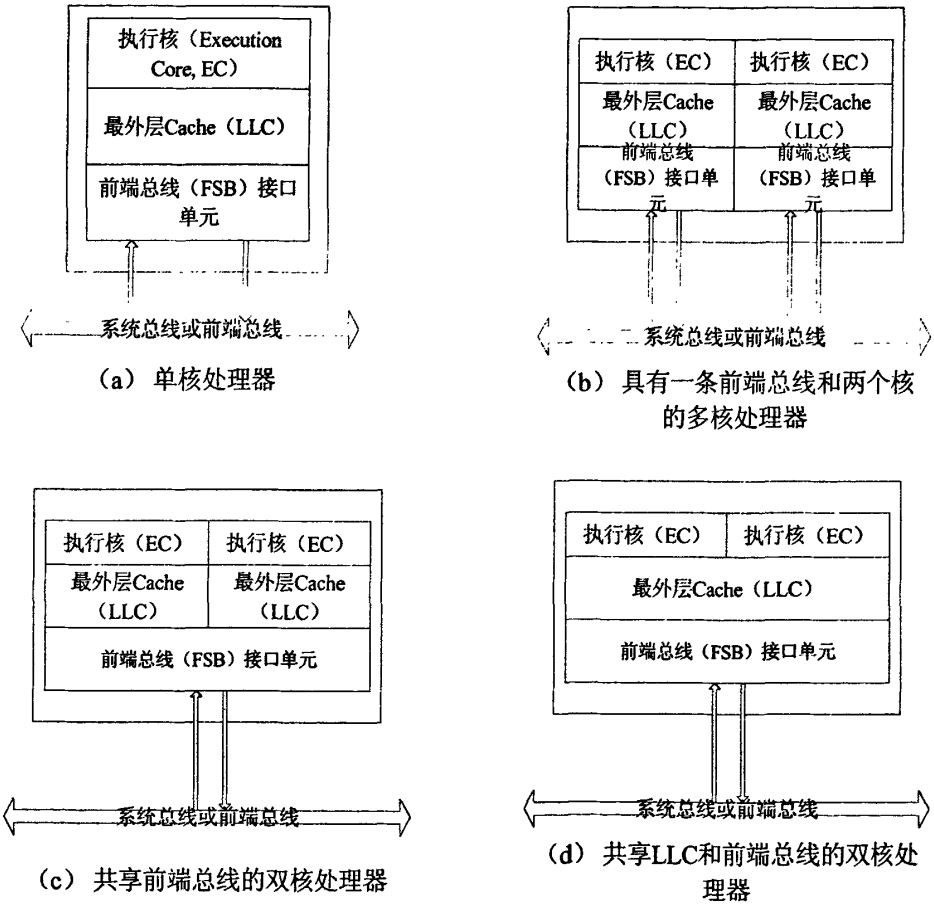


图 1.2 Intel 奔腾双核处理器架构示意图

图 1.2 给出了 Intel[Intel 2005]的一款多核处理器体系结构。图 1.3 以双核处理器为例，给出了一些不同配置的片上多核处理器体系结构[Akhter 2007]。

目前多核处理器设计基本都是将多个相对简单、同构的处理器核集成到同一块芯片上，且这些处理器核往往共享某些片上高速缓存（Cache）。这种设计方法的一大好处是可以充分利用已有成熟的设计，降低研发的风险、周期和成本。

多核处理器本身的系统结构特性，也为性能的挖掘提供了新的机遇，如：芯片内互连网络的通信速度一般远高于片外网络速度；片内共享调整缓存为进程间通信带来方便；等。但另一方面，多核处理器本身的系统结构特性也带来了挑战，如：受制于芯片面积，处理器引脚个数无法增加，使得片外访存成为系统性能瓶颈，等[Ravi 2000][David 2001][Hammand 1997]。



（a）单核处理器　（b）具有一条前端总线和两个核的多核处理器

（c）共享前端总线的双核处理器　（d）共享LLC和前端总线的双核处理器

图1.3 各种不同配置的多核处理器

一般每个集成到多核处理器片内的处理器核，都拥有较完整的资源，如运算器、控制器、私有Cache，等。这些处理器核能够并行的执行多个线程，相较于以往的单核处理器的并发执行方式，多核处理器给挖掘线程级并行提供了巨大空间。

CMP 技术的优势包括概括起来主要有：线延迟小，线程级并行性，设计验证较为容易，功耗低等[Diefendorff 00][Hammond 04]。

1. 线延迟小

随着物理技术的进步，尤其是微尺度工艺的应用和芯片主频的不断提高，芯片外部的互连延迟正开始超过器件延迟，成为制约提高芯片性能和芯片集成度的重要因素。而在多核处理器中，每个处理器核一般只占用芯片上很小一块面积，使互连线的长度缩短，能够有效提高片上互连网络的速度。

2. 线程级并行

在传统的多处理机系统中，并行程序的粗粒度并行性能够较好的被挖掘。在这种机器上，由于处理器之间的带宽和延迟限制，如果通信密度过大（如每执行几百甚至几十条指令即要求进行通信），为对程序整体性能带来极大伤害。这导致并行程序的细粒度并行性不能被很好地挖掘。

而在多核处理器中，由于同一芯片内的互连延迟较小，通信开销对性能的影响降低，进程间的同步变得容易，使得细粒度并行性的挖掘成为可能。

3. 设计验证容易

片上可集成晶体管数目的增加，固然增加了设计空间，带来了性能的提升，但同时也导致处理器的设计和验证的复杂度急剧增加。事实上，在现有的一些已经上市的处理器中，仍然存在着设计错误。如何降低设计和验证的复杂度，成为摆在人们面前的一道难题。一个较好的办法是，充分利用现有的已经比较成熟的设计方案，将它们集成到一起进行复用。而多核处理器正可以做到这一点。

4. 功耗低

处理器的功耗已经成为当前处理器系统结构设计所面临的最大问题。而处理器的功耗与处理器主频正相关，即主频越高，功耗越大，且功耗增加的速度远超过主频提高的速度。在限制功耗的前提下，要提高处理器的性能，则应该考虑利用多核处理器结构：在 CMP 芯片上集成多个主频较低的处理器核，一方面降低功耗，一方面使程序的并行性得到发挥。这样仍能提高系统的整体性能。

1.1.4 并行程序设计模型

并行算法的实现离不开并行编程，并行程序的运行离不开具体的并行计算机。当在具有不同体系结构的并行计算机上执行程序时，要求有不同的编程模式。而

不同的编程模式，则是以不同的并行程序设计模型为基础。根据并行计算机体系结构的差异，可将并行编程模型划分为三大类：即数据并行模型、消息传递模型以及共享变量的并行程序设计模型。[陈国良 2003]：

1. 数据并行模型

该编程模型既可以在 SIMD（Single Instruction Multiple Data，单指令多数据）机器上实现，也可以在 SPMD（Single Process Multiple Data，单进程多数据）机器上实现，这取决于并行粒度的大小。SIMD 程序侧重细粒度并行，SPMD 侧重中粒度并行。对数据并行操作的同步的实现，在程序编译阶段完成。

数据并行模型的特点概括起来有：

单线程：在程序员看来，并行程序的控制流是顺序的；

并行操作：并行操作对象是聚合数据结构，并行程序中的一条语句，可指定多个操作，而这些操作同时作用在不同的数组元素或其他聚合数据结构上；

松散同步：并行程序的同步是隐式的，这种隐含的同步操作事实上存在于程序的每条指令之后；

命名空间全局化：无论物理存储器的分布情况，程序的所有数据均存储于一个单一的地址空间内，对数据的访问操作通过访存实现；

隐式相互作用：由于松散同步，使得通信可藉由变量指派而隐式地完成；

隐式数据分配：数据的分配操作无需程序员的显式指定，这些工作可交由编译器完成（也许会要求在程序中或者编译时传递给编译器一些必要信息）。

2. 消息传递模型

在该模型中，系统中运行在不同处理机上的进程间的通信可藉由互连网络来实现。这里的消息概念包括：数据、指令、同步信息，等等。在该模型中，进程的数据分配和负载平衡工作全部由用户指定。该模型较适合于开发粗粒度并行程序。

消息传递编程模型的主要特点有：

多线程：在该模型中，并行程序由多个进程组成，这些进程的控制流可以相同，也可以不同。除此之外，这些进程的数据流可以相同或不同；

异步并行性：在消息传递模型中，对各进程的同步方法主要依赖于路障和阻塞通信等。各不同进程之间的执行是异步；

分开的地址空间：各进程的地址空间各自独立无关。一个进程的存储空间对其他进程是透明的。进程间通信藉由消息传递来实现；

显式相互作用：进程间的通信操作由程序员显式指定，进程操作的数据等也

全部由程序员指定。进程只在其所拥有的数据上执行计算；

显式分配：计算任务和计算数据的分配均由程序员显式分配。在此情况下，为了减少程序设计和编写代码的复杂度，程序员经常将程序编写成 SPMD 形式。

3. 共享变量模型

在该模型中，各进程的数据集处于同一地址空间，各进程间的通信可藉由读/写内存地址来实现。与消息传递模型不同的是，数据无需程序员显式分配，负载的平衡则既可显式也可隐式地实现。由于进程间通信由共享变量完成，所以进程间的同步必须是显式的。目前关于共享变量模型还没有一个普遍标准。

表 1.1 并行程序设计模型比较

特性	数据并行	消息传递	共享变量
控制流	单线程	多线程	多线程
进程间操作	松散同步	异步	异步
地址空间	单一地址	多地址空间	单地址空间
相互作用	隐式	显式	显式
数据分配	隐式或半隐式	显式	隐式或半隐式

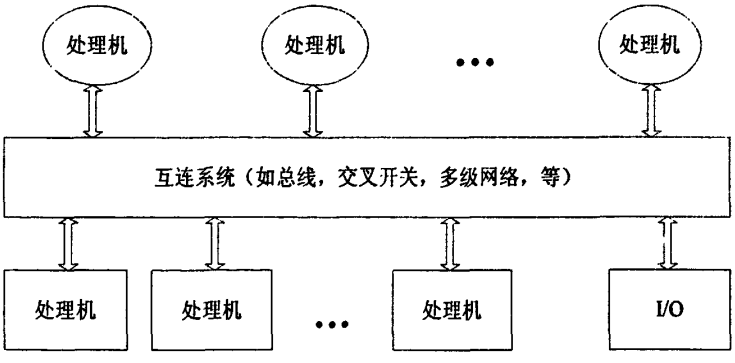


图 1.4 UMA 多处理机模型

1.1.5 并行计算机存储结构模型

并行计算机存储结构模型和并行计算机体系结构模型构成实际并行计算机

体系结构的两个方面[陈国良 2003]。并行计算机存储结构模型基本可以分为五种。

1. 均匀存储访问模型（UMA，Uniform Memory Access）

在该模型中，所有处理器均匀共享存储；所有处理器访问任意存储单元的开销（主要是延迟）相同；所有处理器可带有私有高速缓存。除此之外，所有处理器还可以一定形式共享外围设备。该模式较适于通用或分时应用（图1.4）。

2. 非均匀存储访问模型（NUMA，Nonuniform Memory Access）

该模型的特点是，逻辑上单一的存储器物理上分布(distributed)于各个处理器中，全局地址空间由所有本地存储器共同组成，各处理器访问存储器的开销是不同（即非均匀）。在该模型中，每台处理器可带私有高速缓存，且外设也可以某种形式共享（图1.5）。

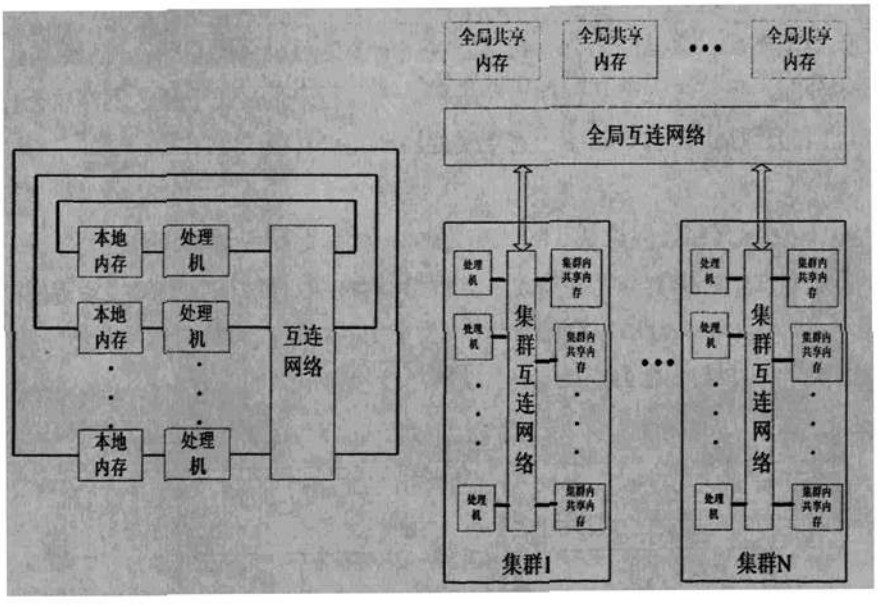


图1.5 NUMA多处理机模型

3. 全高速缓存存储访问模型（COMA，Cache Only Memory Access）

该模型是NUMA的一个特例，它的特点是：各处理器结点中没有存储层次结构，全局地址空间由全部高速缓存共同组成；对高速缓存的访问需利用分布的目录(图1.6中“D”模块)。

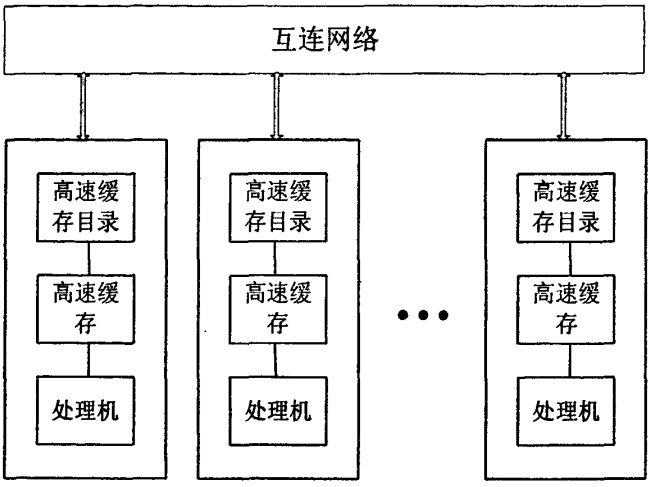


图 1.6 COMA 多处理机模型

4. 高速缓存一致性非均匀存储访问模型（CC-NUMA，Coherence-Cache Nonuniform Memory Access）

该模型本质上就是一个分布式共享存储的多处理机系统（DSM），在该模型中，负载的分配由系统中硬件和软件自动进行，无需程序员显式干预。程序执行过程中，数据的迁移工作由 Cache 一致性硬件自动完成（图 1.7）。

5. 非远程存储访问模型（NORMA，NO-Remote Memory Access）

在该模型中，所有存储器均为各处理器私有，且仅能被属主处理器访问。绝大部分 NORMA 都不支持远程存储访问。该模型中的进程间通信藉由消息传递实现。系统由多个计算结点通过消息传递互连网络连接而成（图 1.8）。

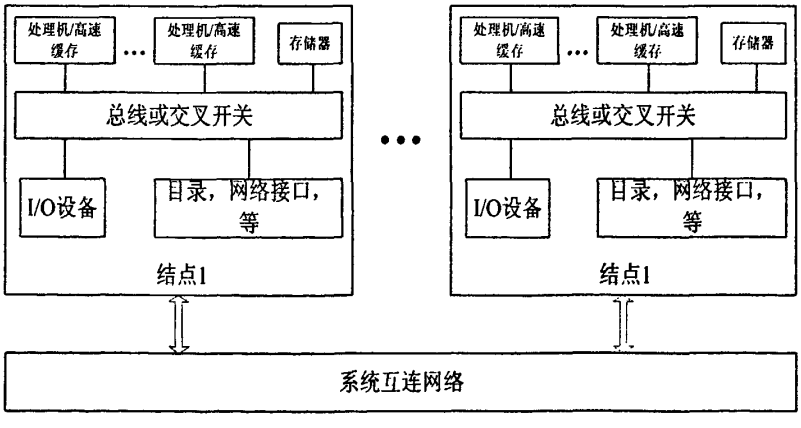


图 1.7 CC-NUMA 多处理机模型

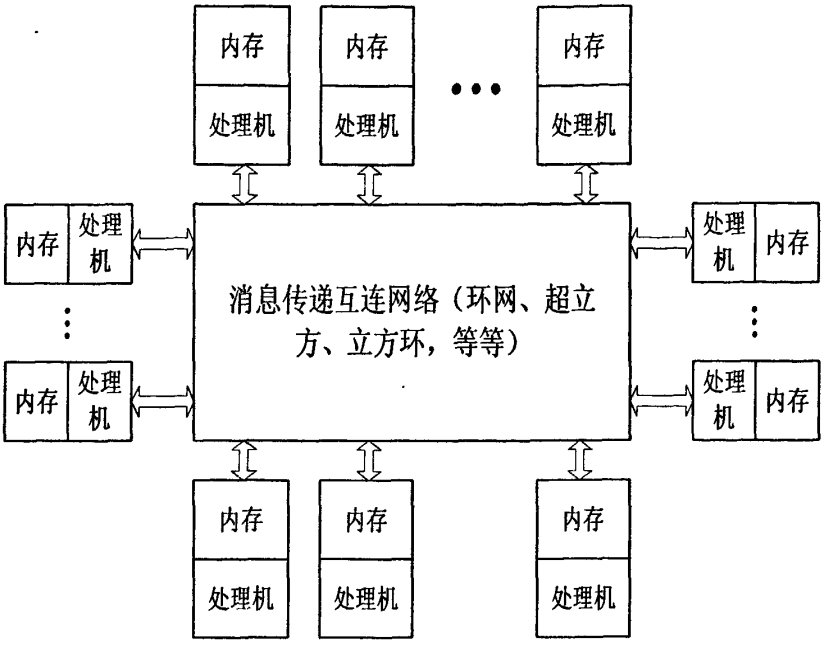


图 1.8 NORMA 多处理机模型

1.2 用多核处理器构建大规模并行计算机

随着多核处理器应用范围日益扩大，使用多核处理器构建大规模并行计算机已经成为主流，在 2008 年上半年的 TOP500 名单中，使用单核处理器的大规模并行计算机只有 11 台[TOP500 2008]。不过在设计制造使用多核处理器构建的大规模并行计算机时有很多需要注意的地方，比如：

1. 多核处理器与传统的 SMP 机器不同

首先在耦合程度上有所不同，比如多核处理器本身往往共享二级甚至三级片内高速缓存。同时，由于处理器引脚的个数限制了多核处理器的对外带宽，使得访存性能对系统的影响更大。

2. 程序的通信行为更难描述

目前似乎还没有很好的工具能对多核处理器内部的核与核之间的通信行为进行较为精确的分析。并且核与核之间的通信带宽、延迟等与处理器之间的通信带宽、延迟等有很大差别。这为进一步利用机器本身的特性优化程序带来的困难。

3. 负载分配

对由多核处理器构建的大规模并行计算机而言，负载分配是以处理器核为单位的，而不在是单个处理器。

4. 维护高速缓存一致性

由于多核处理器内有多个核，且这些核之间往往共享片内二级或三级高速缓存，这要求在多核处理器内也要保证高速缓存的一致性。逻辑上看，在由多核处理器构建的大规模并行计算机中，需要两层一致性协议：多核处理器内的一致性协议，以及多核处理器间的一致性协议。这无疑给多核处理器的高速缓存一致性协议的设计带来了更大困难。与此同时，高速缓存一致性协议对系统性能的影响也更大了。

1.3 研究内容及思路

针对目前多核处理器在并行计算中面临的问题，以及高性能计算机国产化问题，本文的主要研究内容和思路是：

1. 研究了多核处理器中的高速缓存一致性协议及其实现。考虑到片内可集成的处理器核的数量仍将持续增加，本文着重研究了扩放性比较好的基于目录的 MOESI 协议。同时随着多核处理器的普及流行，采用多核处理器构建的并行计算系统也将成为趋势。本文在 M-CMP 环境下研究了多核处理器的一致性协议。由于物理限制等原因，多核处理器片内高速缓存的容量受限。为了提高片内高速缓存的效率，本文研究了多核处理器片内高速缓存的包含与不包含结构，同时针对不包含结构对 MOESI 协议进行了优化。

2. 龙芯处理器是我国首款具有完全自主知识产权的高性能通用处理器，代表了国内处理器设计的最高水平。龙芯多核处理器即将面世，同时采用龙芯多核处理器研制高性能计算机也被中国科大等研究机构纳入研究计划。本文针对科学计算等高性能应用领域，剖析了龙芯多核处理器的性能，并进行了设计空间上的探索，对龙芯体系结构的多核处理器的设计具有参考价值，也对基于龙芯多核处理器的高性能计算机的研制具有重要参考价值。

3. 高性能计算机中的互连网络对系统的性能具有至关重要的作用。本文介绍了一种新型的高性能网络 MPU，包括其网络拓扑、路由算法等。理论上对其性能与目前较具代表的其他高性能网络进行了对比，实现了一个对 MPU 进行仿真的模拟器 MPUS，验证了 MPU 设计的正确性，并预测了其性能。

4. 高性能计算机是一个国家的重要战略资源，其国产化具有重要战略意义。本文紧密结合首台全国产的基于龙芯2F处理器的万亿次计算机KD-50-I的工程实践，研究了KD-50-I的体系结构技术、无盘启动技术、通信库优化技术，等等。同时随着国产龙芯处理器在高性能计算机领域内被逐渐推广，本文研究了实际物理研究用并行计算程序在KD-50-I高性能计算机上的性能和优化情况。本文的工作对国产高性能计算机的推广使用具有重要意义。

1.4 文献资源

为了便于调研、学习和研究，本节中列出一些与计算机体系结构和并行计算技术相关的重要电子文献资源、国际会议文献资源和纸质文献资源以供参考。

1. 电子文献资源

ACM电子文献资源数据库、IEEE电子文献资源数据库、Elserver电子文献资源数据库、Kluwer电子文献资源数据库、Springer电子文献资源数据库、中国学术期刊网，等等。

2. 国际会议文献资源

International Symposum on Computer Architecture(ISCA), International Symposum on Microarchitecture(MICRO), ACM/IEEE Supercomputing Conference(SC/SUPER), IEEE Symposum on High-Performance Computer Architecture(HPCA), ACM SIGPLAN Symposum on Principles and Parallel Programming(PPoPP), IEEE International Conference on High Performance Computing and Communications(HPCC), International Conference on Cluster Computing(CLUSTER), International Symposium on High Performance Computing Systems(HPCS), Hot Chips Symposum(HCS), IEEE International Parallel and Distributed Processing Symposium(IPDPS), ACM Annual Symposium on Theory of Computing(STOC), IEEE Symposium on Foundation of Coumputer Science(FOCS), 等等。

3. 纸质文献资源

Communications of ACM、Journal of ACM、ACM Computing Surveys、IEEE Trans. On Coumputers、IEEE Trans. On Parallel and Distributed Systems、SIAM Journal on Computing、Journal of Parallel and Distributed Computing、Parallel

Algorithms and Applications、Parallel Processing Letter、Software Practice and Experience、Lecture Notes on Computer Science(LNCS)、计算机学报、软件学报、计算机研究与发展、小型微型计算机系统、Journal of Computer Science and Technology，等等。

1.5 本文组织结构

本文共分 7 章。组织如下：

第 1 章为绪论。介绍了本文研究工作的背景和研究现状，包括并行计算机体系结构、多核处理器体系结构、并行程序模型、当前多核处理器设计所面临的问题，等等。随后指出使用多核处理器构建大规模并行计算系统所面临的一些问题，并在此基础上，提出本文研究工作的研究内容和思路。本章最后列出常用的文献资源，并给出本文的组织结构。

第 2 章研究了高速缓存一致性问题。首先介绍了传统的 Cache 一致性问题的一些基本概念和解决方法，随后扩展到多核处理器内的 Cache 一致性问题，以及多核处理器间的 Cache 一致性问题。并在此基础上，提出本文部分研究工作的研究内容和思路。

第 3 章还研究了由多核处理器构建的并行系统及其 Cache 一致性问题。首先研究了由多核处理器构建的共享存储的并行系统（M-CMP），其次研究了在 M-CMP 中实现高速缓存一致性的技术。着重研究了目录 MOESI 协议在 GEMS 中的实现。最后研究了基于不包含策略的片内高速缓存系统，并基于不包含策略对 MOESI 协议进行了优化。

第 4 章研究了龙芯多核体系结构的多核处理器在科学计算领域内的性能，进行了设计空间上的探索，对龙芯多核处理器的设计以及利用龙芯多核处理器的研制高性能计算机等均具有重要参考价值。

第 5 章介绍了 MPU 网络，理论上分析了其性能，实现了一个对 MPU 进行仿真的模拟器，证明了 MPU 设计的正确性，并预测了其性能。

第 6 章研究了 KD-50-I 万亿次计算机系统结构技术以及无盘启动技术、通信库优化等，并给出了 KD-50-I 万亿次计算机的一些性能测试数据。本章还研究了 KD-50-I 万亿次计算机在扫描电子成像中的应用。扫描电镜计算在物理学中有着重要作用。本章研究了该程序在 KD-50-I 上的应用情况，以及优化情况等。

第 7 章总结了本文的研究工作，贡献和创新点等，并对进一步的研究工作进行了展望。

第 2 章 高速缓存一致性

本章摘要 随着片上多核处理器的出现，传统的多芯片间的Cache一致性协议问题被引入到单芯片内部。不仅如此，随着片上多核处理器在并行计算机中的广泛应用，Cache一致性协议的设计和验证更加复杂和困难，同时对系统的性能的影响更大。本章首先介绍了经典的 Cache 一致性问题的一些基本概念和解决方法以及优化方法，随后简要讨论了多核处理器的高速缓存一致性问题，以及互连技术对多核处理器 Cache 一致性问题的影响。并在此基础上，提出本文研究工作的研究内容和思路。

2.1 Cache 一致性问题

共享存储的机器大致可以分为两大类：对称多处理机（Symmetric Multiprocessor，SMP）和分布式共享存储（Distributed Shared Memory，DSM）系统。

SMP 是目前最主要的一种共享存储的并行计算机系统，SMP 系统中的进程间通信一般依赖系统总线来实现。SMP 结构机器在现今的并行服务器中普遍被采用，并且已经越来越多地出现在桌面系统中。同时，SMP 机器也越来越多地作为一个构造模块，被用来构造更大规模的系统[陈国良 2002]。

图 2.1 中的 SMP 机器具有以下四大特性：

1. 对称性

系统中任何处理器均可以对称地访问任何存储单元和 I/O 设备，且具有相同的访存时间。所以这种 SMP 机器也叫做均匀存储访问（Uniform Memory Access，UMA）结构。

2. 单一物理地址空间

所有处理器的存储单元按单一地址空间编址。

3. 高速缓存及其一致性

多级高速缓存可支持数据局部性，且其一致性由硬件来实现。

4. 低通信延迟

处理器间的通信用简单的读/写指令来完成。

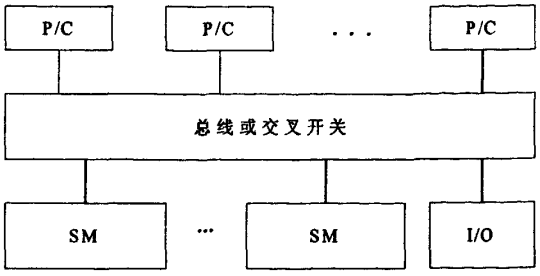


图 2.1 SMP 机器的结构

从分层角度来看，SMP 系统中的单一地址空间由相关硬件直接支持实现。由于存在类似于单处理器环境下串行程序的单地址空间，任何处理器可以用普通的读/写存储指令来高效地访问共享数据，缓存的一致性也由系统硬件自动维护，使得 SMP 对并行编程具有很大吸引力。

DSM 机器（图 2.2）的特征是，把逻辑上共享的存储器分成许多模块，物理上分布于各处理机之中，由这些物理上分布的存储器逻辑地实现共享存储模型。此外，为降低存储共享所引起的访存冲突问题，SM 系统中一般都会使用 Cache。分布存储会引起非一致访存（Non-Univform Memory Access，NUMA），而高速缓存的使用又带来了高速缓存一致性问题---即如何保证同一单元在不同高速缓存中的备份数据的一致。

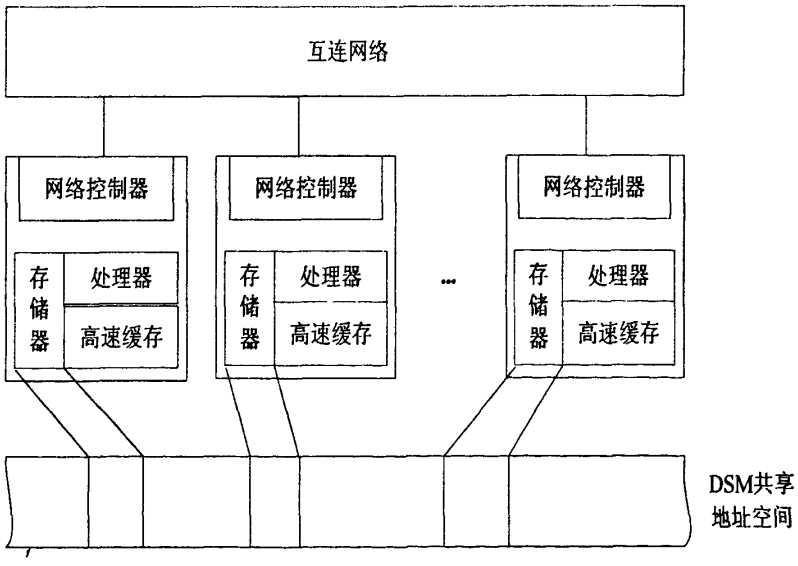


图 2.2 DSM 系统的结构组织示意图

上述共享存储的机器会导致 Cache 一致性问题。比如内存中的某个地址被多个处理器访问，则会使该值在多个 Cache 中出现副本。则当某处理器写该值时，会使得其他处理器的 Cache 中的值不再是最新的（或干净的）。

内存系统的本质是，一个内存系统应该能提供一组保存值的存储单元，当对一个存储单元执行读操作时，应该能返回“最近”一个对该存储单元的写操作所写入的值。在串行程序中，程序员利用内存来将程序中某一点计算出来的值，传递到该值的使用点，实际上就是利用了以上的基本性质。同样，运行在单处理器上的多个进程或线程利用共享地址空间进行通信，实际上也是利用了内存系统的这个性质。一个读操作应返回最近的向那个位置的写操作所写的值，而不管是哪个线程写的。当所有的线程运行在同一个物理处理器上时，它们通过相同的高速缓存层次来看内存，因此在这种情况下，高速缓存不会引起问题。当在共享存储的多处理器系统上运行一个具有多个进程的程序时，希望不管这些进程是运行在同一个处理器上，还是在不同的处理器上，程序的运行结果都是相同的。然而，当两个运行在不同的物理处理器上的进程通过不同的高速缓存层次来看共享内存时，其中一个进程可能会看到在它的高速缓存中的新值，而另一个则可能会看到旧值，这样就引起了 Cache 一致性问题 [陈国良 2002]。

对 Cache 一致性的比较正式的一个定义是：当一个共享存储的机器满足以下条件时，则可以认为该系统是 Cache 一致的[Culler 01]：

1. 任何进程所发出的访存操作被存储器所观察到的顺序必须与进程发出操作的顺序相同；

2. 每个读操作所返回的值必须是最后一次对该存储位置的写操作的值。

以上定义保证了两个属性：写广播和写串行化。写广播指的是写操作被其他所有处理器所观察到。定串行化是指对某一存储位置的所有写操作的顺序，在所有处理器看来都是一样的。

下面分别介绍三种实现 Cache 一致性的技术以及两种 Cache 一致性协议。

2.1.1 侦听技术

由于基于总线的 SMP 机器是通过高速共享总线将商用的微处理器（包括高速缓存）与共享存储器连接起来的，因此在设计中正好可以利用总线的特性来实现 Cache 一致性。总线是一组连接多个设备的线路，总线上的每个设备都能侦听到

总线上出现的事务。

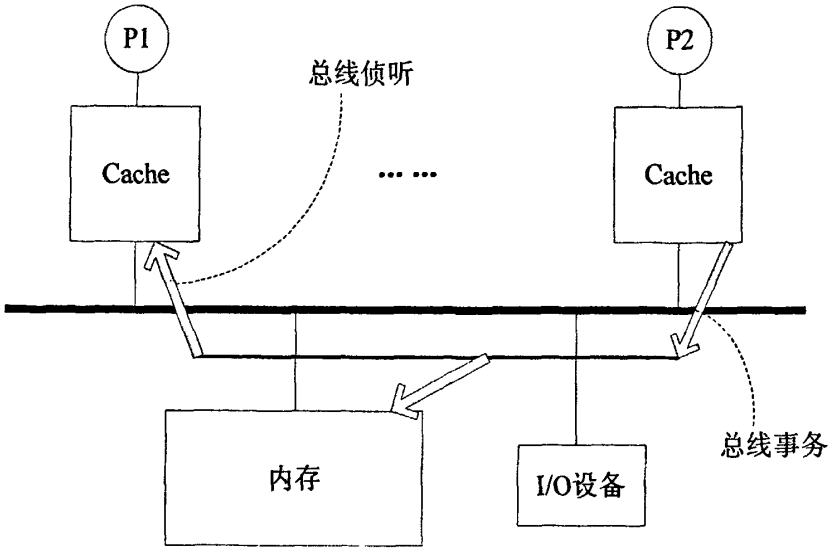


图 2.3 侦听一致性的多处理机

如图 2.3 所示，在侦听 Cache 一致性协议的实现中，当一个处理器向存储系统发出了一个内存读/写请示时，它本地的高速缓存控制器检查自身的状态，并采取相应的动作。比如，读命中，则本地高速缓存直接响应，读缺失，则向总线发出存取内存的事务请求等。所有其他的高速缓存控制器都侦听总线上出现的事务，一旦发现与自己相关的事务------本地高速缓存中有该事务请求内存块的一个拷贝，就执行相应的动作来保证高速缓存一致性。

在侦听技术中正是利用了总线的两个特点来实现一致性协议：所有总线上的事务对所有的高速缓存控制器都是可见的；它们对所有的控制器都是以相同的次序可见的。从而为了实现高速缓存一致性，在侦听一致性协议的设计中只需要保证两点：与内存操作相关的所有的必要的事务都出现在总路线上；控制器能采取适当的措施来处理相关的事务。

一般来说，在侦听高速缓存一致性协议中，每个高速缓存控制器都接收来自两方面的输入：处理器发出的内存请求和总线上侦听到的事务。作为对这些输入的响应，高速缓存控制器可能要根据相应块的当前状态及状态转换图来更新该块的状态，并且也可能要执行一些动作。比如，作为对处理器发出的读请求的响应，高速缓存控制器可能要产生一个总线事务来获得数据，并返回给处理器。有时候，高速缓存控制器必须对总线上侦听到的事务做出响应，比如提供最新的数据给请求者。因此，侦听协议实际上是一组互相协作的有限状态机所表示的分布式算法，它由三部分组成：

1. 状态集合

一个与本地高速缓存中内存块相关联的状态集合。

2. 状态转换图

以当前状态和处理器请求或观察到的总线事务作为输入，输出该块的下一个状态。

3. 动作

与每个状态转换相关的实际动作，这是由总线、高速缓存和处理器的具体设计来决定的。

在侦听技术中，读操作未被总线完全全局串行化，因为读命中不产生总线事务，所以它们可以独立产生。而写操作和读缺失一起被总线序全局串行化，从而它可根据全局的总线序来获得最新写入的值。读命中获得的值或者是同一个处理器中对该位置的一个最近的写操作所产生的，或是在同一个处理器中最近一次读缺失获得的值。因为这两种情况都出现在总线上，所以读命中同样也是按照一致的全局总线序来获得值。可见在侦听技术中，总线序和程序序共同保证了对 Cache 一致性的要求。

2.1.2 目录技术

本文的工作主要基于目录技术，故在此对目录技术加以较详细的介绍。

对于基于共享存储的对称多处理机系统，顺序一一致性模型和侦听一致性协议保证了系统的正确性。但此类机器存在可扩放性差的缺点[胡伟武 01]。当用多级互连网络来构造有数百个处理器的大型系统时，常采用基于目录技术的 Cache 一致性协议。其主要思想是，为每个存储行维持一个目录项，该目录项记录所有当前持有此行备份的处理机号以及此行的相关状态等内容。不论系统是基于侦听的协议还是基于目录的协议，它们基本的高速缓存状态集通常是一样的。通常，目录表的大小正比于每个节点上存储块数和节点数的乘积，这在少于 100 个处理器时还可以忍受，但当处理器数不断增加时，必须寻找一种办法使得目录结构不至于过于庞大，现有的方法是在目录表中保留尽少的存储块的信息（如只保留那些已被高速缓存所缓存的存储块而不是全部的）或者使目录表的每项含有较少的位数。

为了防止目录成为系统瓶颈，可以把目录项分布到各个节点上。这样访问不

同的目录项可以寻址到不同的节点上。在分布式目录表中，每个处于共享状态的高速缓存块的信息只会存放在一个节点粘，这就避免了广播。

在多级网络中，高速缓存目录存放了有关高速缓存行副本驻留在哪里以及处于何种状态的信息，以支持高速缓存一致性。各种目录协议的主要差别是目录如何维护信息和存放什么信息。

根据目录的存放方式，目录技术可以分为两种：

1. 中心目录

用一个中心目录来存放所有高速缓存目录的拷贝。中心目录能够提供为保证一致性协议所需要的全部信息。因此，其容量非常大且必须采用联想方法进行检索，这和单个高速缓存的目录类似。大型多处理器系统采用中心目录会有冲突和检索时间过长两个缺点。

2. 分布式目录[Censier 78]

在分布式目录中，每个存储器模块维护各自的目录，目录中记录着每个存储器块的状态和当前的信息，其中状态信息是本地的，而当前信息指明哪些高速缓存中有该存储器块的拷贝。

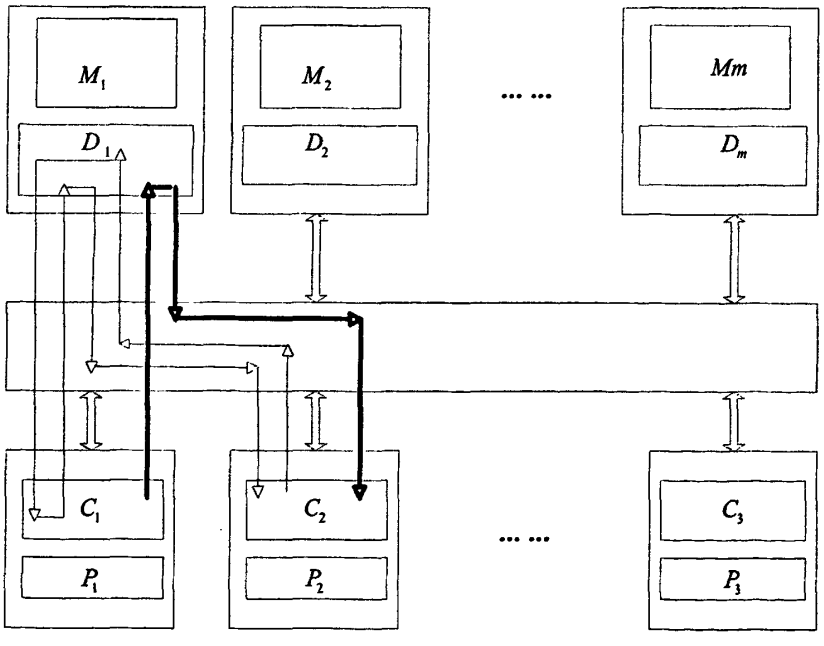


图 2.4 基于目录技术的高速缓存一致性方案的基本原理

图 2.4 中高速缓存C_2的读缺失（图中用细线表示）将产生一个请求，并送给

存储器块，存储器的控制器将该请求再传送给高速缓存 C_1 中的脏拷贝（也就是重写拷贝）。这个高速缓存再把此拷贝写回存储器，于是存储器模块就可以向请求的高速缓存提供一份拷贝（也可以直接将干净的拷贝传给发出请求的 Cache）。在高速缓存写命中时（图 1.4 中用粗线表示），它就发一个命令给存储器的控制器，存储器的控制器再发无效命令给在目录 D_1 的当前向量中有记录的所有高速缓存（高速缓存 C_2 ）。

不使用广播的高速缓存一致性协议，必须将所有高速缓存中每个共享数据块拷贝的地址都存储起来，这张高速缓存地址表，不管它是集中的还是分布的，都称为高速缓存目录。每个数据块的目录项中包含有大量的用来指明块拷贝地址的指针，还包含一个用来说明是否有一个高速缓存允许把有关的数据块写入的脏位。

根据目录的不同组织方式，目录技术可分为位向量目录，有限指针目录，链表目录等。

位向量目录中的每个目录项有一个 N 位的向量，其中 N 是系统中处理机的个数。位向量中的第 i 位为“1”则表示此存储行在第 i 个处理机中有备份。另外，每个目录项有一个改写位，当改写位为“1”时，表示某处理机独占此行，并已将其改写。美国斯坦福大学的 DASH 系统就采用了位向量协议[Lenoski D 1990]。位向量目录所需的目录存储器的容量随处理机的个数 N，共享存储容量 M 的增加而以 $M \times N$ 的速度增加。当 N 很大时实现目录协议的开销很大。在大多数情况下，一个变量每次只被少数几个处理机共享，因此可以用有限个指针指向当前持有此变量的那些处理机。每个指针需要 $\log_2 N$ 位，整个目录所需的存储空间为 $O(M \times \log_2 N)$。由于每个存储行只有有限个指针，当共享某存储行处理机个数多于指针个数时，就导致指针溢出。处理指针溢出的方法有指针替换[Agarwal A1988]，软件支持[Chaiken D 1991]等等。

另一种减少目录存储容量的方法是，把目录分布到各个 Cache 中。具体做法是，把所有持有同一存储行的 Cache 行用链表链起来，链表头在存储行处。当某一存储行被缓存到某个高速缓存中时，就把相应的 Cache 行链到此存储行的链表头所指的链表中。当某一存储行在某个高速缓存中的备份变无效或被替换时，把相应的 Cache 行从此存储行的链表头所指的链表中删去。链表可以是单向的，也可以是双向的[Thapar 1993]。

在上述目录组织方案中，每个目录都有一个固定的宿主结点。处理机方存失效时，可以直接根据失效地址查找相应目录项。上述目录组织结构适合于 NUMA 结构的系统。这种目录组织方式的缺点是，每次访存失效时都需进行远程访问目录。可能宿主目录组织方式提供了另一种解决方案。在此种方式中，每个共享块都有一个持有该共享块最新备份的宿主结点。对于每个共享块，每个处理机都保

有一个可能宿主指针。该指针的含义是该指针“很可能”指向该共享块的宿主结点。如果可能宿主结点没有指向真正的宿主结点，则通过该指针所指的处理机中的可能宿主指针往后找，最终总能找到宿主结点。当处理机收到关于某个共享块的无效信号，或者释放该共享块的宿主权，或者转发该共享块的取数请求时，就修改该共享块的可能宿主指针。

在大多数情况下，可能宿主目录可以减少远程目录的查找次数。因此，可能宿主目录适用于远程访问延迟很长的系统或没有宿主结点的系统。

2.1.3 令牌技术

前面提到的 Cache 一致性技术，无论是侦听还是目录技术，都要求对消息的传递进行协调，以及精心设计状态机的转换来保证其正确性。另外，随着机器规模的增大，互连网络的复杂性也随之提高，也对 Cache 一致性的正确性和效率带来了影响。为了解决这一问题，一种被称为 TokenB 的令牌技术被提出[Martin 2003]。

令牌技术为每个 Cache 块分配一定数目的令牌。当某处理机要写某一高速缓存块时，该处理机必须获得所有令牌；若要读某高速缓存块，则只需获得一个令牌即可。通过这种方法，只需要通过对令牌进行交换和读数即可保证 Cache 的一致性。

令牌技术可以使处理机能够无视消息传递的顺序。通常情况下，处理机按照一个被称为“性能策略（performance policy）”的方式来获取令牌。比方说，一个处理机可以推测自己所需要的令牌目前被哪些处理机所占有，然后直接发消息给这些处理机。

当然这种推测可能会是错误的，获得令牌的请求会失败。为此令牌技术提供了一个“正确性柜架（correctness substrate）”来保证一致性请求最终都能成功。

正确性柜架由两部分构成：安全性和活性。安全性保证每时每刻通过对令牌的计数来维护 Cache 的一致性。活性则保证所有处理器的一致性请求最终都能够得到满足。由于性能策略所使用的一致性请求（称为暂时性请求）可能会失败，正确性柜架提供了一种始终都能成功的一致性请求方式（称为永久性请求）。

TokenB 中的性能策略会广播 GETM 和 GETS 两种一致性消息到系统中所有结点。当收到消息后，其他结点可能会响应以令牌和相关数据。令牌当中的“宿主令牌”指示出应由哪个结点来提供干净的数据。由于 TokenB 实施于无序的网络上，且没有一个同步点，导致竞态条件可能会出现并最终使一致性请求失败。比方说，处理器 P1 和 P5 同时针对某 Cache 行发出了 GETM 请求，处理器 P2 响应 P1 以部分

令牌，而同时处理器 P6 响应 P5 以部分令牌，此时则导致 P1 和 P5 的请求均以失败告终。为了解决此问题，TokenB 中设置了一个超时。当超时发生时，TokenB 会先重试几次，若再失败，则将调用永久性请求。

在令牌技术中的高速缓存替换较为简单。将发生替换的处理机只需将相关的令牌发给相关内存控制器即可，无需再发送其他消息。对令牌的计数可以保证高速缓存的一致性，而不用考虑一致性请求消息与高速缓存块的写回消息之间的竞态。

2.1.4 MOSI 协议

MESI(Modified Exclusive Shared Invalid)协议最初是由 Illinois 大学 Urbana-Champaign 分校的研究者提出的[Papamarcos 1984]。

许多现代微处理器设计中已经实现了 MESI 协议，如 Intel Pentium、i680、PowerPC 601 等。与 MSI 协议[Baskett 1988]比较，MESI 协议增加了一个“互斥（E)”状态。原因如下：小规模的 SMP 机器的一类主要的工作负载是顺序程序，假如采用 MSI 协议，当一个顺序程序先读入一个数据项，然后修改一个数据项时，将会触发两个一致性消息：首先是一个 BusRd（总线读，用来读一个该处理机不打算修改的数据块）事务，用来得到内存块，并置为 S 状态；然后产生一个 BusRdX（总线互斥读，用来获取一块该处理机打算加以修改的数据块）事务，用来将该块状态从 S 变成 M 状态。而在顺序程序中数据项不存在共享者，因此只会在一个高速缓存中有该内存的副本，后一个 BusRdX 事务是不必要的。为了改进此一状况，MESI 协议加入了一个“E”状态，用来表示只有一个高速缓存中有这个内存块，且该块内容没有被修改过。这样的话，只要开始发出 BusBd 事务，得到内存块，并置为 E 状态，就可以直接进行修改，而不需要产生 BusRdX 事务。

MESI 协议由四个状态组成：“M”（修改过，或称脏状态），“E”（互斥干净），“S”（共享）和“I”（无效）。其中，“M”和“I”状态和 MSI 协议中意思是相同的；“E”状态表示只有一个高速缓存中有这个内存块，且该块内容没有被修改过，也就是说，主存中是最新的。共享状态表示有两个或更多的高速缓存中有该块内存的副本。加入“E”状态的一大好处是，当同一个处理器随后对该块进行写操作的时候，不需要产生总线事务或使无效消息等。事实上将“E”状态转化为“M”状态的情况很少发生。并且加入“E”状态的开销很小[John 01]。

图 2.5 中所示是 MESI 协议的状态转换图。表 2.1 对其中的标识进行了解释。

表 2.1 MESI 协议中的事务的意义

PrRd	处理器读
PrWr	处理器写
BusRd	总线读
BusRdX	总线互斥读
Flush	高速缓存中的数据块放到总线上
Flush'	负责提供数据的高速缓存中的数据块放到总线上
——►	处理器发起的事务
·····►	总线侦听器发起的事务

图 2.5 中的记号与 MSI 状态转换图相似，具体的协议转换也与 MSI 协议类似。若在无效状态（I）下有处理器读（PrRd）发生，则根据 S 线是否被置位，来决定是转换到 E 状态还是 S 状态。若置位则转换到 S 状态，否则转换到 E 状态，其中 BusRd(S)表示在 BusRd 事务中 S 线被置位。在 E 状态下，若发生处理器写(PrWr)，将导致由 E 状态转换到 M 状态，不产生总线事务；若观察到一个总线读（BusRd）事务，将导致由 E 状态转换到 S 状态，同时由该高速缓存提供数据；若观察到一个总线互斥读（BusRdX）事务，将导致由 E 状态转换到 I 状态，同时由该高速缓存提供数据。其他状态转换与 MSI 协议类似。

还有一个不同点在于高速缓存和主存中都有最新数据时由谁来提供数据。在 S 状态下，若观察到一个总线读（BusRd）事务或总线互斥读（BusRdX）事务，在进行状态转换的同时，在 MSI 协议中由主存提供数据，而在 MESI 中由处于 S 状态下的某一个高速缓存来提供数据，其他的高速缓存不产生动作。这一点在协议设计时可根据不同出发点来考虑。

另外值得一提的是，MESI 协议要求高速缓存的写回策略。

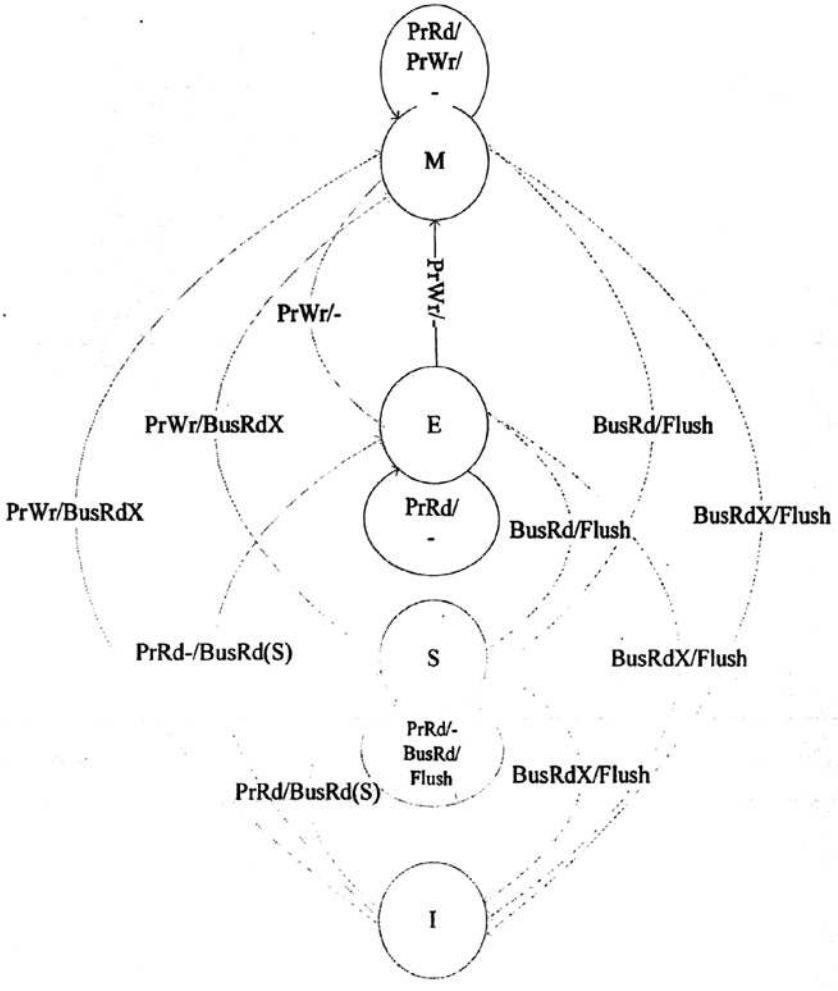


图 2.5 MESI 协议的状态转换图

2.1.5 MOESI 协议

相较于 MESI 协议，MOESI 协议增加了一个“O”（Owned）状态[Sweazey 1988][Zhou 2006][Hennesy 2007]。一个内存块处于 O 状态意味着本地处理机拥有该内存块的同时，其他高速缓存内可能拥有该块的处于“S”状态的副本[Wang 2007]。与高速缓存中副本全部是 S 状态的情况不同的是，最多只能有一个副本处于 O 状态。当有内存块处于 O 状态时，内存中的内容很可能是无效的。

必须指出的是，O 状态的实质就是指明由当前 Cache 负责为所有处理器或内存（当写回时）提供最新的数据。加入 O 状态的目的是，当其他 Cache 需要该数据时，可以直接从当前 Cache 中取数据。这样可以减少对片外带宽的占用，且提高了数据传输速度。在 AMD-760 MPX 中[Johnson 2002]即采用了该协议。

图 2.6 中示意了有两个处理器的 AMD-760 MPX 系统的体系结构。每个处理器

的高速缓存都被映射到内存的某个独立部分。当处理器 1 需要从主存中获取存储在某个数据块中的金融信息时（图中“Line-2”），若该数据块在处理器 0 中处于“Owned”状态，且同时 PCI 总线上的某个主设备要往主存中写数据，在该种情况下 MOESI 协议将带来较好的性能。这是因为处理器 0 是“Owner”，可以直接向处理器 1 提供数据，而处理器 1 也无需读取主存。这样 PCI 总线上的主设备可以并发的写主存（图 2.7）。这种并发性可以提升系统性能。在某些其他的系统中，PCI 总线上的主设备写内存之前必须让处理器 0 完成对主存的写更新。

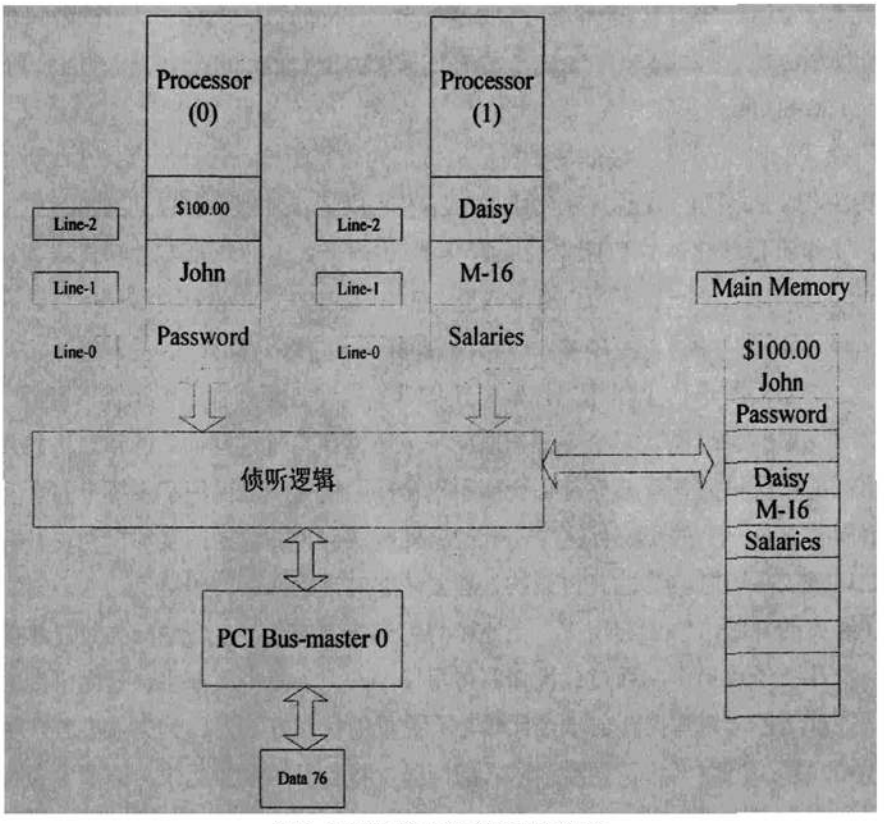


图 2.6 事务发生前高速缓存状态

2.2 一致性协议优化

关于高速缓存一致性协议的研究工作一直是计算机系统结构学科的热点。历史上已经有许多相关研究。限于篇幅，本节将只介绍一些比较具有代表性意义的工作。本文中把这些工作依优化对象划分成三大类：硬件层次的优化、软件层次的优化以及软硬件结合的优化技术等。

2.2.1 硬件优化

高速缓存一致性协议对机器的访存行为至关重要，协议的性能对机器的访存性能也是影响巨大。由于大部分处理器的一致性协议通过硬件实现（比如DASH[Lenoski 1990]，Origin[James 2005]，AlphaServerGS320[Gharachorloo 2000],X1[Cray 1993]，等），且硬件本身的执行速度较快，所以硬件上的优化无疑是提高机器性能的重要途径之一。而当前多核处理器中采用硬件来实现一致性协议更是主流。比如有 IBM 公司的 Power4 处理器[Tendler 2001]、Everest 缓存控制器[Ashwini 2001]，Sun 公司的 Niagara[Kongetira 2005]，HP 公司的 Superdome[Gostin 2005]，Intel 公司的 E8870CC[Briggs 2002]以及 AMD 公司的 Opteron[Keltcher 2003]，等等。

研究者们已经提出过很多基于自适应技术的高速缓存一致性协议。这些协议借助硬件，基于不同的数据共享/访问模式，在程序运行过程中可以动态自适应地对访存操作进行优化。这里所指的数据共享/访问模式主要包括有数据迁移共享（migratory data sharing）[Cox 1993][Stenstrom 1993]、成对共享（pairwise sharing）[IEEE 1993]以及广共享（wide sharing）[Kaxiras 1999]，等。

[Gupta 1992]中的工作指出，数据迁移模式在许多并行程序中是普遍现象。造成这一现象的原因是，多个处理器对同一个数据块轮流进行读写操作。访问并改写由锁进行保护的共享数据往往会展示出这样的模式。锁保证了系统中的众多处理器同时只有一个处于临界区中，此时它拥有访问相关数据的全部权利。只有当该处理器放弃锁时，其他获得锁的处理器才能对该数据进行的访问。

在经典的 MOESI 协议实现中，上述读后写操作首先会产生一个读失效，然后产生一个升级失效动作，通过此操作获得写权限。为了减少所产生一致性事务的数目，针对这种读后写操作的一个比较显而易见的优化方法是，读失效事务产生后，“Owner”在将数据块发送给请求者的同时，将自己的置为无效。请求者会将请求得到的数据块置为“Exclusive”状态。这样当请求者接下来写该数据的时候，不会再产生其他一致性事务。虽然这种优化技术在具有大量读后写操作的程序中效果非常明显，却也有其劣处。[Kaxiras 1999]中指出，对于具有许多广泛共享访存模式的程序，此策略会给性能带来负面影响。

一种比较折衷的策略是，当收到一个读请求时，“Owner”只有在相关数据处于无效的情况下（所有处理器都没有该数据块的备份），才会返回“Exclusive”状态的数据块，以使请求者获得独占的访问权限。

自适应一致性协议中，预取技术较常被采用。预取技术可以有效地隐藏长延迟访存失效带来的开销[Byrd 1999][Nesbit 2004]。预取技术的一大缺点是，增加网络流量和污染缓存，等。一些研究者试图让生产者主动发出消息，把改写后

的数据直接发送给消费者以降低访存延迟[Koufaty 1995]。这些技术大多用在弱一致性或者释放一致性的环境下。

[Mukherjee 1998]中研究了基于推测方法的一致性协议实现。其研究表明，可以在缓存和目录控制器中应用基于地址的两级预测器（two-level predictor）来追踪和预测一致性事务。在此基础上，[Lai 1999]中使用了更为精简的历史纪录表，以改进预测器，并利用不同处理器发出请求的相关性加速读操作。[Kaxiras 1999]中则提出了基于指令的预测技术对多种访存模式进行优化，如数据迁移模式、广泛共享模式和生产者-消费者模式等等。

除了上述技术之外，还有许多工作中利用猜测技术来降低访存失效的延迟。[Lebeck 1995]中提出了动态自无效（dynamic self-invalidation）技术，处理器根据访存的历史信息推测地主动使无效已经拥有地数据块。这样可以一定程度上消除后续写操作时需要发出的无效消息，从而降低了写操作的开销。[Lai 1999]中使用了两级自适应预测器，更加准确地预测应该被无效的数据块和无效的时机。上述所有的技术都取得了很好的效果。这些协议都是实现在处理器内部的。

2.2.2 软件优化

硬件实现一致性协议的最大优点是性能好，但由于硬件本身的特性，使得其不够灵活。其一大缺点是，由硬件实现的一致性协议无法满足具有各种不同访存模式并行程序的需要，不能使特定程序的性能达到最优。

分布式共享存储系统（DSM，Distributed Shared Memory）一直是并行共享存储系统研究的热点领域。在一些分布式共享存储系统中，为了提高协议的灵活性，部分甚至全部利用软件来维护高速缓存的一致性。在[Carter 1991]的工作中，可以在离线时针对应用的通信模式和访存模式，针对性的选择不同的协议及其实现，能充分挖掘程序的性能。而在[Falsafi 1994]中的研究也指出，普遍的，不同的一致性协议更加适用于不同的应用。若协议选择得当，将能使程序运行时的性能得到大幅度的提高。

必须指出的是，由于 DSM 系统中的共享粒度较大（通常以内存页面而不是高速缓存块为共享单位），会增加很多伪共享情况的出现。

2.2.3 软硬结合优化

上文中所述及的软件优化方法和硬件优化方法各有优劣。硬件方法不够灵活，而软件方法则效率较差。这两种方法都能带来性能的提升，但对程序的局部性和伪共享问题过于敏感，使得它们在某些情况下表现出来的性能会非常差[Larus 1994]。于是又有很多研究工作尝试将两种方法结合起来。

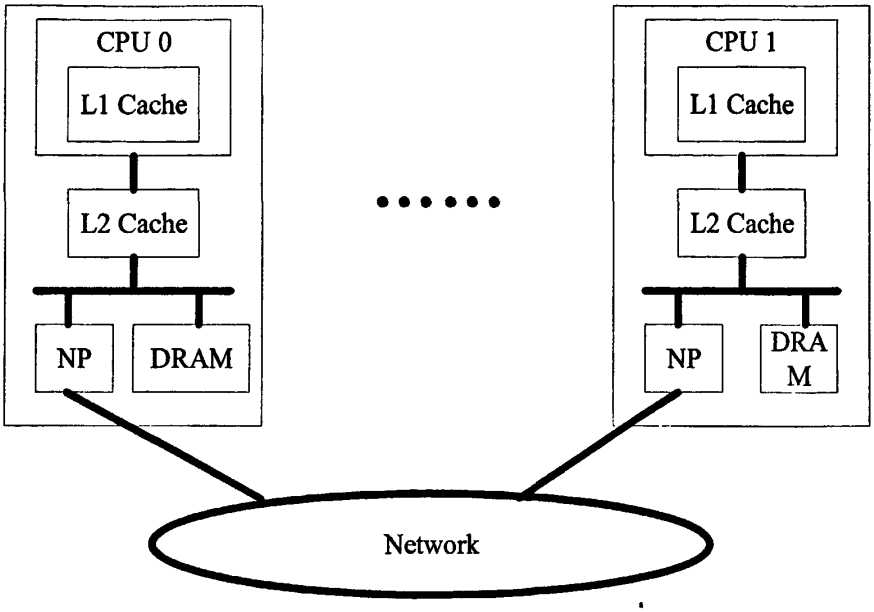


图 2.7 Typhoon 体系结构

在软硬件结合的方法中，比较有代表性的是斯坦福大学的 FLASH 系统（图 2.7，图 2.8）[Kuskin 1998]和威斯康星大学的 Typhoon 系统[Reinhardt 1994]。

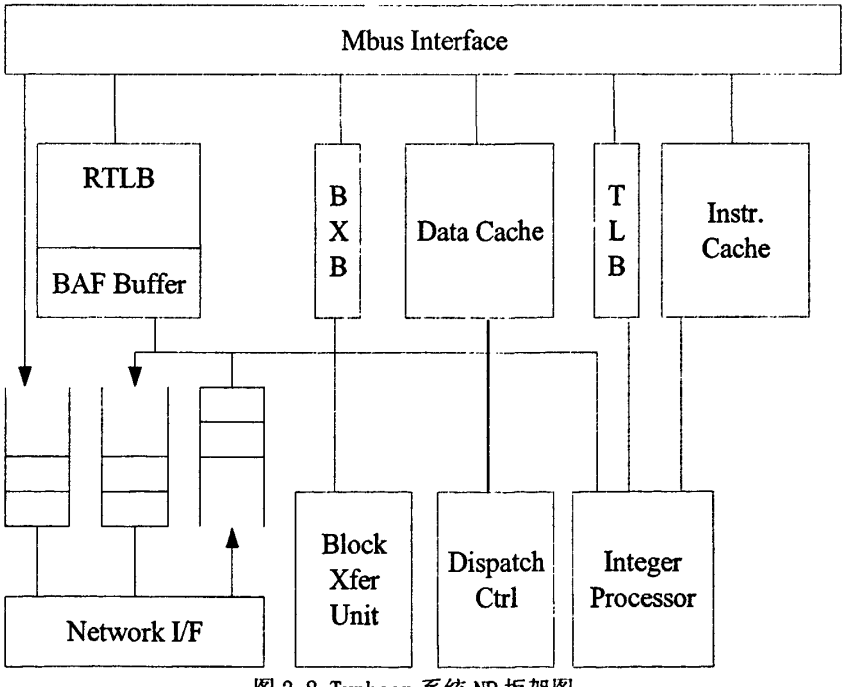


图 2.8 Typhoon 系统 NP 柜架图

Typhoon 和 FLASH 系统的最大创新是，将不同的一致性协议编程实现，再运行于系统中特别配备的专用堣处理器之上。这种方法即充分利用了硬件的高效，又因为将协议用软件实现，提高了系统的灵活性。从另一角度上看，它们是细度度（Cache 块）、硬件速度和软件灵活性结合的较好的传统分布式共享存储系统。在这两个系统中，一致性协议的选择和工作可完全经由软件控制。Typhoon 和 FLASH 系统相对前述软硬件协同设计的系统更加成熟，柔韧性也更强。

图 2.7 展示了 Typhoon 系统结构。Typhoon 中的结点全部是同构的完整的工作站。系统中各结点通过 point2point 高速网络互连。各结点的处理器采用了 Sun 公司的 SuperSPARC 芯片。为提高互连性能，还为各结点定制了专用的网络接口处理器（NP，Network Interface Processor，图 2.8），由总线连接到处理器.

图 2.8 中 RTLB 代表反向 TLB（Translation Lookaside Buffer），BXB 为数据块缓冲区，BAF 缓冲区保存数据块访问例外[Reinhardt 1994]。NP 包含了一个完整的 SPARC 体系定点处理器、I/D Cache、TLB 模块等。采用该专用处理器的目的，能够尽快的处理网络上传来的一致性事务，降低网络事务开销。这种做法可将主 CPU 从高速缓存一致性事务的处理中解放出来，以专注于程序的执行和计算，最终提高系统性能。

2.3 多核处理器（CMP）及 Cache 一致性

2.3.1 在多核处理器中实现 Cache 一致性

要充分的挖掘多核处理器的性能，将主要依靠对软件的并行性的挖掘。目前商用的并行软件中，其编程模型绝大部分都是共享内存式的。运行这些软件的处理器会对同一个物理地址空间进行访问。虽然这些处理器在逻辑上，访问的都是同一个内存，但片上的高速缓存系统对于获得良好的访存性能和提升程序的整体性能却具有关键的作用。

由此而导出的一个关键问题是，无论片上的高速缓存系统如何，必须要使这些共享内存的处理器对内存的访问是一致的，亦即维护高速缓存的一致性问题。对于所有的共享内存的系统来说，高速缓存一致是程序正确性的保障，同时也对系统性能有重要影响。

高速缓存一致性协议不仅决定了共享内存的多处理机之间是如何通信的，还决定了存储系统是如何在处理器、内存、高速缓存之间进行数据的传输的。可以预见的是，只要共享内存的编程模式存在一天，高速缓存一致性对系统的关键影

响也将持续一天。而对高速缓存一致性问题的研究也会一直保持着其重要性。

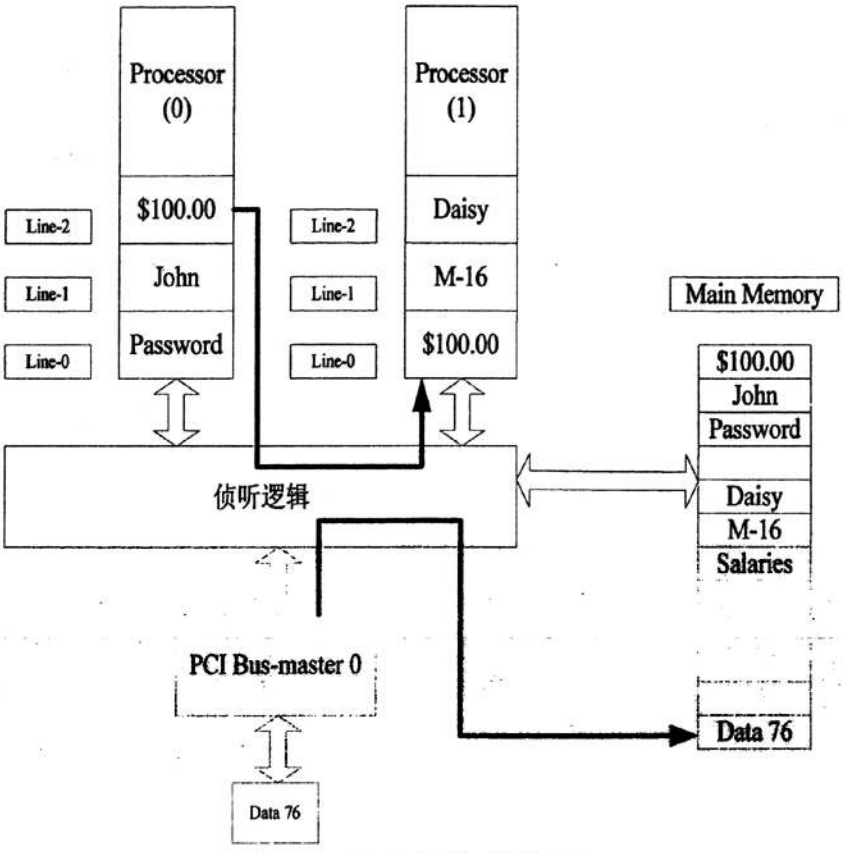


图 2.9 事务发生后高速缓存状态

在单个多核处理器中实现 Cache 一致性的方法，与传统的 SMP 机器有着很多类似的地方。所不同的是，由于多核处理器自身的一些特性（比如一致性的维护主要集中在私有的一级高级缓存中，线延迟对一致性的影响较大，等等）。

虽然历史对高速缓存一致性问题进行过很多研究，但研究的重点是基于 SMP 的，且构成这些 SMP 的处理器是单核处理器。因此随着多核处理器的广泛应用，在 SMP 的环境下研究多核处理器的高速缓存一致性问题有着重大意义。本文的工作将对此问题进行研究。

2.3.2 互连技术的影响

多核处理器的高速缓存一致性，与互连网络的排序特性是紧密相关的。相较于传统的多处理机系统，多核处理器中的互连网络将面对更多的工艺上的局限[Kundu 2006]。片上的互连网络除了与片外的网络之间在物理功能上有差别，片上互连网络还面临着其他一些特有的问题，比如对芯片面积的占用，对功耗的影

响，与处理器核之间的接口，等等[Kumar 2005]。而这些问题在单核处理器的设计中基本是不需加以考虑的。

历史上比较成功的绝大多数多处理机系统都使用总线作为处理器与存储器之间的互连介质。随着多核处理器片上集成的处理器核的个数的不断增加，使用总线将面临着可扩放性差的挑战。历史上的扩放性较好的多处理机系统往往会使用包交换方式互连，尤其当系统网络拓扑是网格型，等。多核处理器将来也许也会采用包交换方式互连，但以目前的工艺水平，在多核处理器片上实现包交换网络开销太大。

除此之外，在包交换互连网络上实现高速缓存一致性协议的开销也非常大，比如会要求多级的消息转发等。在多核处理器上一个比较好的选择是能够实现一个既能提供比总线更好性能，又能比包交换网络简单的片上网络。

已经有很多工作针对互连网络对共享内存多处理机系统整体性能的影响情况做了研究。[Strets 2000]中的研究表明，互连网络的某些特性（如对消息顺序的维护）可以极大的简化一致性协议的实现并且提升系统的性能。[Dai 1999]在网络接口的设计中提出阻塞相关的 FIFO 通道来解决访存请求顺序的问题。[Bilas 1999]中的工作提出使用 GeNIMA 技术来考察特殊的网络特性对 DSM 系统性能的影响。GeNIMA 系统以广播的形式将写操作通知给所有处理器，由于在网络中增加了合适的支持，使得广播的代价得到降低。[Landin 1991]的工作表明可以通过利用一类没有竞态情况出现的网络来消除目录协议中的 ACK 响应消息。

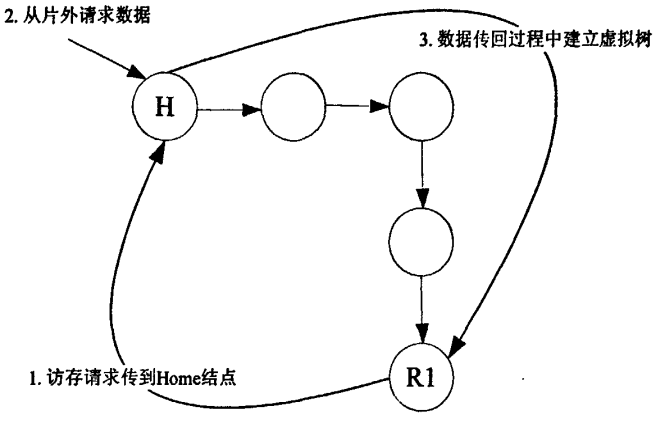
随着片上多核处理器的流行，片上互连网络已经成为一个研究热点，尤其是当片上可集成的处理器核的个数达到成百上千个时，片上互连网络的功耗和延迟问题尤其值得关注。[Kumar 2005]中的研究指出，并行走线、高带宽的总线结构的互连网络功耗将相当于几个处理器核的功耗。该研究表明这样的互连方式对未来的多核处理器/众核处理器研发是不可持续的。同时，由于互连导线的功耗和延迟等问题也影响到片上高速缓存系统的性能，目前已有研究者针对大面积片上 L2 Cache 的设计对互连技术展开了研究[Kim 2002]。

片上互连网络与高速缓存一致性之间的互相影响，也已经成为研究热点。

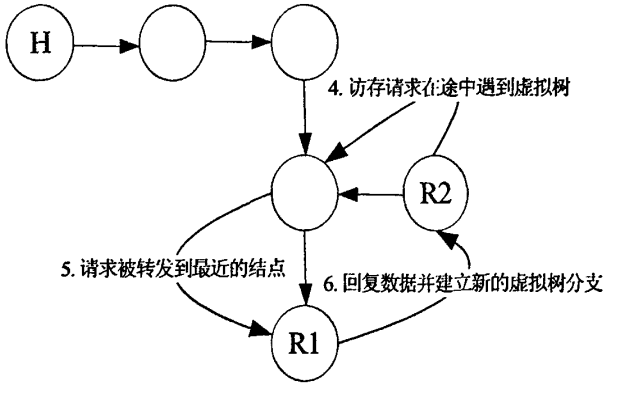
[Eisley 2006]中提出了一种分布式的解决方案，即将一致性协议的部分事务分布片上互连网络路由器中处理。在该实现方案中，缓存块的共享信息被储存在各路由器中，这些路由器以虚拟树的方式被组织起来。第一次访问该缓存块的处理器核成为树的根，缓存块的其他共享者成为树的叶结点。

该设计的工作流程是，当某个访存请示在片上网络传输时，其传输路径覆盖的每个路由器中的共享信息都将被检查。由于共享信息是由树的形式被分布，所以若该路由器所在树中有共享者，则该路由器将选择一个结点提供数据，并将请

求者用新的分支连入虚拟树中。从该流程可以看出，高速缓存块的共享过程就是虚拟树的构建生长过程。



（a） 一个新的访存请求(read1)



（b） 对同一数据块的另一请求(read2)

图 2.10 In-Network 中一致性协议工作示例

具体访存过程如图 2.10 所示。图 2.10（a）展示了当一个数据块尚未进入片上高速缓存时，访存失效后操作的过程以及虚拟树的生长过程。在数据被返回请求者的过程中，相关路由器开始被连接构成虚拟树。

图 2.10（b）中示意当某另外一个处理器访问该数据时，相关请求信息路经虚拟树的某结点（亦即路由器）时，路由器将寻找树中最近的叶结点（即保存有数据拷贝的结点），由该叶结点负责响应请求。在数据传回的过程中，新的虚拟树

分支又被建立。

由于此种方法可以将对一致性请求的处理分散到整个网络中，使请求能够最快地得到响应，减少了对目录访问的压力，使目录成为性能瓶颈的可能性变小。但该方案的一大缺陷是，由于请求地分布处理，会使一致性协议的实现中被引入许多竞态情况，使协议的设计验证变得更加复杂和困难。

目前还有很多工作针对互连导线的特性来研究高速缓存一致性协议。[Cheng 2006]的工作中，基于当前现有物理工艺，利用不同金属层实现了多种具有不同的特性的互连线（图 2.11）。

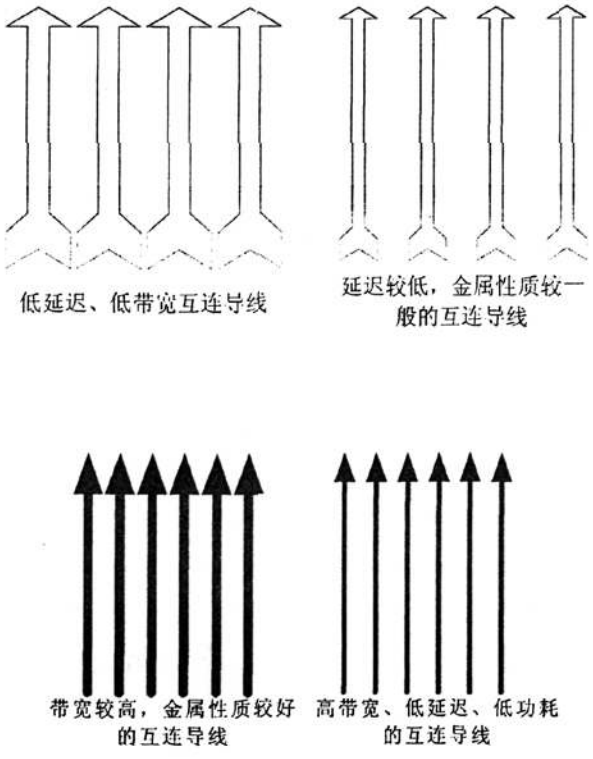


图 2.11 利用不同材料实现的互连导线

互连导线的延迟和功耗等特性由导线的宽度、长度等决定。其基本关系为：导线的宽度和间距较大时，则延迟较低，带宽较小（相同面积可以放置的导线数量减小）；反之，导线的延迟较高，带宽也较高。互连导线的延迟和功耗等特性之间也有类似的关系：当导线所经过的转发器较少时，延迟则较低，但功耗会变大。

[Cheng 2006]的工作中，根据一致性协议中不同的一致性消息类型对带宽、延迟等的具体需求，将相应的消息通过相应的互连线传输。这种方式可以有效的

提高降低延迟提高带宽，系统性能，且降低功耗。比如，将数据写回内存时，由于数据量相对较大，则可以将该操作通过高带宽、高延迟、低功耗的导线完成。而对于同步操作，由于这种操作所传输的数据量并不大，但对并行程序的性能影响非常大，则可以将这种一致性消息通过低带宽、低延迟的互连线来实现。这样做，虽然一定程度上增加了互连网络的功耗，但相对于系统性能所得到的提升，又是非常值得的。

2.4 小结

高速缓存一致问题向来是共享存储多处理机系统的核心技术问题之一，也一直是体系结构技术研究热点。随着片上多核处理器成为处理器设计和应用的主流，关于多核处理器的高速缓存一致性问题研究也已成为热点。

本章首先介绍了传统的 SMP 系统中的高速缓存一致性问题，包括具体协议实现技术（如利用总线进行侦听、利用目录和令牌，等），接着介绍了两个经典的，同时也是与本文工作联系较紧密的的一致性协议：MOSI 和 MOESI 协议。

由于一致性协议基本决定了处理器访存行为，且一致性事务数量大，所以一致性协议的性能对系统的性能影响非常大。本章接下来研究了一些一致性协议的优化技术。这些技术大致可以分为三类，即硬件优化技术、软件优化技术以及软硬件结合的优化技术，等。

片上多核处理器的引入，为一致性协议的设计、验证、实现和性能都带来了新的挑战。原因在于，多核处理器片上的多个处理器核（且处理器核数量将增加至成百上千，甚至更多）之间往往拥有私有 Cache、系统性能对片上网络更加敏感，等。基于此，本章最后简要介绍了目前在多核处理器中实现高速缓存一致协议的基本思想，并研究了互连技术对一致性协议的实现和性能的影响。

第 3 章 多核处理器 Cache 一致性及其包含与不包含

本章摘要 随着多核处理器的广泛应用，基于多核处理器的并行计算系统也已成为并行计算机的主流。这一趋势使传统的基于单核处理器的 SMP 并行计算系统的高速缓存一致性协议面临挑战。本章首先研究了由多核处理器构建的共享存储的并行系统（M-CMP），其次研究了在 M-CMP 中实现高速缓存一致性的技术。据作者所知，GEMS 工具集[Martin 2005]是目前唯一实现了目录 MOESI 协议的研究用开源模拟器。本文着重研究了目录 MOESI 协议的实现。提出了一个基于不包含策略的片内高速缓存系统，在此基础上对 MOESI 协议进行了优化。最后本章研究了多核处理器中的包含与不包含策略。

3.1 由多核处理器构建共享存储的并行系统(M-CMP)

3.1.1 层次化高速缓存一致性协议

无论是计算机研究机构，还是广大计算机厂商，过去都喜欢用商业硬件来构建规模更大、计算能力更强的并行计算机系统。不过过去的主流基本上都是基于单核处理器来构建共享存储的大型并行计算机系统。随着处理器技术的进步，以及多核处理器的日渐流行，无疑将来这样的大型并行计算机系统将主要基于多核处理器来构建。因此，在这样一种环境下来研究多核处理器的高速缓存一致性，具有重大意义。同时，随着多核处理器的引入，从存储的角度而言，系统进一步层次化。层次化的系统，也会要求研究层次化的高速缓存一致性协议。

以往的 SMP 机器，大多是使用商品化处理器的规模较小基于总线的系统。某些商品化的处理器，甚至可以允许使用它们直接搭建 SMP 系统，而无需其他附加的芯片。很多开发者正是利用这种小规模的 SMP 机器来构建更大规模的系统。在这种较大规模的系统内，SMP 机器作为一个结点出现，SMP 内部使用基于总线的技术来维护 Cache 一致性，SMP 之间则使用扩放性较好的目录技术。这正是典型的层次化协议：第一层是 SMP 内部的总线侦听协议，第二层是 SMP 之间的目录协议。

图 3.1 示例了 Sun 公司的 Sun Wildfiir 系统[Hagersten 1999]。图左侧的接口部分的作用是将片内的总线侦听一致性协议与结点之间的目录一致性协议桥接起来。

当总线上产生的某个一致性事务要求第二层的目录协议产生响应时，接口芯

片将会产生一个“忽略”信号，以使该总线事务被从总线序中移除，接着再处理余下的目录协议产生的一致性事务。一旦目录协议处理完所有请求，接口芯片将会把结果放到总线上。

上述的“忽略”信号可以很大程度上减化第一层协议和第层协议之间的交互动作。

现存的大部分使用层次化协议的系统中，第一层协议都是基于总线侦听技术的。[Marty 2008]中讨论了其他方式。

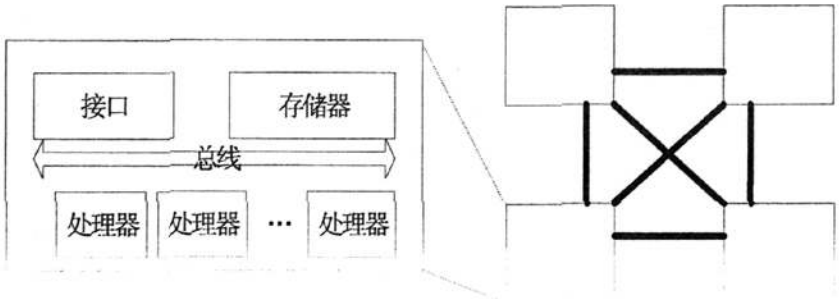


图3.1 利用基于总线的SMP结点构建大规模系统

存储系统向来都是复杂且难于设计、验证的。很显然的例子就是，很多市场上出售的商品里依然有很多设计上的错误和缺陷[Sarangi 2006]。而存储系统的复杂性，很大一部分是因为高速缓存一致性所导致。虽然目前模型检测技术能够帮助在处理器设计阶段就找出并改正很多错误，但随着多核处理器的广泛使用，以及多核处理器本身的愈加复杂，处理器设计的正确性，尤其是高速缓存一致性协议设计的正确性，未来将会面临重大挑战。

3.1.2 写无效和写更新

要维护高速缓存的一致性，必须满足两大充分条件[Hennessy 2007]。一是对数据的修改必须是对所有处理器不透明，即修改是全局的，能被系统中所有处理器观察到，这个条件被称为写传递；二是对同一内存地址的写操作所发生的顺序，对于所有处理器而言，必须是相同的，这个条件被称作写串行性。

当被多个处理器所共享的内存地址中的内容发生改变之后，必须要有消息通知所有相关处理器（修改者除外）。根据所发送的消息的类型，高速缓存一致性协议又可以分为两大种（图3.2）：基于写无效（invalidation-based）的一致性协议，以及基于更新的（update-based）一致性协议。

图3.2（a）中，只有内存保持有数据块a的有效拷贝；在图3.2（b）中，a拷贝被读入P1和P2两个处理器的高速缓存中。若处理器P1发出对a拷贝的写操

作，则基于写无效或写更新的一致性协议将会采取不同的响应。这也是两种一致性协议之间最大的区别。

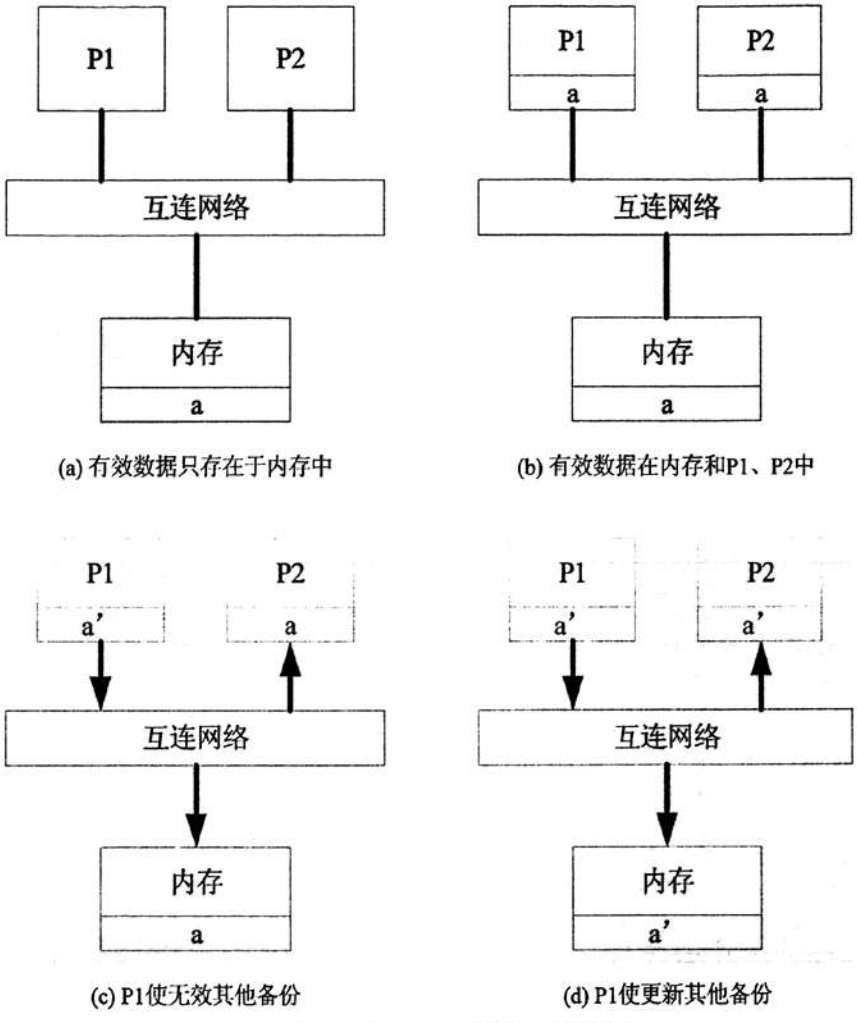


图 3.2 基于无效和基于更新的一致性协议

在基于写无效的一致性协议中，处理器 P1 必须先无效掉所有其他处理器缓存中的共享拷贝，以获得对数据块的独占权（“exclusive”），然后才能修改本地缓存中相应数据块，如图 3.2（c）所示。虽然在一致性协议在处理上述写操作的过程中，会发出一些处理器间的消息，但一旦 P1 获得独占权后，只要没有来自其他处理器的读请求（会使 P1 中拷贝会进入“Shared”或其他状态），处理器 P1 后续的写操作都将可以直接进行，无需再引起多余的一致性事务。

写更新协议的动作则如图 3.2（d）所示。处理器 P1 修改完本地拷贝后，将把更新后的内容发送给其他处理器中的高速缓存，以及内存。在收到这个更新的

消息后，其他处理器将更新本地相应拷贝块。

基于更新的一致性协议保证了系统中所有的数据拷贝都是及时和有效的，从而消除了伪共享带来的一致性无效情况的出现，但此方式可能会消耗大量网络带宽，某些情况下使系统性能大幅下降。关于前述两类协议已经有过不少相关工作。随着处理器中的缓存块位宽的增大，写更新协议会占用更多的带宽资源；另一方面，在写更新协议中实现写操作的串行一致性复杂度相对较高。

在多核处理器中，由于片内带宽相对较高，片内外消息传递速度差距较大，为写更新和写无效协议的设计增加了空间。

3.2 在 M-CMP 系统中实现 Cache 一致性

一个 M-CMP 系统，在本文中，系指由多个多核处理器构建的共享存储的并行计算系统。对于这些系统而言，高速缓存一致性包含两个方面：多核处理器内部的高速缓存一致，以及多核处理器之间的高速缓存一致。

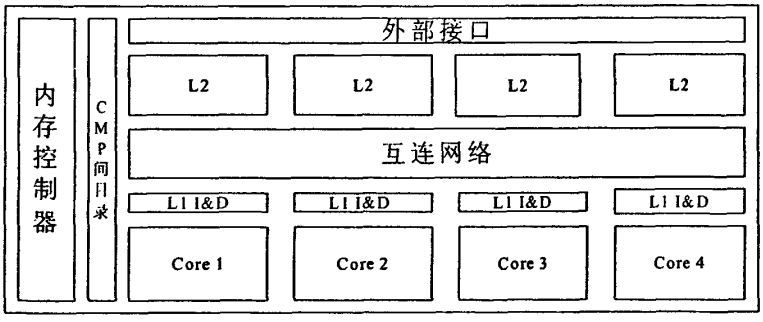


图 3.3 片内多核处理器内部组织。

片内一级指令/数据Cache私有，二级Cache共享。

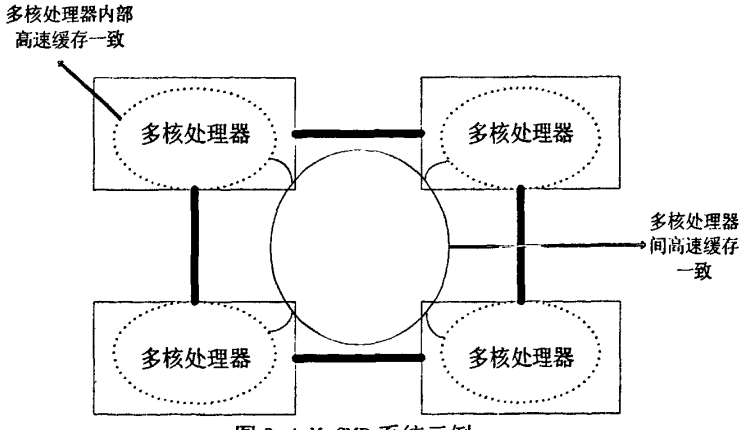


图 3.4 M-CMP 系统示例。

高速缓存一致性可以分成多核处理器片内及片间一致性两部分。

图 3.2 示例一个目前较主流的多核处理器的内部组织结构。图 3.3 示例由四个图 3.2 中所示的多核处理器所构建的一个小规模 M-CMP 系统。

一个比较简单的维护 M-CMP 系统中高速缓存一致性的方法是，不考虑多核处理器片内与片间之间的区别，而对所有层次的高速缓存一视同仁。然而这种方法的一个最大的局限性是，忽视了多核处理器片内通信的高带宽低延迟特性，而将导致系统性能的下降，和片外带宽的浪费。

3.2.1 令牌技术

在 2.1.3 节中提出的令牌技术中，对令牌进行操作的主体全部是由单核处理器构成的节点。由于每个结点中的高速缓存系统全部私有，所以无需考虑多级的高速缓存一致性问题。

本节提出的令牌技术（下文中简称为 TokenCMP）将针对 M-CMP 系统。

在 TokenCMP 技术中，接发令牌的主体是多核处理器内部的单个高速缓存。与 2.1.3 节中类似，每个内存块仍然对应固定个数的令牌。与 TokenCMP 技术对对应的另一种实现方式是，以单个多核处理器为单位来获取和释放令牌，再另外以一套协议来维护片内的一致性。该方式理论上会导致更大的复杂性，比如某个多核处理器内部的每个一级数据 Cache、一级指令 Cache、二级共享 Cache，等等，都将成为获取/释放令牌的主体。对此技术本节不予讨论。该技术将是未来的一个研究方向。

TokenCMP 技术施用于基于写分配且写回策略的片内高速缓存系统。这样意味着，当一个处理器核要读一个高速缓存块时，其一级私有数据（或指令）高速缓存必须拥有至少一个令牌；而当写一个高速缓存块时，其一级私有数据（或指令）高速缓存则必须拥有全部的令牌。

与 2.1.3 节中所述技术类似，TokenCMP 技术也提供永久性请求和暂时性请求两种请求获得令牌的手段。永久性请求的最终结果必然是成功获得令牌，TokenCMP 提供此种手段的主要目的是避免活锁和饥饿现象的出现。当然，永久性请求的实现必然会导致设计复杂度的提高。一旦永久性请求被调用，将在全局范围内保持激活状态，直到获得所有令牌。而对永久性请求的撤消，也是一个全局的操作。无论是请求还是撤消，都要求在全局内进行广播。除此之外，系统中的每个高速缓存还必须为每个永久性请求维护一个记录表，以避免出现各种竞态情况。

必须指出的是，在 M-CMP 系统中，由于多核处理器片内的数据传输数据远快于片间的速度，在 M-CMP 环境下，TokenCMPS 要想取得良好性能，则必须要考虑到这一性能影响因素。在[Martin 2003]中，由于多核处理器片内的任何一次缺失都将导致全局的广播消息到所有私有的高速缓存，使得其在带宽的占用，延迟的

降低等方面无法取得良好的效果。

3.2.2 目录技术

首先研究 GEMS 工具集[Martin 2005]中的目录一致性协议。该目录协议是阻塞型的[Hagersten 1999]。在该模拟器中实现了一个层次化的目录高速缓存一致性协议（下文中简称为基准 CMP 目录协议）。由于目录协议的良好的扩放性，使得片上的核的数目可以足够多，且 M-CMP 系统中的多核处理器数目足够多。

在基准 CMP 目录协议实现中，多核处理器片上有一个片内目录，以维护片内的高速缓存一致性，同时还有一个片间目录，以维护多核处理器片间的高速缓存一致性。图 3.4 中示意了 GEMS 所基于的多核处理器芯片结构。

在图 3.4 中所示多核处理器中，四个处理器核均拥有其私有的一级（数据/指令）高速缓存。二级高速缓存分成四个体（bank），为四个处理器核所共享。每个体（bank）维护一个片内一致性目录，即片内目录实际上存储在相应标识（tag）中的。同时，片上的内存控制器还会维护一个片间的一致性目录。片上互连网络用来在一级高速缓存控制器和片内目录控制器之间传输数据和控制消息。

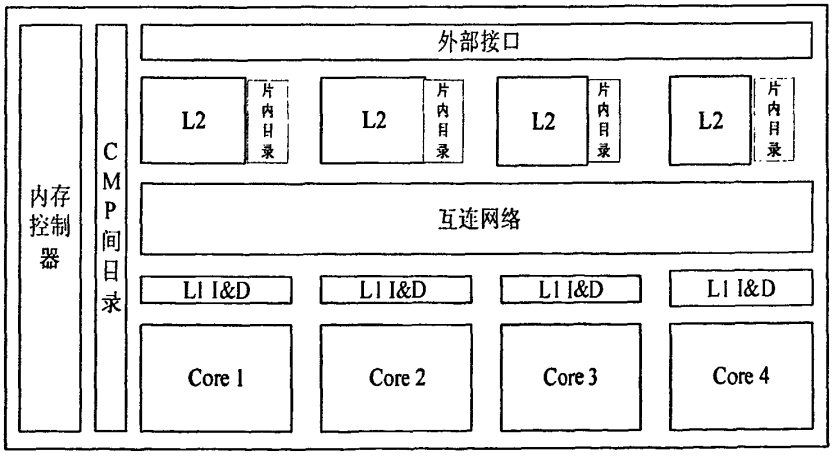


图 3.5 GEMS 模拟器实现目录协议所设计的多核处理器片上结构

在图 3.4 所示多核处理器中，一级高速缓存和二级高速缓存之间是严格包含的。在二级高速缓存中，在每个 Cache 块的标识（tag）里都有一个位向量，用来指示该块在哪些片内一级高速缓存中有副本，除此之外，还有一个指针用来标识哪个一级高速缓存是“Owner”。

CMP 片间目录用来追踪哪些多核处理器中缓存了某个内存块，但并不具体到某个高速缓存。CMP 片间目录会与别的 CMP 片间目录，以及某些片内目录进行消息通信，以维护一致性。

全局的高速缓存一致性通过片内和片间的目录的协同工作来维护。

当一个一级缓存缺失发生时，将会有一个一致性消息被发送到某个二级缓存体（bank），根据当前该块的状态，以及对该块的请求情况（比如对该块目前还有一些未满足的请求），二级缓存可能会进行以下一些动作：直接响应该请求，或将该请求转发到片内某个一级高速缓存，或将该请求转发到某个片间目录（该片间目录由该数据块的宿主内存控制器维护），或将该请求挂起，等等。当响应消息在网络中传输时，相应的高速缓存及目录状态会被更新。

下面研究 GEMS 工具集[Martin 2005]中的片内 MOESI 目录一致性协议。

表 3.1 片间目录所维护的稳定状态

状态	意义
I	无效
ILS	L2中无效，在L1中被共享
ILX	L2中无效，在某个L1中处于独占（E）或修改（M）状态
ILO	L2中无效，某个L1是Owner
ILOX	L2中无效，某个L1是Owner，且其他CMP中没有副本
ILOS	L2中无效，某个L1是Owner，其他L1中有副本
ILOSX	L2中无效，某个L1是Owner，其他L1中有副本，且其他CMP中没有副本
S	L2中处于共享（S）状态，且L1中没有副本
O	L2是Owner，且L1中没有副本
OLS	L2是Owner，L1中有副本
OLSX	L2是Owner，L1中有副本，且其他CMP中没有副本
SLS	L2中处于共享（S）状态，其他所有副本均在L1中
M	L2中处于修改过（M）状态，且独占

基准 CMP 目录协议中的两级目录都实现了所有的 MOESI 协议所要求的状态。片内目录中可能出现的稳定状态如表 3.1 所示。其他的状态（相对于表 3.1 中的稳态，还有很多暂态）及状态间的转换可以通过阅读 GEMS 源代码加以了解。

值得指出的是，在表 3.1 中，ILX，ILOX，ILOSX 等状态用来指示该多核处理器中某个高速缓存块是否处于“Owned”状态，且在其他 CMP 中没有副本。加入这

些状态的一个好处是，当该多核处理器内的某个核要写此高速缓存块时，一致性消息只需在 CMP 片内传送，而无需发到片外。

由于 GEMS 中实现的目录协议是阻塞的，则当某个处理器核的一致性请求到达片内目录时，若之前对同一高速缓存块的请求尚未完成，则该请求会被阻塞。

当处理器核的某个一致性请求成功后，则会发送成功应答消息给片内目录，以解除可能的阻塞，并开始处理被阻塞的请求。同时，目录内的状态会被更新，比如新增一个共享者，或进入某个稳态。

表 3.2 中列出了 GEMS 工具集中实现的 MOESI 协议中的各种一致性请求。

表 3.2 一致性请求类型

消息类型	意义
GETX	以Exclusive状态获得Cache块
GETS	以Shared状态获得Cache块
PUTX	将Cache块置成Exclusive状态
PUTO	将Cache块置成Owned状态
PUTO_SHARERS	置成Owned状态，但同时还有Shared状态副本存在
PUTS	将Cache块置成Shared状态
WB_ACK	写回请求应答（可以写回）
WB_NACK	写回请求应答（不可以写回）
INV	使无效消息

一个多核处理器片内“GETS”一致性请求可能通过以下两种途径得到完成：从片内二级高速缓存获得相关数据，或者从片内某个一级高速缓存获得相关数据。一个片内“GETM”一致性请求只有当获得相关数据并且收到所有片上高速缓存的“ACK”应答时才会成功。片内目录会将共享者的数目随一致性请求转发给 Owner，相应的 Owner 会响应以共享者总数以及相关数据。发出请求的一级缓存控制器会将收到的“ACK”响应的个数与共享者总数进行比较，且当所有共享者都返回“ACK”消息时，一级缓存控制器会发送一个消息给相应的片内目录，以示请求已经完成。

基准目录 CMP 协议中使用了三步写回策略，这样可以对写回操作进行排序，且可以阻止与其他写回请求的总态条件的发生。在三步写回策略中，高速缓存控

制器会发送一个“PUT”消息给目录。在“PUT”请求消息中会指出被写回的数据块是处于“Shared”、“Owned”或是“Exclusive”状态。收到消息后，目录会进入一个忙状态，以阻止一些况态情况的出现，同时进行响应。最后，当高速缓存控制器收到“ACK”应答时，则将数据传送给目录。另外，需注意的是，在将高速缓存块写回或替换时，需先将该高速缓存块使无效。

下面研究基准CMP目录协议中片间一致性协议。

基准CMP目录协议中，片间一致性协议的实现与上述片内一致性协议的实现大同小异。

片间目录存储在相应的DRAM中，且通过内存控制器来访问。片间目录的每个条目中都有一个位向量，用来指示相关数据块在哪些多核处理器中有“Shared”状态的副本，同时还有一个指针指示哪些多核处理器中有“Owned”状态或“Exclusive”状态的副本。必须指出的是，片间目录中的条目仅指出相关副本在哪些多核处理器中，而不具体到哪些高速缓存。这样做一个比较显著的好处是，比如某个由4个8核多核处理器构成的SMP系统中，一个片间目录的条目仅需占用6位（bit），其中4位用来指示共享者，2位用来指示“Owner”。

某个多核处理器所发出的一致性请求会使得片间目录根据目录状态响应请求者以相关数据，或者发出一些相应的一致性消息到其他多核处理器。片间目录也是阻塞型的。

当某个片内目录收到从某片间目录发来的“INV”一致性请求时，则必须首先完成使无效工作，然后才能响应该该请求，或返回相关数据块。比如，一个片间的“INV”请求，必须产生片内的“INV”一致性消息（这些消息将发送给片内共享者，或“Owner”）。在整个CMP响应以前，必须收集该CMP片内的所有应答消息，并对应答计数。

与片内一致性协议的实现类似，片间一致性协议也要求三步写回策略。

3.3 基于不包含策略的Cache一致性协议

由于多核处理器（CMP）中处理器核（core）的数量仍在持续增加，而同时处理器芯片I/O引脚的个数并不遵循摩尔定律[Tao 2008]，限制了处理器芯片对外的访存带宽。在这种背景下，处理器芯片的对外带宽将会成为处理器性能的瓶颈。提高片内Cache命中率，减少对内存的访问次数于是对于提高系统性能具有重大意义。而同时，限制Cache面积的增加，对于限制处理器的功耗同样非常重要[Zhang 2007]。下面将对采用不包含策略Cache的多核处理器的片内存储系统进行分析研究。

同时，如前文中所述，随着CMP的广泛使用，采用CMP构建的并行计算系统

也越来越多。这使得将Cache一致性的研究仅局限于单个CMP内部已经不能满足提高性能的要求[Marty 2005]。因此，本文还试图在一个由四个四核的CMP构成的SMP系统中研究Cache一致性对性能的影响。

3.3.1 片内高速缓存的包含(inclusive)性

片内多级cache的包含性可以分为以下三种：

1. inclusive（真包含）

上层Cache中的数据必然在下层Cache中有副本,比如所有L1 Cache中的数据必须在L2 Cache中有副本,而L2 Cache中的数据必然在L3 Cache(如果有L3 Cache)中有副本等等。

2. non-inclusive（非真包含）

上层Cache中的数据在下层Cache中可能有也可能没有副本。

3. exclusive（不包含）

上层Cache中的数据与下层Cache中的数据没有交集。比如AMD Opteron处理器就采用了此种策略，使片上Cache的面积理论上增加了1.128倍。

在我们的研究目标机器中，与[Barroso 2000]中类似，我们采用了不包含Cache，即一级高速缓存中的数据在二级高速缓存中没有副本，但片内其他一级高速缓存中可以有副本。这样做在提高片上Cache容量的同时，还可以降低维护一致性所需的开销。比如一种较极端的情况是，在真包含方式中，若二级高速缓存中的某块数据被丢弃，则所有包含此块数据的一级高速缓存都要采取相应动作。当片内处理器核和一级高速缓存的数量较大时，此种情况亦会带来较大的开销。

当然不包含方式也有其劣处，比如需要随时对一级和二级高速缓存中的数据进行同步，以保证严格的不包含等等。不过正如前文所述之理由，在多核处理器中采用不包含策略仍然值得我们去研究。

3.3.2 协议实现

本文中采用了基于目录的MOESI协议。

在本文的工作中，由于采用不包含策略，在目录中无需对同时出现在一级和二级高速缓存中的数据块加以特别的状态来进行标识，这样使得我们可以对MOESI协议的具体实现做出改进，减少了中间状态的数量（表3.3中给出了改进后的稳定状态），以及由此带来的维护一致性开销的减小。另外，我们的设计中，

二级高速缓存事实上扮演着一个“观测中心”的角色，因为所有一级高速缓存的通信（如请求，响应，等）都需经过二级高速缓存。这使得二级高速缓存可以在 CMP 间进行一致性协议通信的时候发挥更大作用，比如可以准确及时的对消息进行转发。

必须指出的是，“O”状态的实质就是指明由当前高速缓存负责为所有处理器或内存（当写回时）提供最新的数据。加入“O”状态的目的是，当其他高速缓存需要该数据时，可以直接从当前高速缓存中取数据。这样可以减少对内存带宽的占用，且提高了数据传输速度。在 AMD Athlon Mp 处理器中即采用了该协议。在单 CMP 系统中，只由“O”状态高速缓存负责为其他高速缓存提供最新的数据，在我们的实现中对此进行了扩展：即“O”和“S”状态的高速缓存均可以为其他高速缓存提供最新的数据。这样做的好处显而易见。比如 CMP1 中某高速缓存块处于“O”状态，而 CMP2 中的相应的高速缓存块处于“S”状态，则当 CMP2 中的某个核从所处 CMP 中获取最新数据的速度将远快于从 CMP1 获取最新数据[Kumar 2005]。

表 3.3 改进后的稳定状态及其意义

状态	意义
I	Invalid状态
LS	在L1中处于Shared状态
LX	在L1中处于Exclusive或Modified状态
LO	在L1中处于Owner状态，且其他CMP中有Shared状态数据
LOX	在L1中处于Owner状态，且其他CMP中没有Shared状态数据
S	在L2中处于Shared状态
O	在L2中处于Owner状态，且其他CMP中没有Shared状态数据
OS	在L2中处于Owner状态，且其他CMP中有Shared状态数据
M	在L2中处于Exclusive或Modified状态

当对处于“O”状态的数据进行写时，传统的 MOESI 协议实现方式会有两种一致性消息可能被触发。一是其他“S”状态的高速缓存块被使无效，一种是被更新。写更新占用较大的数据带宽，而使无效方式往往会忽略数据的局部性，并导致后来的高速缓存失效。为了尽可能地将存储访问满足于多核处理器片内，我们

提出了一种基于 location-MRU（mostly-recently-used）的实现方式。即根据其他 Cache 块中相关的处于“S”状态的数据的 MRU 状态以及高速缓存块的位置来决定采取使无效方式还是更新方式。工作流程如下：

1. 数据块 A 处于“O”状态，写请求。

2. 查目录，发出无效或更新消息，A 进入“OM”中间状态（等待并收集应答消息）。

3. 对本 CMP 内的所有“S”状态的副本使无效；对其他 CMP,若存在一个共享者，则使无效；若存在多个共享者，则对所有共享者的状态（这里的状态指的是，共享者的访问频度和 MRU 状态。参见 3.3.4 节）进行比较，挑选出一个候选者进行更新，置为“S”状态，其他的共享者置为无效。

GEMS 工具集[Martin 2005]中通过二级目录来维护一致性，这是因为同一个数据块，在一级和二级高速缓存中的状态可能不同。在 GEMS 工具信中，每个多核处理器的内存控制器需要维护一个目录以保证多核处理器间的一致性，同时，与前文中所述基准 CMP 结构类似，每个二级高速缓存体（Bank）还要维护一个目录以保证多核处理器片内的一致性。由于我们的工作中一级和二级高速缓存之间采用不包含策略，所以我们采用一级目录来实现（图 3.5）。这样做，不仅可以省去两层目录之间的消息接口，同时也简化了目录查找工作。比如，在前文基准 CMP 目录协议中，当一级高速缓存发生读/写失效时，需要发送一个消息到相应的二级高速缓存体（Bank）所维护的目录，接下来该目录可能会发消息给多核处理器片间目录，或者将消息转发到其他一级高速缓存。在这里，一致性消息会消耗大量带宽,并且会造成较大延迟。而在我们的设计中，只需要发送一次消息到目录就可以准确得到所需的信息。

同时,采用一级目录带来的存储开销并不大。比如在 Power6 处理器[IBM 2008]中,片内所有一级高速缓存(包括指令和数据 Cache)的总大小为 64*2*2 = 256 KB，其中数据高速缓存大小为 128 KB，而所有二级 Cache 的总大小为 4*2 = 8 MB.假设片内一级高速缓存块大小为 128 字节,同时假设目录内每个条目的大小为 16 bit，则当采用一级目录时，内存控制器所维护的目录的大小相对于二级目录情况下仅增加了 16 KB，仅占二级高速缓存容量的 0.2%。考虑到物理上的延迟，这一部分目录可以存储在离一级高速缓存较近的地方。

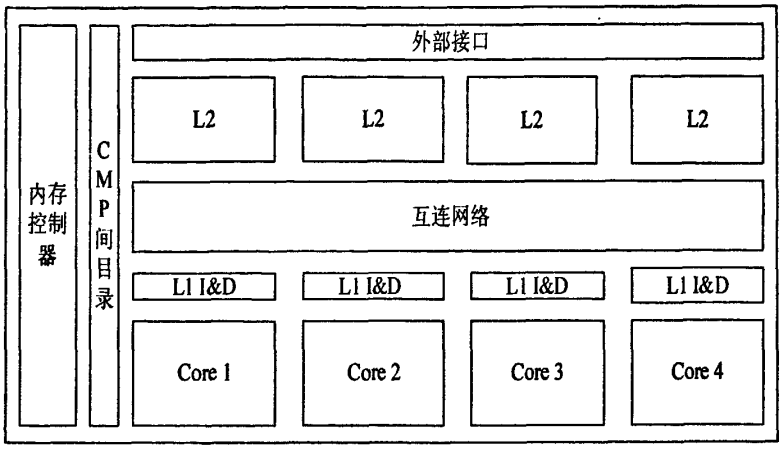


图 3.6 目标多核处理器结构

3.3.3 Cache 块的分配与替换策略

在本文的工作中，关于高速缓存块的分配和替换策略主要侧重于一级高速缓存上。这是因为据我们所知以往的研究中关于这方面的工作相对较少。同时因为一级高速缓存访问命中率的提高对于性能的提高相较于二级高速缓存的为大。

由于对一级指令高速缓存的访问具有良好的时空局部性，对一级指令高速缓存仍然采用 LRU（Least-Recently-Used）策略。下面讨论一级数据高速缓存的分配和替换策略。

当一次失效发生时，相关的数据将被直接取入一级高速缓存。由于采用不包含策略，二级高速缓存更象是一个大的 victim buffer。当一个一级高速缓存块被替换时，若其他一级高速缓存中没有副本，则该块将被视为 MRU 的块置入二级高速缓存中，否则直接丢弃。

对于一个组内的一级高速缓存块，我们不仅记录其 MRU 状态，与[Dybdahl 2006]中类似，还有一个计数器 Counter，Counter 的值越大，则该块被替换的优先级越低。与[Dybdahl 2006]中不同的是，我们另外为每个块增加了一个共享标识位 SC，当 SC 值为 1 时，表示片内其他一级高速缓存中还有该数据块的副本；若 SC 的值为 0，则表示该数据块是片内唯一。SC 位于目录内，这样便于更新 SC 的值。当 SC 的值为 0 时，我们认为该块被替换出去的优先级较低。这是因为通过片内网络在一级高速缓存之间传输数据的速度远快于从片外和二级高速缓存中取数据的速度[Kumar 2005]，所以我们尽可能将唯一的副本留在一级高速缓存内。考虑到替换算法实现的复杂性，在挑选被替换的对象时，我们把 SC 值为 0 的数据块的 Counter 值增加 60（这是一个经验数据，我们相信还可以有优化空间）。

3.3.4 实验

据作者所知，威斯康星大学所开发的 GEMS 工具集是目前唯一一个实现了 MOESI 协议的模拟器。所以本文的实验工作主要基于 3.0.31 版本 simics [Virtutech 2008] 和 2.1 版本 gems（加载了 opal 模块，参见[Martin 2005]）工具集，模拟了一个由 4 核 CMP 构建的 4 路 SMP 机器。由于一致性协议的实现和正确性验证非常困难，目前我们并无法保证我们的协议设计的完全正确，以及对 GEMS 模拟器修改的完全正确[Sarangi 2006][Joshi 2003] 。这一部分工作将会是未来研究的一个重点。

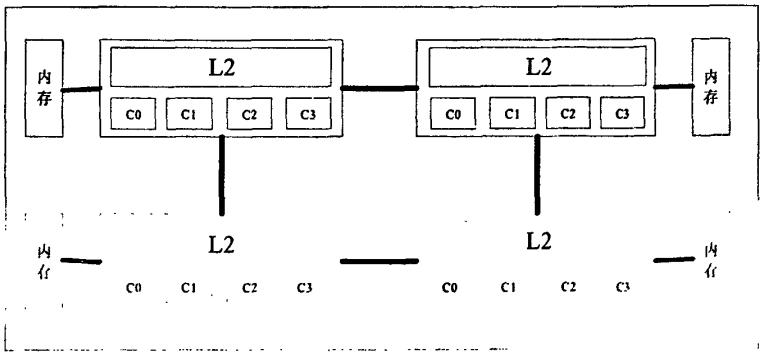


图 3.7 目标机器体系结构

在本文的设计中，每个 CMP 中的 L1 Cache 分为指令 Cache 和数据 Cache，大小均为 16KB。L1 Cache 每个核私有。L2 Cache 被片内所有核共享，大小为 16MB。其中 L1 Cache 为 4 路组相连，L2 Cache 为 16 路组相连，分成 4 个 bank。Cache 块大小为 64 字节，内存大小为 16GB，为四个 CMP 共享（图 3.6）。片内网络为点对点结构，速度为 16GB/s，片间网络速度为 8GB/s。

实验所用 benchmark 是由斯坦福大学开发的 splash2[Woo 1995] ，我们选择了其中九个程序，分别是：Barnes-Hut，FFT，Radix，LU-Contiguous，LU-Noncontiguous，Ocean-Contiguous，Ocean-Noncontiguous，Water-spatial，Water-nsquared。splash2 程序对体系结构底层变化非常敏感，十分适合用来做体系结构方面的研究。

为了节约实验时间以及更好的了解程序并行部分运行时的性能，我们利用 simics 提供的 API 接口，在编译 splash2 程序时加入相关制导语句，这样使得我们可以在程序进入并行部分时再加载 gems 模块，以及在并行部分结束时即退出模拟以提高实验速度(事实上由于模拟速度太慢，我们每个程序最多都只运行了十亿条指令)。我们的数据全部来自于上述并行部分执行时的统计。图 3.7 给出了每千条指令片内 Cache 失效次数的统计。图 3.8 给出了每个程序运行相同指令数时的时间加速比。这里的时间指的是“Ruby_cycles”[Martin 2005] 。

从图中可以看出，对于一些数据集较大的程序，比如 Ocean，其效率的提升较大。这正是由于片内 Cache 失效次数减少的结果。

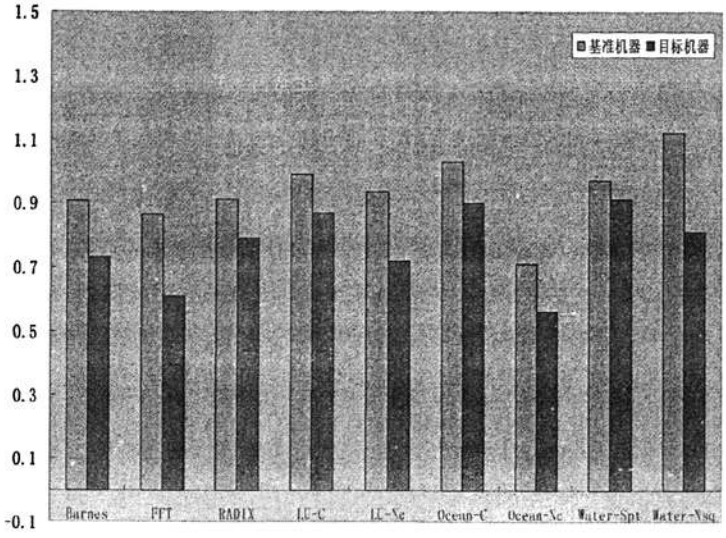


图 3.8 每千条指令片内高速缓存失效次数

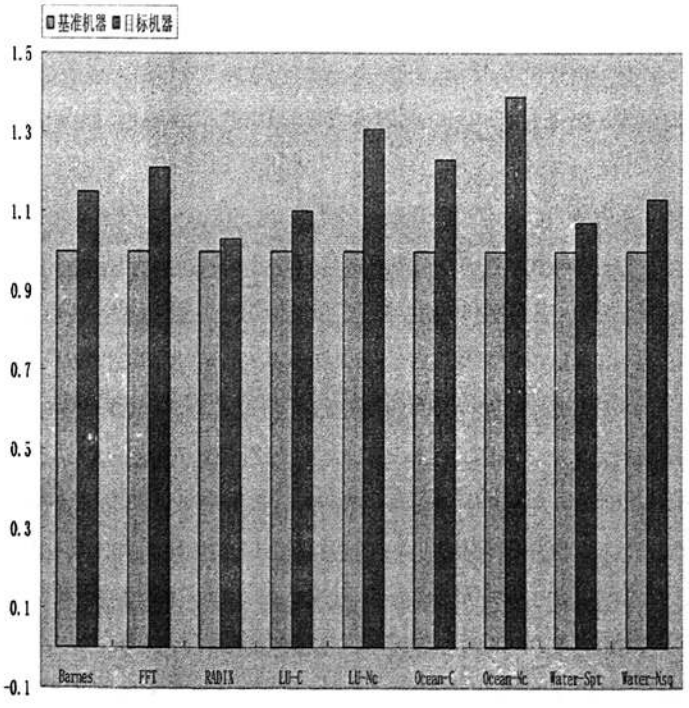


图 3.9 加速比

3.4 多核处理器中的包含与不包含

随着物理技术的发展，处理器主频的增速远超过 DRAM 存储器访问速度的提高，使得存储访问速度已经成为制约系统性能的瓶颈。为了解决这一问题，高速缓存在缓解处理器主频与内存访问速度之间的矛盾方面发挥了重要作用。

高速缓存的特点是面积小，容量小，同时访问速度快。高速缓存用来存储接下来最有可能被用到的数据[Smith 1982]。

对片上高速缓存而言，面积较小则访问速度较快，与处理器之间的速度差异也较小，但面积小的同时，也会限制高速缓存的命中率。同时，考虑到生产成本以及功耗等问题，片上高速缓存的面积也不可能做地过大[Baer 1988]。另外一种提高片上高速缓存命中率的方法，则是发展新的策略以充分利用现有的固定面积的片上高速缓存。

目前的处理器设计中，基本上都在处理器芯片中集成了多级高速缓存，这种设计方法可以认为是一种在片上高速缓存命中率和访问时间之间的一种折衷。而这些多级高速缓存内的数据的关系，可以大致分为包含，非真包含，不包含等三种。比如 Intel Pentium 4 Willamette[Intel 2003]的片上高速缓存采用了包含策略，而 AMD Athlon Thunderbird[AMD 2003]则采用不包含策略。采用包含方式时，降低了面上高速缓存容量的使用效率；而采用不包含方式，理论上则增加了片上高速缓存的容量。

3.4.1 单核处理器中的包含与不包含

在[Jouppi 1993]的工作中，研究了一个直接映射（direct-mapped）的 4 路组相联基于不包含策略的片内高速缓存设计。与传统的基于包含策略的片内高速缓存相比较，其性能有了很大的提高。不包含策略事实上可以使用在任何具有多级存储结构的处理器中。比如在[Wong 2002]中，研究了在网络存储设备中的基于不包含策略的高速缓存。

AMD 公司在 Athlon 和 Duron 两款处理器中实现了片上两级基于不包含策略的高速缓存系统，其体系结构如图 3.10 所示。二级高速缓存中的内容是由于冲突等原因从一级高速缓存中被替换出来的存储块。当一级高速缓存中发现一次失效时，将被替换出的高速缓存块被移动到牺牲性缓冲器（Victim Buffer）中，与此同时，二级高速缓存中的内容被索引。

在 Athlon 和 Duron 中，一级高速缓存大小为 128KB，且分为指令和数据两部分，大小均为 64KB。此容量基本保证了一级高速缓存的命中率，使得牺牲性缓冲器被完全写满的可能性不大。处理器可以在适当的时机（比如空闲时）将牺牲性缓冲器中的内容刷新到二级高速缓存中。但有时候会导致较高的二级高速缓存延

迟，比如当牺牲性缓冲器中没有空闲空间时，发生了一级高速缓存失效而同时二级高速缓存命中。在这种情况下，必须严格执行三步写策略，即先将牺牲性缓冲器中的内容写回到二级高速缓存中，再将一级高速缓存中的被替换块写入到牺牲性缓存器中，最后才将相关数据块从二级高速缓存中移动到一级高速缓存中。

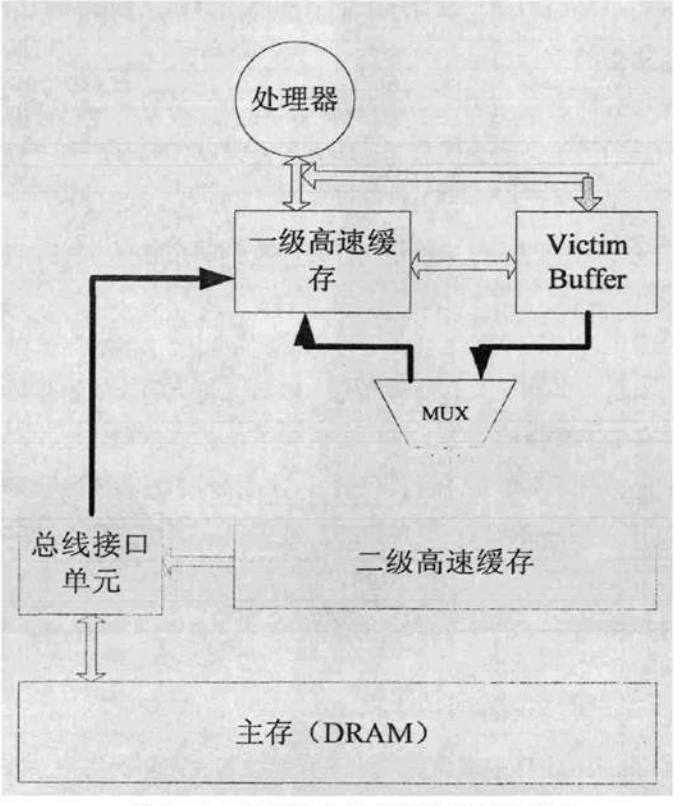
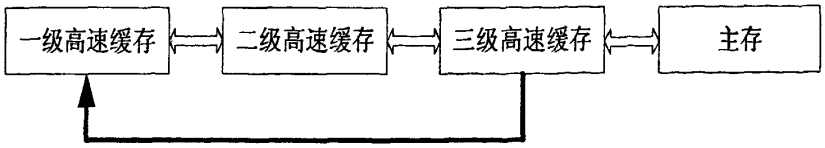


图 3.10 基于不包含策略的处理器体系结构

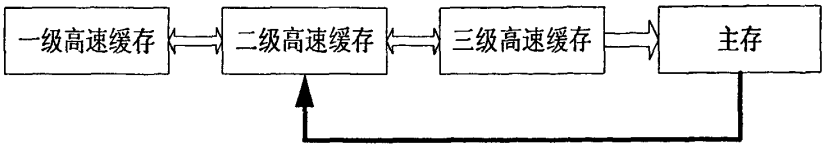
高速缓存的包含与不包含策略可以被推广到多级高速缓存中。一个多级高速缓存的设计方式可以有很多种，比如一级和二级高速缓存之间采用包含策略，而同时二级和三级高速缓存之间采用不包含策略，或者反之（图 3.11）。

若多级高速缓存中的数据块大小相同，则在它们之间传输数据时显得较为容易。否则，容易导致部分包含性。比如当一级、二级高速缓存之间的块大小不等且采用组相联方式时，或者二级高速缓存中的组的数目小于一级中的时，则会出现非真包含情况[Culler 2001]。

在对称多处理机中，采用真包含方式有利于降低维护高速缓存一致性的开销和复杂度。而若在该环境中使用基于不包含策略的高速缓存的处理器，则会增加片上面积的开销，以及访问高速缓存时的延迟。在 AMD-760 MPX [Johnson 2002] 中，一级高速缓存使用了侦听端口来维护一致性。



一、二级高速缓存之间不包含，同时一、二级中的内容均为三级缓存的子集



一、二级之间包含，三级和二级高速缓存之间不包含

图 3.11 三级高速缓存的包含策略方案

3.4.2 维护包含性

维护包含性不是一件容易的事情，即使在单核处理器中。以下两种情况必须加以考虑[陈国良 2002]：首先，必须保证一旦当二级高速缓存中的一个内存块被侦听到的总线事务或者一致性消息置为无效时，这个无效信息必须传到一级高速缓存；其次，当由于处理器的内存引用，而将新的内存块放入到一级和二级高速缓存时，如果要发生替换动作，此时必须要注意包含性的维护。初看起来，包含性似乎能被自动满足，因为所有在一级高速缓存中的缺失，都要到二级高速缓存中去，问题在于，两级高速缓存可能会选择不同的块替换出去。实际上，许多情况下，包含性不能被自动维持。下面来看一些情况[陈国良 2002]。假设一级高速缓存的相联性为 a1，组（set）的数目为为 n1，块大小为 b1，因此，一级高速缓存的总容量为 s1 = a1xb1xn1 。设二级高速缓存的对应参数分别为：a2，n2，b2 和 s2，并且假设所有的参数值为 2 的幂。

1. 第一种情况

假设一级高速缓存和二级高速缓存都是 2 路组相联高速缓存，具有相同大小的块（b1 = b2），二级高速缓存比一级高速缓存大 k 倍。并且假设一级高速缓存和二级高速缓存采用的替换策略都是最近最少使用（LRU，Least Recently Used）策略。现在来看一下以下不遵守包含性的情况。考虑三个内存块 m1，m2 和 m3，它们映射到一级和二级高速缓存中的同一个组。假设某个时刻，m1 和 m2 正处于一级高速缓存和二级高速缓存的一个组，再假设处理器要读内存块 m3，此时，一级高速缓存和二级高速缓存中都要将 m1 和 m2 中的一块替换出来。因为二级高速

缓存并不知道一级高速缓存中块的使用情况，因此很可能二级高速缓存替换出内存块 m2，而一级高速缓存却替换出另一个内存块 m1。实际上，只要一级高速缓存不是直接映射高速缓存，且采用 LRU 替换策略，则不管二级高速缓存的参数如何，总是会违反包含性。

2. 第二种情况

假设一级高速缓存分为指令和数据两部分，则即使一级高速缓存是直接映射的，并且具有同一个二级高速缓存，也有可能会破坏包含性。一个指令块 m1 和一个数据块 m2 在二级高速缓存中冲突，而在一级高速缓存中并不冲突，因为指数和数据在不同的地方。如果 m2 存在二级高速缓存中，而这时处理器读了内存块 m1，则 m2 将会被从二级高速缓存中替换出来，但不会从一级高速缓存中被替换出来。

3. 第三种情况

两级高速缓存具有不同的大小。考虑一个微型系统具有直接映射的一级高速缓存和二级高速缓存（a1 = a2 = 1），一级高速缓存和二级高速缓存的块大小分别为 1 个字节和 2 个字节（b1 = 1，b2 = 2），组的数目分别为 4 和 8（n1 = 4，n2 = 8）。因此，一级高速缓存容量为 4 个字节，位置 0、4、8、…映射到组 0、1、5、9、…映射到组 1，等等。二级高速缓存容量为 16 个字节，位置 0&1、16&17、32&33、…映射到组 0，位置 2&3、18&19、34&35、…映射到组 1，等等。可以看到，一级高速缓存中能同时包含位置 0 和 17，而二级高速缓存却不能同时包含这两个位置，因为在二级高速缓存中，它们映射到同一个组 0。

实际上，有些高速缓存配置能自动维持自动性。只要配置满足以下条件：一级高速缓存是直接映射的；二级高速缓存可以是直接映射的或组相联的，并且可以采用任何替换算法，只要放入一级高速缓存的块同时也一定在二级高速缓存中；块大小必须是相等的，一级高速缓存的组数可以小于或等于二级高速缓存中的组数。

但是，许多在实际中使用的高速缓存配置，在执行替换时并不自动维持包含性。包含性是通过扩展在高速缓存层次中传播一致性事件的机制来实现的。例如，一旦当二级高速缓存中的一块被替换出去，该块的地址被传给一级高速缓存，一级高速缓存将对应的块置为无效。

另外，在基于总线的系统中，处理总线事务和处理器写的功能也应被加强。二级高速缓存负责侦听总线，有些侦听到的总线事务既与二级高速缓存相关又与一级高速缓存相关，因此这些总线事务必须被传给一级高速缓存。比如，二级高

速缓存由于侦听到的一个消息而将某数据块置为无效，如果该块在一级高速缓存中有拷贝，则该消息必须也被传递给一级高速缓存。

在一级高速缓存的写命中的情况下，所做的修改应该被传给二级高速缓存，使得它能够在必要时提高最新的值。一种方法是，一级高速缓存采用写直达，且这种方法的一个额外好处是易于在单个周期内完成写操作。然而，写操作消耗了二级高速缓存的很大部分带宽，因此为了不让处理器停下来，必须在一级高速缓存和二级高速缓存中设置写缓冲（Write Buffer）。另外，一级高速缓存采用写回缓存也能够解决这个问题，因为只需要让二级高速缓存知道该块已经被修改了，当实际需要向总线提供该块的最新值时，可从一级高速缓存得到。这可以通过在二级高速缓存中同时置位修改位（“Modified”）和无效位（“Invalid”），引入Modified-but-Stale状态来实现。

3.4.3 多核处理器中的非真包含

片上多核处理器体系结构的出现，给处理器芯片的设计带来了一些很严峻的挑战，尤其是对存储系统方面。把多个处理器核及其私有/公有高速缓存集成在一块芯片上时，相较于以往的单核处理器设计，带来挑战的同时却也为设计者带来了更大的设计空间。

随着半导体技术的进一步发展，片上可集成的处理器核的数目仍然会迅速增加，这必然导致对内存的访问的次数、频率的加大和访问模式的复杂化；同时受限于芯片对外引脚的数目，对外访存带宽无法有效提高，这一瓶颈对系统的性能的影响将更加严重。

这此背景之下，对多核处理器片上高速缓存系统的包含性进行研究也开始引起了人们的重视。

在[冯昊 2008]中，对多核处理器中采用非真包含方式（non-inclusive）的片内高速缓存的设计和性能进行了研究。该工作中目标多核处理器体系结构模型为Dance-hall结构（图3.12），其片上高速缓存结构如图3.13所示。

该设计的工作原理如下：

1. 当图3.13中L2 Cache由于L1 Cache的缺失而命中时，则该命中高速缓存行与L1 Cache中被替换出来的进行交换（图3.13①）。

2. 由于设计了L1 Cache与下级主存间的独立数据通路，当L1 Cache缺失且在L2 Cache中继续缺失之后，处理器会继续访问主存（图3.13②），并将找到的数据直接从主存读入L1 Cache(图3.13③)，而无需经过L2 Cache。同时将L1 Cache

中被替换出的 Cache 行写入 L2 Cache 中（图 3.13①）。L2 Cache 中被替换出的数据将直接写入主存。

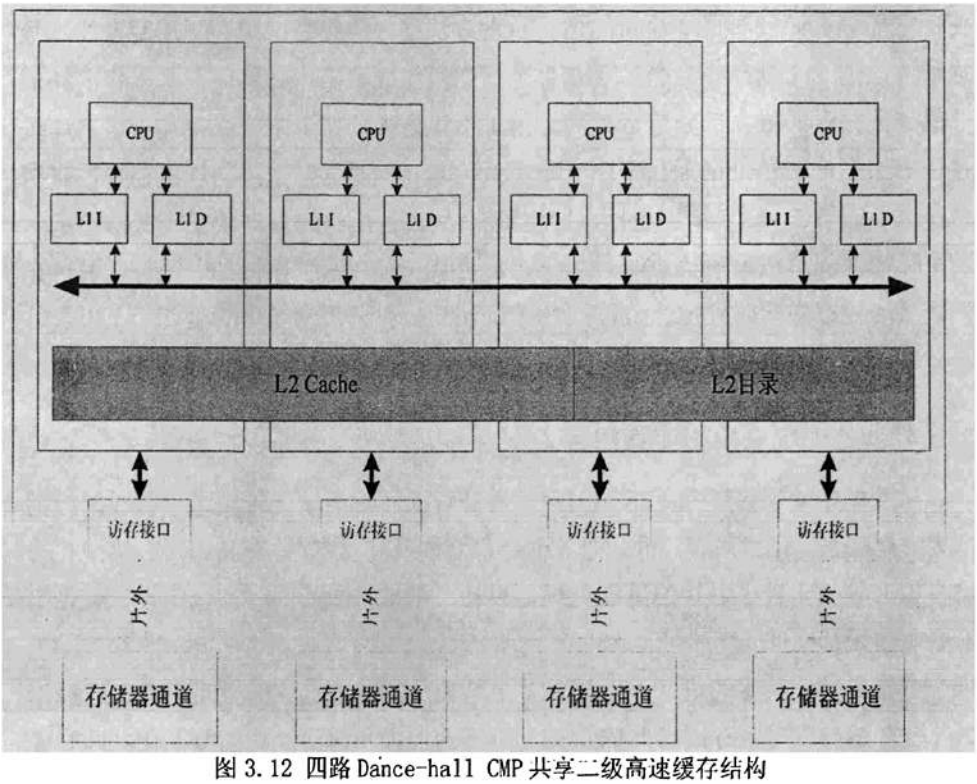


图 3.12 四路 Dance-hall CMP 共享二级高速缓存结构

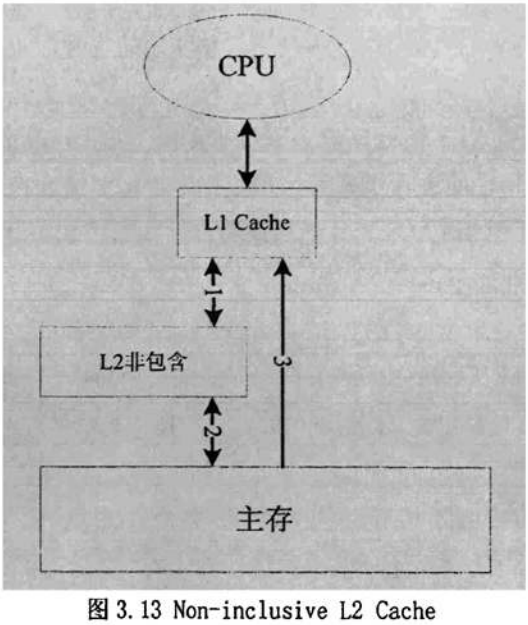


图 3.13 Non-inclusive L2 Cache

由此可见，non-inclusive Cache 的工作原理与普通基于包含策略的 Cache 工作原理有很大不同。在 non-inclusive Cache 中，L2 Cache 只接收上层高速缓存中被替换出的 Cache 行，并且命中时 Cache 行将进行对换，因此上级 Cache 中的数据并不一定存储在本级 non-inclusive Cache 中（注：non-inclusive Cache 并不能使所存储数据与上级 Cache 完全不同。单处理器情况下，当两级 Cache 的 Cache 行大小不同，或组数不同时，数据可能有重复。共享存储的多处理器情况下，由于不同处理器的 Cache 可同时保存同一数据副本，当某一副本被替换到下级 non-inclusive Cache 之后，也存在 non-inclusive 包含上级 Cache 数据的情况）。这种消除冗余数据副本的存储方式使得 non-inclusive Cache 具有增大 Cache 逻辑容量的作用。由上级容量为 C1 的 inclusive Cache 和本级容量为 C2 的 non-inclusive Cache 组成的 Cache 层次结构，实际相当于最大逻辑容量为 C1+C2 的普通基于包含策略的高速缓存。但在硬件上，non-inclusive Cache 的控制逻辑也要较 inclusive Cache 复杂。

[冯昊 2008]的工作中，研究了三级高速缓存采用不同包含性配置情况下多核处理器的性能。该工作使用了 splash2 基准测试程序集和 SESC 模拟器[Park 2001]。其研究结果表明，采用 non-inclusive 方式设计高速缓存，可以有效地减小片外访存次数，提高多核处理器的性能，并节省片上资源。随着半导体技术的发展，当有更多片上资源可以利用时，将片上最低级 Cache 设计成 non-inclusive 方式，能有效降低片外缺失率，减小程序平均访存时间，提高多核处理器的整体性能。

3.4.4 多核处理器中的不包含

从处理器的角度来看，基于不包含策略的 Cache 结构意味着，数据要么在 L1 中，要么在 L2 中，但不可能同时出现两级高速缓存中（以下研究不考虑三级或更多级高速缓存的情况）。由于当发生冲突或容量不够时，高速缓存块会从 L1 Cache 中被移动到 L2 Cache 中，所以事实上 L2 Cache 扮演着一个大容量的牺牲性高速缓存（victim cache）的角色。除此之外，L2 Cache 事实上还扮演着一个同步中心的角色：所有关于 L1 Cache 的同步消息（请求，响应，等）都须经过 L2 Cache。由于 L2 Cache 在不包含策略中的这一特殊功能，使得它可以在片间一致性的实现中发挥重要作用，比如可以精确地将一致性消息进行转发，等。

当 L2 Cache 收到某 L1 Cache 发出的一致性请求消息时，所请求的数据可能在 L2 Cache 中，也可能在片内某其他 L1 Cache 中，还可能在片外的某个存储器内。无论是上述哪种情况，L2 Cache 均能准确的进行消息转发，以及将响应数据进行转发。

当 L1 Cache 所请求的数据被返回时，根据 L2 Cache 及相关影子标识（shadow tag）中的内容，可以确定该数据块的一致性状态。比如该数据块可能处于“Shared”状态---同时在其他某 L1 Cache 中处于“Modified”，“Owned”，等状态。事实上，在请求消息被转发，以及数据块被响应的过程中，相关数据块是处于一个“暂态”。

不包含策略最大的好是，可以充分利用片上面积。由于只有极少量数据被复制（比如标识），相对于包含和非真包含两种策略，不包含策略可以使更多的不同的数据存在于片上高速缓存中。但不包含策略也有其劣处，除了需要随时对 L1 Cache 和 L2 Cache 中的数据进行同步外（参加 3.3.2 节），有时还会出现这样一种情况：

比如某 L1 Cache 所请求的数据在片内某另一 L1 Cache 中，则消息传递的跳数为 2：L1 Cache --- L2 Cache --- L1 Cache。而在包含方式中，只需一跳：L1 Cache --- L2 Cache。这是因为所有 L1 Cache 中的数据都在 L2 Cache 中被“包含”。

虽然上述情况看起来似乎不包含策略使得获取数据的速度下降了，但同时考虑到片上所存储的不同的数据的数量有了增加，所以本文仍相信基于不包含策略构建片上高速缓存系统值得研究。事实上，在上述例子中，不包含方式需要两跳以获取数据，但在采用包含方式的处理器中，性能也许会更差：由于片上数据相对较少，也许需要到片外去获取数据。

在实现上，不包含策略显得较为复杂。比如当某个 L1 Cache 的数据请求消息被 L2 Cache 转发到某个 L1 Cache 时，该 L1 Cache 正在将该数据写回。针对这种情况的一些解决办法是，使协议要求该 L1 Cache 在写回前将数据块置为一个被锁定的“暂态”，或者直接响应以数据块，以提高性能，且紧接着取消本次数据块写回操作（另一种可能是，若不取消本次写回操作，则应使协议允许暂时性的出现数据块同时出现在 L1 Cache 和 L2 Cache 中的情况）。

3.4.5 实验及分析

1. 实验环境

我们使用了由 NASA Ames Research Center 开发的 NPB 3.3 OpenMP 版本[NASA 2008]作为测试程序。NPB 包含有五个核心程序，是应该比较频繁的一些算法。这五个测试程序主要测试特点分别介绍如下[袁伟 2005]：

① IS

整数排序（Integer Sort）程序，主要测试整数运算性能和集合通信（Collective Communication）性能，对通信延迟很敏感。

② EP

密集并行计算（Embarrassingly Parallel）程序，通信很少，主要测试数学函数的浮点运算性能。

③ MG

三维多重网格计算（3-D Multigrid）程序，采用多重网格算法求解三维 Poisson 方程，要求处理器数目必须为 2 的幂次，主要测试规则（Structured）的非连续存储访问集合通信和点到点通信。

④ CG

共轭斜量法计算（Conjugate Gradient）程序，主要测试不规则（Unstructured）的集合通信和点到点通信。

⑤ FFT

快速傅利叶变换（Fast Fourier Transform）计算程序，用 FFT 求解三维偏微分方程，主要测试集合通信。

由于使用 GEMS 工具集模拟速度太慢，我们只选择了 MG 和 CG 两个程序进行了测试。对其他程序的测试将是未来工作的一个重点。

实验目标机器分别为一个四核处理器芯片和一个八核处理器芯片。其中四核芯片的 L2 Cache 被所有核共享，大小为 8MB，八核处理器芯片的 L2 Cache 为 16MB。其余配置与 3.3.4 节相同。

2. 结果及分析

由于 GEMS 工具集[Martin 2005]并未完全实现 sparc 指令集[Sun 2004]，所以本文中还在模拟器中自己实现了某些指令。同时由于 Simics 模拟器[Virtutech 2008]的限制，实验进行的时间并不长，平均每个程序运行了大约 24 小时左右。

图 3.14 给出了每千条指令片内高速缓存的失效次数。实验结果肯定了采用不包含策略对降低 Cache 的失效次数有着显著作用。由于仿真试验本身的不确定性，在此仅对试验结果进行粗略的分析。

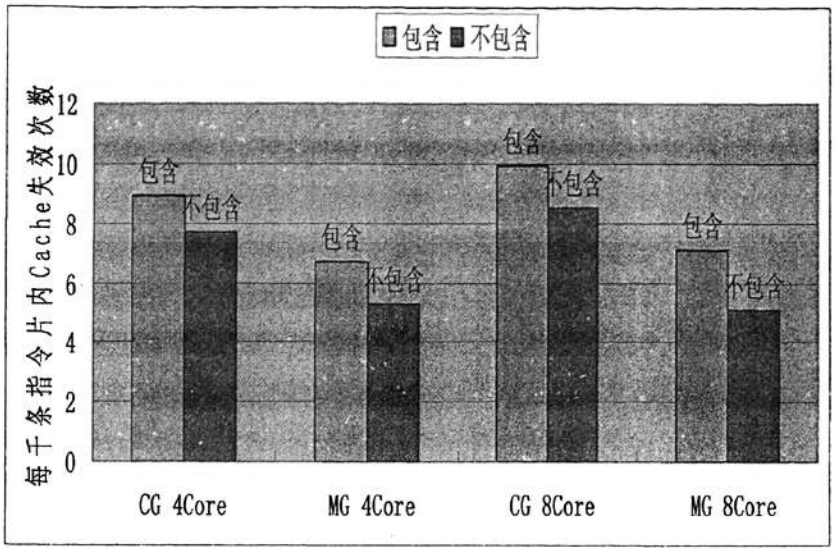


图 3.14 测试结果（每千条指令片内 Cache 失效次数）

可以看出，随着片内处理器核数量的增加，即使增加了片内高速缓存的总容量，片内 Cache 的失效次数也会明显增加。同时由于 MG 程序对存储较为敏感，采用不包含策略时，其性能提升也相对较大。

事实上，对包含性策略进行设计分析相对于仅仅降低高速缓存失效次数或提高性能更为复杂。在真包含，非真包含以及不包含等策略间进行比较分析时，还有很多其他的因素必须加以考虑。比如，片内 L1 Cache 和 L2 Cache 之间的通信带宽因素。当 L2 Cache 不能满足一致性请求时，则可能会将一致性消息转发给片内某 L1 Cache，此时 L1 Cache 和 L2 Cache 之间的通信带宽对性能的影响便会显现。除此之外，L1 Cache 的对外端口数目有时也会成为性能瓶颈。比如，L2 Cache 转发了大量的一致性消息到某 L1 Cache，而同时该 L1 Cache 所属的处理器核正试图访问 L1 Cache。增加 L1 Cache 的端口数目也许并不是一个比较好的解决方法。因为这位占用大量片上面积，增加设计验证的复杂度，且会降低 L1 Cache 的访问速度。

对不包含策略的实现是一件比较困难的工作，复杂性非常高。其原因非常多，不过最主要的是，本文的工作是建立在一个现成的基于真包含策略的模拟器之上，对现有模拟器的修改越多，其复杂性越高，可能的错误也越多，比如对 L1 Cache 写回操作的实现。同时由于包含性与高速缓存一致性协议是紧密联系在一起的，更增加了设计实现的复杂性和难度。

考虑到仿真实验的时间和正确性，上述工作未考虑 M-CMP 情况以及 L1 Cache 和 L2 Cache 容量的变化。这将是未来工作的一个重要方面。

本章的研究工作也存在某些局限性。

1. 测试程序较少

本文仅测试了两个基准测试程序。事实本文工作曾尝试运行更多的基准测试程序，但运行并不成功。

2. 运行时间短

由于软件模拟本身速度很慢，以及模拟对象规模很大。本文认为，将来基于硬件的仿真实验工具将成为主流。

3. 高速缓存一致性的设计和验证的难度

GEMS 中使用 SLICC 来为高速缓存一致性协议建模。本文利用 SLICC 修改实现高速缓存一致性协议时消耗了大量工作量。且对协议的正确性本文未加以充分验证。

由于上述及其他原因，本文的实验数据存在一定的不确定性。但是，本文仍然相信，单从性能角度而言，不包含策略能为性能的提升起作用，是一个值得研究的课题。同时，本文研究的过程中，也发现了很多在研究基于不包含策略的片上高速缓存系统时应注意的问题。

3.5 小结

大规模并行计算系统的构建，其体系结构与性能等和所使用的处理器是紧密相关的。随着多核处理器的广泛使用，基于多核处理器构建的大规模并行计算系统也已开始成为主流。在这种系统中，由于 Cache 层次的增加，以及处理器核的数量急剧增加，其高速缓存一致性协议的设计验证更加困难，对系统性能的影响也更大。在 M-CMP 环境下研究高速缓存一致性问题具有重大意义。

本章首先研究了 M-CMP 系统以及在 M-CMP 中实现高速缓存一致性的两种技术：令牌技术和目录技术。随后研究了基于不包含策略的多核处理器中的高速缓存一致性协议及其实现，相较于传统经典协议，本文的工作对协议做出了优化，带来了性能的提升。

本章接着研究了传统单核处理器中高速缓存的包含与不包含性，在此基础上引入多核处理器中的包含与不包含性问题，着重研究了不包含方式，并给出了实验和分析。实验结果指出，在多核环境下，基于不包含方式的片内高速缓存系统

虽然一定程序上增加了复杂性，但能带来较好的性能提升。

第 4 章 多核处理器片上高速缓存系统性能研究

本章摘要 多核处理器的片上高速缓存系统对处理器的性能具有重要影响。在多核处理器时代，基于多核处理器研制高性能计算机已经成为潮流。目前我国已经有若干研究机构和企业已经将研制采用我国具有完全知识产权的龙芯 3 号多核处理器的高性能计算机列入日程。为此，研究龙芯体系结构的多核处理器的片上高速缓存在科学计算等领域的应用具有重要意义。本章对龙芯体系结构的多核处理器的片上高速缓存的负载行为进行了评测和分析，并依此对其设计空间进行了一些探索。

4.1 研究背景

片上 Cache 一直是计算机系统结构技术的研究重点。在多核处理器时代，片上 Cache 的作用更加突出。对多核处理器片上 Cache 的负载行为进行评测，对于理解多核处理器的特性以及更进一步地优化处理器设计，具有重要意义。

高性能计算机是一个国家的重要战略资源，是综合国力的集中体现。高性能计算机的国产化，对于一个国家的经济建设、科技发展和国防事业等均具有重要意义。龙芯系列微处理器是我国完全具有自主知识产权的通用高性能微处理器。目前已经有基于龙芯处理器的国产高性能计算机面世，如中国科学技术大学和中国科学院计算技术研究所等研制的 KD-50-I 万亿次机。同时基于龙芯 3 号多核处理器的高性能计算机的研制工作已经被一些机构或单位（如中国科学技术大学、曙光公司，等）列入研制计划中。

在此背景下，针对科学计算等应用领域，研究龙芯体系结构的多核处理器在这些领域中的性能，并依此探索其设计空间，具有重要的学术意义和应用前景。

4.1.1 龙芯 3 号处理器

龙芯 3 号处理器是由中国科学院计算技术研究所所开发的片上多核微处理器，是一款高性能、低成本、低功耗的多核 CPU。目前的龙芯 3 号处理器片上集成了 4 个处理器核，每个核拥有私有一级指令和数据 Cache。后续将会有 8～16 核芯片面世。概括而言，龙芯具有以下特点[Hu 2008]：

1. 扩放性好

片上网络采用 mesh 网和交叉开关（crossbar）组成，每个交叉开关具有四个方向的互连，将处理器核、L2 Cache 等连接在一起。缓存一致性协议基于扩放性好的目录技术。

2. 处理器核和 L2 Cache 可重构

片上共享二级 Cache 可被配置成内置 RAM，与 DMA 直接通信。支持内存页面在 L2 Cache 和内存间的迁移。片上处理器核可异构，可集成 64 位 4 发射的通用处理器核，以及其他多用途处理器核，以用来支持 Linpack 数学计算、信号处理等。

3. 低功耗

龙芯 3 号中所使用的工艺和技术能有效降低功耗。除此之外上，龙芯 3 号中的电源管理也很具特色，如模块级门控时钟、温度传感，等。

4. X86 二进制翻译加速

龙芯 3 号处理器在硬件上支持对 X86 二进制的加速翻译，其加速比可达 10 倍之高。龙芯 3 号的指令集中加入了一些能够完成 X86 指令功能的指令，且这些新指令完全符合 MIPS 指令格式。新加入的指令共计 200 多条，但实现这些指令的硬件开销很小。

5. 主频较高

龙芯 3 号处理器基于 65nm 技术，主频大于 1GHz。

4.1.2 龙芯 3 号片上 L2 Cache

龙芯 3 号处理器的片上 L2 Cache 被片内所有处理器核所共享，物理上分布于不同的结点中。根据地址，每个结点的 L2 Cache 分为可并行访问的 4 个缓存体，每个体容量均为 512KB，采用 4 路组相联方式。片上所有结点的 L2 Cache 位于统一的地址空间内。L2 Cache 的地址和内存的地址分布一致，这样可以降低内存与 L2 Cache 之间的通路设计并降低访问 L2 Cache 失效时造成的延迟[王焕东 2008]。

龙芯 3 号通过基于目录的高速缓存一致性协议来维护片内 L1 Cache 的一致性。所有 L2 Cache 的 Cache 块均有一个固定的宿主（home）结点，相应的缓存块目录由宿主结点进行维护。所实现的一致性协议中，每个缓存块具有三种可能的

状态，分别是：无效、共享（可读）和独占（可读写）。

4.2 龙芯 3 号 L2 Cache 负载行为评测

片上高速缓存系统对于处理器的性能至关重要，尤其在多核处理器环境下，由于各个核上的应用程序访存行为模式可能不同、数据集变大、访存通路压力更大等原因，对片上缓存系统提出了更高的要求[Beckmann 2006]。

为了更好的理解龙芯 3 号多核处理器片上缓存系统的工作特征和性能，并进一步的挖掘其设计空间，有必要利用基准测试程序测试应用程序对龙芯 3 号片上 Cache 的压力，和分析龙芯 3 号片上 Cache 对程序性能的影响，最终找出设计优化空间。

4.2.1 评测方法

如前文中所述，为了更好的研究科学计算应用在龙芯 3 号上的性能等，本文选择的基准测试程序是由斯坦福大学所开发的 Splash2 包[Woo 1995]。

Splash2 是比较经典的多线程并行科学计算程序包，线程间经常需要频繁地对共享变量进行同步操作。Splash2 中的所有程序均来源于实际科学和工程应用，包括有海洋模拟、数的排序、快速傅利叶变换，等等。这些程序分成两大部分：核心程序（Kernels）和应用程序（Applications）。Splash2 程序对体系结构底层变化非常敏感，非常适合用来做体系结构方面的研究。

表 4.1 参数设置

参数	值
处理器核数目	4
处理器特征	4发射，乱序
分支预测器	GShare，4096条目大小PHT
功能单元	2个定点单元，2个浮点单元，1个内存控制器
L2 Cache MSHR大小	8
ROQ大小	64
Cache 块大小	64B
L1指令Cache	64KB，4路组相联，3周期延迟
L1数据Cache	64KB，4路组相联，3周期延迟
共享的L2 Cache	4MB，4路组相联，10周期延迟

实验平台我们使用了龙芯模拟器 Sim-Godson[张福新 2007]和 GEMS 工具集[Martin 2005]。Sim-Godson 是一款专为龙芯处理器所设计开发的模拟器，具有速度快、灵活性好和准确性高等优点。Sim-Godson 在 3.0GHz 的 Pentium4 微机上的运行速度约为 500K 条指令/s。我们利用 GEMS 工具集实现了 4.1.2 节中所述 L2 Cache，同时在 Sim-Godson 的基础上，针对多核环境对模拟器做出了一些修改，并利用 Sim-Godson 收集数据。能对龙芯 3 号处理器的片上 Cache 进行较好的模拟仿真。实验参数如表 4.1 所示。表 4.2 中给出了测试程序的参数。

表 4.2 测试程序参数

应用程序	问题规模
FFT	65536 particles
LU	512*512 matrix
LU_nc	128*128 matix
Radix	262144 keys
FMM	256 particles
Ocean	130*130 array，1e-7 error tolerance
Ocean_nc	130*130 array，1e-7 error tolerance
Cholesky	Lshp

4.2.2 负载行为评测

龙芯 3 号 4 核处理器目前片上二级高速缓存的容量为 4MB，相对许多其他处理器片上 L2 Cache 的容量较小。因此，并行程序在龙芯 3 号处理器上对 L2 Cache 的竞争更加激烈，L2 Cache 的性能对整体性能的影响也更大。

对 L2 Cache 的负载行为进行评测，较好的办法是将对 L2 Cache 块的共享行为特征刻划出来[Bradford 2006]。把对 L2 Cache 块的共享分为三种：

1. 独占访问

缓存块只被一个处理器所访问。

2. 共享只读

多个处理器共享缓存块，但对该缓存块只读，不写。

3. 共享读写

缓存块为多个处理器所访问，且这些处理器中至少有一个将写该缓存块。

显然对于龙芯 3 号处理器的 L2 Cache 而言，共享只读的访问越多，性能会越好。对于独占访问，如果数量大，则消耗的 L2 Cache 容量越大，竞争越激烈，对性能影响越大。而如果共享读写的数量大，则会引发大量的一致性开销，引入线延迟，同样对性能有不良影响。

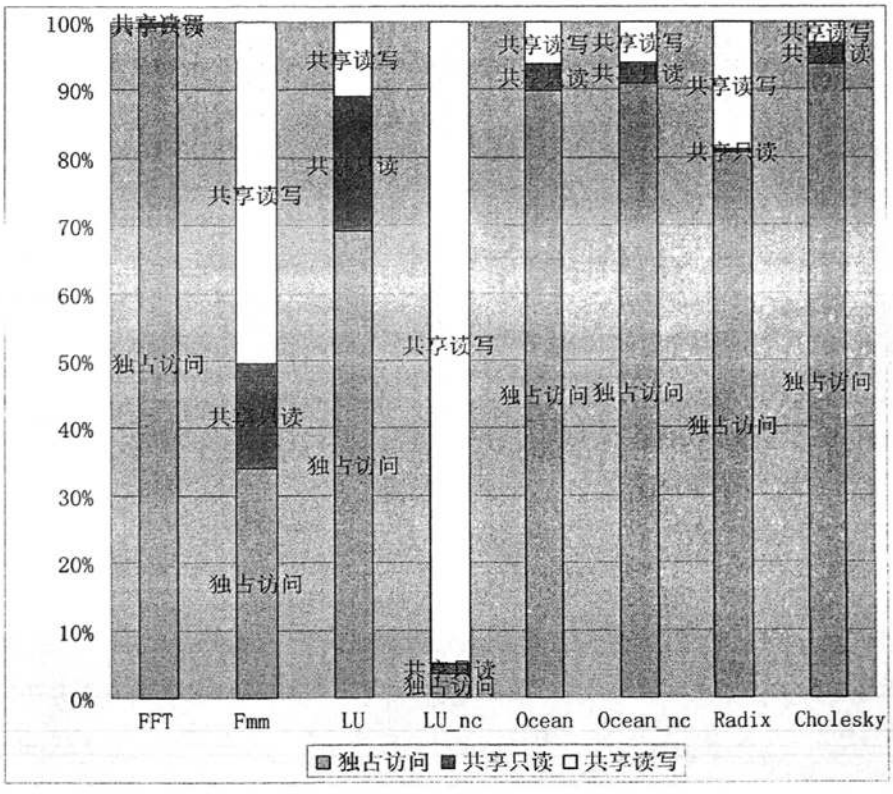
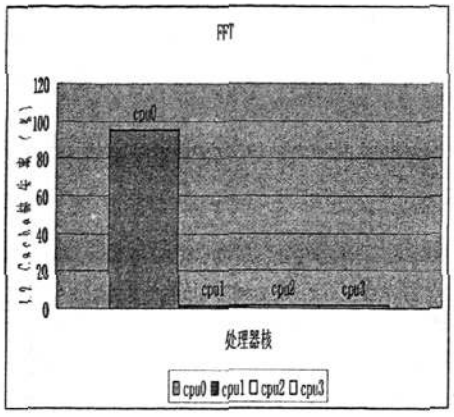


图 4.1 L2 Cache 块的共享行为剖析

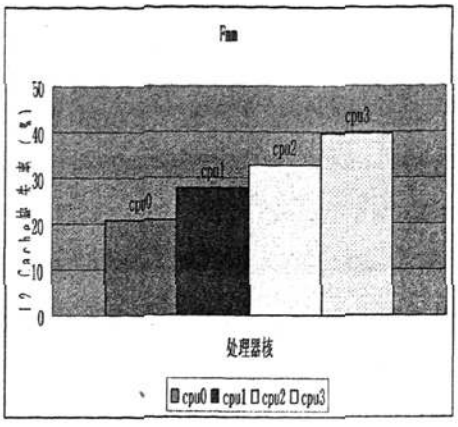
从图 4.1 可以看出，对于 Splash2 应用，独占访问所占的比例平均最大。在这种情况下，大容量的 L2 Cache 的性能将会更好。对高速缓存块的共享读写操作也占有一定比例。最节省存储空间的共享读操作所占比例最小。

图 4.2 中给出了所有程序运行时不同处理器核造成的 L2 Cache 的缺失率。从图中可以看出，总是有一个处理器核的缺失率特别高，这里原因有二，一是因为程序尚未进入并行阶段、进行初始化时将新鲜数据不停调入片内 Cache 所造成；

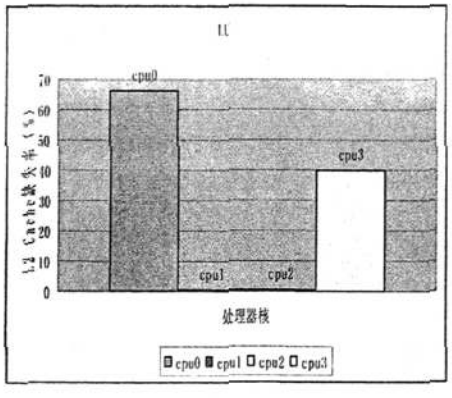
二是在程序结束时，主线程的归约处理等操作，会对 L2 Cache 造成较大压力。



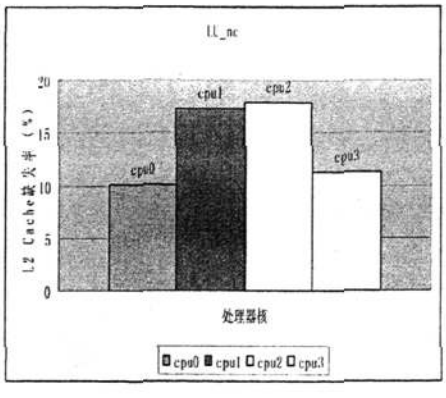
(a) FFT



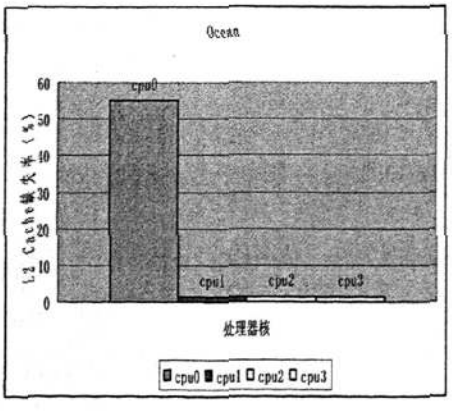
(b) FM



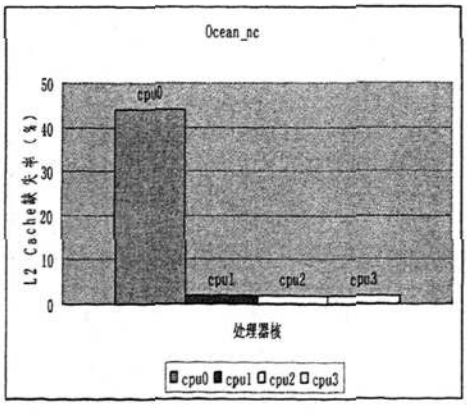
(c) LU



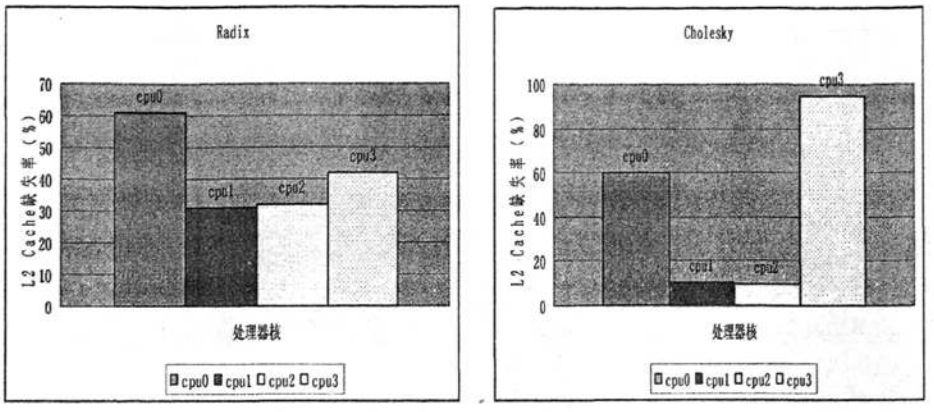
(d) LU_NC



(e) Ocean



(f) Ocean_nc



(g) Radix　　(h) Cholesky

图 4.2 L2 Cache 缺失率

对于 FFT 程序，虽然其独占访问操作数量非常多，但这些数据具有较好的时空局部性，且数据集不大，故总的来说，其 L2 Cache 缺失率不高，龙芯 3 号处理器表现出了较好的性能（图 4.2a）。对于 Fmm 程序，共享读写操作占了较大比例，这造成了 Cache 块的经常性失效，同时独占访问也占了相当比例，故使得 L2 Cache 的缺失比例较高且较平均（图 4.2b）。

LU 程序的目的是用块分解算法将一个稠密矩阵分解成一个上三角矩阵和一个下三角矩阵。其实现方式是，将待分解数组中的不连续内存块分配到连续的内存空间里，并将这些块均匀分布到各处理器/核的内存中。LU 程序中存在着大量的存储（store）指令，这些指令经常会竞争同一组中的缓存块，对于采用 LRU 替换策略的 L2 Cache 而言，会严重影响 L2 Cache 的缺失率。虽然 LU 程序中的独占访问操作数较多，但由于程序本身的数据局部性较好、复用率高，很多情况下会使访存操作在 L1 Cache 直接命中，所以可以看到一个较为奇怪的现象，即有两个处理器核的 L2 Cache 缺失率较高，而同时另两个核的缺失率相对较低（图 4.2c）。

LU_nc 程序是 LU 程序的非连续实现，即数据非连续的分布于各处理器/核的私有空间里，所以如图 4.1 所示，LU_nc 中的共享读写操作占据了绝大部分比例。另外，由于共享 L2 Cache 容量为 4M，相对我们所选择的数据集来说偏小，故所有 4 个处理器核的缺失均偏高（图 4.2d）。

对于 Ocean 和 Ocean_nc 程序，它们对于高速缓存块的访问方式均类似，独占请求居绝大多数，共享只读和共享读写占的比例很小（图 4.1）。图 4.2e 和图 4.2f 中显示，总有一个处理器核的缺失率较高，相应这是由于主线程的串行工作部分

所导致。然而幸运的是，Ocean 和 Ocean_nc 中的独占访问都具有良好的时空局部性，同时这两个程序的数据集大小恰好以 4M 为一个临界点，故总的看来，L2 Cache 的缺失率不高。

Radix 程序的工作流程可以分为三步：本地信息的建立、全局信息的建立以及数的交换排序。其中第一步和第二步的工作量相对不大，时间开销主要集中在第三步上，且第一步完成之后均会有一个利用路障(barrier)进行同步的过程[Woo 1995]。从图 4.1 中可以看出，独占访问占大多数，共享读写次之，共享只读最少。这是由于在数据集信息的建立过程中，以及数的交换排序过程中对数的访问模式以独占访问最多的原因。程序的第三步会有很多的全局通信操作，这非常有利于共享片内缓存的处理器。但在交换的过程中，对 Cache 的容量有一定要求。图 4.2g 中显示，4 个处理器核的 L2 Cache 缺失率均偏高，相信主要原因还是在于 L2 Cache 的容量偏小。

稀疏矩阵的 Cholesky 分解是一个经典的不规则应用。在不规矩应用程序中，同步操作对整体性能具有重要影响。Cholesky 程序中存在大量对内存的不规则访问，这对程序的局部性有负面影响[Shigehisa 1999]。Cholesky 中的共享对象有两种，一种是矩阵中的非零元素，另一种是任务队列。Cholesky 中的通信操作也占据较大一部分比例。图 4.1 中显示，独占访问所占比例最高，这是由于 Cholesky 中对非零元素的处理方式所造成。而从图 4.2h 中可以看出，4 个处理器核的 L2 Cache 缺失率两个较高，另两个相对较低。究其原因，一方面是独占操作对 L2 Cache 容量的压力较大，另一方面，应该是通信操作过程中，处理器核上的进程处于忙等状态，此时的 L2 Cache 缺失率几乎为 0 所造成。

4.3 龙芯 3 号 L2 Cache 设计空间探索

4.3.1 共享和私有 L2 Cache

目前的龙芯 3 号片上 L2 Cache 为所有 4 个处理器核共享。这种当前主流的片上二级高速缓存的实现方式一般是物理上分布，逻辑上共享。为了避免热点（hot-spots）的出现，Cache 行交叉往往根据地址的高位或低位分布于各 Cache 片（slice）上。龙芯 3 号 4 核处理器中采用的即是高位交叉。然而随着处理器核数和 Cache 容量的增加，远地访问开销也会加大，并影响到性能。现在已经有一些工作针对 L2 Cache 的组织方式，如私有 L2 Cache、混合式 L2 Cache 等[Zhao 2008]。

基于龙芯 3 号模型，我们成功实现了其私有 L2 Cache 结构，并在此基础上运行了 Splash2 中的若干程序。图 4.3 中给出了共享和私有 L2 Cache 结构上的性能(IPC)比较。

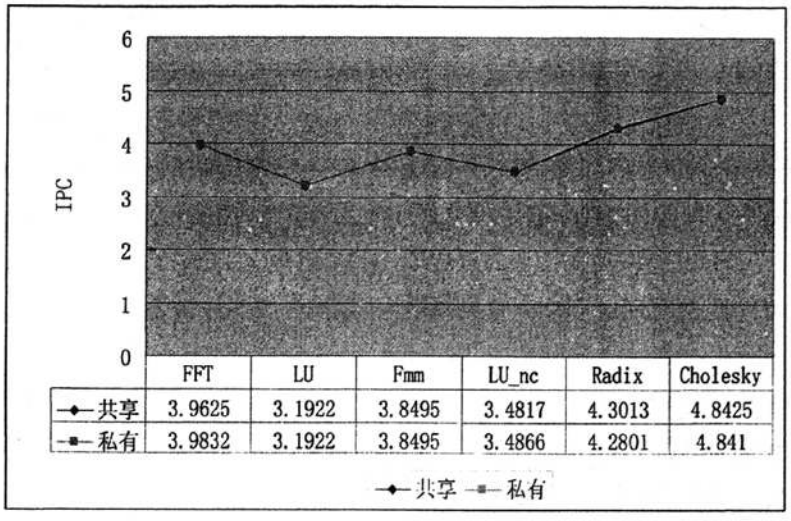


图 4.3 共享和私有 L2 Cache IPC 比较

所有程序的性能差异不大，两种组织方式下的表现各有优劣。FFT 和 LU_nc 程序在私有结构下的 IPC 稍微有所提高，Radix 和 Cholesky 程序则反之，而同时 LU 和 Fmm 程序的性能基本一样。

以 FFT 程序为例，其独占访问操作占绝大多数，私有 L2 Cache 应该是降低了其远程访问的开销，使性能有所提升。但总的来说，我们认为，这些程序在两种结构下之所以性能差别都不大，一个重要的原因还是 L2 Cache 的容量较小所致，使得 L2 Cache 的组织方式对网络开销的影响远小于由于容量所限所导致的访问失效产生的影响。

4.3.2 L2 Cache 容量

前文中已经指出，4MB L2 Cache 的容量似乎是影响 Splash2 程序性能的一个重要因素。目前的许多多核处理器的片上 L2 Cache 的容量已经可以达到数十兆乃至上百兆，如 IBM 的 Power6 处理器等。

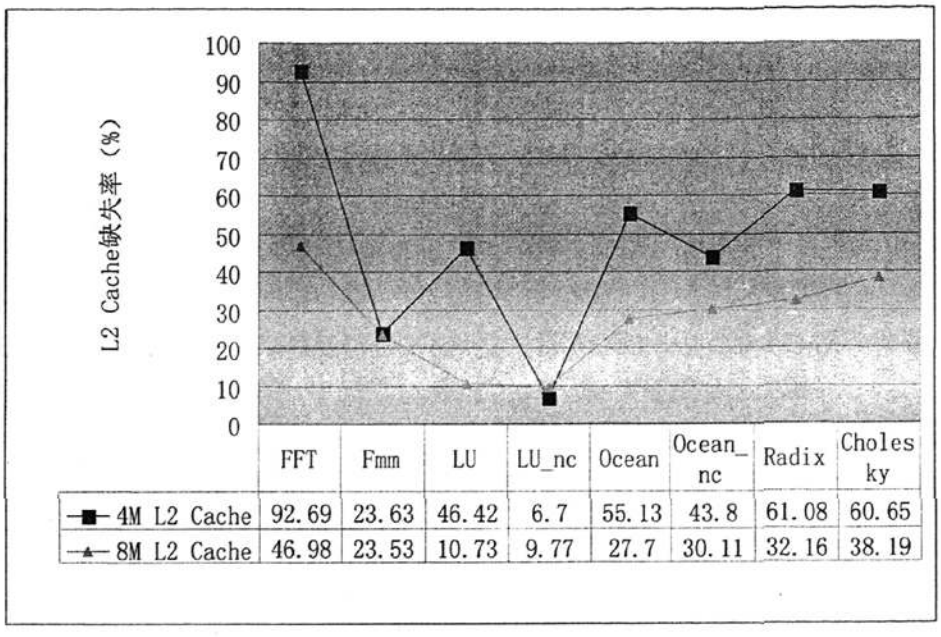
本节将 L2 Cache 的容量提升到 8MB，并在给出了其性能与 4MB 时的对比。

从图 4.4 中可以看出，在 8M 的配置下，L2 Cache 的缺失率有着明显的降低。其中 LU 程序的 L2 Cache 缺失率降低幅度可以高达 75%左右，其余如 FFT、Radix、Cholesky 的缺失率也降低了 50%左右。

LU_nc 程序的表现最奇怪，在 8M 的情况下 L2 Cache 的命中率居然有所下降。

结合图 4.1 和图 4.2 分析，我们认为这是由于共享数据分布于处理器各相邻 Cache 片中，在存在大量共享读写的情况下，对远程 Cache 片中更大量数据的频繁访问使数据的一致性受到破坏，最终使整体的命中率受到影响。

基于上述分析，总体而言，我们认为选择更先进的物理工艺，并提高片上 L2 Cache 的容量，是一个在不过多牵涉结构设计的前提下能有效提高龙芯 3 号多核微处理器性能的好办法。



	FFT	Fmm	LU	LU_nc	Ocean	Ocean_nc	Radix	Cholesky
4M L2 Cache	92.69	23.63	46.42	6.7	55.13	43.8	61.08	60.65
8M L2 Cache	46.98	23.53	10.73	9.77	27.7	30.11	32.16	38.19

图 4.4 8MB 和 4MB L2 Cache 的缺失率

4.3.3 L2 Cache 体的不平衡访问

我们发现，龙芯 3 号处理器对片上 L2 Cache 的各个体（bank）的访问较不均匀，某些体的利用达到 100%，而某些体中被使用的 Cache 块数只有数十个。经分析，其原因可能是高位交叉所致。

目前的龙芯 3 号 4 核处理器中，采用的是高位交叉寻址。即处理器地址的高位代表 Cache 体的体号，低位表示体内的地址。这造成一种情况，即其他处理器核的数据经常性的出现在 0 号处理器核的 L2 Cache 体内，造成了其他处理器核访问本地 L2 Cache 体的缺失率较高（却不影响整体 L2 Cache 的命中率），需要到 0 号处理器核处获得数据，增加了通信量，对性能具负面影响。

为了更深刻地研究此一现象，我们在 16 核处理器的平台上，剖析了 Splash2

中的一些程序的数据在 L2 Cache 的各个体上的分布情况。处理器的 L2 Cache 总容量为 16M，其他参数与表 4.1 中所示相同。

表 4.3 中给出了各程序在各个 Cache 体上所使用的 Cache 块的数目。

表 4.3 各 L2 Cache 体使用块数

程序	本地L2 Cache使用块数															
	P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15
FFT	16384	268	268	268	268	268	268	268	268	268	268	268	268	268	268	268
Fmm	6854	1197	1328	1423	1	0	0	2	0	0	0	0	0	0	0	0
LU	16384	14	14	14	14	15	14	14	14	14	15	14	14	14	14	15
LU_nc	5407	15	15	18	14	15	15	18	15	15	16	15	15	15	15	16
Ocean	16384	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29
Ocean_nc	16384	36	36	36	36	36	36	36	36	36	36	36	36	36	36	36
Radix	16384	139	139	139	139	139	139	139	139	139	139	139	139	139	139	139
Cholesky	16384	33	30	26	29	29	26	28	28	27	28	25	27	26	26	29

从表 4.3 中可以看出，L2 Cache 的各个体的利用率比较不平衡。我们认为采用原地复制策略（in-situ replication,[Chishti 2005]）应能较好地解决此一问题。其核心思想是，当数据从一级 Cache 中被替换出来时，可以将其复制到本地的 L2 Cache 体中，若数据需要再次被访问，则可以直接在本地命中（假设数据是干净的），从而减少远程数据访问操作。原地复制策略虽然会在 L2 Cache 中造成一定的数据冗余，但对照表 4.3 中的数据，相信对性能的正面影响会远大于负面影响。

4.4 小结

本章主要研究了基于龙芯体系结构的多核处理器的片上 L2 Cache 的性能。我们选择了科学计算基准测试程序包 Splash2，对这些程序在 L2 Cache 上的负载行为进行了评测，刻画了对 L2 Cache 块的访问模式，并结合程序特点对 L2 Cache 进行了分析。

随后本章在前述研究的基础上，对 L2 Cache 的设计空间进行了探索，包换共享 L2 Cache 和私有 L2 Cache 的比较、L2 Cache 的容量对性能的影响、数据分布模式的剖析等。

第 5 章 高性能计算机互连技术研究

本章摘要 在高性能计算机中，互连网络对应用程序的性能和系统的可扩放性具有至关重要的影响。目前已经有大量工作针对高性能计算机中的互连网络，其中较具代表性的有 BlueGene\L 中使用的 3-D Torus 网络等。本章研究了一种非常新颖的网络拓扑：MPU。内容包括其数学原理、路由算法等，还研究了为 MPU 开发的一个大型并行仿真工具 MPUS 的架构、工作流程等。MPUS 的主要任务是功能验证设计的正确性。仿真结果显示 MPU 的设计正确，且具有良好的可扩放性。

5.1 背景

利用高性能计算机进行计算，已经成为科学研究突破创新和工程应用领域解决实际困难的最有力且性价比最高的手段。事实上，高性能计算已经与理论研究和实验研究并列的第三大科学发展途径。

然而，人类对高性能计算的需求是永无止境的。今天的计算能力永远无法满足明日之需求，比如计算机模拟应用，其所需的计算能力是当前能提供的计算能力的成千上万倍[Keyes 2003]。然而，随着处理器时钟频率继续提高的空间有限，当前的大规模并行处理（MPP，Massively Parallel Processor）高性能计算机的性能与系统中互连网络的关系更加紧密。比如在高性能计算机历史上均有重要地位的 BlueGene\L 和 Cray RedStorm\XT3[Blumrish 2003][Scott 1996]中，均对系统互连网络的设计实现做了大量工作。

而在高性能计算互连网络的设计中，网络所采用的拓扑结构与路由算法，则是影响网络性能的关键因素，从基础上决定了大规模并行应用的效率 [Duato 2003]。在现有的许多高性能并行计算机（尤其是那些针对细粒度并行应用的计算机）中，系统的性能和应用的效率，往往受制于系统网络性能更甚于处理器性能。这使高性能计算机中的互连网络的设计显得愈加重要。

在本章中，我们研究了一种称为 MPU（Master Processing Unit）的互连结构 [邓越凡 2006]。MPU 具有高带宽、低延迟、扩放性好的特点，同时利用 MPU 可以大大提高计算密度。本章内容包括 MPU 的设计原理、路由算法、MPU 模拟器的实现，等，最后给出了 MPU 的性能分析。

5.2 MPU 互连结构

MPU 互连结构是一种 K 维的类 mesh 拓朴结构，在 MPU 中，每个处理单元（如一个处理器，或一个集成其他部件的计算结点，等）可视为被置于 K 立方（K-dimensional cube）的中心点位置，有2^k个邻居结点。处理单元与所有的邻居结点均互连在一起。换言之，一个 K 维 MPU 是由 2 个 K 维 mesh 网紧耦合而成，每个 mesh 网中的任意结点，均位于由另一 mesh 网中的相邻结点所构成的若干多维立方体的中心位置---即与所环绕的所有多维立方体的顶点直连。MPU 能在系统扩放性与结点耦合度间取得较好的平衡。

5.2.1 数学模型

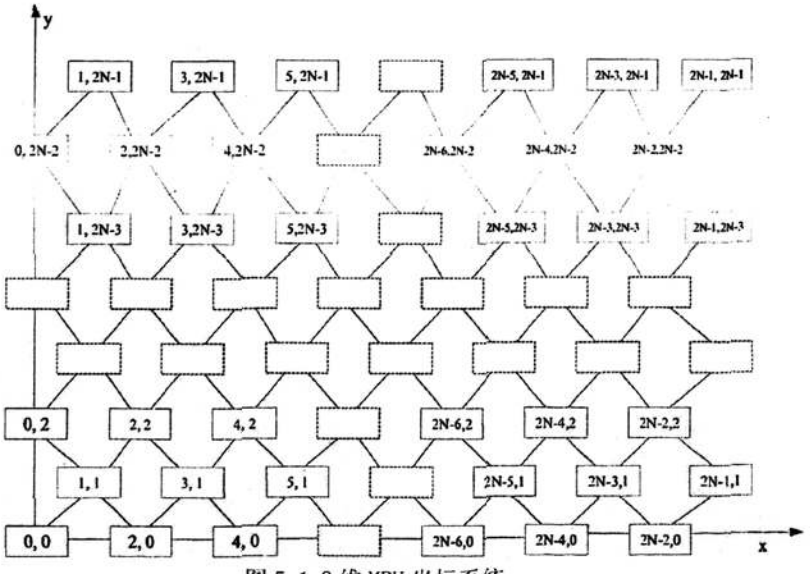


图 5.1 2 维 MPU 坐标系统

一个 MPU 的数学模型可以标记为：MPU($N_1 \times N_1 \times L \times N_k$)。其中每个坐标点($X_1, X_2, ..., X_k$)代表一个处理结点，如果其坐标满足：

$$0 \le X_i \le 2N_i - 1 \text{ 和 } X_i \bmod 2 = 0 \quad (1 \le i \le K)。$$

每个坐标点（$X_1, X_2, ..., X_k$）表示一个通信结点，如果其坐标满足：

$$0 \le X_i \le 2N_i - 1 \text{ 和 } X_i \bmod 2 = 1 \quad (1 \le i \le K)。$$

基于上述坐标系统，K 维 MPU（$N_1 \times N_1 \times L \times N_k$）系统的拓扑连接法则为：每个结点（$X_1, X_2, ..., X_k$）直接互连到其$2^k$个邻居结点（$y_1, y_2, ..., y_k$），它们的坐标满足：

$$y_i = (x_i + 1) \bmod 2N_i \text{ 或 } y_i = (x_i - 1 + 2N_i) \bmod 2N_i \quad (1 \le i \le K)。$$

因此，每个结点度（即直接互连的邻居结点个数）为 2^k，且每个结点位于 2^k 个邻居结点组成的 K 维立方体的体心。

图 5.1 给出了一个二维 MPU 的坐标系统。图中的每个奇数坐标点（x, y）均代表一个通信结点，每个偶数坐标点（x, y）代表一个处理结点。

5.2.2 设计实现

在一个现实的完备的 MPU 系统中，应包括 N 个处理结点，N 个通信结点（系统中共有 2N 个结点），以及互连架构。每个处理结点中集成有本地网络路由单元，提供与其他通信结点直接互连的功能。其中的每个处理结点均位于 2^k 个相邻通信结点所组成的 K 维立方体的体心，并同时与前述 2^k 个相邻节点互连；反之，每个通信结点，也均位于由 2^k 个相邻处理结点所组成 K 维立方体的体心，并同时与上述 2^k 个相邻结点互连。

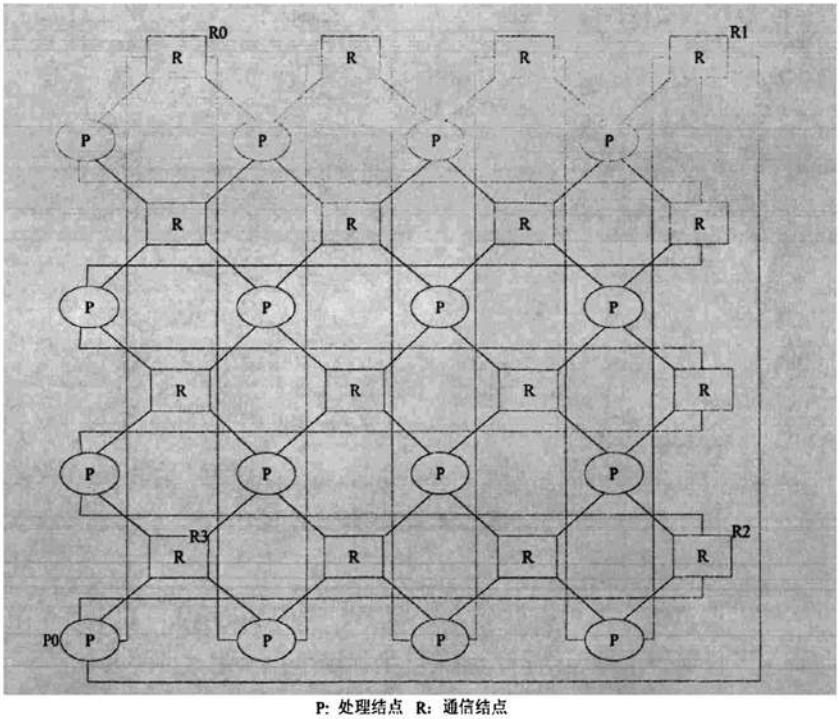


图 5.2 MPU（4x4）系统例图

一个 MPU 系统本身既可视为是一台高性能可扩展的并行计算系统，又可视为是一个高性能高耦合的超级结点，借助相关附加网络系统，可组建成一个完备的更大规模的超级计算机系统。

图 5.2 中示例了一个具体的 MPU（4x4）系统。该系统中，总共有 16 个处理

结点和 16 个通信结点，共 32 个结点。每个结点都能完成对消息的输入、输出和转发操作。对于不相邻的结点之间的通信，藉由若干跳途经若干中间结点完成。图中的每个结点均位于由相邻 4 个结点所构成的正方形的中心位置。位于图中网络拓朴系统边界的结点，则位于由相邻 4 个结点和拓扑循环映射结点所构成的虚拟正方形的中心位置，且与这 4 个相邻结点点对点连接。例如图 5.2 中的处理器 P0，即位于由相邻的结点 R3 和拓扑循环映射结点 R0、R1、R2 所构成的一个虚拟正方形的中心位置。

当将 2 维 MPU 扩展到 3 维时，每个通信结点均处于由 8 个相邻处理结点所构成的立方体的体心。与 2 维情况下类似，在网络的边界处的相应拓扑循环映射结点，与部分相关的位于边界处的处理结点共同组成了立方体，使边界通信结点同样位于某虚拟立方体的体心位置。同样，所有的处理结点也位于由相关通信结点构成的立方体的体心位置。

在 MPU 中，所有的处理结点构成了一个虚拟的网格，同样所有的通信结点也形成了一个虚拟网格。MPU 正是利用体心拓扑连接方式和拓扑循环性质，能够将这两个虚拟网格系统进行紧密的耦合嵌套。

5.2.3 路由算法

MPU 系统中的结点由两大类构成：计算结点（简称为 PU）和通信结点（简称为 SU）。计算结点的主要任务是计算、生产和消费网络消息，通信结点的主要功能是对消息进行解包分析，并根据网络状况动态地找出最佳的下一跳目的结点。

当在应用程序运行过程中，某个计算结点要与其他计算结点通信时，由于在任意两个计算结点间没有直接网络连接，故消息总需要根据源结点和目的结点的地址，经由若干个通信结点转发。

MPU 中的消息传递方式类似于虫蚀路由[陈国良 2002]，所有的消息均分片传送，这样能提高网络的利用率和性能等。路由算法是 MPU 的核心技术之一，本文中不予详细介绍。下面给出了两个简单的例子（图 5.3 所示。图中圆形表示通信结点，圆形表示计算结点）：

1. 计算结点 1 向计算结点 4 发送消息

消息将仅经过通信结点 0，由其转发。

2. 计算结点 1 向计算结点 9 发送消息

消息传递路径为：P3 -> S2 -> P6 -> S5 -> P9。

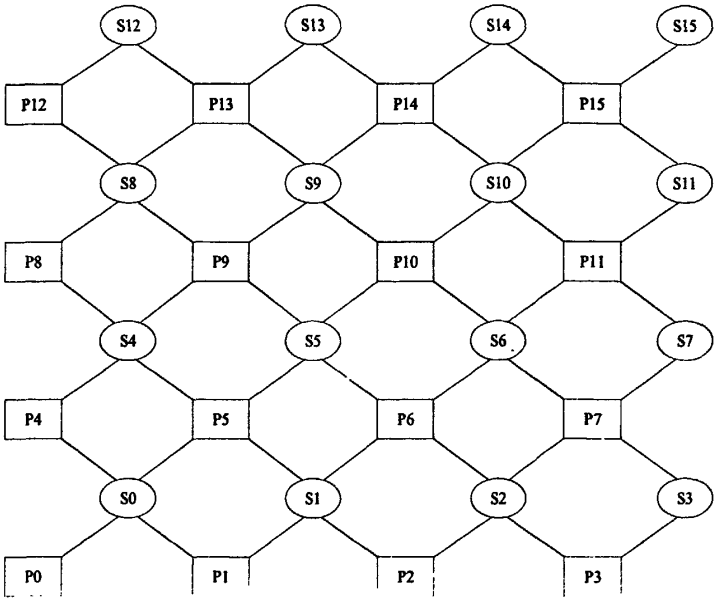


图 5.3 一个 4X4 的 MPU 系统平面图

5.2.4 对网络性能的理论分析

下面将在理论上从高性能计算机系统中对网络进行评价的一些重要参数出发，对 MPU 网络拓扑的性能进行分析。

首先简要介绍将要比较的参数的意义（具体介绍可参见[陈国良 2002]）。

1. 结点度数

指与某结点直接相连的结点的个数，即为该结点的度数。

2. 网络直径

即网络中任意两个结点间的最大距离，或跳数（hop）。一跳就是相邻结点之间的距离。

3. 结点间平均距离

指互连网络中，某结点到其他所有结点的距离的平均。单位同样是跳（hop）。

4. 等分宽度

指当将互连网络等分成两个子集（这两个集合中的结点数相同）时，所需要切断的最少连接的个数。

5. 等分带宽

即上述等分宽度的定义中，要被切断的边线上的总带宽。

6. 连线数目

指互连网络中，将各个结点互连起来所需的边线的个数。

表 5.1 与 3-D Torus 的比较

网络参数	2 维 MPU	3 维 MPU	3 维环绕拓扑（3-D Torus）	2 维 MPU 与 3-D Torus 之比	3 维 MPU 与 3-D Torus
网络维数	2	3	3	0.67	1
网络类型	间接网络	间接网络	直接网络		
处理结点数目	N^2	N^3	N^3	1/N	1
结点度	4	8	6	0.67	1.33
网络直径	N	N	3N/2	0.67	0.67
结点间平均距离	N（2 N^2 +1）/3（N^2 -1）	N^2（3 N^2 +2）/4（N^3 -1）	3 N^4 /4（N^3 -1）	4（2 N^2 +1）（ N^3 -1）/9 N^3（N^2 -1）	1+（2/3 N^2 ）
等分宽度	4N	8 N^2	2 N^2	2/N	4
等分带宽	NP	N^2 P	1/（N^2 P）	3/N	3
连线数目	4 N^2	8 N^3	3 N^3	4/3N	2.67

表 5.1 中给出了 2 维和 3 维 MPU 网络与 IBM BlueGene 中使用的 3-D Torus 网络拓扑（即 N 元 3 立方结构，相应地可扩展到 N 元 K 立方结构，等[Blumrish 2003]。）的性能对比。在表 5.1 中数据的计算中，假设每个结点总共有 P 条信号引脚用于和邻居结点进行通信（其中不包括用作其他用途的引脚，如 I/O 引脚，等）。

从表 5.1 中，可以分析出 MPU 的若干特点，或优点：

1. 网络直径固定

随着网络维数的增加，3-D Torus 网络的直径是线性增加的，而 MPU 中网络直径保持固定不变。因此，在 MPU 系统中，远程通信的距离和延迟被有效地减少了，对提高系统的可扩放性具有关键作用。

2. 等分宽度大

在互连网络的维数和规模相同的情况下，MPU 系统的等分宽度是 N 元 K 立方网络的 2^{k-1} 倍。这说明，在条件相同的情况下，MPU 系统中可以提供更多的网络连接以供结点间通信使用。网络连接冗余的增加，对系统健壮性和可用性具有重大意义。

3. 等分带宽高

在网络维数和规模相同的情况之下，MPU 系统的等分带宽是 N 元 K 立方环绕网络的 K 倍。换言之，在 MPU 系统中，结点通信所能使用的带宽更高。

4. 耦合度高

在 MPU 系统中，随着维数的增加，系统中结点的度数的增加以指数方式进行。结点度数的增加提高了系统的耦合度，但另一方面，由于物理上的限制等原因，高维数的 MPU 系统目前在实现上尚有困难。但我们相信 2 维到 3 维的 MPU 系统目前已经足够使用。

5. 通信延迟低

在 MPU 系统中，通信结点数目较多，且均匀分布于网络中，这可以有效地降低结点间的通信延迟，且可以将处理结点从网络消息处理的重荷中解放出来。

5.3 对 MPU 的并行仿真

前文中对 MPU 的设计原理进行了研究，并从数学上分析了 MPU 的性能。但现代高性能计算机体系结构的理论研究和工程实践均要求，将目标机器形成实物之前，应对设计进行仿真实验，以验证设计的正确性，并对性能进行预测。本文下面将研究针对 MPU 所开发的一个大型系统级并行仿真模拟器（以下称为 MPUS），内容包括其体系结构、与 MPICH2 的交互、工作流程等各方面。

MPUS 的主要任务是，对目标计算机的设计进行验证，仿真目标计算机的行为，并评测其性能，为目标计算机的设计与改进提供一个仿真验证平台。同时进行其他有关目标计算机系统设计的研究工作。

MPUS 模拟器在基于 MPU 技术的高性能并行计算机的研制中走到了重要作用。MPUS 上可以运行基于 MPI 标准的并行程序，同时完全实现了目标计算机的拓扑结构和路由算法。在 MPUS 的设计实现过程中和结束后，为目标机器的研发团队提交了大量技术文档和最终进行过优化的代码。通过在仿真器上运行 Linpack 程序，可以验证目标计算机的设计思想和模拟目标计算机的工作特性。该模拟器在目标计算机的初期研发阶段起到了举足轻重的作用，

5.3.1 模拟器架构

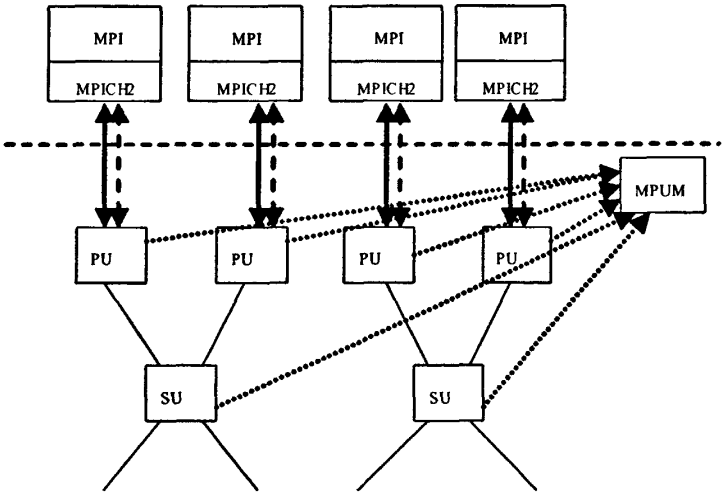


图 5.4 MPUS 架构

图 5.4 中给出了模拟器（MPUS）的架构。对图中各个模块简要介绍如下：

1. MPI Application:

基于 MPI-2 标准的上层应用。本文的工作中所选择的 MPI 实现是 MPICH2 [MPICH2 2009]。

2. MPICH2

MPICH2 原来的实现是完全基于 CP/IP 协议，数据包的路由等功能完全由 TCP/IP 控制。我们的实现必然会导致对 MPICH2 的修改。但在我们的源代码修改过程中有一个原则，即无论如何修改，所有的修改总是对上层应用是透明的。

3. PU

PU 是一个对处理结点进行仿真的进程。其主要任务是对来自上层应用的消息进行接收和发送工作，以完成进程间的通信。

4. SU

SU 是对通信结点进行仿真的进程，其主要任务是对路由器的功能仿真。SU 和 PU 共同构成了 MPUS 的硬件层。

5. MPUM

MPUM 是所有 PU 和 SU 的管理进程。MPUM 对提高系统的扩放性具有重要作用，同时也是 PU 和 SU 进行交互的主要控制者。MPUM 在系统启动阶段帮助所有的进程交换信息和建立关联。MPUM 进程的 IP 地址和端口是固定配置的。当启动过程中所有 PU 和 SU 进程均初始化完毕之后，这些进程将向 MPUM 发送消息，以登记注册本进程的相关信息。MPUM 将利用这些信息帮助构建整个网络的拓扑。

5.3.2 建立拓扑

下面以一个 4X4 的 MPU 系统为例，研究 MPUS 的建立拓扑的过程。

MPUM 进程可以在模拟平台中的任意结点上启动，一般选择服务器结点。当 MPUM 进程启动完毕之后，将进入监听状态，等待来自所有 PU 进程和 SU 进程的注册消息。

接下来将启动 PU 和 SU 进程（在 MPUS 的实现中，出于性能和易用性的考虑，PU 和 SU 进程的启动不通过特殊命令进行，而是将启动信号内置入 MPICH2 的启动命令：“mpiexec”命令）。启动完毕之后，所有 PU 和 SU 进程将发消息到 MPUM 以注册自己。注册的信息包括本进程的角色（PU 还是 SU）、网络地址（IP 和端口号），等等。MPUM 收到注册后，将为每个 SU 和 PU 进程分配一个进程标号，以及最重要的，发送相关的拓扑信息：即哪些进程彼此之间是互相直接连接的。当所有进程完成注册之后，PU 和 SU 将根据 MPUM 返回的消息进行拓扑的计算和连接的建立。至此，MPUS 基本上启动完毕。

接下来则可以使用“mpiexec”命令启动应用程序。在 MPUS 中，每个应用进程都是和某个 PU 进程绑定在一起的。这要求修改 MPICH2 中的 MPI_init()函数。修改过后的 MPI_init（）函数被调用时将会使应用进程向 MPUM 发送消息并接收由 MPUM 返回的消息(消息内容主要包括该应用进程将绑定的 PU 进程的相关信息，如进程标号、IP 地址、端口号，等等)。该 PU 进程即扮演着此 MPI 应用程序的消息通信代理人角色。MPUM 进程会维护一个记录表，表中的条目记录了每个 MPI 应用进程和相应代理 PU 进程的相关信息。

在标准 MPICH2 的实现中，MPI 应用进程间的连接采用点对点方式。但在 MPUS 中，进程间的通信方式必须做出改变：

当某 MPI 应用进程发起通信时，将首先把消息发给本进程所连接并绑定的 PU 进程，该 PU 进程根据消息中的路由信息（如目的 MPI 应用进程的进程标识，等），计算出目的 MPI 应用进程所绑定到的 PU 进程，接下来把消息转发给相连的某 SU 进程，由该 SU 启动路由计算过程，并在计算完毕后向目的 PU 进程逐步发起连接。

在目的 MPI 应用进程所绑定到的 PU 进程收到源 PU 进程发起的连接请求后，会向对应的 MPI 应用进程建立连接。

上述过程完毕后，目的 PU 进程会向源 PU 进程返回连接建立成功信号。这样两个 MPI 应用进程间即可进行通信（图 5.5)。

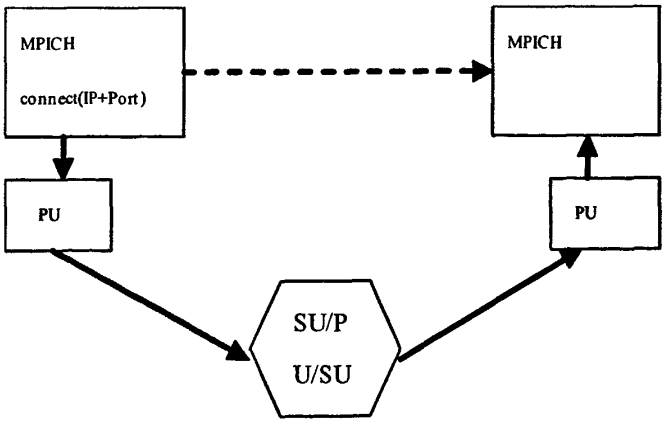


图 5.5 MPUS 中连接建立过程

5.3.3 对 MPICH2 的修改

在标准 MPICH2 对 MPI 的实现中,源应用进程在向目的应用进程发送消息前会先建立一个 TCP 连接，即在 MPICH2 中的消息传递是面向链接的[MPICH2 2009]。但在 MPUS 中，由于消息传递模式的不同（参见前文所述），故需对 MPICH2 的源代

码进行修改。修改主要集中在两部分：MPICH2 的初始化函数 MPI_Init()和 MPICH2 中的所有消息传递函数。

1. MPI_Init()函数

在 MPI_Init()函数中，需加入的功能主要有两部分。一是登记功能，即将应用进程的相关信息（包括进程标识号、IP、端口号，等）注册到 MPUM 进程；二是绑定功能，即在收到 MPUM 进程返回的消息后，找到本进程将绑定至的 PU 进程的相关信息，并绑定（图 5.6）。

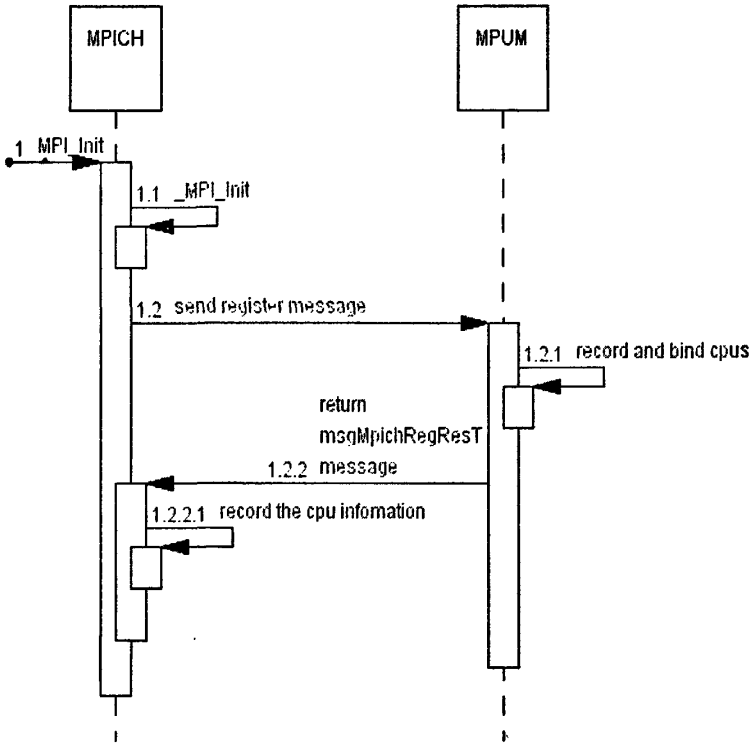


图 5.6 修改后 MPI_Init()函数工作流程

2. 消息传递函数

在消息发送函数中，当有一个应用进程消息需被发送到另一应用进程时，源应用进程会首先与绑定到的 PU 进程建立 TCP 连接（需注意的是，这样的 TCP 连接是唯一的）。接下来源应用进程的消息都将经由此连接直接发送到相应 PU 进程，并由其转发。

转发的消息最终会到达应用进程所绑定到的 PU 进程，该 PU 进程也会与相应

应用进程建立 TCP 连接，并将消息转发给应用进程。

对 MPICH2 实现的所有修改对上层应用均透明。

5.3.4 仿真结果

本文的实验工作在一台拥有 32 个主频为 1GHz 的 PA-RISC8800 CPU 的刀片服务器上进行。该服务器的网络拓扑为星形结构，路由结点位于中央位置。由于服务器中路由结点性能较高，且路由操作最多只需进行一次，故星形网络拓扑对性能影响很小。

服务器中每个 CPU 均拥有 2G 内存，运行的操作系统为 HP-UNIX 11.11。服务器使用以太网进行结点互连，平均网络带宽高于 1Gbps，延迟小于 50us。

所使用的基准测试程序为 NPB。NPB 中共有五个核心程序，基于条件所限，我们只测试了 IS 和 EP 两个程序在 MPUS 上运行情况[Yuan 2005]。为测试系统的可扩放性，选择 Mflops/s/processor 作为衡量单位，同时分别使用了 8、16 和 32 个 CPU 运行程序。

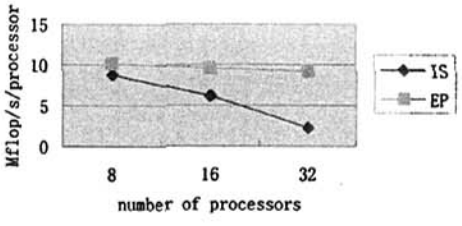


图 5.7 测试结果

在 IS 程序中，对归约操作的调用十分频繁，故 IS 对通信延迟非常敏感。EP 程序中的通信操作相关较少，故使得 Mflop/s/processor 值对处理器个数较不敏感（图 5.7）。NPB 中所有程序均能在 MPUS 上运行结束并得出正确结果。充分考虑程序的运行时间和负载压力，可以证明 MPUS 模拟器工作正常，且未发现 MPU 的设计缺陷；同时，测试数据也指出 MPUS 具有较好的扩放性。

5.4 小结

本章研究了一种新型的高性能计算机专用网络拓扑结构 MPU，包括其数学模型、技术实现、路由算法等。从理论上分析了 MPU 的性能，并就其性能与目前先进的 3-D Torus 网络进行了数学上的对比分析。本章还介绍了为 MPU 开发的大型

并行模拟器 MPUS 的情况，包括其架构、实现、工作流程等，最后给出了仿真结果。

第 6 章 KD-50-I 万亿次计算机系统结构技术

本章摘要 人类对高性能计算能力的需求是永无止境的，世界上各个先进国家都在大力发展高性能计算技术。高性能计算机在国家的经济建设、国防建设、科学技术研究等各领域均发挥着关键作用，而高性能计算机的研制水平更是一个国家综合国力的体现。KD-50-I 是首台全国产的万亿次计算机，从机器最核心的处理器到外围的交换机，机架等均实现了国产化。KD-50-I 在我国高性能计算的发展史上有着其一定的地位。本章首先详细研究了 KD-50-I 的系统结构技术，以及无盘启动技术、通信库优化等，并给出了 KD-50-I 万亿次计算机的一些性能测试数据。本章之后还研究了实际物理学研究中用到的扫描电镜成像程序在 KD-50-I 上应用及优化情况。

6.1 KD-50-I 万亿次计算机介绍

6.1.1 研究背景

高性能计算广泛应用于航天、国防、能源、物探、生物、气象、抗灾等关系国计民生的重要领域，是衡量一个国家高科技水平的试金石。高性能计算机的研制能力是实现高性能计算的基石，是一个大国实力的新象征。当前国际上最高水平的高性能计算机主要是美国、欧洲以及日本研制的，其计算能力为百万亿次量级，目前正在研制千万亿次超级计算机。

国内高性能计算机的主要研制单位有中科院计算所、国防科大、联想研究院等，高性能计算机的主要产业代表有曙光、联想、浪潮等。目前，国内高性能计算机的计算能力为十万亿次量级，与国际先进水平有相当的距离。其中中科院计算所与曙光公司研制的曙光 4000A 超级计算机在 2004 跻身 TOP500 前十名，代表着国内目前的最高水平。

高性能计算机作为国家的战略资源，其自主性格外重要。目前国内高性能计算机的核心部件—CPU 还是国外进口，这对国家的战略安全是一个重大的威胁。因此，研制基于国产高性能处理器的高性能计算机，是打破国外垄断、提升我国高性能计算机研制水平的重要战略。

“十五”期间，国产高性能通用处理器的研制取得重大突破。其中，中国科学院计算技术研究所研制的 64 位龙芯 2F 处理器代表着国内高性能通用处理器设

计的最高水平。龙芯 2F 处理器采用四发射超标量的 RISC 结构，兼容 MIPS III 指令集，包含 2 个定点部件、2 个浮点部件和 1 个访存部件，支持寄存器重命名、动态调度、转移猜测等乱序执行技术，支持全流水浮点乘加指令和 SIMD 短向量指令，支持 40 位的虚地址和物理地址访问，片内集成 512K 二级缓存、DDR2 内存控制器和 PCI-X/PCI 控制器。龙芯 2F 采用 90nm 工艺设计制造，芯片面积 42mm^2，主频达 1GHz，支持 DDR2 667，SPEC CPU2000 性能基准测试程序分值超过 500，而功耗不到 7 瓦。采用龙芯 2F 处理器设计高性能计算机，不仅在性能上是可行的，而且其低功耗小型化的特点对研制新一代高效能计算机系统具有很强的优势。

KD-50-I 是基于国产高性能通用处理器龙芯 2F 的万亿次高性能计算机系统，率先实践个人高性能计算机的理念。该系统具有低功耗、低占地面积、高计算密度三大特点，解决了板上机群、高密度计算节点、轻量操作系统等关键技术，对未来研制国产千万亿次计算机系统及提高其自主创新性具有很强的示范作用。

6.1.2 技术路线

KD-50-I 的一大目标是研制个人用得起的高性能计算机，因此低成本、低功耗是本项目的基本技术路线。

1. 低成本

目前，高性能计算机的主要成本来自处理器，通常占整个计算机造价的 80%。本项目通过采用低成本国产的龙芯 2F 处理器，使得处理器的成本不再是计算机的主要成本。

KD-50-I 实现低成本的主要手段有：通过龙芯处理器的低功耗、低面积和高集成度的特征设计高密度的计算节点，直接降低硬件板卡、散热系统的成本以及机柜的体积和占地面积；通过板上机群的设计来减少系统交换设备的购置；通过研制轻量级操作系统来减少处理单元的复杂度，直接降低节点的设计成本，等等。

2. 低功耗

现有高性能计算机的高能耗使得维护成本远高于设计成本，而且系统的稳定性也受到影响。高性能计算机的能耗主要来自处理器。KD-50-I 采用低功耗的龙芯处理器，能够将处理器的功耗降低一个数量级，并大幅度削减散热系统的能耗。

KD-50-I 的节点结构采用平面设计，有利于提高散热效率，从而进一步降低散热系统的能耗。

6.1.3 体系结构简介

KD-50-I 的体系结构如图 6.1 所示。KD-50-I 采用单个龙芯 2F 处理器芯片的处理单元（简称 PE，也即计算单元）；单个交换底板上集成 6 个 PE，采用两个 4 端口千兆以太网节点交换机（也即交换芯片）连接；一个计算节点（1U）包含两块交换底板，共享电源和散热风扇。整个系统包含 28 个计算节点，总共 336 个 750MHz 龙芯 2F 处理器，通过机柜交换机互联，与系统中的其它处理单元以及主控服务器通信，可达到万亿次的计算能力。并且可满足功耗和成本的设计要求。

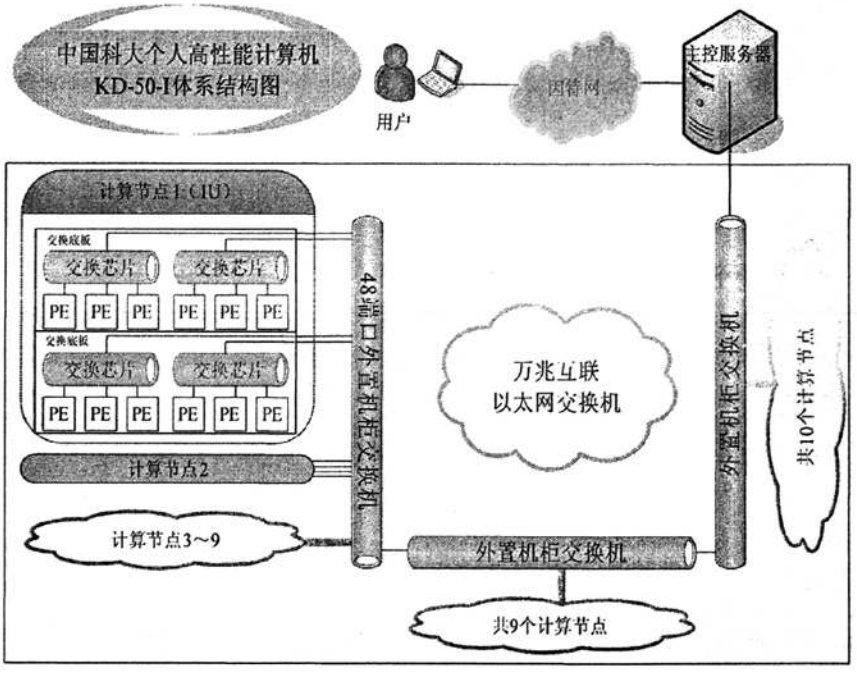


图 6.1 KD-50-I 计算机体系结构

6.2 KD-50-I 万亿次计算机系统结构

6.2.1 总体结构

KD-50-I 主要结构特点如下：

1. 采用网络集中存储，取消计算节点本地硬盘，处理单元部件化。

在 KD-50-I 系统中，除服务节点外，其它处理单元的本地硬盘可以取消，外移到服务节点，并通过网络进行访问。硬盘分作系统盘和数据盘。KD-50-I 的服

务节点同时也是存储节点，作为整个系统的虚拟系统盘，通过网络引导实现系统中其它所有节点操作系统的远程加载。服务节点对外提供网络存储设备接口，共同构成系统的存储空间，它们可作为本地数据盘建立各处理单元的私有数据空间，也可作为共享存储设备建立系统全局数据空间。这样做的优势是明显的：由于存储集中，大大方便了系统管理；全局虚拟系统盘的建立简化了节点操作系统的管理，便于实现系统的动态部署；处理单元本地硬盘的外移（取消）便于提高计算节点的密度和可靠性。

分析表明，在现有大规模的机群系统中，节点机中的本地硬盘是影响系统可靠性的一个重要因素。

在 KD-50-I 系统中，计算节点成为系统的计算部件，存储节点成为系统的存储部件，所有节点可以不再是一个功能通用的完整的计算机。

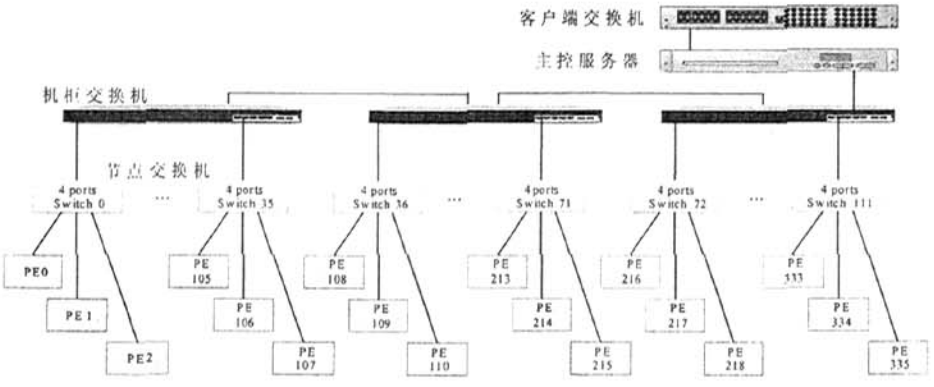


图 6.2 KD-50-I 系统整体结构

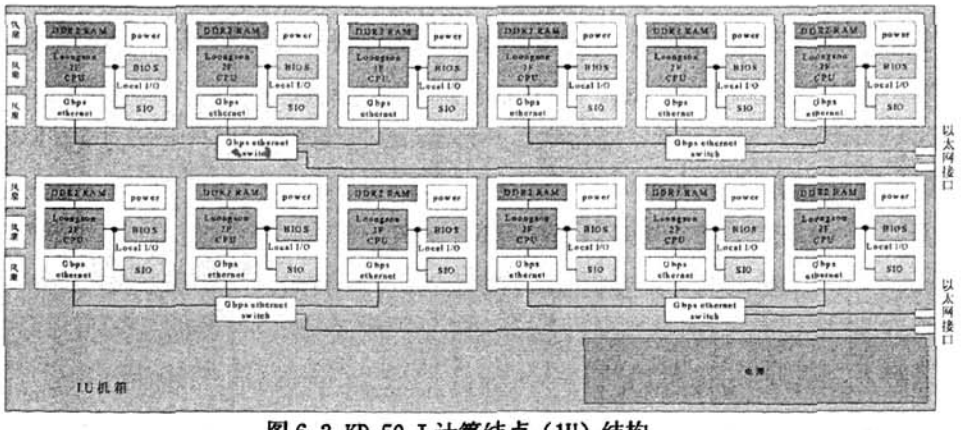


图 6.3 KD-50-I 计算结点（1U）结构

2. 研制专用的处理单元操作系统，支持提高整个系统的应用饱和性能。

传统的机群系统中，各节点都运行通用的操作系统。在 KD-50-I 系统中，为提高应用的饱和性能，各处理单元可根据自身实现功能的需求和应用的特点动态部署专用的处理单元操作系统。在 KD-50-I 中，为提高科学计算应用的性能，为处理单元研制了一种轻核心的 Linux 操作系统，该操作系统根据科学计算的特点，对通用操作系统进行裁减、改造。

KD-50-I 系统的总体结构如图 6.2 所示。处理单元之间通过节点交换机和机柜交换机互联。每个计算节点通过 4 个千兆网络接口去连接机柜交换机。机柜交换机之间采用专用的万兆堆叠接口实现高带宽连接。主控服务器为系统提供磁盘存储、系统引导、用户等录、任务调度等功能。

6.2.2 计算结点

整个系统以 1U 节点作为系统的基本部件，在一个节点内放置 12 个龙芯 2F 处理单元，每个处理单元具有 1GB 内存，处理单元内置千兆以太网卡。

在节点内设置 4 个 4 端口千兆以太网交换机，每三个处理单元连接一个交换机，输出一个千兆以太网口。

结点内的 12 个处理单元、4 个交换机共用一套电源和散热系统。节点的机箱内布局如图 6.3 所示。

6.2.3 处理单元

每个处理单元包括一个龙芯 2F CPU，1GBDDR2 内存，RTL8110 千兆以太网卡，BIOS Flash，串口芯片，实时时钟芯片以及各种电源变换电路等。处理单元如图 6.4 所示。

CPU 工作在 750MHz 时钟频率下，单 CPU 的计算能力是 3Gflops。

PCI 接口为 32bit@66MHz，接口带宽为 264MB/s。

DDR2 内存的总线带宽可达到 667MB/s。

SIO（串行接口）为处理单元调试接口。RTC 为系统提供硬件时钟，并提供存储配置信息的 SRAM。

电源模块为单元上各器件提供适合的供电电压。

处理单元通过双排针连接器与节点内的底板连接。

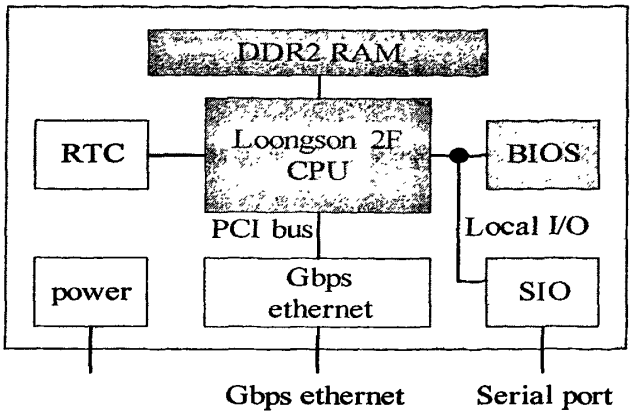


图 6.4 采用龙芯 2F 处理器的处理单元（PE）结构

6.2.4 交换底板

交换底板一方面实现处理单元之间的网络互联，另外也为各个处理单元提供电源接口。在一个底板上设置两个千兆以太网交换芯片，每个交换芯片连接 3 个处理单元，对外输出 1 个千兆网络接口。交换底板的逻辑结构如图 6.5 所示。

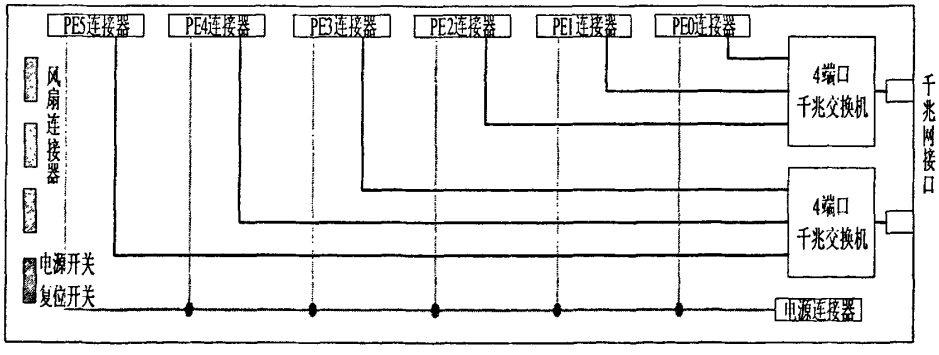


图 6.5 计算节点内交换底板逻辑结构

6.3 KD-50-I 万亿次计算机网络启动技术

KD-50-I 的处理单元主要配置为：750MHz 龙芯 2F 处理器，1GB DDR2 内存，RTL8110 千兆以太网芯片，无显卡、键盘、鼠标和硬盘等。

由于所有处理单元为无盘的，所以操作系统必须通过网络来引导至内存中运行。这要求在 KD-50-I 处理单元的 BIOS 程序（PMON [PMON 2007]，图 6.6 中给出该程序的源代码结构。）中实现网络引导功能。具体到 PMON 程序，本文的研究工

作就是为之开发 RTL8110 网卡驱动程序，并利用 TFTP 传输协议使其在相关初始化等工作完成后自动下载并启动操作系统。

龙芯 1 号和 2 号使用的 BIOS 是在 PMON2000[PMON 2007]的基础上改进的，添加了对硬盘的支持、EXT2 文件系统的支持和显卡的支持，修复了 Debug 功能，扩展性也得到提高，比较容易移植到新的系统。

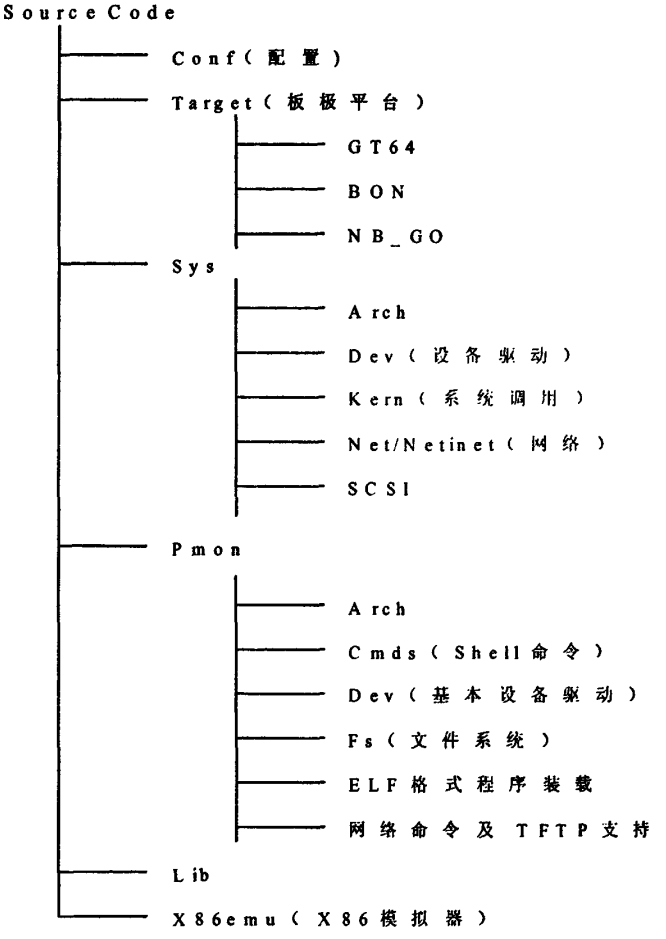


图 6.6 PMON 源程序的目录结构

PMON 源程序的目录结构如图 6.6 所示。RTL8110 的驱动程序文件位于 sys/dev/pci 目录下。

6.3.1 RTL8169 芯片

RTL8169 网卡芯片[Realtek 2007]是台湾 realtek 公司生产的一款千兆以太网卡，具有以下特点：集成的 10/100/1000M 传输器；工作模式自适应（auto-negotiation）能力；支持 PCI2.3 版本协议，可以工作在 32 位，33/66MHz

模式下；支持全双工流控(IEEE 802.3x)；与 IEEE 802.3，IEEE 802.3u，IEEE 802.3ab 完全兼容；支持收发缓冲（FIFO)；支持 PCI 消息触发中断（PCI Message Singled Interrupt）等等。

6.3.2 RTL8169 收发原理

RTL8169 支持一种新的基于描述符（descriptor）的收发缓冲管理策略对在内存中开辟环形缓冲区进行管理，该策略可以有效减少对 CPU 资源的要求。环形缓冲区的组织和工作方式与[与 SONG 2007]中的描述类似。RTL8169 最多可以分别支持 1024 个连续的发送/接收描述符队列，即意味着可以有三个描述符队列，分别为：高优先级发送描述符队列，普通优先级发送描述符队列以及接收描述符队列。在我们的驱动程序中，考虑到实际情况，未实现高优先级发送描述符队列。每个描述符占用四个连续的双字（double word）大小内存。下面分别对发送和接收描述符中的主要内容进行研究。

1. 发送描述符

TX_BUFFER_ADDRESS_HIGH 寄存器为 32-bit 大小。存放该描述符所指向的发送缓冲的 32 位高地址。

TX_BUFFER_ADDRESS_LOW 寄存器为 32-bit 大小。存放该描述符所指向的发送缓冲的 32 位低地址。

OWN 位为 1-bit 大小。当该位为 1 时，表示该描述符所指向的发送缓冲中有数据，网卡可以进行发送。当该位为 0 时，表示上层协议可以调用驱动程序向相应缓冲中写入数据；写完数据后驱动程序会将该位置成 1。当网卡将相应缓冲中数据发送完毕后，会自动将该位清 0。初始状态时该位为 0。

EOR 位为 1-bit 大小。当该位为 1 时，表示该描述符是发送描述符队列中的最后一个。每个描述符队列中有且只有一个描述符中的 EOR 位被置 1。

FS 位为 1-bit 大小。RTL8169 支持对包的分片。FS 位指示该描述符所对应缓冲区中的数据是否为某个包的第一个分片。是为 1，否为 0。

LS 位为 1-bit 大小。LS 位指示该描述符所对应缓冲区中的数据是否为某个包的最后一个分片。是为 1，否为 0。

Frame_Length 寄存器为 16-bit 大小。描述该描述符所对应的发送缓冲中的数据帧的大小。

2. 接收描述符

RX_BUFFER_ADDRESS_HIGH 寄存器为 32-bit 大小。存放该描述符所指向的接

收缓冲的 32 位高地址。

RX_BUFFER_ADDRESS_LOW 寄存器为 32-bit 大小。存放该描述符所指向的接收缓冲的 32 位低地址。

OWN 位为 1-bit 位大小。当该位为 1 时，表示该描述符所指向的接收缓冲归网卡所有，即网卡尚未向缓冲中填充数据。当该位为 0 时，表示该描述符所指向的接收缓冲中已经存有数据，可以由驱动程序传递到上层协议栈。初始时 OWN 位由驱动程序初始化为 1；当 RTL8169 向缓冲中填充数据之后会自动将其置为 0。

EOR 位为同发送描述符中 EOR 位。

Buffer_Size 为 14-bit 大小。指示该描述符对应的接收缓冲区的大小。该值应小于 8KB。

FS 位同发送描述符中 FS 位。LS 位同发送描述符中 LS 位。

Frame_Length 寄存器为 14-bit 大小。当 OWN 为 0 且 LS 为 1 时，该域表示收到的包的大小（包括 CRC 校验位）。

当对 RTL8169 进行初始化时，会在内存中分别建立普通优先级发送描述符队列和接收描述队列，并为每个描述符分配一段缓冲；描述符中的各标识位的初始化工作也将进行。在 RTL8169 的初始化过程中还有一项重要工作，即将每个描述符队列的起始（物理）地址写入到 RTL8169 的相应寄存器中。

当上层协议栈有数据要发送时，会调用驱动程序中的发送函数。该函数会在发送描述符队列中找到 OWN 位为 0，即指向的缓冲区空闲的发送描述符，将数据填入其所指示的发送缓冲。然后会将 OWN 位置 1，以及对描述符中的其他标识位进行适当设置。RTL8169 会轮询发送描述符队列，把 OWN 位为 1 的描述符所指向的缓冲中的数据从内存中取走，并将 OWN 位置为 0 和对描述符中的其他相关标识位进行适当设置。

当 RTL8169 从网络上接收到数据后，会从接收描述符队列找出 OWN 位为 1 的描述符，随后将数据填入该描述符对指向的接收缓冲，并将 OWN 位置为 0 和对描述符中其他相关标识位进行适当设置。在完成上述一系列动作之后，RTL8169 会触发一个中断，随后即由驱动程序中的接收函数将数据从接收缓冲中取出并上传到上层协议栈。接收函数还会对相关接收描述符中的标识位进行适当设置。本文在 PMON 中的 RTL8169 驱动程序实现工作中并未实现 NAPI 机制。

6.3.3 PMON 中 RTL8169 驱动程序源代码研究

除了最重要的数据收发功能，本文的工作还在本文所开发驱动程序中为用户提供了命令行接口，可以对网卡的工作模式（比如千兆全双工模式，等等）进行

设置等工作。限于篇幅，本文对这些工作不做介绍。

当 PMON 程序启动时，会对 PCI 设备进行轮询。RTL8169 驱动的入口点为函数是 r8169_attach( )；

该函数主要工作有：

1. 调用 RTL8169_init_board( ) 函数。这个函数的主要工作是为 RTL8169 网卡分配内存空间等。

2. 设置描述 RTL8169 的结构(struct rtl8169_private)中的各个函数指针，如工作模式的设定函数、状态检测的函数等。

3. 设置 MAC 地址寄存器。当 RTL8169 附有一个 EEPROM 时，默认的是从 EEPROM 中读取 MAC 地址。写 MAC 地址，则不论是否已从 EEPROM 中读取过 MAC 地址，MAC 地址都将以此时所设置的值为准。

4. 调用 rtl8169_hw_phy_config( ) 函数，对物理寄存器 (PHY Registers) 进行设置。

5. 调用 if_attach( ) 函数和 ether_ifattach( ) 函数将 RTL8169 插入到设备列表中。

6. 调用 rtl8169_open( ) 函数，打开 RTL8169。该函数主要任务是：在内存中开辟接收/发送描述符队列并调用 rtl8169_init_ring( ) 函数对描述符队列进行初始化；调用 rtl8169_hw_start( ) 函数，主要是对 RTL8169 中的寄存器进行设置。

7. 调用 pci_intr_map( )和 pci_intr_establish( )两个函数进行中断映射及将 RTL8169 中断加入到中断表中。

下面研究发送和接收函数的分析。

1. 发送函数：rtl8169_start_xmit( )函数。其主要工作有：会首先判断发送缓冲是否还有空间。若无则返回，有则继续。调用 rtl8169_xmit_frags( ) 函数，该函数会将上层协议栈传来的数据包从队列中取出，拷贝到发送缓冲中，并

释放原来的数据包所占用的内存空间。完成数据的复制后，该函数还会对描述符中的 OWN 等标志位进行设置。

完成上述工作后，即写 RTL8169 寄存器，通知其有包等待发送。

2. 接收函数：rtl8169_rx_interrupt()。当有数据被网卡写入到接收缓冲时，会触发一个中断，并调用此函数。其主要工作有：检测描述符中的 OWN 位，以判断是该描述符所指向的缓冲中是否有数据。否则返回，是则继续。判断包是否分片。在本文的工作中，无论收发，均不支持包的分片。为保证正确性，我们所分配的收发缓冲区大小均大于 1536 个字节，即以太网上所能传输的最大的包的大小。在此处，是则出错返回，否则继续。进行校验和检查。调用 getmbuf() 函数，为接收缓冲的数据在内存中另外分配一块空间，随后从接收缓冲中将数据复制到此空间中。调用 ether_input( ) 函数，将数据传输到上层协议栈。

6.3.4 网络启动

当 PMON 启动完成之后，会处在命令行交互方式状态，等待用户输入。我们跟踪 PMON 的执行流程，发现在源程序中，此时停留在/pmon/common/main.c 函数中的一个无限循环处。于是我们在 main()函数里，调用 PMON 原有的 do_cmd()函数，分别完成为网卡设定 IP 地址，从服务器上下载内核，启动操作系统等工作。代码示例如下：

1. 设定 IP：

```
netboot_line = "ifaddr rtk0 10.2.5.68\0";
do_cmd(netboot_line);
```

2. 下载内核：

```
netboot_line = "load tftp://10.0.0.1/vmlinux.32\0";
   //假设服务器 IP 地址为 10.0.0.1
do_cmd(netboot_line);
```

3. 启动内核：

```
netboot_line = "g rdinit=/linuxrc console=ttyS0,115200
ip=10.2.5.68\0";
do_cmd(netboot_line);
```

此处引号内的内容是传递给内核的参数，比如、ramdisk 文件，输出设备，波特率，IP 地址等。

完成了以上三大步，则 PMON 将系统交给操作系统，进入 Linux，并可以进行自动挂载 NFS 工作等等。考虑到计算单元不必要也不可能使用完整的 Linux 操作系统，我们对使用的 Linux 操作系统进行了适当裁剪[7]，比如去除了不必要的驱动程序等等。

6.4 文件系统及系统软件

KD-50-I 高性能计算机的系统软件以开源软件为主，具有兼容性强、易维护、易升级、易使用等特点。

KD-50-I 的处理单元操作系统为 Debian GNU/Linux 无盘系统，采用稳定高效的 2.6.18 内核（图 6.7）。

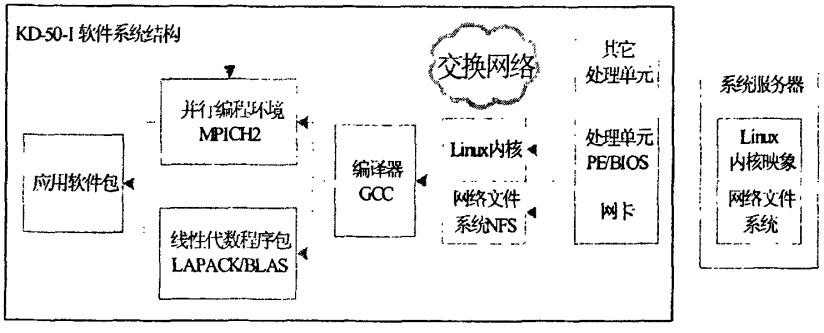


图 6.7 KD-50-I 软件系统结构

6.4.1 文件系统

在最初的设计中，采用的是利用 busybox[BusyBox 2007]制作 ramdisk 的方式，不仅将操作系统内核，还有文件系统也全部放置在计算单元本地的物理内存中。但由于在 KD-50-I 中，每个计算单元只有 1G 大小的物理内存，以致 ramdisk 方式下内存吃紧，甚至出现因为内存不够而导致进程中途被杀掉的情况。另外这种方式会导致网络启动时间大大增加，并且对系统中服务结点的压力也大增。

在我们最终的实现中，采用的是利用 NFS 将所有结点的文件系统分别挂载到系统中的服务器上。这样可以缓解内存的压力，同时也使得文件的共享和拷贝变得容易。但这种方式也导致了一个较不理想的后果，即对系统的网络压力加大。

这种方式的实现方式如下：

BIOS 对系统进行初始化后，自动从服务器上下载内核并启动。当修改过的操作系统启动完成后，会自动挂载 NFS。由于此时网络数据量较大，会出现竞争的情况，所以各计算单元在进行挂载动作前会有一个不同的时延。同时，挂载动作是重复执行的，直到挂载成功为止。

6.4.2 精简操作系统

在传统的机群系统中，所有节点都采用通用的操作系统，这样针对特定类型的应用，实际性能可能会不高。为此，KD-50-I 系统研制经过优化和改进的节点操作系统，针对科学计算类应用在内存分配、进程调度等方面进行改进，同时对不必要的功能进行裁减，提高 LINPACK 等开放源代码应用的实际性能。

KD-50-I 上研制的精简操作系统主要工作包括：

1. 改进内存管理

在尽量保证应用运算中所需的大块的内存在物理内存中也连续的前提下，底层通信软件协议得到了简化，通信性能有改善。

2. 裁剪操作系统

为了减少操作系统的各种资源开销，将操作系统中跟高性能计算无关的模块都去除掉[Gu 2003]。通过裁减，将通常情况下 1.4M 到 1.6M 的核心精简到 780K 左右。

3. 核心线程去除

Linux 作为一个通用操作系统，越来越倾向于使用更多的核心线程来完成各种管理工作，但是线程越多就越易于影响那些同步操作比较频繁的应用，形成系统“噪音”。为此，KD-50-I 精简操作系统中修改了 Linux 核心代码，将各个线程的工作融入到核心的其它部分，以最大限度的降低核心线程“噪音”。通过修改，最终生成的核心只有一个核心线程 keventd。

此外，还在调度策略、页面调整和系统配置优化等方面进行了尝试。

6.4.3 编译/并行运行环境

KD-50-I 使用 GCC 4.2.编译器，支持 C、C++、FORTRAN77/90/95 等编程语言。同时还提供支持龙芯 2F CPU 乘加指令的 gcc 3.4.6 编译器。

KD-50-I 提供 MPICH2 并行环境，支持 C、C++、FORTRAN77/90/95 的 MPI 并行。

KD-50-I 采用交叉编译的方式进行编译，用户只需在登录节点上用交叉编译命令进行编译即可，无需登录到处理器单元上进行。

6.4.4 数学库

KD-50-I 提供针对龙芯体系结构优化的 ATLAS 数值函数库等[顾乃杰 2008]。

6.4.5 资源管理和作业调度

采用 TORQUE[Torque 2008]和 Maui[Maui 2008]进行资源管理和作业调度，利用 Ganglia 进行系统运行监控[KD50 2009]。

6.5 通信协议分析和优化

高性能是机群通信系统优化的主要目标。为了优化机群通信系统，需要综合考虑其点到点通信性能，以及多个进程在不同通信模式同时通信时的全局均衡通信性能。MPI 利用基本通信库的基本通信功能，为用户应用程序提供一个基于消息传递的并行编程环境（图 6.8）。全局通信一般采用点对点通信实现，点对点通信性能提高了，全局通信性能也会随着提高[杨晓奇 2008]。

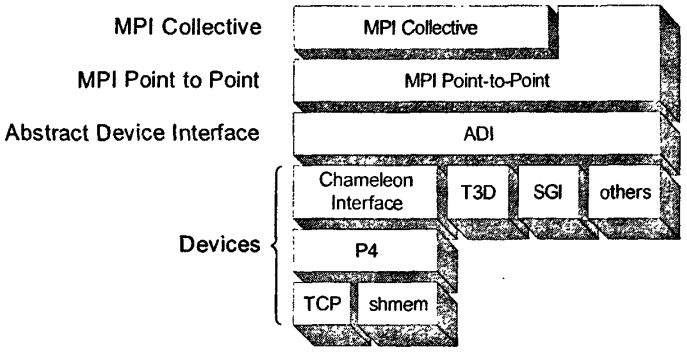


图 6.8 MPICH 软件层次图

6.5.1 点对点通信

点到点通信性能是评价一个机群通信系统的最基本的性能指标，一个好的机群通信系统必须有好的点到点通信性能。点对点通信属于应用层，从网络的层次上看，影响点对点通信性能的因素有[杨晓奇 2008]：

1. 应用层：消息传递库实现点对点通信的算法；

2. 网络层，传输层：网络协议；

3. 物理层，数据链路层：网络传输、交换设备，网络的带宽、延时是影响点对点通信性能的最基本因素。

随着 I/O 处理速度和网络传输速度的提高，TCP/IP 协议中不必要的开销已经成为通信的瓶颈。

6.5.2 网卡参数调整

调整 TCP 参数，优化网络性能。因为 TCP 对所有网上数据的传输进行控制，所以调整 TCP 参数是优化网络性能的重要的一步。

通过调整网卡的 PCI 参数，优化网卡对 PCI 总线占用的时间片，极大地提高了传输带宽。

336 个处理单元全部由以太网交换机互连，如果某些处理单元网卡的 MAC 地址不能被交换机学习进地址转发表，发送给这些 MAC 地址的数据包会被发送到机柜交换机的所有端口，严重影响整体通信性能。挑选网卡 MAC 地址，确保所有处理单元网卡的 MAC 都能学习进交换机地址转发表。

影响网络性能的 TCP 参数有许多，对于本项目这种低延迟、高带宽网络，启用 Linux kernel 中的 tcp_low_latency 选项，禁用了 tcp_timestamp 选项。

通过以上调整，点到点的 TCP 传输带宽从最初的 200Mbps 提升到 500Mbps，延迟也得到很大改善[杨晓奇 2008]。

6.5.3 任务卸载（TASK OFFLOAD）

为了提高性能，利用网卡的硬件特性，检测网卡提供的硬件特性，TCP/IP 传输层可以将校验和(Checksum)任务卸载给拥有适当任务卸载能力的网卡。让硬件进行 Checksum，这样就可以减少整个软件的开销，而且对于 CPU 而言可以减轻负载，而且比软件检测要快。可以通过修改网卡驱动中网络设备属性和操作系统内核来实现，从内核的角度去理解可以说只是在完成对伪首部的字节校验，对于设备而言则是完成剩下数据的校验[杨晓奇 2008]。

6.5.4 全局通信路径优化算法

MPI 定义了 14 个全局通信操作，其中常见的全局通信有广播(Broadcast)、散发(Scatter)、收集(Gather)、组收集(Alltogether)、归约(Reduce)，完全交

换(AlltoAll)，同步(Barrier Synchronization)等操作。实践证明[Rolf 1999]，MPI 的函数开销时间的 40%消耗在是 MPI AllReduce 和 MPI Reduce，而且大约 25%程序执行时间是在节点数目不是 2 的幂的情况下。全局通信路径优化自调整算法根据机群网络拓扑特点，消息大小进行优化。目的是对长消息减少带宽使用，而对短消息减少延迟[杨晓奇 2008]。

1. Altogether

MPICH 组收集采用环(Ring)算法，所有进程按各自的进程号大小顺序构成一个逻辑环。进程号为 rank 的进程，它的左边进程的进程号是(rank-1) %size，右边进程的进程号是(rank+1) %size，其中 size 为进程总数。组收集操作执行时各进程循环 size-1 次调用消息接收发送操作，每一次从左边进程接收数据，同时向右边进程发送上一次接收到的数据(第一次发送的是各进程原有的数据)。环形算法对长消息(>512KB)和机群数目是任意的情况下最有效。

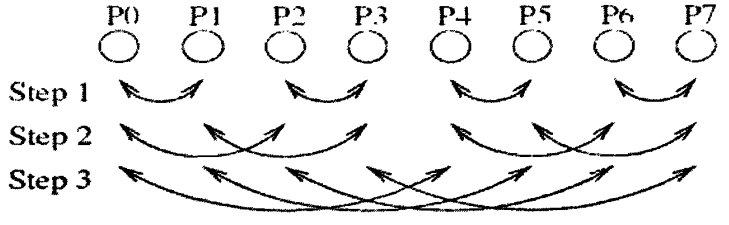


图 6.9 Recursive Doubling 算法

图 6.9 显示 Recursive Doubling[Rajeev 2005]算法对 8 个节点的情况。第一步在步长为 1 的节点间交换数据，第二步在步长为 2 的节点之间交换节点自己本身的数据以及上一次接收到的数据。第三步在步长为 4 的节点交换节点自身的数据以及前两步接收的数据。对节点数目 n 是 2 的幂的情况下，需经过 lgp 步，全部时间为：

$$t_{recursive-doubling}=\lg p\alpha+\frac{p-1}{p}n\beta$$

该算法对中小消息情况且机群节点数目是 2 的幂的情况最有效。

图 6.10 显示 Bruck 算法[Bruck 1997]对 6 个节点的情况。首先每个节点拷贝输入的数据到发送缓存的最顶部。对第 k 步，节点 i 把自己目前所拥有的数据发给目的节点（$i-2^k$），接收来自节点（$i+2^k$）的数据。整个过程持续$\lfloor \lg p \rfloor$次。如果节点数目不是 2 的幂，需要加上一个额外步骤，每个节点发送前$(p-2^{\lfloor \lg p \rfloor})$块和它所接收的数据。最后，虽然各个接收了全部数据，但数据没有在合适的位置上，需要对数据的顺序进行调整。$t_{bruck}=\lceil \lg p \rceil \alpha+\frac{p-1}{p}n\beta$，此算法对短消息(< 80 KB)和集群

节点数目不是 2 的幂的情况下最有效。

2. Broadcast

MPICH 采用扁平树，适合短消息。$t_{tree}=\lceil \lg p \rceil(\alpha+n\beta)$，[Thilo 1999]中提出对长消息，可以采用先把消息分段，MPI_SCATTER 给其他节点，然后在 MPI_ALLGETHER 把分段的消息收集到每个节点上，最终达到散发的效果，$t_{van}=(\lg p+p-1)\alpha+2\frac{p-1}{p}n\beta$。该算法对长消息、节点数目 logP>2 的情况较好。

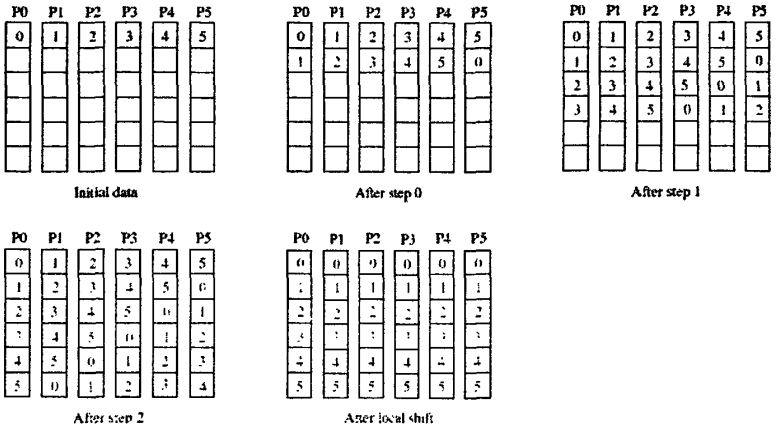


图 6.10 Bruck 算法

3. All-to-All

MPICH 对通讯不进行调度。每一个进程用循环发出 MPI_Irecv，然后用循环发出所有 MPI_Isends，接着跟着 MPI_Waitall。不使用循环变量作为作为接收和发送目标进程号，每一个进程使用 $(rank+i)\%p$ 计算源和目标进程。新的 all-to-all 算法[Rajeev 2005]根据消息的大小不同使用不同的算法。对短消息(<=256bytes)使用 Bruck 算法，$t_{bruck}=\lceil \lg p \rceil\alpha+(\frac{n}{2}\lg p+\frac{p}{n}(p-2^{\lfloor \lg p \rfloor}))\beta$，对长消息且节点数目是 2 的幂的情况，使用双交换（pairwise-exchange）算法[Rajeev 2005]，此算法使用 p-1 步。对第 k 步，每个进程计算它的目标进程 rank^k（异或操作），然后和目标进程直接交换数据。对节点不是 2 的幂的情况，对第 k 步，每个进程从 rank-k 接收数据，向 rank+k 发送数据。对这两种情况，数据都是直接在源节点和目标节点交换，不经过中间步骤。

4. Reduce-Scatter

Reduce-Scatter 是归约操作的一种，与普通归约不同的是，它把归约的结果散发到所有节点。MPICH 首先采用二项树把结果归约到 rank 0，然后执行线性散

发。这个算法执行 P/2 步，$t_{binomial}=(\lg p+p-1)\alpha+(\lg p+\frac{p-1}{p})n\beta+n\lg p\gamma$。对短消息采用 recursive halving 算法（图 6.11）[Rajeev 2005]，它和 allgather 算法中采用的 recursive-doubling 类似。首先，每个进程和距离自己 $\frac{p}{2}$ 的节点交换数据。第二步每个进程和距离自己 P/4 的节点交换数据。这个过程一直迭代 lgp 步，每步二分通信的数据，$t_{rec\_half}=(\lfloor\lg p\rfloor+2)\alpha+2n\beta+(1+\frac{p-1}{p})n\gamma$。如果节点数目不是 2 的幂的情况，归约那些最接近 2 的幂数目节点，偶数节点发送数据给它们的邻居奇数节点(rank+1)。这些奇数节点在 recusive-havling 算法中计算自己和它们左节点结果，最后把结果发回给他们的左邻居节点，$t_{rec\_half}=(\lfloor\lg p\rfloor+2)\alpha+2n\beta+(1+\frac{p-1}{p})n\gamma$。对长消息，采用成对交换(pairwise exchange)算法。此算法运行 p-1 步，第 k 步每一个进程发送消息给 rank+k,从 rank-k 接收数据，执行局部归约。

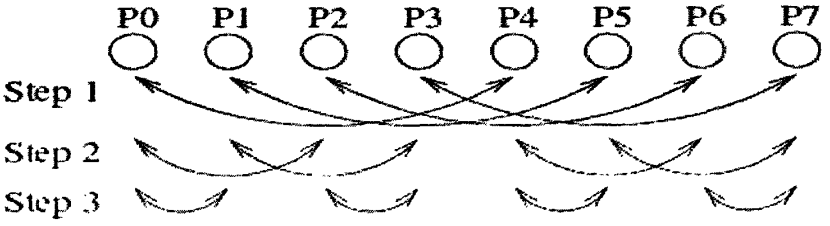


图 6.11 Recursive halving 算法

5. Reduce 和 Allreduce

MPI_Reduce 执行全局归约操作，将结果发送给指定的根节点。MPI_Allreduce 返回结果给所有节点。对 MPI_Reduce MPICH 采用扁平树算法，执行 lgp 步，每步数据通讯的数目是 n。因而 $t_{tree}=\lceil\lg p\rceil(\alpha+n\beta+n\gamma)$，MPI_Allreduce 只是 MPI_Reduce 之后再加上广播（broadcast）操作。对短消息，扁平树算法很好。对长消息，Rabenseifner[Rolf 2007]提出执行 reduce-scatter(recursive halving 算法），然后 gather(采用二项树算法），因而 $t_{rabenseifner}=2\lg p\alpha+2\frac{p}{p-1}n\beta+\frac{p}{p-1}n\gamma$.

6.6 通信性能

NetPipe 和 PMB 完成对 MPI 环境的一个详细报告和测试数据的分析，它能提供机群中 MPI 环境的详细数据。

6.6.1 点对点通信

评价一个机群通信系统点到点通信时最基本的性能参数是通信延迟和通信带宽。通信延迟是指消息从发送方到接收方全过程花费的时间。通信带宽是指在点到点通信过程中，单位时间从发送方传送到接收方的消息字节数。使用 NetPIPE 测试软件，测试不同节点间的通信性能如下（表 6.1）[杨晓奇 2008]：

表 6.1 点对点通信性能

测试项目	参与测试结点	TCP 最小延迟(us)	TCP 最大带宽(Mbps)
同一个交换芯片内2节点	Node0101，Node0102	33.29	401
同一个交换机内2节点	Node0101，Node0201	37.12	402
相邻交换机下2节点	Node0101，Node1401	38.80	403
不相邻交换机下2节点	Node0101，Node2801	39.82	398

表 6.2 全局通信性能

处理器数	64	128	256	336
SendRecv(MB/s)	26.90	26.00	27.89	27.41
Reduce (us)	3296	5226	4992	4627
Allreduce (us)	4459	4830	5217	6917
Allgather (us)	64909	133886	261693	334914
Allgatherv (us)	65271	134594	259421	329559
Alltoall (us)	248526	376146	5024150	5934214
Bcast (us)	9290	10471	11713	18687
Barrier(us)	5042	8072	8059	8384

6.6.2 全局通信

评价一个机群通信系统全局通信性能，主要通过以下几个指标：广播（Broadcast），归约（Reduce），全归约（AllReduce）和完全交换（AlltoAll）。使用 PMB 2.2 测试软件，对 KD-50-I 不同规模的计算节点进行如下测试：

1. 16384 字节的 SendRecv、Allreduce、Reduce、Allgather、Allgatherv、

Alltoall、Bcast 测试，得到相应规模的带宽或延迟。

2. Barrier 测试，得到同步延迟。

测试结果如表 6.2 所示[杨晓奇 2008].

6.7 扫描电子显微成像计算程序在 KD-50-I 上的应用及优化

研究在 KD-50-I 上的应用,对于推广国产高性能计算机的使用有着重要意义。以下两节结合物理学实际应用，研究了扫描电镜成像模拟程序在 KD-50-I 上的应用及优化情况。

在物理学中，简单地说，电子束就是指在真空管中被观察到的电子流。可以有多种方法产生电子束，比如在真空管中安装正负两个金属电极，在两个电极间形成一定的电压降。电子束最早由德国科学家发现。由于电子本身的特性，电子束可以通过电磁手段被施加以聚焦、加能等操作。

扫描电子显微镜是这样一种电子显微镜：通过高能电子束与样品的相互作用来形成电子图像。扫描电子显微镜在科学研究和生产实践中有着广泛的应用，比如在材料分析技术方法中，电子束被广泛应用于研究材料的晶体结构、组成部分、电子结构、表面形貌、内部缺陷等各种微观性质。所以，电子束与物质的相互作用是凝聚态物理研究中的一个非常重要的领域，构成了许多检测和分析工作的物理基础[Li 2005]。

扫描电子显微镜的背散射电子像与材料组分密切相关，一般用来研究材料的组分分布，而二次电子像具有高灵敏度、高空间分辨率、二维剖析能力、测量简便等优势，是用于研究纳米尺度的样品和器件等的强有效工具。理解电镜衬度的形成机理及对影响衬度关系的各种因素进行模拟分析，是将该分析工具进一步发展至实用化的必须环节。

在建立恰当的物理模型的基础上，借助 Monte Carlo 计算模拟方法模拟扫描电子显微镜的成像，可模拟具有复杂空间结构和材料组成样品的扫描电镜成像，验证电子与固体相互作用的散射模型，可用于研究样品的二次电子和背散射电子信号的形成和发射过程，探讨影响衬度的各种物理因素，探索提高衬度的有效途径等，能作为扫描电镜在实际应用中的辅助手段，促进对扫描电镜成像的理解以及在实际中的应用。

6.7.1 扫描电子显微镜的工作原理

扫描电子显微镜（SEM，Scanning Electron Microscopy）主要由如下几个基本部分组成[Li 2005]：

1. 控制台

扫描电子显微镜是一个庞大的系统，其绝大部分均可以通过控制台来加以控制。控制台能完成的功能主要有：控制电子束（如电子束方向、电子枪，等）、图像处理（如线性或非线性信号放大、静态帧存储以及图像显示，等）、扫描初始化和同步、真空系统控制等。事实上这几部分的功能基本上就是一台扫描电子显微镜的全部功能。一个传统的控制台由各种专用逻辑电路构建而成，每块电路均放在一个大的盒子中。而目前较先进的控制台则出于节省成本的考虑，往往采用 PC 机或小型工作站来构建，当然得在这些通用计算机里加上一些控制卡，以将控制信号输出到相关设备。

2. 真空系统

电子束总是在真空系统中移动着的。一个真空系统，一般包括电子光学系统、电子束操控系统、信号采集系统以及真空泵，等等。

3. 电子枪

任何扫描电子显微镜都有电子枪，电子枪就是电子束的产生装置。将电子从原子或其他料子中剥离开以形成电子束，有两基本方法：一种是施加足够大的能量将电子拽开；一种是将粒子置入强电磁场中。

使用扫描电子显微镜，能使由电子枪发射并被数千伏高压加速的电子束与物体表面或内部的原子相互作用，以产生各种电光子信号（如二次电子（SE，Secondary Electrons）、背散射电子（BSE，Back-scattered electrons），等）。这些信号都需要特殊的设备来采集、放大并最终形成图像。

在最普遍/标准的电子和光子信号采集模式---二次电子成像模式---下，扫描电子显微镜可以产生清晰度极高的三维图像，而这些图像可以揭示 1 至 5 nm 尺度下的物理细节。扫描电子显微镜的放大倍数一般介于 25 倍到 250,000 倍之间，其分辨率可以达到最先进的光学显微镜的 250 倍。由于这种方法产生图像的物理原理及所产生图像的高清晰度/分辨率，使得扫描电子显微镜在研究物体的表面结构等领域大显身手。迄今为止分辨率最高的扫描电子显微镜是 JEOL 的 JSM-7700F 色差修正显微镜，在 5 千伏时可以得到最有最高分辨率和放大倍数的二次电子图像[JEOL 2005]，而且由于其焦深长，可对形貌变化范围较大的区域同时成像，所以是研究物体表面形貌特征的最强大和普遍使用的工具[Wells 1974]，特别适合

用于微尺度下物质材料的表面观察。

背散射电子系由电子束在物体表面的弹性反射所形成。由于背散射电子信号的密度与能谱范围内的原子数紧密相关，故利用背散射电子得到的扫描电子显微图像可以较好的揭示不同元素在物体内部的分布情况。很多利用二次电子散射无法得到的图像可以利用背散射电子形成。

扫描电子显微镜的工作原理最早是在 1935 由 Knoll[Knoll 1935]首先提出的。1938 年 Ardenne[Ardenne 1938]设计了一个利用扫描电子束穿过薄膜样品的电子来形成图像的装置，这是实际意义上的第一台扫描透射显微镜。而第一台真正意义上的扫描电子显微镜则是由 Zworykin[Zworykin 1942]等在 1942 年所开发，其主要构成部件包括：一个倒向腔体（底部有一个电子枪）、三个静电透镜以及一个在第二和第三个透镜之间的扫描线圈，该扫描电子显微镜使用光电倍增管来采集信号。1948 年，剑桥大学的 Oatley 在 Zworykin 的工作基础上开始构建新的扫描电子显微镜，McMullen[McMullen 1952]在他的博士论文中描述了这方面的工作。Smith[Smith 1956]针对电子光学方面做了一些改进：利用非线性信号处理以提高图像质量。其后的主要进展是对信号收集过程进行改进，在 Zworykin 使用的发光屏/光电倍增管中加入一个光导管，使得闪烁体和光电倍增管之间直接进行光学耦合，可大大提高效率。这项工作是由 Everhart 和 Thornley 完成的，因此该探测器被命名为 Everhart-Thornley[Everhart 1960]探测器。剑桥大学的 Pease 和 Nixon 把所有这些改进都融合进一个装置中，该装置利用了倒向腔体、电磁透镜、双偏折扫描系统、补偿器线圈以及 Everhart-Thornley 探测器等，它是第一台商业电子扫描显微镜（即 1965 年剑桥科学仪器公司的 Mark I）的原形[Pease 1965][Li 2005]。

6.7.2 Monte Carlo 方法及其应用

Monte Carlo 方法最早是由美国 Alamos National Laboratory 实验室的科学家所提出，系指一类计算算法的集合，这类算法都是依赖于重复随机抽样方法来计算其结果。Monte Carlo 方法经常用于模拟仿真物理和数学系统的特征。由于 Monte Carlo 方法对计算的重复和随机数的产生非常依赖，故 Monte Carlo 方法非常适合于在计算机上进行计算。

Monte Carlo 方法非常适合用于模拟拥有大量的随机耦合度的复杂系统，如流体力学、分子结构、材料学研究等。除此之外，Monte Carlo 由于在对输入具有随机性的系统的模拟上具有强大力量，所以该方法还广泛应用于金融、数学领域等。

由上可见，Monte Carlo 方法是适合于模拟电粒子在固体中的运动过程等问

题的。在物理实验中，可以通过利用电子束来撞击研究对象来获得电子与对象之间的一些互动信息。但由于目前的科技水平和物理条件所限，整个互动的过程非常难以捉摸，尤其在一些具体现象方面。这就必须寻求其他方法来得到一些从实验中无法详细具体获得的信息。在这方面，由于 Monte Carlo 方法本身的特点，可以起到重大作用。Monte Carlo 模拟方法本身实际上是一种理想的计算机实验，但仿真结果数据却可以帮助理解所要研究的现象[Li 2005]。

电子输运的 Monte Carlo 模拟是建立在对散射过程的随机描述之上的[Binder 1992]。电子在物体内的碰撞过程，可以用 Monte Carlo 方法来建模，其科学原理在于物理学中的散射截面概念。

电子在物体内的运动轨迹有些类似于“Z”字形。电子发生散射时就会改变方向。根据 Monte Carlo 方法的原理，电子发生散射时的散射角和能量损失等都可以利用随机数表征。于是，对电子轨迹的描述某种意义上即相当于对大量的复杂公式的排列组合计算。同时，由于 Monte Carlo 的内在特性，统计误差与随机事件的数目的平方根成反比关系，故需要模拟大量的电子轨迹。

建立在电子相互作用的微观机制之上的电子散射模型对于模拟结果有着非常重要的影响。现在已经有许多科学家提出了很多的固体中电子散射的 Monte Carlo 模型，可以用来研究不同的物理问题。但不管哪种模型，对所使用的计算机性能均提出了较高要求，比如多变量的激发函数会要求占用大量的内存空间[Li 2005]。

Monte Carlo 模拟的具体过程可以参见[Valkealahti 1984]。模拟过程中所采用的抽样步骤可参见[Ding 1996] [Li 2005]。

6.7.3 扫描电子显微成像模拟

理论上，人们可以通过模拟 SEM 中采集到的信号的产生过程来模拟扫描电子显微镜衬度像。计算机仿真技术和模拟方法不仅有助于改进扫描电子显微镜，还特别有助于理解扫描电子显微镜中的成像机制和图像衬度形成机理。

随着计算机仿真和模拟技术的不断前进，尤其是计算方法和计算机性能的进步，使得在计算机上模拟扫描电子显微镜成像成为现实。

人们使用计算机对扫描电子显微成像进行模拟的主要方法是 Monte Carlo 方法，而使用 Monte Carlo 方法计算的主要对象则是电子散射轨迹和电子/光子信号的产生与发射过程[Shimizu 1992]，换句话说，Monte Carlo 计算方法的输入集主要是电子入射的相关信息和材料的基本信息，而输出则是电子散射轨迹和信号的产生与发射过程的数学描述。

在使用 Monte Carlo 方法进行模拟计算时，对于所研究的样品的外表特征和

物质结构特征必须加以考虑，并根据不同情况采取各种不同的处理方法[Li 2005]。例如[Hovington 1997]的工作中，所开发的程序模拟嵌入到均匀基底内的单个球样品，该程序的物理模型建立在 Mott 弹性散射以及 Joy 和 Luo 所提出的连续能量损失近似之上，通过模拟电子束在固体样品内部的单散射行为特征来研究扫描电子显微镜中的 X 射线、背散射电子和二次电子等，该程序可以用来做点分析、线扫描和二维图像研究等。[Radzimski 1995]中的工作模拟了多层多元素结构的样品的二维背散射电子图像，该程序采用 Rutherford 散射截面和 Bethe 阻止本领仿真单散射的过程，可以用来研究扫描电子图像与轰击电子束的尺寸、探测设备的参数之间的关系。[Howell 1996]的工作中开发了一个研究样品表面宏观形貌与电子背散射之间的交互影响的程序，该程序可以用来模拟某些具有平整表面、圆环或者用波纹构造的粗糙表面的电子图像。[Lowney 1995]的工作中所开发的 Monte Carlo 模拟程序主要用来模拟衬底上光刻线结构的电子图像，还可用于仿真一个简单 V 形槽的二维背散射电子和二次电子图像。[Seeger 2003]的工作中前述程序进行了扩展，可以模拟较复杂样品的表面的背散射电子和二次电子图像，该工作中采取了一种近似方法，即用许多三角形对样品表面进行构建。该程序的优点在于能够处理具有较复杂的表面的样品，而缺点则是对样品表面的建模过程或数学描述过程非常复杂，难度也大，另外该方法对样品内部的非均匀三维结构的处理不够好。在[Yan 1998]的工作，为了能对一些比较复杂的样品结构的背散射电子和俄歇电子图像进行处理，利用 Monte Carlo 方法开发了一个三维的模拟程序。该程序能处理的几何结构较多。程序要求必须能够解析表示研究样品的边界，这限制了程序的应用范围。另外该程序不能模拟二次电子成像。

由于对二次电子发射的进行精确计算非常困难，并且所需的计算量极大，因此严格意义上说，上述研究工作中的大部分都还不能得到二次电子像（除非做很粗糙的近似，但这样的效果不好）。另外，模拟构造样品的三维复杂结构也非常困难，这限制了 Monte Carlo 方法的应用的广度和深度，所以目前的研究一般都注重于模拟研究样品的背散射电子图像，且要求这些样品的成分结构简单、分布均匀。

因此，不仅是从并行计算方法还是从扫描电子显微成像的应用前景（如纳米测量学和真实器件的成像研究）来说，对具有非均匀成分、结构形貌非常复杂的样品的进行高效准确的模拟，获得高质量扫描电子显微图像，就成为了具有重要意义的工作。

6.7.4 消息传递接口（MPI）

并行计算程序的实现必须要通过相应的并行语言来实现，目前较主流的并行

编程方法主要有三种：PVM（并行虚拟机）、MPI（消息传递接口）和 OpenMP。

MPI 的英语全称是 Message Passing Interface，中文即消息传递接口，是消息传递编程方式的一种标准，其目的是为用户提供一个实际可用的、可移植的、高效的和灵活的消息传递接口库。MPI 编程是广泛应用于消息传递并行计算的一种方法，适合于存储分布模型上的并行计算 [陈国良 2002]。目前关于 MPI 的实现有很多种，不同的机构或单位可能会有不同的实现。尽管有许多不同的实现方法，但其基本思想仍是通过消息传递来实现进程间通信[都志辉 2001]。在本章的工作中，将利用 MPI（消息传递接口）对扫描电子显微镜成像 Monte Carlo 并行模拟程序在 KD-50-I 进行研究。

MPI 的特点概括起来主要涉及有以面三个方面：

1. MPI 不是一种语言，而是一个标准和共享库。MPI 仅仅是一个并行库，经过编译后形成共享库形式。仅有 MPI 是无法实现并行计算的。但 Fortran、C 等语言可以通过各种手段加以调用编译后形成的 MPI 共享库，使用库所提供的 MPI 编程函数，从而实现并行计算。

2. MPI 是一种标准或规范，是一个消息传递的标准。而不是特指某一个对它的具体实现。MPI 的历史可以追溯到 1992 年。迄今为止，几乎所有的并行计算机商都提供对 MPI 的支持，可以免费得到 MPI 在不同并行计算机上的实现。国际上的研发团体还开发了一些免费版本的 MPI 实现，主要有 MPICH、LAM 和 CHIMP 等[陈国良 2002]。

3. MPI 建立了一种消息传递编程模型，并成为这种模型的代表以及事实上的标准。MPI 不是一个独立的自包含的系统，而是建立在本地并行程序设计环境之上，其进程管理和 I/O 均由本地并行程序环境实现。MPI 的实现和提供的函数虽然规模庞大，但其核心思想是提供了进程间通信的另一种形式。

MPI 的目标概括起来有：较高的通信性能；较好的程序可移植性；强大的功能。这三个方面对于 MPI 的具体实现来说，体现了实现的质量，但同时这三方面目标其实是互相冲突的，必须在三者之间取得较好的平衡。

MPICH2 共提供有 200 多个编程函数，但所有功能可由 6 条基本语句构成的子集完成（如表 6.1 所示）：

表 6.3 MPI 六条基本语句

语句	描述
MPI_INIT	MPI初始化
MPI_COMM_RANK	确定自己的进程标志符
MPI_COMM_SIZE	确定进程总数
MPI_SEND	发送一条消息
MPI_RECV	接收一条消息
MPI_FINALIZE	结束MPI计算

这些实际上是 MPI 中最基本的六条语句，包含初始化并启动计算、结束整个计算过程、获得进程 ID 号以及发送和接收 MPI 消息等。如果不考虑编程难度和运行时的效率问题，MPI 并行编程非常容易掌握，因为仅使用上述六条语句便能完成计算任务。

6.7.5 并行化成像模拟实现

使用 Monte Carlo 方法进行模拟，必须要对不同的实验条件（如电子能量、电子入射角度和研究对象的物理特征等）进行大量的模拟，才可以降低统计误差。另外，在扫描电子显微成像研究中，还需要模拟大量的电子入射点。电子在研究对象中的轨迹数是入射电子数目的数十倍，因此模拟计算量非常大。例如：使用串行程序在单处理机上模拟在基底内部含有一个空心球的研究对象的线扫描图（设分割点为 300 个），其中每个入射点的入射电子数为10^4，则在一台 1.2GHz 的 PC 机上运行时间大约为 4 个小时；而对于一个面扫描（分割为 300×300 个点）所需要的时间则大约是 1200 小时左右，即约 50 天，7 个星期多。这样串行程序所需的计算时间对于研究者而言是不可接受的，因此需要对计算方法进行改进。主要有两种改进方法：一是对现有串行程序进行优化，如对数据集进行合理分块、改变程序运行过程等，以提高程序效率；二是利用并行计算。由于串行程序效率的提升有限，且并行计算技术已经发展地较为成熟，能够花费较少的时间和其他代价得到不错的研究结果，所以利用并行计算技术成为首选。[Li 2005]中用 MPI 实现了三种不同的并行程序：

1. 按入射点分割

在该并行实现中，使用 MPI 模型编程。其基本思想是，将并行计算机中的计算结点与电子入射点相互对应，每个计算节点的任务是依次分别计算不同入射点的电子运动轨迹。每当一个入射点的相关计算完成后，计算任务所依赖的处理器将利用 MPI 消息机制将计算结果发送给主计算结点上的管理程序。而当主结点完成自身的模拟计算任务后（事实上主结点也可以不进行模拟计算，而只进行数据的分配和搜集等工作），将对所有从从结点发送过来的数据进行检测和整理，之后再把计算结果写入某文件中，这就完成了一轮计算。接下来主结点会为所有计算结点重新分配新的数据，启动下一轮计算，并等待接收计算结果。整个过程依此类推。这种计算模式是典型的 SIMD（单指令多数据流）模式。另外，整个计算过程中，轮与轮之间的阻塞同步可以依赖 MPI 接口轻易完成。按照入射点进行分割的方法是最简单直接的并行实现方法。

2. 主从模式并行化实现

与上述按照入射点分割方式类似，在该 MPI 并行实现中，每个入射点相关的计算任务仍是分配到不同的计算结点上并行进行。但第一种并行实现方法中有一个很明显的缺陷，即轮与轮之间是严格同步的，即必须本轮计算全部完成，所有计算结点才可以进入下一轮计算。这种方式在一些异构的机器上，或者负载较不平衡的机器上，会严重影响性能。主从模式并行化实现正是对第一种方法的优化，消除了前述缺陷。

本并行实现的工作流程基本可以描述为三步：主结点分配任务，发送数据；计算结点进行计算，完成后以异步模式将数据发送给主结点；主结点收到某结点的计算结果后，对该结点重新分配任务和发送数据，再将刚收到的数据进行处理。

可以看出，整个计算过程是异步的，各计算结点不会因为等待其他速度较慢的结点而停止计算，进行空等。这样可以大大地减少处理机空转时间，能提高整个程序的效率，并节约资源。

3. 以电子为单位进行分割

前面两种并行化方法的基本思想其实是相似的，即将与电子在球面的入射点相关的计算任务分配到不同的处理机上。而按照电子数目进行分割，则提供了另一种并行化思路。其基本思想是，每个计算结点将不再计算每个入射点相关的任务，而是对每一个电子入射点，将所有需进行模拟的电子集中起来，以电子为单位进行处理机分配，理想情况下是，处理机个数足够大，每个电子相关的计算任务都能分配到一个不同的处理器上。当一个电子入射点相关的任务完成后，即一

轮计算已经完成，则计算其他某一个入射点的相关任务，即进入下一轮，依此类推，直至结束。轮与轮之间的同步利用 MPI 提供的接口可以轻易的实现。

在该并行化实现中需注意一点，即必须使各个计算结点之间的随机数不存在任何关联，否则计算结果得不到保证。[Li 2005]中通过将随机化产生的初始数组元素随机分配到各计算结点作为随机数算法产生随机数的种子来解决这一问题。

表 6.2 中给出了前述三种 MPI 并行化实现在 HP RX2600 高性能计算集群上的运行时间对比，其中计算结点数为 8。从表中可以看出，主从模式并行化实现性能相对比较好。经分析其原因在于，由于采用异步计算方式，减少了计算结点的等待任务分配和数据的时间，增加了计算时间所占比重，从而提高了程序的整体运算效率。

表 6.4 三种 MPI 并行实现性能对比

MPI并行模型	运算时间（S）
按入射点分割	771.54
主从模式并行化实现	617.38
以电子为单位进行分割	634.18

6.7.6 性能优化

本文根据 KD-50-I 万亿次计算机的特点，以及扫描电子显微成像模拟程序的特点，将优化工作主要集中于两个方面：

1. 单机性能优化

单机的性能可以通过提高 Cache 的命中率、优化指令级流水和编译优化选项组合等技术来优化。

(1) 高速缓存命中率

高速缓存的命中率对程序运行性能的影响非常大。由于 KD-50-I 万亿次计算机中使用的龙芯 2F 处理器中的高速缓存系统使用的是随机替换策略，故本文的工作主要使用了下标对换、循环合并、循环展开等方法，这些方法可改善对数据调整缓存的访问的时空局部性。同时，将数据集进行适当调整，使其能适合数据高速缓存块的大小，可以减少访问数据高速缓存时的冲突，使高速缓存的访问失效

次数得到降低，提高计算性能。

(2) 指令级流水

由于考虑到龙芯 2F 处理器的浮点运算能力较强，对乘加指令的实现较好，因此我们可以把编译后的分开的乘法和加法指令合并成一条乘加指令。这样可以减少指令数目，提高计算性能。本文主要对计算量较集中的循环进行了优化工作，主要方法是尽可能多的进行指令合并、使用乘加指令，调整指令顺序，等。除此之外，根据程序特点，对于一些调用较频繁的函数，则可以采用内联（inline）技术，等。

(3) 编译优化选项组合

KD-50-I 上所使用的 MIPS 编译器提供了较多的优化选项，科学合理的使用这些选项，往往能给程序的性能带来较好的提升。通过反复试验，我们确定的最优编译优化选项组合为：-mno-branch-likely -O3 -mips3。

2. glibc 优化

Glibc 库是提供系统调用和基本函数的 C 库，对 Glibc 库的优化程序，不仅对应用程序的性能具有重要意义，还能很大的影响整个系统的性能。该程序的实现中频繁地调用了 glibc 库函数。我们针对龙芯处理器对 glibc 库进行了优化，带来了较好的效果。

3. 通信优化

并行程序中，无论是采用 TCP/IP 通信协议进行通信的 MPI 编程模型，还是共享内存的 OpenMP 编程模型等，通信性能对程序整体性能的影响都很大。影响并行程序通信性能的因素有很多，但主要有两点，一个是机器本身的通信性能，一个是程序本身的通信性能。

对于 KD-50-I 上的应用程序优化而言，机器本身的通信性能是外在条件，已经无法优化。所以本文的主要工作集中在通信数据结构和通信模式的优化上。通信数据结构的优化能够减少通信的数据传输量，降低对带宽的要求；通信模式的优化则可以减少通信操作发生的次数，甚至即便在通信次数未减少的情况下，改善通信的性能，提高程序通信效率（如将通信较频繁的计算进程放到一个计算结点内，等）。

MPI 标准提供了很多强有力的手段，比如在对分布于不连续的内存的数据进行发送时，可以利用 MPI 提供的函数将数据进行发送前打包，再一次发送出去。这样可以减少系统调用的次数，降低开销。

另外，由于大数据量的归约操作对网络拓朴和性能极其敏感，为避免网络堵

塞的情况的出现，当归约数据量较小时，应直接使用 MPI_Reduce()函数进行规约操作。

4. 负载平衡

扫描电子显微成像模拟程序本身的工作特点是，在任务开始阶段，将数据分发到各个计算结点，任务结束阶段，再归约收集数据。整个程序的计算过程中，通信量并不太大，较为适中。所以针对程序主要的优化工作集中在广播（MPI_Brocast()）和归约（MPI_Reduce()）操作上[Li 2007]。

表 6.5 同一计算条件在 KD-50-I 上的不同 CPU 的运行时间及加速比

结点数目	12	24	48	96	168
运行时间（秒）	5280.9	3886.3	1294.3	662.9	432.4
加速比（优化前）	12.00	16.31	48.96	95.60	146.37
加速比（优化后）	12.11	19.32	48.96	95.79	150.37

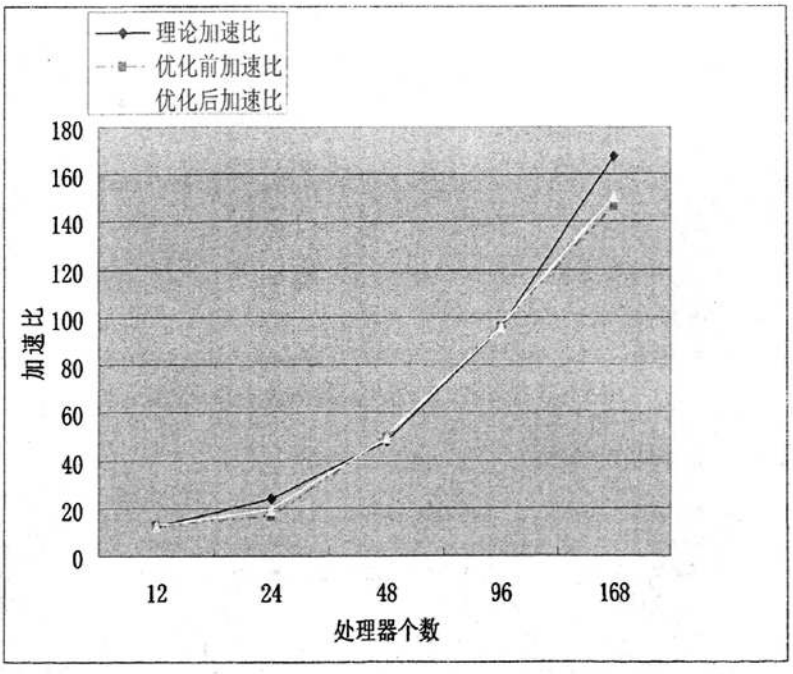


图 6.12 加速比

表 6.5 和图 6.12 给出了加速比。该程序使用交叉并行编译器编译源程序，并分别利用作业高度软件 TORQUE 提交 12、24、48、96、168 和 336 个 CPU 计算同一个题目。测试条件是，入射电子能量为 3KeV，每个入射点的入射电子数目是 1000 个，样品为半径为 5 纳粹的金球体，扫描分割点数为 10201 个，获得的计算结果正确无误。

6.8 小结

KD-50-I 是首台全国产的万亿次计算机，从机器最核心的处理器到外围的交换机，机架等均实现了国产化。本章研究了 KD-50-I 的系统结构技术，包括总体结构、计算结点结构、处理单元结构以及交换底板结构等。随后介绍了 KD-50-I 的网络启动技术、文件系统和系统软件、通信库优化等，并给出了 KD-50-I 万亿次计算机的一些性能测试数据。

扫描电子显微成像技术在物理学研究上有着重要意义。本章研究了扫描电子显微成像技术的物理原理、数学方法、MPI 并行实现等。最后研究了该程序在 KD-50-I 上的应用及优化情况。实验证明，KD-50-I 能够很好地完成任务。本文的工作对于推广国产高性能计算机的使用有着重要意义。

第 7 章 结束语

本章摘要　本章对全文的研究工作、贡献和创新点进行总结，并对下一步工作方向进行了展望。

7.1 本文主要工作

随着半导体技术的发展，以及传统提高处理器性能的手段逐渐失效，多核处理器已经成为现代处理器体系结构发展的趋势。这一趋势不仅为处理器芯片自身的设计带来了新的挑战，同时也为基于多核处理器的高性能计算机体系结构设计带来了新的挑战。

高性能计算机系统市场大部分为国外厂商所垄断，其价格昂贵，使用范围受限，存在很多机构买不起或用不了高性能计算机的情况。另外，高性能计算机作为国家的战略资源，是衡量一个国家科技水平的重要标志，其自主性格外重要。

本文的研究工作主要分为两大部分：一是研究多核处理器中的高速缓存一致性协议，以及多核处理器中的高速缓存的包含与不包含性；二是结合工程实践出发，理论上地研究了首台全国产万亿次计算机 KD-50-I 的体系结构技术以及通信库优化技术，等。具体而言，本文的工作主要有：

1. 多核处理器中的高速缓存一致性协议研究

在多处理器系统中，高速缓存一致性协议的设计不仅对系统的正确性至关重要，且对系统的性能有着重要影响。而随着多核处理器的引入，为高速缓存一致性协议的设计引入了一些新的问题。比如随着处理器芯片内高速缓存数目的增加，以及片内互连网络的复杂化，对高速缓存一致性协议的设计的难度越来越大，而验证协议的正确性更是困难。事实上一些已经商品化的处理器的一致性协议中就存在着错误。除此之外，不仅要保证高速缓存一致性协议的正确性，还要注意协议的可扩放性，以及尽量减小其对性能的影响。本文研究了多核处理器中的高速缓存一致性，且基于可扩放性考虑以及多核处理器本身的特点，重点研究了 MOESI 协议及其实现。随着多核处理器的流行普及，基于多核处理器构建的高性能计算机（M-CMP）也必然会成为主流，对 M-CMP 的体系结构研究将会成为重点。而这种新的结构在带来性能的提高的同时，也带来了许多新的挑战。比如在单核处理器的设计中，只需考虑处理器芯片之间的高速缓存一致性，而引入多核处理器之后，不仅要考虑片间的一致性，还需要考虑片内的高速缓存一致性，等等。针对这一问题，本文在由多核处理器构建的并行计算系统中研究了多核处理器的一致性协

议。据本文调研所知，目前针对 M-CMP 环境下进行多核处理器一致性协议的研究尚不多。

2. 多核处理器片内高速缓存的包含性研究

由于物理工艺以及功耗等因素的限制，片内高速缓存的面积的增加有限。提高多核处理器片内高速缓存的效率，或命中率，对于提高处理器的效率有着重大影响。同时，随着半导体技术的进一步发展，片内处理器核的数目还会继续增加，对存储访问的压力随之会继续加大，但处理器芯片的对外引脚个数，限制了处理器的对外访存带宽。在此背景下，研究提高多核处理器片内高速缓存的性能，对提高系统的性能有着重大意义。本文研究了片上高速缓存的包含与非包含策略。据调研，目前针对此方面的研究还较少。

3. 多核处理器片上 Cache 的性能评测

龙芯系列高性能通用微处理器是我国完全具有自主知识产权的处理器。目前已经有一些研究机构或企业将研制基于龙芯多核处理器的高性能计算机列入研发日程。为了更好地研究龙芯体系结构的多核处理器在科学计算等领域的性能，本文对其片上 L2 Cache 的表现进行了评测。并依此对其设计空间进行了一些探索。

4. 高性能计算机互连网络研究

高性能计算机中的互连网络对系统性能具有至关重要的影响。本文研究了一种先进的新型互连网络：MPU。研究内容包括其数学模型、网络拓朴和路由算法等。本文在理论上对 MPU 的性能与其他当前先进的互连网络进行了对比研究，证明 MPU 具有良好的性能。本文还研究了专为 MPU 所开发的一款大型并行模拟器 MPUS，包括其原理、架构和工作流程等，利用 MPUS 证明了 MPU 设计的正确性及可扩放性等。

5. KD-50-I 万亿次计算机体系结构技术及其通信优化研究

高性能计算机作为国家重要战略资源，其自主性格外重要。KD-50-I 是我国第一台基于国产高性能通用处理器龙芯的全国产万亿次计算机，具有低功耗、低占地面积、低成本、高计算能力等优点。KD-50-I 对我国未来研制千万亿次计算机系统及提高其自主创新性具有示范作用。本文紧密结合工程实践，研究了 KD-50-I 的体系结构技术，包括其硬件体系结构、以及无盘启动技术、通信库优化等方面，并给出了 KD-50-I 万亿次计算机的一些性能测试数据。本文针对 KD-50-I 的设计，及提高其性能和可用性方面做了大量工作。为国产万亿次计算机的推广应用做出了贡献。

6. KD-50-I 在电子显微成像技术中的应用研究

随着高性能计算机性能地不断提高，以及应用的推广，在物理研究中应用地也越来越广泛。扫描电子显微成像模拟技术是材料研究中的一个重要手段。计算机模拟方法不仅有助于改进扫描电子显微镜，特别是有助于理解扫描电子显微镜中的成像机制和图像衬度形成机理。本文研究了扫描电子显微成像模拟程序在 KD-50-I 上的应用，并对其进行了优化。本文工作提高了应用程序运行效率，为 KD-50-I 在不同领域的应用，提供了积极示范。

7.2 本文主要贡献和创新点

本文主要有以下贡献和创新点：

1. 多核处理器高速缓存一致性协议

研究了多核处理器中的高速缓存一致性协议，尤其是 MOESI 协议。MOESI 协议相对 MESI 协议增加了一个“Owned”状态，由于多核处理器片上网络的特点，增加了系统设计和性能提升的空间。目前对 MOESI 协议在多核处理器中的实现的相关研究较少。本文研究了该协议的实现，并结合网络环境对协议的实现进行了优化。

在由多核处理器构建的并行计算系统（M-CMP）中，传统的由单核处理器构建的 SMP 机器中的一致协议协议无法解决问题，要求引入层次化的一致性协议，以解决片内和片间的一致性问题。本文在 M-CMP 环境中研究了多核处理器的一致性协议和协议对性能的影响。

2. 多核处理器片内高速缓存的包含性

由于物理、功耗等因素的限制，多核处理器片上高速缓存的容量受限，如何充分利用有限的片上存储空间对提高系统性能和降低功耗具有重大意义。本文研究了片上基于包含和不包含策略的高速缓存的性能，并提出了一个基于不包含策略的片上高速缓存体系结构。在此结构中，片上一级高速缓存和二级高速缓存中的数据没有交集，可以有效提高片上存储容量。

片上高速缓存的结构对一致性协议具有影响。我们发现，采用不包含结构不仅能提高片上高速缓存的容量，还为一致性协议的优化带来了机会。本文研究了优化后的一致性协议，能有效减少 MOESI 协议的中间状态数量和网络通信量，并提高系统性能。

3. 龙芯体系结构多核处理器在科学计算中的应用

本文率先对科学计算在龙芯体系结构多核处理器上的表现进行了评测、剖析。并依此对其设计空间进行了探索，提出了一些有益的想法。

4. 一种先进的高性能计算机网络

本文研究了一种新型的高性能计算机网络 MPU。在理论上对其性能进行了分析对比。着重研究了为 MPU 的专门开发的大型并行模拟器 MPUS。在高性能计算机模拟器并行化等方面做出了创新和贡献。

5. KD-50-I 万亿次计算机体系结构技术及通信库优化

本文紧密结合工程实践，研究了 KD-50-I 的体系结构技术，特别是无盘启动技术、系统软件架构等。另外，对 KD-50-I 的通信性能进行了优化，为以后的基于龙芯处理器的高性能计算机研制打下了坚实基础。

7.3 进一步工作

多核处理器时代的到来是一个不可逆转的历史潮流。新的工艺及技术对体系结构设计提出了新的要求。随着龙芯处理器本身的发展进步，特别是龙芯多核处理器即将推出，基于新的龙芯处理器的高性能计算机的研制工程已经被提上日程。综上，未来的进一步研究工作包括：

1. 多核处理器片内网络与高速缓存一致性

随着片内可集成的处理器核的数目还将持续增加，处理器核之间、处理器核与高速缓存之间以及与内存之间的通信速度对片内网络的设计提出了挑战。而一致性消息也是在片内网络上进行传输。所以，结合片内网络，研究其对访存性能的影响和对高速缓存一致性协议的影响，将会是未来研究的一个重点。

2. 新的仿真工具的研究

本文中的工作主要基于 GEMS 工具集，这是一个主要以 C++语言编写的模拟器。其运行速度非常缓慢，尤其是当模拟的处理器核较多，内存容量较大时，其速度更是无法忍受。因此，对计算机体系结构的研究必然会要求新的研究工具，比如并行化的模拟器，甚至硬件模拟等。

3. 基于龙芯多核处理器的高性能计算机研究

龙芯多核处理器即将推出，为未来的全国产高性能计算机带来了新的机遇和挑战。将来会针对新的高性能计算机体系结构技术、数学库优化以及通信优化等各方面做更深入的研究。

参考文献

[陈国良 2002] 陈国良,吴俊敏，章锋，章隆兵. 并行计算机体系结构[M]. 北京：高等教育出版社. 2002.

[陈国良 2002] 陈国良，安虹，陈崚，郑启龙，单久龙. 并行算法实践[M]. 北京：高等教育出版社. 2002.

[陈国良 2003] 陈国良. 并行计算---结构.算法.编程[M]. 北京：高等教育出版社. 2003.

[邓越凡 2006] 邓越凡. MPU 系统设计报告[M]. 上海：上海红神信息技术有限公司. 2006.

[都志辉 2001] 都志辉. 高性能计算之并行编程技术---MPI 并行程序设计[M]. 北京：清华大学出版社. 2001.

[冯昊 2008] 冯昊，吴承勇. CMP 体系结构上非包含高速缓存的设计及性能分析[J]. 计算机工程与设计. 7(29):1595-1611. 2008.

[顾乃杰 2008] 顾乃杰，李凯，陈国良，吴超. 基于龙芯 2F 体系结构的 BLAS 库优化[J]. 中国科学技术大学学报. 38(7)：854-859. 2008.

[胡伟武 2001] 胡伟武. 共享存储系统结构[M]. 北京：高等教育出版社. 2001.

[王焕东 2008] 王焕东，高翔，陈云霁，胡伟武. 龙芯 3 号互连系统的设计与实现[J]. 计算机研究与发展. 45(12):2001-2010. 2008.

[杨晓奇 2008] 杨晓奇，郑启龙，陈国良，张俊霞. 国产万亿次高性能计算机 KD-50-I 的通信优化[J]. 小型微型计算机系统. 已录用，稿件编号：0800182.

[袁伟 2005] 袁伟，张云泉，孙家昶，李玉成. 国产万亿次机群系统 NPB 性能测试分析[J]. 计算机研究与发展. 42(6):1079-1084. 2005.

[张福新 2007] 张福新，章隆兵，胡伟武. 基于 SimpleScalar 的龙芯 CPU 模拟器 Sim-Godson[J]. 计算机学报. 68-73. 2007.

[Agarwal 1988] Agarwal A, Lim B, Kranz D, Kubiatowicz J. APPIL: A processor Architecture for Multiprocessing[C]. Proc of the 17th Annual International Symposium on Computer Architecture. 1988.

[Akhter 2007] S. Akhter, J. Roberts. Multi-Core programming---increasing performance through software multi-threading[M]. 北京：电子工业出版社. 2007.

[Almasi 1989] Almasi. G. S. , GA Highly Parallel Computing. Benjamin-Cummings publishers, Redwood, CA. 1989.

[AMD 2003] AMD. http://www.amd.com/us-en. 2003.

[Ardenne 1938] M. von Ardenne. Z. Tech. Phys. 109:553. 1938.

[Ashwini 2001] Ashwini K. Nanda, Anthony-Trung N., M. M. Michael, D. J. Josep. High-Throughout Coherence Control and Hardware Messaging in Everest. IBM Journal of Research and Development. 45(2):229-244. 2001.

[Baer 1988] J. L. Baer, W. H. Wang. On the inclusion properties for multi-level cache hierarchies[C]. Proc of the 15th International Symposium on Computer Archtecture. 73-80. 1988.

[Barroso 2000] Luiz Andre Barroso, Kourosh Gharachorloo, Robert McNamara, et. at. Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing[C]. ISCA 2000. 2000.

[Baskett 1988] Baskett F, jermoluk T, Solomon D. The 4D-MP graphics superworkstation: computing + graphics = 40 MIPS + 40 MFLOPS and 100,000 lighted polygons per second[C]. Proc 33rd IEEE Computer Society

International Conference-Compcon'88. 468-471. 1988.

[Beckmann 2006] B.M.Beckmann. Managing wire delay in Chip Multiprocessor caches[D]. PhD thesis. Uni of Wisconsin. 2006.

[Bilas 1999] A.Bilas, C.Liao, J.P.Singh. Using Network Interface Support to Avoid Asynchronous Protocol Processing in Shared Virtual Memory Systems[J]. SIGARCH Comput. Archit. News. 27(2):282-293. 1999.

[Binder 1992] K.Binder, Ed. The Monte Carlo Method in Condersed Matter Physics[M]. Springer-Verlag, Heidelberg. 1992.

[Blumrish 2003] M.Blumrish, D.Chen, P.Coteus, A.Gara, M.Giampapa, P.Heidelberger, S.Singh, B.Steimmacher-Burrow, T.Takken and P.Vranas. Design and analysis of the BlueGene/L Torus Interconnection Network[J]. Computer Science. 2003

[Bradford 2006] Bradford M. Beckmann, M.R.Marty, D.A.Wood. ASR: adaptive selective replication for CMP caches[C]. In 39th Annual IEEE/ACM Symposium on Microarchitecture. 2006.

[Briggs 2002] F.Briggs, M.Cekleov, K.Creta, M.Khare, S.Kulick, A.Kumar, L.P.Looi, C.Natarajan, S.Radhakrishnan, L.Rankin. Intel 870: A Building Block for Cost-Effective, Scalable Servers. IEEE Micro. 22(2):36-47. 2002.

[Bruck 1997] Jehoshua Bruck, Ching-Tien Ho, Schlomo Kipnis, Eli Upfal, and Derrick Weathersby. Effcient algorithms for all-to-all communications in multiport messagepassing systems[J]. IEEE Transactions on Parallel and Distributed Systems. 8(11):1143-1156. 1997.

[BusyBox 2007] BusyBox. http://busybox.net/. 2007.

[Byrd 1999] G.Byrd, M.Flynn. Producer-Consumer Communication in Distributed Shared Memory Multiprocessors. Proc of the IEEE. 87(3):456-466.

1999.

[Carter 1991] J.B.Carter, J.K.Bennett, W.Zwaenepoel. Implementation and Performance of Munin. In Proc of the 13th ACM Symposium on Operating System Principles. 152-164. 1991.

[Censier 1978] Censier L M, Feautrier P. A new solution to coherence problems in multicache system[J]. IEEE Trans. Computers. C-27(12):1112-1118. 1978.

[Chaiken 1991] Chaiken D, kubiatowitz J, Agarwal A. Limitless directories: a scalable cache coherence scheme[C]. Proc of the 4th International Conference on Architectural Support for Programming Languages and Operating Systems. 1991.

[Cheng 2006] L.Cheng, N.Muralimanohar, K.Ramani, R.Balasubramonian, J.B.Carter. Interconnect-Aware Coherence Protocols for Chip Multiprocessors. In Proc of the 33rd Annual International Symposium on Computer Architecture. 339-351. 2006.

[Chishti 2005] Z.Chishti, M.D.Powell, T.N.Vijaykumar. Optimizing replication, communication, and capacity allocation in CMPs. In Proc of the 32nd International Symposium on Computer Architecutre. 2005.

[Cox 1993] A.Cox, R.Fowler. Adaptive Cache Coherency for Detecting Migratory Shared Data. In Proc of the 20th Annual International Symposium on Computer Architecture. 98-108. 1993.

[Cray 1993] Cray Research. CRAY T3D System Architecture Overview. Cray Research HR-04033 edition. 1993.

[Culler 2001] David E. Culler. Parrallel Computer Architecture[M]. Beijing: China Machine Press. 1999.

[Dai 1999] D.Dai, D.K.Panda. Exploiting the Benefits of Multiple-Path Network in DSM Systems: Architectureal Alternatives and Performance Evaluation[J]. IEEE Transactions on Computers. 48(2):236-244. 1999.

[David 2001] F.David. F., S.Ravi, G.Serban, et al. Proposed NIST standard for role-based access control[J]. ACM Transactions on Information and System Security. 4(3):224-274. 2001.

[Diefendorff 2000] Diefendorff K, Dubey P, Hochsprung R, et.al. AltiVec extension to powerPC accelerates media processing[J]. IEEE Micro. 20(2):85-95. 2000.

[Ding 1990] Z.J.Ding. PhD thesis[D]. Osaka Univ. 1990.

[Ding 1996] Z.J.Ding, R.Shimizu. Scanning[J]. 18(92). 1996.

[Duato 2003] J.Duato, S.Yalamanchili, L.Ni. Interconnection Networks - and engineering approach[M]. Morgan Kaufmann Publishers. 2003.

[Dybdahl 2006] Haakon Dybdahl, Per Stenstrom, Lasse Natvig. An LRU-based replacement algorithm augmented with frequency of access in shared chip-multiprocessor caches[C]. MEDEA' 06. 2006.

[Eckhardt 1987] R.Eckhardt. Special Issue[J]. Los Alamos Science. 15(131). 1987.

[Eisley 2006] N.Eisley, L.S.Peh, L.Shang. In-Network Cache Coherence. In MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture. 321-332. 2006.

[Everhart 1960] T.E.Everhart, R.F.M.Thornley. J.Sci.Instr[J]. 37(246). 1960.

[Falsafi 1994] B.Falsafi, A.R.Lebeck, S.K.Reinhardt, I.Schoinas.

Application-Specific Protocols for User-Level Shared Memory. In Proc of the Supercomputing. 380-389. 1994.

[Gharachorloo 2000] K. Gharachorloo, M. Sharma, S. Steely, S. V. Doren. Architecture and Design of AlphaServer GS320. In Proc of the 9th International Conference on Architecture Support for Programming Languages and Operating Systems. 13-24. 2000.

[Gostin 2005] G. Gostin, J.-F. Collard, K. Collins. The Architecture of the HP Superdome Shared-memory Multiprocessor. In Proc of the 19th Annual International Conference on Supercomputing. 239-245. 2005.

[Gu 2003] GU Yong-feng, CHEN Zhang-long. Approach to Tailoring Embedded Linux[J]. Journal of Chinese Computer Systems. 24(9): 1697-1700. 2003.

[Gupta 1992] A. Gupta, W. Weber. Cache Invalidation Patterns in Shared-Memory Multiprocessors. IEEE Transactions on Computer. 41(7):794-810. 1992.

[Hammand 1997] L. Hammand, B. A. Nayfeh, K. Olukotun. A single-chip multiprocessor[J]. IEEE Computer, 30(9):79-85. 1997.

[Hammond 2004] Hammond L, Wong V, Chen M. et. al. Transactional memory coherence and consistency[C]. the 31th Annual International Symposium on Computer Architecture. 32(2):102-114. 2004.

[Hagersten 1999] E. Hagersten, M. Koster. WildFire: a scalable path for SMPs. Proc of the 5th IEEE Symposium on High-Performance Computer Architecture[C]. 172-181. 1999.

[Hennessy 2007] John L. Hennessy, David A. Patterson. Computer Architecture A Quantitative Approach[M](4th edition). 209. Beijing: China Machine Press, 2007.

[Howell 1996] P.G.T.Howell. Scanning[J]. 18(428). 1996.

[Hovington 1997] P.Hovington, D.Drouin, R.Gauvin. Scanning[J].19(1). 1997.

[Hu 2008] Hu W.Wang J.GaoX.etal.Miero-architecture of Godson-3 multi-core processor[C]. In Proc of the 20th Hot Chips. 2008[2008 — 11—20]. http: //www. hotchips. org/hc20/main—page.htm

[Hwang 1998] Kwang K., Xu Z. W. Scalable parallel computing: technology, architecture, programming[M]. MaGraw-Hill. 1998.

[IEEE 1993] IEEE. IEEE Standard for Scalable Coherent Interface: IEEE Std. 1993.

[Intel 2003] Intel. http://www.intel.com. 2003.

[Intel 2005] Intel. Intel pentium dual-core processor overview. http://www.intel.com. 2005.

[IBM 2008]IMB. http://www.research.ibm.com/journal/rd/516/le.html

[James 2005] James L, Daniel L. The SGI Origin: A cc-NUMA Highly Scalable Server. In Proc of the 24th Annual International Symposium on Computer Architecture. 1997.

[JEOL 2005] JEOL. http://www.jeol.com/sem_/semprods/jsm7700f.html. 2005.

[John 2001] John L. Hennessy, David A. Patterson. Computer Architecture A Quantitative Approach[M]. Beijing: China Machine Press. 2006.

[Johnson 2002] J.J.Johnson. The AMD-760 MPX Platform for the AMD Athlon MP Processor. AMD White Paper. 2002.

[Joshi 2003] R. Joshi, L. Lamport, J. Matthews, S. Tasiran, et. al. Checking Cache-Coherence Protocols with TLA+. Formal Methods in System Design[J], 22(2):125-131. 2003.

[Jouppi 1993] N. P. Jouppi, S. J. E. Wilton. Tradeoffs in two-level on-chip caching[J]. WRL Research Report. 1993.

[Kaxiras 1999] S. Kaxiras, J. R. Goodman. Improving cc-NUMA Performance Using Instruction-Based Prediction[C]. Proc of the 5th IEEE Symposium on High-Performance Computer Architecture. 161-170. 1999.

[KD50 2009] KD-50-I. http://kd50.ustc.edu.cn. 2009.

[Keltcher 2003] C. N. Keltche, K. J. McGrath, A. Ahmed, P. Conway. The AMD Opteron Processor for Multiprocessor Servers[J]. IEEE Micro. 23(2):66-76. 2003.

[Keyes 2003] D. E. Keyes, A science-based case for large-scale simulation[J]. Office of Science U.S. Department of Energy. 2003.

[Kim 2002] C. Kim, D. Burger, S. W. Keckler. An Adaptive, Non-Uniform Cache Structure for Wire-Delay Dominated On-Chip Caches[C]. In Proc of ASPLOS. 211-222. 2002.

[Knoll 1935] M. Knoll. Z. Tech. Phys. 16:467. 1935.

[Kongetira 2005] P. Kongetira, K. Aingaran, K. Olukotun. Niagara: A 32-Way Multi-threaded Sparc Processor[J]. IEEE Micro. 25(2):21-29. 2005.

[Koufaty 1995] D. A. Koufaty, X. Chen, D. K. Poulsen, J. Torrellas. Data Forwarding in Scalable Shared-Memory Multiprocessors[C]. In Proc of the 9th International Conference on Supercomputing. 255-264. 1995.

[Kumar 2005] R. Kumar, V. Zyuban, and D. Tullsen. Interconnections in

multi-core architectures: understanding mechanisms, overheads and scaling[C]. Proc of the 32nd Annual International Symposium on Computer Architecture. 2005.

[Kundu 2006] P.Kundu. On-die interconnects for next generation CMPs[C]. Workshop on On- and Off-Chip Interconnection Networks for Multicore Systems. Dec.2006.

[Kuskin 1998] J.Kuskin, D.Ofelt, M.Heinrich, J.Heinlein, R.Simoni, et.al. The Standford FLASH Multiprocessor[C]. In Proc of the 25th Annual International Symposium on Computer Architecture. 485-496. 1998.

[Lai 1999] An-Chow Lai, B.Falsafi. Memory Sharing Predictor: The Key to a Speculative Coherent DSM[C]. In Proc of the 26th Annual International Symposium on Computer Architecture. 172-183. 1999.

[Landin 1991] A.Landin, E.Hagersten, S.Haridi. Race-free Interconnection Networks and Multiprocessor Consistency[J]. SIGARCH Comput.Archit.News. 19(3):106-115. 1991.

[Larus 1994] J.R.Larus. Compiling for Shared-Memory and Message-Passing Computers[J]. ACM Letters on Programming Languages and Systems. 2(1-4):165-180. 1994.

[Lebeck 1995] A.R.Lebeck, D.A.Wood. Dynamic Self-Invalidation: Reducing Coherence Overhead in Shared-Memory Multiprocessors. In Proc of the 22nd Annual International Symposium on Computer Architecture[C]. 48-59. 1995.

[Lenoski 1990] Lenoski D, Laudon J, Gharachorloo K, et.al. The directory-based cache coherence protocol for the DASH multiprocessors[C]. Proc of the 17th Annual International Symposium on Computer Architecture. 148-158. 1990.

[Li 2005] Li Huiming. Studies on the Simulation of Scanning Electron

Microscopy Image and of Electron Spectroscopy[D]. University of Science and Technology of China. 2005.

[Li 2007] Li Hui, Wu Junming, Chen Guoliang. MPUS: a scalable parallel simulator for RedNeurons parallel computer[C]. Proceedings of the 2nd international conference on Scalable information systems. 2007.

[Lowney 1995] J. R. Lowney. Scanning[J]. 17(281). 1995.

[Ly 1995] T. D. Ly, D. G. Howitt, M. K. Farrens, A. B. Harker. Scanning[J]. 17(220). 1995.

[Martin 2003] M. M. K. Martin, M. D. Hill and D. A. Wood. Token coherence: Decoupling Performance and Correctness[C]. Proc of the 30th Annual International Symposium on Computer Architecture. 182-193. 2003.

[Martin 2005] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, et. al. Multifacet's general execution-driven multiprocessor simulator(gems) toolset[J]. Computer Architecture News(CAN). 92-99. 2005.

[Marty 2005] Michael R. Marty, Jesse D. Bingham, Mark D. Hill, et. al. Improving Multiple-CMP Systems Using Token Coherence[C]. Proceedings of the 11th International Symposium on High-Performance Computer Architecture. Feb, 2005.

[Marty 2008] Michael R. Marty. Cache Coherence Techniques for Multicore Processors[D]. University of Wisconsin-Madison. 2008.

[Maui 2008] Maui - PBS Integration Guide. http://www.clusterresources.com/products/maui/docs/pbsintegration.shtml. 2008.

[McMullen 1952] D. McMullen. PhD thesis. Cambridge Univ. 1952.

[Mukherjee 1998] S.S.Mukherjee, M.D.Hill. Using Prediction to Accelerate Coherence Protocols. In Proc of the 25th Annual International Symposium on Computer Architecture. 179-190. 1998.

[MPICH2 2009] MPICH2. http://www.mcs.anl.gov/research/projects/mpich2/. 2009.

[NASA 2008] http://www.nas.nasa.gov/Resources/Software/npb.html

[Neil 2005] Neil Vachharajani, Matthew Iyer, Chinmay Ashok, et.al. Chip multi-processor scalability for single-threaded applications. ACM SIGARCH Computer Architecture News. 33(4):44-53. 2005.

[Nesbit 2004] K.J.Nesbit, J.E.Smith. Data Cache Prefetching Using a Global History Buffer. In Proc of the 10th International Symposium on High Performance Computer Architecture. 96-105. 2004.

[Papamarcos 1984] Papamarcos M, patel J, A low overhead coherence solution for multiprocessors with private cache memories[C]. Proc of 11th Annual of International Symposium on Coumputer Architecture. 348-354. 1984.

[Park 2001] Park J.S., S.Ravi, A.Gail-Joon. Role-based access control on the web[J]. ACM Transaction on Informantion and System Security. 4(1):37-71. 2001.

[Paul 1988] Paul Sweazey. Shared memory systems on the futurebus[C]. the 33th IEEE Computer Society International Conference. 505-511. 1988.

[Pease 1965] R.F.M.Pease, W.C.Nixon. J.Sci.Instr. 42 (81) . 1965.

[PMON 2007] PMON2000 Boot Firmware, http://www.opsycon.se/pmonmain. 2007.

[Radzimski 1995] Z.J.Radzimski, J.C.Russ. Scanning[J]. 17(276). 1995.

[Rajeev 2005] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. Optimization of Collective Communication Operation in Mpich[J]. the international journal of high performance computing application. 2005.

[Ravi 2000] Sandhu Ravi, F.David, K.Richard. NIST model for role-based access control: towards a unified standard[C]. Proc of the ACM Workshop on Role-Based Access Control. 47-63. 2000.

[Realtek 2007] Realtek Semiconductor Corp. Integrated Gigabit Ethernet Controller (LOM) (MiniPCI) Datasheet, Rev.1.3[Z].2007.

[Reinhardt 1994] S.K.Reinhardt, J.R.Larus, D.A.Wood. Tempest and Typhoon: User-Level Shared Memory[C]. In Proc of the 21st Annual International Symposium on Compouter Architecture. 325-336. 1994.

[Rolf 1999] Rolf Rabenseifner, Automatic MPI counter pro_ling of all users: First results on a CRAY T3E 900-512. In Proceedings of the Message Passing [C]. Interface Developer's and User's Conference 1999 (MPIDC '99). 77-85. 1999.

[Rolf 2007] Rolf Rabenseifner. New optimized MPI reduce algorithm. http://www.hlrs.de/organization/par/services/models/mpi/myreduce.html

[Sarangi 2006] S.R.Sarangi, A.Tiwari, and J.Torrellas. Phoenix: Detecting and Recovering from Permanent Processor Design Bugs with Programmable Hardware[C]. Proceeding of the 39th Annual IEEE/ACM International Symposium on Microarchitecture. 2006.

[Seeger 2003] A.Seeger, C.Fretzagias, R.Taylo. Scanning[J]. 25(264). 2003.

[Shigehisa 1999] Shigehisa Satoh Kazuhiro, Kazuhiro Kusano, Yoshio Tanaka, Motohiko Matsuda, Mitsuhisa Sato. Parallelization of Sparse Cholesky Factorization on an SMP Cluster[C]. In Proc. HPCN Europe 1999, LNCS 1593.

[Shimizu 1992] R. Shimizu, Z. J. Ding. Rep. Prog. Phys. 55(487). 1992.

[Smith 1956] K. C. A. Smith. PhD thesis. Cambridge Univ. 1956.

[Smith 1982] A. J. Smith, Cache memories[J]. ACM Computing Surveys. 1982.

[SONG 2007] SONG Youquan, GAO Xiaopeng, LONG Xiang. Design and Optimization of PCI Ethernet Adapter Driver Program in Embedded System[J]. Computer Engineering. 33(2): 264-266. 2007.

[Strets 2000] R. Strets, S. Dwarkadas, L. Kontothanassis, U. Rencuzogullari, M. L. Scott. The Effect of Network Total Order, Broadcast, and Remote-write Capability on Network-based Shared Memory Computing[C]. In Proc of the 6th International Symposium on High Performance Computer Architecture. 265-276. 2000.

[Stenstrom 1993] P. Stenstrom, M. Brorsson, L. Sandberg. An Adaptive Cache Coherence Protocol Optimized for Migratory Sharing. In Proc of the 20th Annual International Symposium on Computer Architecture. 109-118. 1993.

[Sun 2004] Sun Microsystems. UltraSPARCIII Cu user's manual(version 2.2.1)[M]. http://www.sun.com/processors/manuals/usIIIv2.pdf.

[Tendler 2001] J. Tendler, S. Dodson, S. Fields, H. Le, B. Sinharoy. POWER4 System Microarchitecture. IBM Server Group Whitepaper. 2001.

[Thapar 1993] Thapar M, Delagi B, Flynn M. Linked list Cache Coherence for scalable shared memory multiprocessors[C]. Proc of the 7th International Parallel Processing Symposium. 1993.

[Thilo 1999] Thilo Kielmann, Rutger F. H. Hofman, MAGPIE: MPI's Collective Communication Operations for Clustered Wide A rea Systems[C]. In Proceeding of the Seventh ACM SIGPLAN Symposium on Principles and Practice

of Parallel Programming. 1999.

[Tao 2008] Jie Tao, Marcel Kunze, and Wolfgang Karl. Evaluating the Cache Architecture of Multicore Processors[C]. Proceedings of the 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing. 2008.

[TOP500 2008] TOP500. http://www.top500.org/stats/list/31/procgen. 2008.

[Torque 2008] TORQUE Resource Manager. http://www.clusterresources.com/pages/products/torque-resource-manager.php. 2008.

[Valkealahti 1984] S.Valkealahti, R.M.Nieminen. Appl.Phys[J]. 35(51). 1984.

[Virtutech 2008] Virtutech AB. Simics Full System Simulator. http://www.simics.com/

[Wang 2007] Wang Haixia, Wang Dongsheng, Li Peng, Wang Jinglie, Li Congming. Reducing network traffic of token protocol using sharing relation cache[J]. Tsinghua Science and Technology. 12(6):691-699. 2007.

[Wells 1974] O.C.Wells, A.Boyde, E.Lifshin, A.Rezanowich. Scanning electron microscopy[M]. McGraw-Hill. 1974.

[Wong 2002] T.M.Wong, Gregory R.G., J.Wilkes. My cache or yours? Making storage more exclusive[C]. In Proc of the 2002 USENIX Annual Technical Conference. 2002.

[Woo 1995] Woo.S.C, Ohara.M, Torrie.E, et.al. The SPLASH-2 Programs: Characterization and Methodological Considerations[C]. ISCA' 95. 1995.

[Yan 1998] H.Yan, M.M.EL.Gomati. Scanning[J]. 20(465). 1998.

[Yanof 1989] A.W.Yanof, Ed.Electron-Beam, X-Ray, and Ion-Beam Technology[M]. Bellingham. Washington, D.C. 1989.

[Yuan 2005] Yuan Wei et al. Performance Analysis of NPB Benchmark on Domestic Tera-Scale Cluster Systems[J]. Journal of Computer Research and Development, 2005, Vol.42, No6, pp.1079-1084.

[Zhang 2007] Li Zhang, Chris Jesshope, On-Chip COMA Cache Coherence Protocol for Microgrids of Multithreaded Cores[C]. Euro-Par 2007 Workshops, 38-48. 2007.

[Zhao 2008] Li Zhao, R.Iyer, M.Upton, D.Newell. Towards Hybrid Last Level Caches for Chip-Multiprocessors[M]. ACM Sigarch Computer Architecture News. 2008.

[Zhou 2006] Zhou Ming-De, 64-bit MicroProcessor System Programming[M]. Beijing: TsingHua Press, 2006.

[Zworykin 1942] V.K.Zworykin, J.Hiller, R.L.Snyder. ASTM Bull[M]. 117:15. 1942.

致谢

首先感谢我的导师陈国良院士。是陈老师将我从一个懵懂少年引领入计算机科学的殿堂，教我学习、研究、做人做事。陈老师对我无论是学习上，还是生活上，都倾注了大量关爱。在陈老师的指引下，我有机会参与了许多重大科研项目，使我得以在实践中锻炼自己，凝练自己的学术思想，有机会接触我国计算机领域内的各位专家。从中的学习将令我受益一生。陈老师自己作为我国计算机科学领域内的泰斗，其渊博的学识、极富创意的思想、认真的工作精神、谦逊和善的为人，等等诸多优良品质，值得弟子永远学习，且将永远激励我前进。

吴俊敏副教授于我可算是亦师亦友。其学术上的精深造诣，生活中的幽默风趣，令我有相见恨晚之感。犹记得当年在吴老师的带领下，与几位师兄走南闯北的激昂潇洒。更记得吴老师的车技是如何步步提高的。感谢吴老师对我在苏州撰写博士论文期间给予的指导和帮助。

感谢计算机系系主任顾乃杰教授，您的数学和算法水平令我在课堂上陶然其中；特别是今年“难得”遇上百年一次的金融危机，您对我的去向问题的关心令学生感动。感谢中国科大网络中心主任杨寿保教授，实在很难将您同“杀手”联系起来；您同顾老师一样关心我，替我写推荐信，等等。感谢郑启龙副教授。您的学识和风度，令学生仰慕，而对我的关心和帮助，令我觉得如此温暖。感谢张俊霞老师，在北京一起生活工作的半年多时间里，您的用功刻苦，令我汗颜，从您身上我学习到了很多。感谢计算机系安虹副教授、徐云副教授、孙广中讲师、赵莉莉老师、卢建平老师、李春生老师，等等。

感谢中国科大网络中心张焕杰老师和李会民老师（现在中科院青岛生物能源与过程研究所）。张老师对 Linux 操作系统和计算机网络的精通程度，让我高山仰止，始知天外有天，获益良多。李老师对物理学、操作系统以及高性能计算的研究水平，令人吃惊。张老师和李老师是中国科大卧虎藏龙的最好例证之一。

感谢中国科学院计算技术研究所的孙凝晖研究员、胡伟武研究员等老师。我在计算所的学习工作和生活，得到了你们的很大帮助，让我有机会在我国最先进的高性能计算机研究基地和我国最先进的通用微处理器研究基地学习进步。感谢计算所的王剑研究员、安学军老师、章隆兵副研究员、高翔博士、陈云霁博士，等。

感谢任劲永师兄（现在北京）、李征师兄（现在日本）、李黄海师兄（现在北京）、王文涛师兄（现在上海）、鲍春健师兄（现在深圳）。我们在中国科大华为研

究所共享了一段美好的时光，那里的饭菜真好。当然，不得不提，是你们，让我在很多方面从一个很菜的菜鸟变成了一个也许不是那么菜的菜鸟。

感谢杨晓奇师兄（现在北京）、李凯博士、隋秀峰博士、邹丰富硕士、方维博士、吴超博士，等。难忘我们一起为 KD-50-I 奋斗的日子。那段历程将是我们一生的财富。

感谢中国科大国家高性能计算中心 506 实验室的所有师弟们。他们是吴超博士、方维博士、龙柏博士、廖银博士、苗乾坤博士、齐鸣博士、袁晶博士，等。我们是一个团结友爱、严肃活泼、共同进步的集体。

随着论文工作的结束，我的二十年校园生活也将画上句点，人生就要踏入新的征程。在这里要感谢我的亲人们。特别感谢我的父母、奶奶和大舅爷爷，等。是你们给我生活上的照顾，学习中的鼓励，让我一步步成长，一点点成熟，成为一个能对国家人民有用的人。你们的恩情永记我心。

攻读学位期间发表和录用的论文

[1] Li Hui, Wu Junming, Chen Guoliang, Sui Xiufeng. MPUS: a scalable parallel simulator for RedNeurons parallel computer[C]. Proceedings of the 2nd International Conference on Scalable Information Systems. vol. 304. 2007.

[2] 李晖，李凯，吴俊敏，孙广中，陈国良. KD-50-I 中的无盘启动技术、文件系统架构及 BLAS 库优化[J]. 小型微型计算机系统. 已录用. 文章注册号：0800523.

[3] 李晖，吴俊敏，陈国良. 一种基于不包含策略的多核处理器 Cache 设计[J]. 中国科学技术大学学报. 已投出.

攻读学位期间参与的科研项目

[1] 大规模分布式系统中间件研究(华为公司项目,2004 年 9 月至 2005 年 7 月)。本人负责分布式锁的实现及后期代码优化工作。

[2] 红神超高扩展高密度计算技术研究(国家“863”项目,2005 年 9 月至 2007 年 1 月)。本人参与开发了一个大型并行模拟器,独力开发了一个大型路由模拟器,并参与机器的设计工作,指出了一些缺陷并提出改进方法。

[3] “KD-50-I”项目(教育部“985 工程”平台建设项目)。本人负责系统软件、网络启动技术、BIOS 程序开发、通信优化等工作(2007 年 7 月至 2008 年 1 月)。一期项目结束后,在中科院计算技术研究所使用龙芯 IP 核开发了一个四核处理器 FPGA 仿真平台(2008 年 5 月至 2008 年 12 月)。

[4] 基于多核平台的并行片上多处理器仿真技术研究(国家“863”项目,2008 年 10 月至 2009 年初)。参与前期调研及环境搭建等工作。