分类号_____ 密级_____

UDC _____ 编号_____

中 国 科 学 院
博士学位研究生学位论文

基于 SMP 结点的机群通信系统关键技术的研究

马 捷

指导教师 李国杰 院士 研究员
中国科学院计算技术研究所

申请学位级别 博士学位 学科专业名称 计算机系统结构

论文提交日期 2001 年 4 月 论文答辩日期 2001 年 5 月

学位授予单位 中国科学院计算技术研究所

答辩委员会主席 钱德沛 教授

摘　要

在现代超级计算机和超级服务器体系结构的研究中，机群系统逐渐成为一种主流的结构。而基于 SMP 结点的机群系统以其较高的性能价格比和较好的容错能力成为当前研究的一个热点。作为一个机群系统，其通信协议的性能是影响系统整体性能的一个关键因素。基于 SMP 结点的机群系统由于其结构的变化，其通信协议也与早期的基于单处理器的机群系统不同。因此，研究基于 SMP 结点的机群通信系统方面的问题在设计和实现一个高性能的机群系统中占有重要的地位。

本文首先全面分析了机群体系结构以及机群通信系统的发展现状，并介绍了一些相关的研究。然后，从机群系统互连性能的角度上将机群通信协议抽象为一种层次结构，并基于这种层次结构给出了用于基于 SMP 结点的机群系统的多重通信协议模型以及组成多重通信协议的基于共享内存的通信协议和半用户级通信协议。

随后，本文重点讨论了组成多重通信协议的两个协议层。文章首先讨论了基于共享内存的通信协议的不同实现方式以及提高通信性能的一些关键问题。然后讨论了采用半用户级通信协议对通信系统的安全性和可靠性的影响、对系统的可移植性的支持以及对异构网络环境的支持。文章还进一步采用进程代数的方法对多重通信协议进行形式化描述及验证。

性能是评价一个通信协议的指标之一。本文给出了在曙光 3000 上实现的一个多重通信协议——BCL-3 的评价与性能分析。通过分析多重通信协议的点到点通信性能、LogGP 模型参数以及一组集合操作的性能来综合评价通信系统的性能。实验表明 BCL-3 系统具有良好的性能与可扩展性。

最后，本文分析了限制通信系统性能的一些因素，并给出了多网卡通信协议的模型和一个双网卡通信协议的实例。通过设计多网卡系统来进一步提高通信系统的性能。

关键词 基于 SMP 结点的机群　机群通信系统　用户级通信　可扩展流量控制

Research on Key Issues of Communication System on Cluster of SMP's

Ma Jie (Computer Architecture)
Directed by Professor Li Guojie

Cluster is a widely used architecture in modern super-computer and super-server. Since cluster of SMP's (CLUMPS) has high ratio of performance and cost and high fault tolerance ability, it becomes a hot point in current research fields. Communication system is a key factor that affects the actual performance of CLUMPS. Since the architecture of CLUMPS is different to uni-processor cluster, the communication system is different, too. The research on communication system of CLUMPS plays an important role in the research and development of CLUMPS.

After analyzing all aspects of cluster architecture and cluster communication system, this paper introduced a system hierarchy model by analyzing connection performance. A multi-protocol mechanism, which consists of a shared memory based protocol and a semi-user-level protocol, is discussed under this model.

Then this paper concentrates on the key issues of the two protocols of the multi-protocol system. Some key issues are discussed to improve the performance of shared memory based protocol. The security, reliability, portability and heterogeneous network environment support of the semi-user-level protocol are also discussed. Process Algebra is introduced to describe and verify the protocol.

Performance is a key factor to evaluate a communication protocol. This paper presents the evaluation model and performance analysis of BCL-3, which is the Basic Communication Library of DAWNING-3000. Point to point communication performance, LogGP communication parameters and collective communication performance are used to evaluate the whole system. The tests showed good performance and scalability of the protocol.

Finally, this paper discussed the factors that affect the communication performance. A multi-card protocol model and a double-card protocol prototype are introduced to improve the performance.

Keywords: CLUMPS, Cluster Communication System, User-level Communication, Scalable Flow Control

目 录

第一章 引 言

近年来基于机群(Cluster)的计算技术的发展成熟使得可扩展(Scalable)机群系统在并行计算方面应用得越来越广泛。无论是从工程技术角度还是从经济学的角度来看，使用单处理器(Uniprocessor)工作站建立机群系统的方法逐渐被使用对称多处理器(Symmetric Multiprocessor，SMP)工作站所代替。这种新的基于SMP 结点的机群系统被称为 CLUMPS。SMP 结点的紧偶合共享内存结构为开发高性能应用程序提供空间。但是，CLUMPS 与基于单处理器的机群系统有着较大的差异，现有的技术已不再能很好地适用于 CLUMPS。采用原有技术有时不但不能提高，反而会降低应用程序的性能。因此，需要对应用于 CLUMPS 上的技术进行研究。

本文主要针对设计 CLUMPS 上的机群通信协议方面的问题进行了深入的研究。通过按照机群系统互连性能的不同对 CLUMPS 建立了一个层次结构。并基于这个层次结构提出了多重通信协议的概念，将 CLUMPS 上的机群通信协议划分为用于 SMP 结点内部通信的基于共享内存的通信协议和用于 SMP 结点间的半用户级通信协议。在设计结点间的通信协议时，采用了一种新的半用户级通信协议的模式，提高了系统的安全性和可靠性。

本章首先简单地介绍了计算机体系结构的发展情况以及机群结构的发展情况，然后从机群通信协议的角度介绍了相关的一些研究，并简单地介绍了本文研究的课题背景。最后，介绍了本文的研究目的、主要贡献及文章的内容组织。

1.1 机群系统体系结构的发展

1.1.1 计算机体系结构的分类与发展

从一九四六年世界上第一台数字计算机 ENIAC 出现到现在的五十多年中，电子器件制造技术的进步和计算机体系结构的发展极大地推动了计算机性能的提高。在计算机从电子管发展到 VLSI 的四个阶段中，器件的性能的提高使得

计算机的处理能力飞速增长。目前，器件的性能虽仍在不断发展，但已难以满足计算的需求。研究并行可扩展的计算机体系结构成为提高计算机系统性能的一个重要途径[1][2]。

按照用指令流和数据流进行分类的 Flynn 方法，计算机系统可分为四大类：单指令流单数据流系统(SISD)、单指令流多数据流系统(SIMD)、多指令流单数据流系统(MISD)、多指令流多数据流系统(MIMD)。

SISD 系统是指传统的顺序处理计算机。后三类从广义上来说均属于并行处理系统，其中，SIMD 系统是多个处理单元在同一控制器下工作的计算机系统，系统中各处理单元执行相同的指令流，但作用于不同的数据流上，如向量计算机和阵列计算机均属此类。由于系统在同一控制器的控制下进行工作，各计算部件必须执行相同的指令，就限制了这类系统的应用范围。MISD 系统在概念上说来可以存在，但这种在指令级并行、数据级串行的系统目前尚无实例。

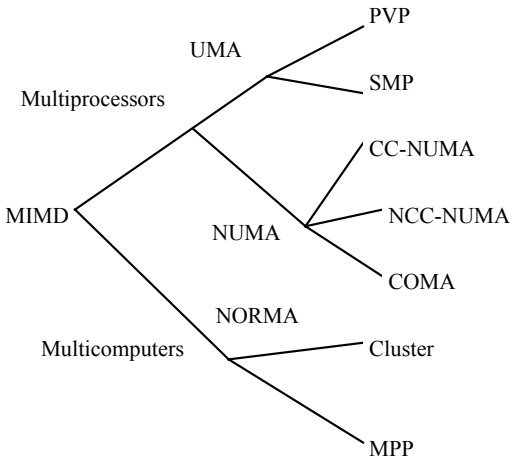

图1.1 MIMD 计算机的分类

MIMD 系统则是多个处理机各自执行不同的指令流，并分别作用于不同的数据流上。这类系统是实现作业、任务、指令和数据各个级别全面并行的理想结构，是当今并行处理的主流系统。根据各处理机间耦合程度的不同，MIMD 系统又可分为共享存储(Shared Memory)的多处理机系统(Multiprocessor)和分布存储(Distributed Memory)的多计算机系统(Multi-computer)两类。前者各处理机通过总线、开关阵列或多级网络等方式共享一个公共的存储器，分散在各个处理器中的应用程序的各部分之间可以方便地通过共享的存储变量来交换数据并

实现各种互斥和同步操作。由于受到存储器带宽的限制，这类系统的可扩展性(Scalability)较差，一般很难达到较大的规模和非常高的性能。因此，多计算机系统逐渐成为计算机体系结构研究的一个主要方向。

最初的多计算机系统多属于大规模并行计算机(Massively Parallel Processing，MPP)类型。这类系统结点一般是高性能专用处理部件。每个结点独立不成构成一个计算机。MPP 的代表机型有 Cray 的 T3D/T3E、Thinking Machine 的 CM-2/CM-5、Intel 的 Paragon XP/S 和 Fujitsu 的 VPP500 等，国家智能计算机研究开发中心研制的曙光-1000 并行系统也采用了 MPP 这种结构，在早期，MPP 在超级计算机市场上取得了很大的成功。由于采用了专用部件，整个系统的成本较高，同时也很难有一个统一的标准。随着多计算机规模的不断增大，计算机系统的成本(性能价格比)成为在设计计算机系统时需要考虑的一个重要因素。由于批量生产可以降低成本，使用商品化部件作为结点的机群系统能够有效地提高系统的性能价格比。机群系统逐渐成为设计超级计算机和超级服务器的一种主流的结构。

1.1.2 机群系统

机群系统是利用高性能通信网络将一组计算机(结点)，按某种结构连接起来，并在并行程序设计及可视化人机交互集成开发环境支持下，统一调度，协调处理，实现高效并行处理的系统。一般地，每个计算机结点是一台高性能工作站或高档 PC 服务器。除了满足由交互用户单独地使用每个结点的任务外，所有的机群结点必须能一起工作，如同一个单一集成的计算资源。机群中每一个结点是一台完整的计算机，它有自己的处理器、高速缓存、磁盘以及 I/O 适配器，并具有一个完整、标准的操作系统。机群系统采用单一映象(Single System Image，SSI)技术实现单一集成计算资源的概念，使得机群系统更易于使用和管理。结点间的连接可采用商品化的网络(如以太网等)或其它高速网络(如 Myrinet 等)[3]。

当前，SMP 结点逐渐成为组成机群系统的主流设备。无论是在高端的服务器中，还是在低端的 PC 机群上，采用 SMP 结点能够提供更高的性能价格比。越来越多的研究都集中在基于 SMP 的机群系统(CLUMPS)[4]。

CLUMPS 是近二十年来高性能计算机体系结构发展的一个潮流。不论是超

级计算机、大型机，还是个人微机系统，其体系结构的发展都逐渐汇集到 CLUMPS 结构。图 1.2给出了这样一个发展的趋势。在初始的阶段，不同体系结构的系统按其各自的发展趋势向多处理器结构发展。大型机逐渐发展为基于 SMP 的服务器；向量机逐渐发展为 MPP 结构，进而发展为 CC-NUMA 结构的服务器和超级计算机；微机也向多处理器协同工作(处理器池)的方向发展，逐渐演变为 NOW 结构。随着处理器的性能的不断提高，系统的整体性能也随之提高。

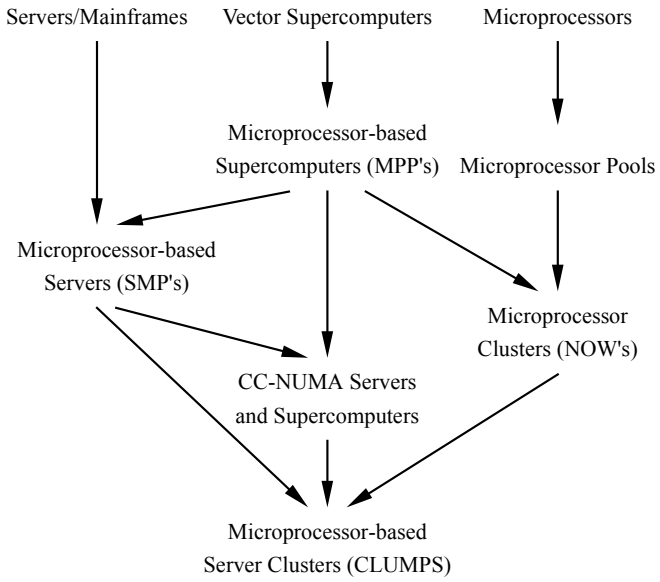

图1.2 体系结构的发展集中于 CLUMPS 结构

之后，器件的发展不再能满足系统性能提高的要求，大规模的处理器池或处理器阵列的扩展性和效率也制约着其性能的提高。因此，NOW、SMP 服务器和 CC-NUMA 系统逐渐汇集到 CLUMPS 结构。CLUMPS 优于 SMP 服务器和 CC-NUMA 结构之处在于 SMP 服务器与 CC-NUMA 结构的单一系统结构能够高效地支持应用，但是同时也会出现单一故障点的问题。虽然一些机器提供了系统分区(Partition)的功能，但是 Cache 一致性的共享内存使得系统中的故障会影响整个系统的运行。而 CLUMPS 则能较好地解决这一问题。CLUMPS 能够以较低的价格提供一个较优的性能和较好的容错能力。对比 NOW 结构，由于采用具有多处理器 SMP 结点，CLUMPS 能够以较少的结点数提供较高的性能。这能够有效地降低系统的管理开销。同时，采用 SMP 结点也能提高 SMP 结点

内部处理器间的通信性能，从而使整个系统的性能提高[7]。

1.2 机群通信协议的发展

对于并行处理系统，人们希望要有较高的结点运算速度，系统的加速比性能接近线性增长，并行应用程序的开发要高效、方便。目前，机群系统大多采用商用高性能工作站或高档 PC 服务器，结点的运算速度问题不是很突出，因而主要的研究方面是在提高系统的并行效率和使系统更方便使用上。在这里，建立一个高效的通信子系统是一个关键的问题。

通信子系统是并行计算机系统的重要组成部分，它完成系统中各结点之间的数据传递功能，其性能的好坏直接影响到并行计算的加速比和效率。这是因为并行计算时间是由各结点计算时间和结点间数据通信时间两部分组成，如果通信时间所占的比例过大，则必然使得并行计算的加速比下降，整个系统的效率也不会高[3]。高效的机群通信近几年来始终是人们研究的一个热点，可以说高效的机群通信既是推动机群系统发展到今天的一个主要动力，同时也是其继续发展的一个重要基础。

传统机群系统的通信是建立在普通低速的局域网互连和 TCP/IP 协议的基础上。其通信性能低下，只适于运行串行程序和通信量不大的粗粒度并行程序。随着高性能通信网络和通信软件的出现以及它们在机群机间通信中的运用，机群系统才真正成为一种较为理想的并行处理平台并得到了较大的发展。其中具有代表性的通信系统有 VMMC、U-Net、Active Messages、Fast Sockets、SP2 MPL 以及最近的 GM、BIP、PM 等。

最初设计的通信系统相互之间差异较大，各个系统为了提高性能，在软、硬件方面都进行了很多改进。例如：美国 Princeton 大学计算机系的 Kai Li 教授等设计 Shrimp 系统为了消除传统通信机制和网络接口对通信性能的制约，使用了 VMMC[10][11][12][13]通信机制，并为支持该通信机制的实现研制了专用的网络接口。同时，它的实现对操作系统有较大的依赖。这一时期的研究主要集中于对协议结构的设计上。

随着对机群通信系统研究的深入，各个通信系统的协议都集中于基于消息传递的模式之上。其中较有代表性的是美国 Cornell 大学计算机科学系的 Eicken 等人在 1995 年设计的 U-net[14][15][16]的通信系统。U-net 的主要设计目标就是在不

削弱保护机制的前提下，通过允许多个用户进程对网络接口硬件的直接访问，取得较小的通信延时和较大的通信带宽，并增加在其上实现高层通信协议的灵活性。之后的 GM、BIP 等系统，以及在曙光天潮系列机(曙光 2000-I、曙光 2000-II 和曙光 3000 等)上所使用的 BCL 系统，都在一定程度上采用了这种模式。

美国 California 大学 Berkeley 分校的 Eicken 等人在 1992 年提出的 Active Messages[17][18]也是一种有代表性的系统。它可以看成是一种轻量的远程过程调用(RPC)，其基本设计思想就是让消息的发送方预先指定好接收方用于处理该消息的函数(Handler)，消息到达接收方时，这个预先指定的消息处理函数被调用以处理到达的消息。与传统的 Send/Receive 通信机制相比，Active Message 在实现上的最大改进在于它不需要任何缓存。在通信的接收方，消息的数据可以直接送入用户程序预先为它分配好了的接收内存块，或者到达的是一个消息处理函数可以直接处理并立即做出应答的小消息。因此，Active Message 在消息接收方无须缓存。传统的消息传递软件中通信启动开销通常很大，为了降低通信的平均开销，往往采用大消息包，以便启动一次通信能传送尽可能多的数据。但对于 Active Message 来说，通信的启动软件开销要小得多，通常可以采用比较小的消息包。因此对于消息发送方，通信硬件提供的缓存对 Active Message 的小消息包来说就已经足够了，无须再由软件来提供缓存。Active Message 非常自然地避免了传统通信机制存在的由于复杂的缓冲区管理、慢速的内存数据拷贝和流量控制造成的大量软件开销，并且也没有程序运行时消息发送操作和接收操作时间上的不匹配问题。

目前在机群通信协议的研究方面出现了许多基于 Myrinet[21][22][23][24][25][26]网络设备硬件的系统，其中较有代表性的有 Myricom 公司自己推出的 GM 协议、法国的 LHPC 实验室研究的 BIP 协议和智能中心研制的 BCL 协议。

GM[27][28]是一个商品化的通信协议，它与 U-net 类似，采用基于端口(Port)的消息传递方式。它提供了阻塞式和非阻塞式两类消息传递机制，并为上层实现 TCP/IP 协议提供了特殊支持。

BIP[29][30][31][32]是一种精简的低层次通信协议，其目标是提供一个高效的硬件访问方式和零存储拷贝通信。由于 BIP 只支持很简单的通信功能，因此它不能直接提供给最终用户程序员使用，BIP 仅为高层的通信软件(如 MPI 等)提供一个具有较好功能性能比的开发环境。BIP 的 API 是一个经典的消息传递界面。它同时提供阻塞式和非阻塞式通信接口；其通信的可靠性由网络硬件保证；提

供有序的消息传递；提供错误检测功能，但不提供错误恢复功能。

BCL[34]是曙光天潮系列机上使用的通信协议。BCL 通信协议是一种基于消息传递的机群通信协议。它通过结点间相互收发消息来完成机群间的通信、同步。BCL 支持阻塞和非阻塞的通信语义，提供了完善的流量控制、错误纠正等功能，具有很高的可靠性和容错能力。

VIA[35](Virtual Interface Architecture)是由 Intel、Compaq 和 Microsoft 三家公司共同推出的一个用户级通信界面，它力图为机群上的用户进程提供一个高性能的、从内存到内存的网络传输标准，目前已经得到超过一百三十家业界领先公司的支持。VIA 大量借鉴了 AM、U-Net、FM 和 VMMC 等用户级通信技术的思想，并在设计中使之更加完善。

当机群系统逐渐发展到以 SMP 结点作为主要计算设备时，仅考虑机间通信的通信协议已不再适应当前的系统配置。为此，Active Messages、BIP 和 BCL 相继展开了相关的研究。Active Messages 推出 AM-II[40][41]，BIP 推出 BIP-SMP[42]，BCL 推出 BCL-3[44][45][46]。

综上所述，基于单 CPU 结点的机群通信协议已发展得较为成熟，目前常见的几种协议结构都较为类似。VIA 的提出又更进一步统一了这类机群通信协议的结构。随着 SMP 结点的出现，基于 SMP 结点的机群通信协议成为高效通信系统研究的一个热点。研究机内与机间通信问题，提高通信的速度和效率，对于提高整个系统的性能具有非常重要的意义。

1.3 课题背景

本文的研究工作是结合国家智能中心承担的“国家 863 计划”重点课题：“高性能计算机及应用系统——曙光 3000 系统”(课题编号为 863-306-ZD01-01)进行的。

超级服务器是一类覆盖面非常广的高性能计算机系统。它的基本特点是：性能高，规模大、服务面广。浮点运算速度在每秒数十亿次到数千亿次，每分钟可接待数千到数万个查询访问，内存容量在数个 GB 和 TB 之间，处理机个数为几个到上千个。一台超级服务器是一个巨型计算机系统；可广泛用于大规模科学工程计算、商务计算(事务处理和决策支持等)及网络应用(网络服务、存储服务、媒体服务及信息服务等)。为了能提高机群的性能，研究高性能的机群

通信协议就成为设计实现整个系统的一个重要的问题。通信系统的性能指标关系到整个系统的性能。

1.4 本文的贡献及内容组织

1.4.1 本文的主要工作及贡献

本文主要研究基于 SMP 结点的机群通信系统的关键技术。重点在于提高通信系统的性能和研究结合 SMP 结点特点的通信系统。结合曙光 3000 超级服务器通信系统 BCL-3 的设计和实现工作，对机群通信系统的体系结构、系统的安全性、可靠性、系统的可扩展性、高可用性以及通信系统形式化表示与验证、系统在复杂的通信模式下的通信性能进行了研究。

本文的主要贡献有：

本文从互连性能的角度将基于 SMP 结点的机群系统抽象为一个层次结构模型。这种层次结构模型对于设计实现机群通信系统、优化集合操作性能以及扩展多网卡互连结构都有一定的指导作用。

本文给出了一个基于 SMP 结点的机群系统的多重通信协议结构。在机群系统的层次结构模型的基础上设计的多重通信协议系统能够充分地利用 SMP 结点的特性，对于不同互连结构的计算单元间采用不同的通信模式，从而有效地提高系统通信硬件的利用率，提高系统的总体性能。

本文提出了半用户级通信的概念，并实现了半用户级的通信协议。半用户级通信综合了用户级通信和核心级通信的特点。它采用核心级通信的方式来保护系统的重要数据与重要操作，从而保证了系统的安全性、可靠性。同时，半用户级通信沿用了用户级通信的思路，通过减少进入核心的操作、减少通信关键路径上的开销来降低通信延迟。半用户级通信协议在不影响(或较少影响)通信性能的前提下提供了保证系统安全性、可靠性的机制。采用半用户级通信模式还可以有效地提高系统的可移植性，并对异构网络环境提供支持。

本文采用进程代数的形式化方法描述了通信过程，并采用相应的验证工具对通信协议进行验证。通过形式化的方法描述通信协议对协议的设计及实现有一定的帮助。

本文给出了一组通信协议性能测试的方法。通过测试点到点的通信性能来

确定通信系统的基本性能指标；采用 LogGP 模型来分析影响通信性能的因素；通过测试一组集合操作性能来评价机群系统在复杂通信模式下的性能，从而给出一般应用程序在机群系统上运行性能的评价。

本文提出了细粒度多网卡通信协议的模型。在基于 SMP 结点的机群系统的层次结构模型的基础上细化了通信的层次，设计了双网卡通信协议。并对多网卡通信协议设计实现中的关键问题进行了讨论。

最后，在本文的理论指导下，设计和实现了用于曙光 3000 的机群通信协议 BCL-3。该协议从实践的角度上验证了多重通信协议的设计思路，并且在系统的安全性、可靠性、可扩展性和高可用性方面给出了实践的证明。

1.4.2 内容组织

本文主要研究了基于 SMP 结点的机群通信协议的关键问题。论文的第一章简要地介绍了机群系统及机群通信系统的发展状况。同时，对论文的研究背景、主要工作及论文的内容组织给出了简要的介绍。

在第二章中，给出了基于 SMP 结点的机群系统的层次结构模型。这一章中通过分析层次结构模型中的各个层次，给出了一个基于 SMP 结点的机群系统的多重通信协议结构。并对多重通信协议中的各个协议子层进行了讨论，给出了设计中的关键问题。同时，本章还简要介绍了用于多重通信协议设计和实验的基础——BCL 系统，包括前期设计实现的 BCL-1、BCL-2 系统以及采用多重通信协议思想设计的 BCL-3 系统。

在第三章中，给出了多重通信协议的详细设计与一些关键问题的讨论。本章首先讨论了实现多重通信协议应注意的一些问题，然后分别讨论了组成多重通信协议的两个部分——基于共享内存的通信协议和半用户级通信协议。基于共享内存的通信协议结合了 SMP 结点内部通信的特点，提供了一个高效的机内通信协议。半用户级通信协议结合了用户级通信协议和核心级通信协议的特点，在不影响(或较少影响)通信性能的前提下提高了系统安全性、可靠性，同时增强了系统的可移植性和对异构网络环境的支持。

在第四章中采用了进程代数的方法，用 PROMELA 语言描述了多重通信协议，并用协议验证工具 Spin 对通信协议进行了验证。这一章中，我们首先对进程代数、PROMELA 语言和协议验证工具 Spin 进行了简单介绍，然后采用形式

化的方式描述了协议系统。

在第五章中我们给出了多重通信协议 BCL-3 在曙光 3000 平台上的性能数据及分析。这一章中首先给出了通信协议的评价方法，并从点到点通信性能、LogGP 模型参数和集合操作性能等几个方面综合评价了通信协议的整体性能指标。

在第六章中给出了多网卡通信协议的模型。这一章中首先讨论了限制通信性能的主要因素，然后扩展了在第二章中提出的基于 SMP 结点的机群系统的层次结构模型，给出了多网卡通信协议的结构模型。最后，讨论了设计实现多网卡通信协议的一些关键问题，并给出了一个双网卡通信协议的设计和原型系统的性能指标。

最后，在第七章中总结了全文，并对进一步的工作进行了展望和讨论。

第二章 基于 SMP 结点的机群通信结构

本章列举了在设计实现一个多重通信协议(Multi-protocol)时的一些关键问题。文章首先简单介绍了传统的通信系统的结构，指出了基于 SMP 结点的机群系统与传统机群系统的不同，给出了多重通信协议的一个层次结构模型。然后，文章简要介绍了 BCL 通信协议的相关情况。BCL 是本文中实现多重通信协议的一个基础。最后，在多重通信协议的层次结构模型下，文章列举了设计多重通信协议的一些关键问题，并在后续的章节中讨论了实现一个多重通信协议的一些细节问题。

2.1 通信结构概述

传统的机群结构如图 2.1所示。每个结点有一个处理器，结点间通过高速互连网络连接起来。由于结点上只有一个处理器，一般情况下所运行的应用程序在每个结点上也只加载一个进程。这样，机群通信系统主要需考虑结点间的通信。机群各结点间通过高速互连网络连接在一起，结点间的通信开销相同(有的互连网络结点间的通信开销不尽相同，但在同一数量级上)。这样，应用程序的各个进程之间从软件的角度上可以看作是对称的。进程在进行通信时，无须考虑进程所处的结点。

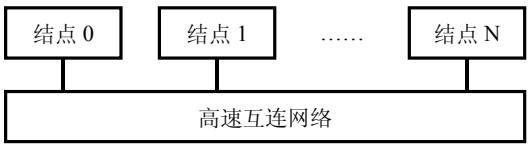

图2.1 传统机群结构示意

当使用 SMP 结点组成机群(如图 2.2所示)时，高速互连网络仍可以看作一个对称的结构，即结点间的通信开销对称。由于每一个 SMP 结点中包含了多个处理器，应用程序可以在每个处理器上加载多个进程。这样，应用程序中各个进程之间不再对称，即每个进程在与其它进程进行通信时需要考虑所处的物理

位置。

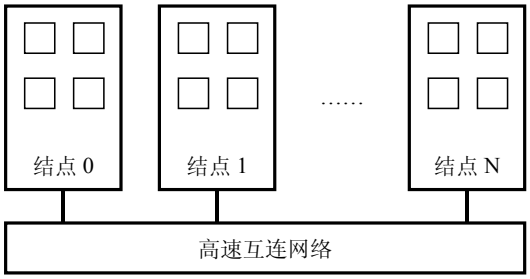

图2.2 基于 SMP 结点的机群结构示意

不同结点间进行通信需要通过互连网络，同一结点内部不同处理器间的通信可以直接通过共享内存进行。这样，结点内部的通信开销比结点间的通信开销小。在早期的通信协议中，如 BCL-1 及 BCL-2，虽然支持同一结点内多进程的通信模式，但对于结点内通信与结点间通信没有进行区分。所有进程间的通信，不区分其拓扑结构统一通过互连网络进行。这样设计的优点在于保证了整个协议接口的一致性，为机群系统提供了一致的通信界面。但是，采用这种方式无疑降低了系统的效率，无法充分利用系统所提供的性能。

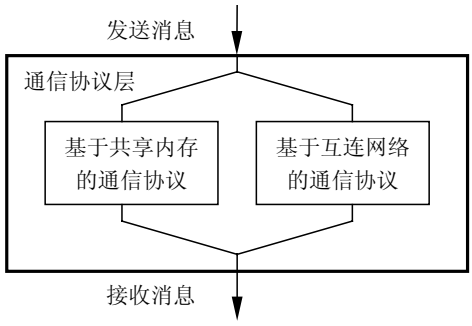

图2.3 多重通信协议示意

为了能够充分利用系统的硬件性能，我们可以根据处理器间的互连性能将机群系统看作一个层次结构。同一 SMP 结点内部看作一个高速互连的结构；不同结点间看作一个相对低速的互连结构。在设计机群通信协议时采用多重通信协议(Multi-protocol Layer)。如图 2.3中所示，同一结点内的进程间传递消息通过基于共享内存的通信协议，不同结点间的进程间传递消息通过基于互连网络的通信协议。通过采用这样一个二层结构，可以有效地描述基于 SMP 结点的机

群通信系统的结构。

多重通信协议的不同协议层采用相同的应用程序接口，对上层软件提供统一的界面。程序员将通信协议层看作一个黑盒子，开发应用程序时无须考虑通信进程间相对的位置关系。在进行通信时，通信协议层自动地将不同相对位置间所需传递的消息分发给不同的协议完成。这样，机群物理上的拓扑结构对应用程序员透明。

本文中所实现的多重通信协议系统是在已有的 BCL-1 及 BCL-2 协议的基础上完成的。扩展后的协议称为 BCL-3。它在 SMP 结点间的通信(远程消息)实现中仍采用类似于 BCL-2 中的模式，由消息的发送方将消息发送请求写入网络接口卡(Network Interface Card, NIC)，由 NIC 发送消息。同时，对通信过程及系统的流量控制等方面做了进一步的改进。在 SMP 结点内部的通信(本地消息)实现中扩展了原有的协议，采用了基于共享内存的通信方式。本地消息不再通过 NIC 传递，而直接由消息的发送方将消息写入共享内存中，然后由消息的接收方读出。在消息的接收方，BCL-3 采用相同的应用程序接口来接收消息。接收消息时将查询从网络和共享内存中传递的消息。

2.2 BCL 通信协议简介

BCL 通信协议是由国家智能计算机研究开发中心开发的高效机群通信协议。主要应用于曙光天潮系列机。它同时支持智能中心自行研制的通信硬件 PMI-8/16 和 Myricom 公司生产的通信硬件 Myrinet。图 2.4给出了在曙光 3000 上的通信软件协议层次，处于最底层的是 BCL-3 协议。BCL-3 除支持上层的通信协议 PVM/MPI 外，还支持 TCP/IP 和分布式共享存储(DSM)及机群文件系统等应用。同时，也支持用户应用程序直接使用其 API。

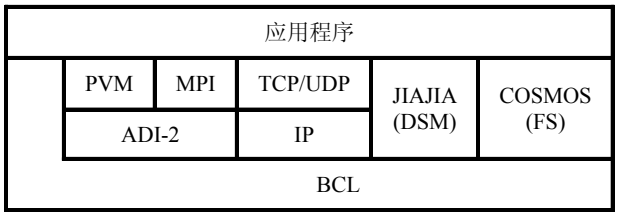

图2.4 曙光 3000 上通信软件协议层次

BCL 通信协议是一种基于消息传递的机群通信协议。它通过结点间相互收发消息来完成机群间的通信、同步。BCL 支持阻塞和非阻塞的通信语义，提供了完善的流量控制、错误纠正等功能，具有很高的可靠性和容错能力。它与其它实验性的通信协议的不同之处在于，BCL 已成功地应用于曙光天潮系列机，并已投入正常运转。

BCL 协议中通信的基本单位是端口(Port)。每个参与通信的进程首先需要注册一个通信端口，端口号与进程所在结点(Node)号在通信过程中唯一标识该进程。所有的消息通信都需要消息的发起方(消息发送进程)指出消息的接收方的结点号与端口号。

随着曙光天潮系列机的研制，BCL 通信协议从用于曙光 2000-I 上的 BCL-1、用于曙光 2000-II 上的 BCL-2 发展到用于曙光 3000 上的 BCL-3。在这里分别作简单介绍。

2.2.1 BCL-1

BCL-1 是用于曙光 2000-I 的通信协议。它的消息传递过程分为三个阶段：消息的发送、消息在网络上的传输和消息的接收。其中，消息发送是指消息由用户空间(User Space)，经核心缓冲(Kernel Buffer)，传入网络接口上的输出消息队列(OFIFO)的过程。消息在网络上的传输是指消息由网络接口上的输出消息队列(OFIFO)，经系统高速通信网，传至接收节点网络接口上的输入消息队列(IFIFO)的过程。消息的接收指接收方将消息由网络接口上的输入消息队列(IFIFO)，经核心缓冲(Kernel Buffer)，传入用户空间(User Space)的过程。

BCL-1 使用存储映射的方法，将核心中的 DMA 空间映射到用户空间，发送或接收原语可直接操纵，减少进出核心的次数，实现用户级消息传递。BCL-1 细化了消息通信的处理过程，区分小消息和大消息，小消息可以用直接 I/O 方式读写网卡，因为 DMA 启动开销大，这样有利于减小软件延迟；大消息则直接传递到用户缓冲区，减少数据拷贝，提高带宽。BCL-1 还采用了动态流量控制技术、支持多任务，并有一定的容错能力。此外，BCL-1 中还使用滑动窗口协议实现了消息应答、出错和超时重发的功能，支持可靠的消息传递。

但是，在 BCL-1 协议的数据通路需要经过二次系统缓冲区。这两次内存拷贝增加了通信的开销。同时，内存拷贝的带宽也限制了通信系统的理论带宽上

限。为此，我们加入了内存零拷贝的技术，设计了 BCL-2 协议。

2.2.2 BCL-2

BCL-2 是用于曙光 2000-II 的通信协议。为避免消息的内存拷贝开销，有效地提高通信带宽，BCL-2 在消息传递过程的设计上与 BCL-1 有所不同。其主要区别就在于 BCL-2 实现了消息传递路径上的内存零拷贝(Zero-copy)，很好地避免了消息传递中的系统核心缓存过程。BCL-2 的消息传递过程分为三个阶段：消息的发送、消息在网络上的传输和消息的接收。其中，消息发送是指消息由用户空间(User Space)传入网络接口上的输出消息队列(OFIFO)的过程。消息在网络上的传输是指消息由网络接口上的输出消息队列(OFIFO)经由网络传至接收节点网络接口上的输入消息队列(IFIFO)的过程。消息的接收指接收方将消息由网络接口上的输入消息队列(IFIFO)传入用户空间(User Space)的过程。与 BCL-1 相比，BCL-2 减少了消息传递通路上的两次内存拷贝。

BCL-2 是基于通信协处理器的底层通信协议。由于协处理器的参与，通信是完全用户级的，消息传递过程不用进核心，大部分工作由协处理器完成，可以做到通信与计算的重叠，提高系统效率。BCL-2 支持真正的消息传递零拷贝，对于大消息将用户虚存锁在内存中，实现用户缓冲区之间的直接 DMA。另外，还提供了 BCL 缓冲区管理的 API，开放 BCL 缓冲区，从而减少上层软件 PVM、MPI 内存拷贝的次数。因为一些必要的打包拆包操作可以直接在系统缓冲区中进行了。BCL-2 采用了页表管理的优化技术。优化页表管理的目的是减少昂贵的内存锁定与解锁(pin/unpin)操作，提供虚地址和实地址之间转换的缓存(Pin-down Cache)，同时协调系统的总体性能，控制占用物理内存的总量。BCL-2 采用了消息接收重定向技术。在无消息预收请求的情况下，系统会将消息暂存于系统缓冲区。在系统将消息存入系统缓冲区的过程中，如果用户进程发出接收请求，并生成页表，则系统可以将未接收的部分及时进行重新的定向操作，直接接收到用户缓冲区，从而减少内存拷贝次数。

此外，BCL-2 协议仍旧保持了 BCL-1 协议中所具有的流量控制、可靠传输和容错能力，提供了阻塞和非阻塞式的消息通信原语。

2.2.3 BCL-3

BCL-3 是用于曙光 3000 的通信协议。它继承了 BCL-2 内存零拷贝与优化页表管理的技术，并更进一步地优化了主机处理器与通信协处理器之前的负载，提高了通信的性能。BCL-3 的消息传递过程分为三个阶段：消息的发送、消息在网络上的传输和消息的接收。其中，消息发送是指消息由用户空间(User Space)传入网络接口上的输出消息队列(OFIFO)的过程。消息在网络上的传输是指消息由网络接口上的输出消息队列(OFIFO)经由网络传至接收节点网络接口上的输入消息队列(IFIFO)的过程。消息的接收指接收方将消息由网络接口上的输入消息队列(IFIFO)传入用户空间(User Space)的过程。与 BCL-2 的不同之处在于 BCL-3 消息传递的三个阶段的工作全部交由通信协处理器完成，有效地减少了主机处理器的负载，降低了通信过程的开销。

BCL-3 采用了多重协议结构，实现了对 SMP 结点的支持。它对 SMP 结点内部进程间的通信采用基于共享内存的消息通信协议，对于 SMP 结点间进程的通信采用基于互连网络的通信协议，从而提高了系统的整体效率。BCL-3 协议有效地平衡了协议的功能与性能，使得它在满足高层通信软件和其它系统软件对通信的需求的同时，将通信硬件的性能尽可能多地提供给上层应用。为了更好地支持上层应用，BCL-3 除提供点到点消息传递通信机制，支持 PVM/MPI 的实现外，还提供了远程内存访问(RMA)操作和通信事件处理中断触发方式，有效地支持了机群文件系统 COSMOS、分布式共享存储软件 JIAJIA 和 IP 协议的实现。其中通信事件处理的中断触发方式通过标准 UNIX 系统调用(select)提供给用户，这极大地方便了基于 Socket 通信的应用程序到 BCL-3 的移植。BCL-3 实现了可扩展流量控制、基于超时重发的可靠数据传输，以及故障检测、结点分离/重入和异常处理等功能。BCL-3 采用了半用户级通信协议(Semi-user-level Protocol)实现结点间的消息传递。在这种协议不再强求完全用户级通信过程，而采用用户级通信的思想，尽量减少通信开销(进入核心的次数)。同时，协议还吸取了核心级通信的优点，对重要数据结构及操作过程进行保护，保证了系统的安全性和可靠性。此外，采用了半用户级通信协议还能够从软件层次的角度屏蔽硬件影响，实现对异构网络环境(Heterogeneous Network Environment)的支持，提高系统的可移植性。BCL-3 实现了可扩展的流量控制和超时重发机制，并采用了更有效的检错纠错技术，保证了数据的可靠传输。针对 SMP 机群结构

的特点对机内通信和 PVM/MPI 提供的集合通信进行优化。

BCL-3 同时专门抽象出 ADI(Abstract Device Interface)层方便地支持 PVM、MPI，解决了底层协议在 API 完整性、性能、上层软件移植方便上的矛盾。

2.3 多重通信协议

在基于 SMP 的机群系统中，针对机内通信与机间通信的不同，需要采用多重通信协议。本节简单介绍组成多重通信协议的两个部分，并给出在设计实现多重通信协议时所需要考虑的一些关键问题。

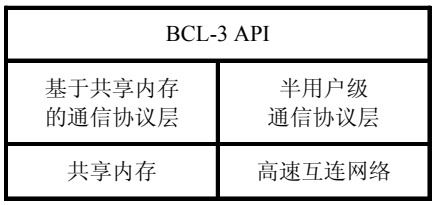

图2.5 BCL-3 多重通信协议

图 2.5给出了 BCL-3 多重通信协议的结构。图中左半部分表示通过基于共享内存的通信协议层进行同一 SMP 结点内部不同进程间的消息传递，右半部分表示通过半用户级通信协议层使用高速互连网络进行结点间的消息传递。

2.3.1 基于共享内存的通信协议

在实现 SMP 结点内通信时，可以采用不同的通信机制。图 2.6给出了两种常见的实现机内通信的方法。

第一种方法是采用共享内存的方法。消息发送方(进程 A)将待发送的消息送入共享内存中，并通过同步机制通知消息接收方(进程 B)接收消息。进程 B 发现消息到达后，从共享内存中取出相应的消息。采用这种方法进行机内进程间通信，不需要进入核心空间，可以实现完全的用户级通信。第二种方法是采用直接在不同进程的发送和接收数据缓冲区间进行拷贝。这样，进程间的通信只需要消息发送方(进程 A)将消息数据送入消息接收方(进程 B)的进程空间，无须进程 B 做任何操作。同时，采用这种方法还可以减少一次数据的拷贝。

对比两种通信方法，采用基于共享内存的通信协议可以减少通信过程中因

进入核心空间而带来的开销，从而降低通信延迟。采用直接访问接收方进程用户空间的方法可以减少一次内存拷贝，从而提高通信带宽。在本文中，由于所实现的 BCL-3 通信协议需要提供较高的安全性和可靠性，因此，采用基于共享内存的通信协议。为了降低因两次内存拷贝造成的开销，在 BCL-3 的基于共享内存的通信协议中采用了流水线等技术。详细情况将在下一章中进行讨论。

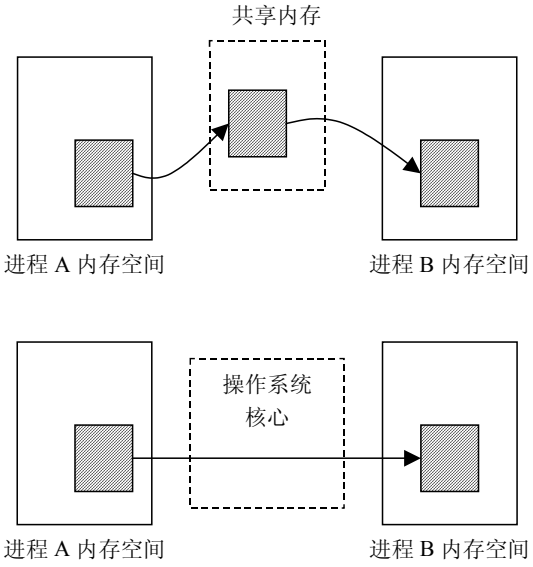

图2.6 实现 SMP 结点内部通信的几种方法

2.3.2 半用户级通信协议

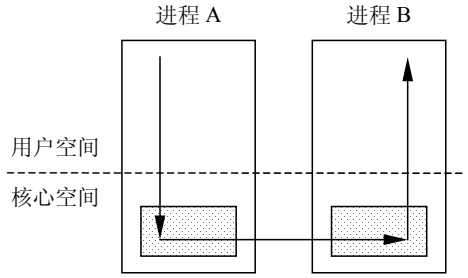

图2.7 核心级通信

目前各种通信协议普遍采用两种通信结构，即核心级通信和用户级通信。

核心级通信(图 2.7)需要通信双方在通信过程中进入操作系统核心空间。通信协议主要在核心中实现，对网络接口卡(NIC)的访问只能在核心中进行，NIC 与主机间控制的传递主要通过中断来实现。传统的通信协议，如 TCP/IP，主要采用核心级通信结构实现，该结构在通信关键路径上引入了两次进出核心和中断处理的开销，对通信延迟有较大的影响。

在用户级通信(图 2.8)中，通信双方无须进入操作系统核心空间。它们在用户空间中直接访问 NIC。通信事件处理的触发多采用查询方式而非中断方式。这样，用户级通信消除了通信关键路径上的大量开销，降低了通信延迟。由于能提高通信性能，因此近年来在机群系统中主要采用用户级通信实现高效的机间通信协议。

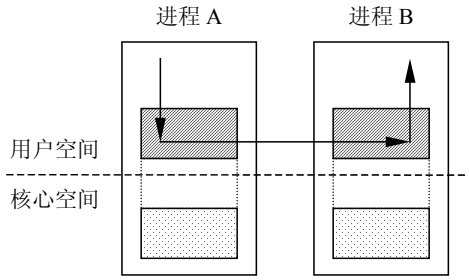

图2.8 用户级通信

但是，采用用户级通信协议也有其缺点。首先，采用用户级通信协议时，通信过程中的所有操作均在用户进程空间进行。这样，就需要将系统(核心)的一些数据结构映射到用户空间，在用户进程所在的安全级别下进行操作。当用户程序出错而发生误操作，或有恶意用户进行破坏时，系统容易被破坏。由于系统数据结构中包含了与本进程(或并行任务)无关的其它进程(或并行任务)的相关信息，某一用户(并行任务)的失误将会影响到其它用户(并行任务)的执行。这样就很难保证系统的安全性和可靠性，也无法保证并行任务间的相互独立性。

为了能保证系统安全、可靠以及并行任务间相互独立，又能降低系统开销，本文提出了一种半用户级通信的概念。即采用进入核心的方式来保证系统的安全、可靠，隔离不同并行任务的数据结构，同时采用用户级通信的思路来减少通信关键路径上的开销。

如图 2.9所示，阴影部分表示通信关键路径上所需要操作的数据结构。在半

用户级通信协议中，系统将通信关键路径上所需要的数据结构分为两部分，其中重要的一部分放在系统核心空间中；而相对不重要、对其它进程无影响的数据结构放在用户空间中。在通信过程中，用户进程在发送消息时需要访问核心及对 NIC 进行控制。并将待发送的数据将传递到其它进程之中。因此，在半用户级通信协议中，把消息发送过程中的一部分重要的数据和过程放入核心，以保证系统安全性、可靠性和任务间的独立性。在接收消息时，由于只涉及读取数据，不会影响到系统中的其它进程，因此该操作无须进入核心。

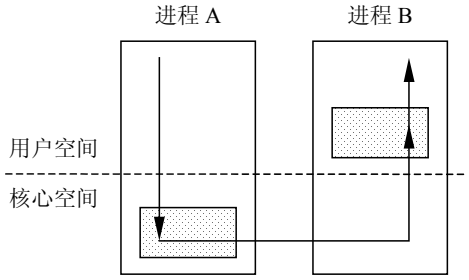

图2.9 半用户级通信

同用户级通信结构相比，半用户级通信在通信关键路径上只增加了一次进入核心的开销。目前这一开销已经很小，不会对通信性能造成明显影响。同时，采用进入核心空间的方法还可以保护系统的安全性、可靠性，保障并行任务间的独立性。此外，半用户级通信可以屏蔽系统的硬件信息。由于涉及硬件(NIC)的操作均集中在进入核心空间的处理过程之中，因此在用户级通信函数中不再出现与系统通信硬件有关的操作。整个通信协议可以支持异构网络环境。即对于不同的通信硬件使用不同的核心通信系统，整个系统使用相同的用户级通信库(函数)。这样，一方面可以用一种协议互连不同的通信硬件设备，另一方面也可以使得上层应用程序二进制代码可移植。最后，从实现的角度来说，用户级通信协议需要将核心空间的数据映射到用户空间。这在实现上需要依赖所使用的软件平台(操作系统)。不同的平台可能使用不同的方法进行内存映射，有的平台可能不支持从核心空间到用户空间的内存映射。这就对系统的实现和移植造成了困难。采用半用户级通信协议在实现方面对软件平台(操作系统)没有特殊的要求，有利于系统的移植。

2.3.3 多重通信协议设计中的关键问题

业界在总结了近年来的各种高效通信协议的基础上，由 Compaq、Intel 和 Microsoft 共同提出了 VIA(Virtual Interface Architecture)标准。VIA 在总结了各个研究单位的工作后，力图提出一个高性能的通信协议标准。但是，VIA 所提出的规范中没有给出对 SMP 结点的特殊支持。同时，在协议安全、可靠性方面也没有相关研究。因此，在这方面还有许多关键性的问题需要进行更进一步的研究。

多重通信协议研究的关键在于如何设计实现高效的多重通信协议的各协议层，以及多重通信协议中不同协议间相互组合的问题。本文中的多重通信协议由基于共享内存的 SMP 结点内通信协议和半用户级结点间通信协议组成。基于共享内存的通信协议是针对 SMP 结点而设计的结点内通信协议。在设计中需要考虑如何提高通信性能，如何处理多进程同时访问临界资源，如何降低数据拷贝造成的影响。半用户级通信协议是在用户级通信协议 BCL-2 的基础上设计而成的。它着重在不影响性能的前提下保护部分重要数据结构，以提高系统的安全性、可靠性。在当前的高效通信协议中，多数都采用用户级通信协议。为了能够使得半用户级通信协议能够尽量减少因进入核心空间而带来的开销，在设计机间通信时需要考虑如何组织需要保护的重要数据结构，减少对这些数据结构的操作。同时，组合两个通信协议时还需要考虑协议接口的一致性问题，以及不同协议之间的协调问题。

在一般的高效通信协议中引入对 SMP 结点的支持和对安全性的考虑后，会带来一些新的问题。首先是如何设计一个高效的基于共享内存的通信协议。由于是采用共享内存的方法，就需要考虑对共享资源的并发访问问题。一般对共享资源的访问需要采用锁机制或 PV 操作等互斥机制来控制并发访问。本文中的结点内通信采用 Lock-Free 的算法，通过设计数据结构和相关操作减少不必要的互锁操作，从而降低因访问临界资源而带来的额外开销。在第18页图 2.6 中可以看出，对比直接在不同进程的发送和接收数据缓冲区间进行拷贝的方法，采用共享内存的方法实现结点内通信会增加一次额外的内存拷贝过程。如何提高这一部分的性能也是研究多重通信协议的一个关键问题。其次，如何设计一个高效的半用户级通信协议。半用户级通信协议需要将对重要数据结构的操作限制在核心环境之中，这就需要合理安排数据结构和对数据结构的操作。最后，

多重通信协议是将两个不同的协议组合，为上层提供了一个统一的界面。因此，多重通信协议的应用程序接口应是一致的。为了达到一致的接口，并减少因接口而造成的额外开销，我们在设计协议时着重考虑了协议中探询的开销。下面给出了协议设计中的几个关键问题。

- **并发访问数据**

在使用共享内存进行并行计算时，总会出现数据被并发访问的情况。当多个进程同时需要访问同一数据区域时，为了保证访问的一致性，需要对被访问的数据进行保护。保护一般采用对数据的原子操作或在访问临界资源时加锁等。在通信的关键路径上使用这些方法保护数据将会增加通信的开销，使通信延迟增大。

为了减少访问共享资源时的互斥开销，在基于共享内存的通信协议中采用了 Lock-Free 算法，通过在算法方面进行设计达到互斥的目的。这样，不但减少了通信关键路径上的开销，同时也使得系统避免了死锁。

- **内存拷贝**

对比直接在不同进程的发送和接收数据缓冲区间进行拷贝的方法，采用共享内存的方法实现结点内通信会增加一次额外的内存拷贝过程。为了降低内存拷贝对性能的影响，在协议的设计中采用了流水线的方式使得额外的一次内存拷贝的过程隐藏在另一次内存拷贝过程之中。

- **数据分布**

在设计多重通信协议的数据结构时，首先要考虑降低通信关键路径上的开销。同时，还要考虑到每个协议层间的数据关系以及数据保护的问题。

降低通信关键路径上的开销是研究高效通信协议的一个关键问题。这主要包括两方面的问题，一是减少进入核心的次数，一是减少在通信关键路径上的数据交换(数据拷贝)次数。在采用用户级通信协议时，所有的通信操作均在用户级完成，没有进入核心的开销。当采用半用户级通信协议时，由于部分重要数据结构须进入核心空间进行操作，因此增加了通信的开销。为了使这一通信开销降低，我们需要尽量减少通信关键路径上进入核心的次数。这就需要合理安排系统的数据结构。在多重通信协议中，我们将重要的数据结构以及对其的操作组织在发送消息的过程之中，而在接收消息时不再进入核心空间。

在关键路径上的数据交换主要包括消息体数据和系统控制数据两部分。多重通信协议采用内存零拷贝技术来减少机间通信关键路径上传递消息体数据的开销，并采用流水线的技术来减少机内通信关键路径上传递消息体数据的开销。对于系统控制数据则需要设计合理的数据分布来减少开销。由于机间通信采用了半用户级通信协议，因此部分关键数据结构需要在用户空间与核心空间之间传递。过多的控制类数据交换(拷贝)次数也会降低系统的性能。为此，在多重通信协议中，我们将数据按照具体需要分别放置在用户空间和核心空间中，以减少其交换的次数。

- **探询开销**

多重通信协议中消息发送与接收采用非阻塞语义，每个发送和接收请求都需要一个相应的探询过程以确定其执行完成。由于在多重通信协议中，结点内部与结点间所采用于通信机制不同，因此消息的探询方式也不同。对于远程消息，消息的发送和接收结束均由 NIC 的状态决定，而对于本地消息，消息的发送和接收结束只涉及本地共享内存的状态。一般的通信协议多采用分别查询 NIC 状态和本地共享内存状态来判断消息发送与接收的结束。由于查询 NIC 状态需要较长的时间，这样，分别查询会造成对查询速度快的结点内通信性能的影响。本地消息和远程消息的通信开销取决于查询 NIC 和共享内存状态中速度较慢的一方。

造成这一现象的根本原因在于探询不同消息的开销不同。为此，在多重通信协议中采用了特殊处理，使得不同消息的探询开销相近。这样，就解决了远程消息与本地消息探询处理时间上的不平衡问题，减小了探询开销。

- **应用程序接口**

多重通信协议虽然在协议结构上采用了两组协议分别处理本地消息和远程消息的传递，但对上层软件应提供一致的协议服务。这样，应用程序员就可以将整个通信协议层看作一个黑盒子，无须关心通信是发生在哪一对进程之间，需要使用哪一层协议。整个通信协议将系统的硬件拓扑结构屏蔽起来，让上层软件无须考虑系统的体系结构。然而，多一层软件的封装将增大通信过程中的开销。因此，设计一个统一的应用程序接口，而又不影响不同协议的性能也是设计多重通信协议的一个关键问题。

第三章 多重通信协议

本章着重探讨多重通信协议在设计和实现上的一些问题。首先，从整体的角度讨论了多重通信协议设计实现中的一些问题，包括所使用的通信机制、多重协议间的协同以及对性能的影响等。然后，文章分别从多重通信协议的两个协议层上进行讨论，研究了基于共享内存的通信协议和半用户级通信协议在设计和实现上的一些问题。最后，讨论了多重通信协议的可扩展性问题。

3.1 多重通信协议

由 SMP 结点构成的机群系统从互连性能的角度上可以看作两级互连结构。第一级是 SMP 结点内部互连，采用共享内存的机制，不同处理器共享同一内存。第二级是 SMP 结点间的互连，通过高速互连网络进行消息传递。为了能够有效地利用硬件的性能，并且对上层提供一个统一易用的应用程序接口，在通信协议层我们采用了多重通信协议结构。多重通信协议由基于共享内存的 SMP 结点内通信协议和基于互连网络的半用户级通信协议组成。两重协议支持了不同互连结构的通信硬件。为了能够对上层提供一致的接口，这两重协议采用了相同的通信机制和应用程序接口。在这一节中将对多重通信协议的通信机制和应用程序接口进行讨论，并分析采用多重通信机制后带来的一些问题。

3.1.1 通信机制

在 BCL-3 中，进行消息通信的基本单元是端口(Port)。每一个进程可以创建一个端口同其它进程进行通信。每一个进程由其所在的结点号(Node)与其创建的端口号标识。在整个并行应用中，由结点号与进程号组成的偶对唯一标识一个进程。每一个端口有一个消息发送请求队列(Sending Request Queue，SRQ)、一个消息接收缓冲池(Receiving Buffer Pool，RBP)和相应的完成事件队列：消息发送完成队列(Sending Event Queue，SEQ)和消息接收完成队列(Receiving Event Queue，REQ)。消息接收缓冲池由一组消息通道组成。在 BCL-3 中定义

了三种通道类型：系统消息通道(System Channel)、一般消息通道(Normal Channel)和开放消息通道(Open Channel)。

当一个进程发送一个消息给另外一个进程时，消息发送进程首先需要建立一个消息发送请求。消息发送请求中包含了待发送消息的数据或其缓冲区地址以及消息接收进程的标识(结点号与端口号偶对)。然后，消息发送进程将该消息发送请求插入消息发送请求队列中。在实施正式的消息传递前，消息接收进程需要准备好其相应的接收缓冲区(接收通道)。当消息接收完成后，系统将会产生一个消息接收完成事件，并插入消息完成事件队列，以通知消息接收进程。同样，在消息发送方当消息发送完成后，系统也将产生一个消息发送完成事件通知相应的进程。

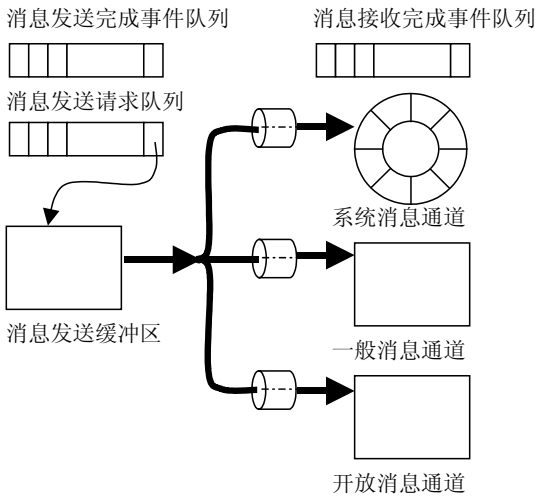

图3.1 多重通信协议的通信机制

图 3.1给出了多重通信协议的通信机制的示意图。图中显示了三种不同的通道类型：系统消息通道、一般消息通道和开放消息通道。系统消息通道用于传递小消息。每一个进程只有一个系统消息通道。系统消息通道在进程初始化端口的同时进行初始化，并在进程通信过程中始终有效。每一个系统消息通道有一个系统消息缓冲池。这个缓冲池按照环形队列的结构组织。当一个小消息到达时，该消息体自动地放入环形队列中的第一个空闲块内。如果缓冲池中已没有空闲块，则接收到的消息被丢弃。接收进程按顺序在系统消息缓冲池中接收消息。在接收进程接收到一个消息后，相应的缓冲块将被回收到系统消息缓冲

池中。

一般消息通道用于传递大消息。在 BCL-3 中，大消息的传递是按照 Rendezvous 语义进行，即仅当消息接收进程预先准备好消息接收缓冲区时，消息发送进程才能保证正常完成。每一个进程(端口)可以有若干个一般消息通道。每一个一般消息通道有一个由用户指定的消息接收缓冲区。这个消息接收缓冲区需要在每次使用该通道进行消息传递前初始化。当消息传递结束时，消息发送进程和消息接收进程将接收到相应的消息发送/接收结束事件。

开放消息通道是用于进行远程存储访问(Remote Memory Access，RMA)操作的。每一个进程(端口)可以有若干个开放消息通道。每一个开放消息通道有一个由用户指定的消息接收缓冲区。与一般消息通道不同的是，这个消息接收缓冲区只需在第一次使用该通道进行消息传递前进行初始化。当远程存储访问操作结束时，RMA 的发起者将收到相应的操作结束事件。

3.1.2 统一的应用程序接口

采用多重通信协议的一个主要的目的就是将两种不同的互连特性屏蔽起来，为上层提供一个一致的接口。为此，提供一个统一的应用程序接口是多重通信协议必须满足的一个条件。

3.1.3 探询开销问题

在高速通信协议层中，通常使用探询(Poll)的方式来检测消息的到达。在多重通信协议中，需要对系统中不同通信协议层使用不同的探询手段。这无形中增加了通信路径上的开销。同时，在不同通信协议中探询消息到达的方式不尽相同，其开销也不同。对于基于共享内存的通信协议，探询过程只需要涉及结点内共享内存的访问，其访问速度较高。对于需要访问网络接口卡的结点间通信，其探询过程就可能需要涉及 I/O 设备，访问所需的时间也较长。为了降低通信开销，在 AM-II 中就采用了 Adaptive Polling 的方式，对于不同的协议中的探询采用不同的速率进行。即多探询访问速度快的协议层而减少对访问速度慢的协议层的探询次数[7]。

在本文中，我们采用了另一种方法来降低探询的开销。对于访问速度较慢的设备我们并不直接去探询其状态，而采用由通信设备回传消息接收事件的方

式，将消息到达的事件转发至访问速度较快的介质中。这样，两重协议的探询开销相近，就不存在因探询开销不平衡而降低通信性能的问题。

3.2 基于共享内存的通信协议

本地消息通过共享内存进行传递。SMP 结点的共享内存和 Cache 一致性保证了通信协议的正确性。消息通信采用了基于通道(Channel)的消息传递。两个进程间通过特定通道进行消息传递。在 BCL-3 的共享内存通信协议中，消息的传递分为两种情况，一种是采用完全异步的通信方式，通过缺省的系统消息通道(System Channel)进行消息传递；另一种是采用 Rendezvous 的通信方式，通过程序中指定的一般消息通道(Normal Channel)或开放消息通道(Open Channel)进行消息传递。一般地，在进行小消息传递时，由于所需的系统缓冲区较小，通常采用第一种方式。采用这种方式可以减少因同步消息发送方与接收方而带来的开销，从而提高通信性能。在进行大消息传递时，由于同步开销相对于消息传递时间来说对通信性能影响较小，因此采用第二种方式。这种方式有助于减少因消息发送与接收方不匹配而造成的开销(如内存拷贝等)。

在设计共享内存通信协议时，主要须考虑共享内存的使用和两种通信机制的设计。由于共享内存是由多个进程并发访问的内存区域，因此，需要有相应的并发访问控制机制。如何减少因访问临界区而带来的开销是设计数据结构和算法时考虑的问题。下面将对共享内存通信协议中数据结构在内存中的分布和两种通信机制分别进行介绍。

3.2.1 内存分布

图 3.2给出了基于共享内存的通信协议的数据结构在内存中的分布情况示意。进程 A 和进程 B 的通信控制结构分布在各自的内存空间中。两进程相互交换数据的数据传输队列位于两进程的共享内存中。进程的控制结构由一组队列组成，包括消息发送请求队列、消息接收请求队列、消息发送完成事件队列和消息接收完成事件队列。

数据传输队列是用来在进程间传递消息体数据的。任意两个相关进程之间有一组双向数据传输队列。数据传输队列由一个系统消息缓冲环和一个数据交换队列组成。系统消息缓冲环是系统消息通道使用的缓冲环。它是一个环形的

先进先出队列。通过系统消息通道传递的消息首先暂存在这个环形队列中，然后由消息接收函数接收到用户缓冲区。数据交换队列是一般消息通道和开放消息通道使用的缓冲队列。这类消息采用 Rendezvous 的通信方式进行传递，消息发送时其接收方缓冲区已准备好。数据交换队列只是用来传递消息包的介质。消息发送方与接收方采用 Lock-Free 算法使用该缓冲区，完成消息的传递。采用数据交换队列可以使一般消息通道消息的发送与接收过程重叠起来，从而隐藏内存拷贝过程。

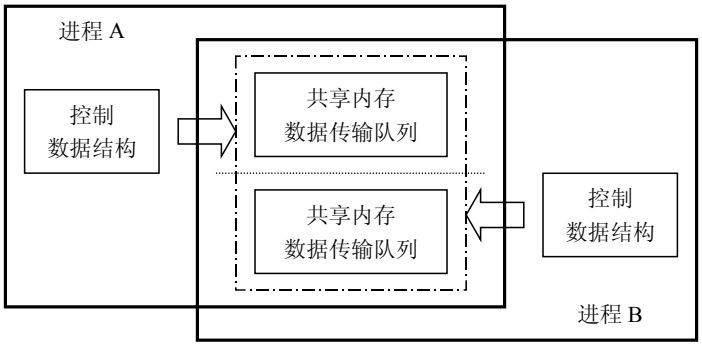

图3.2 基于共享内存的通信协议数据结构分布

3.2.2 通信机制

在基于共享内存的通信协议中，根据消息的长度采用两种不同的通信机制进行消息传递。对于长度较短的消息(小消息)采用系统消息通道的机制，这种通信机制可以减少通信关键路径上的开销，从而降低通信的延迟。对于长度较长的消息(大消息)采用一般消息通道的机制，这种通信机制可以一次传递较多的数据，从而有效地利用系统硬件的通信带宽。

系统消息通道的通信机制

系统消息通道使用系统消息缓冲环来进行数据传输。图 3.3给出了一个单向的系统消息通道通信结构。

系统消息通道的使用方式是一个典型的“生产者-消费者”方式。每一个系统消息缓冲环由多个“生产者”(发送消息进程)和一个“消费者”(接收消息进程)并发访问。由于只有一个“消费者”进程，因此缓冲区读指针可以放在用户

进程的空间中，由单一进程访问。访问时无须加锁。多个“生产者”进程可能并发地向系统消息通道内发送消息，因此缓冲区写指针在共享内存空间，由多个进程并发访问。该指针为临界资源，对它的访问需要互斥。当系统消息缓冲环填满时，消息发送程序将被挂起。在下一次通信操作时将激活被挂起的发送程序。

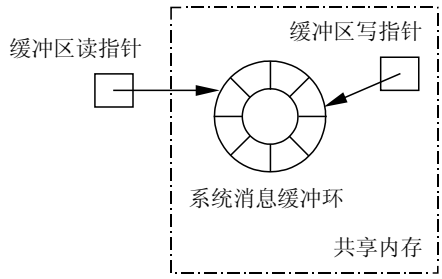

图3.3 基于共享内存的系统消息通道通信机制示意

一般消息通道和开放消息通道的通信机制

一般消息通道和开放消息通道使用数据交换队列来进行数据传输。图 3.4 给出了两个进程间数据交换队列的结构示意。

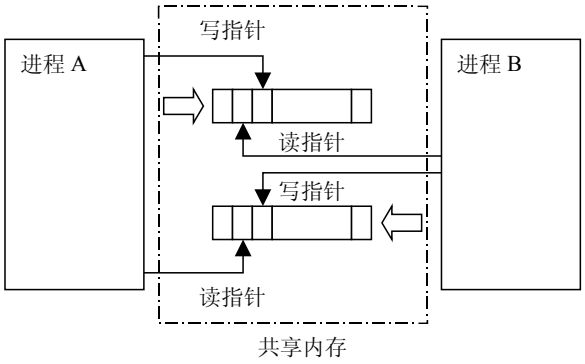

图3.4 基于共享内存的一般消息通道和开放消息通道的通信机制示意

与系统消息通道所使用的系统消息缓冲环不同，一般消息通道和开放消息通道中所使用的数据交换队列只是用于一对交互进程之间的通信。也就是说，任何一个数据交换队列只有一个数据的“生产者”和一个数据的“消费者”。因

此，在操作数据交换队列时无须加锁，从而减少了通信的开销。当数据交换队列填满时，消息发送程序将被挂起。在下一次通信操作时将激活被挂起的发送程序。

3.2.3 Lock-Free 算法

由于共享内存中的消息缓冲区允许不同进程并发访问，就需要采用一些原子操作以控制不同进程对临界资源的访问。设计一个高效的共享内存访问策略是提高基于共享内存的通信协议性能的一个关键问题。

传统的并发数据访问算法采用临界区的方式来防止并发访问间的相互干扰：进程需获取一个独占(Exclusive)锁以进入临界区，从而防止多个进程同时进入临界区。这样，拥有锁的进程若因其它原因而长时间不能释放锁，就会造成其它进程无限期的等待。其它进程采用自旋锁(Spin Lock)的方式获取锁会造成处理器资源的浪费[55]。目前，有很多技术可以用来解决这方面的问题，如 Preemption-safe Locking 等。虽然，Lock-Free 算法在一般的情况下性能并不好，但是用于队列(Queue)、堆(Heap)、栈(Stack)、计数器(Counter)等数据结构时，可以达到较高的性能[56][57]。

3.2.4 大消息传递的流水线算法

第30页图 3.4给出了大消息传递的通信机制。在采用一般消息通道进行消息传递时，消息体通过系统数据交换队列进行数据传递。在 BCL-3 协议中，数据交换队列可以看作是联系两个进程的单向通信管道。在两个进程间进行通信时，每一个通信方向上都有一个数据交换队列。两个进程间不同的通信通道共享这两个数据交换队列。

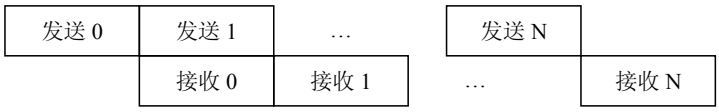

图3.5 大消息传递的流水线算法

在使用共享内存进行消息通信时，消息数据需要经过两次内存拷贝。为了能减少内存拷贝对通信性能的影响，在 BCL-3 中采用了流水线算法进行一般消

息通道的通信。图 3.5给出了流水线算法的示意图。数据交换队列按照循环队列进行组织。当进行大消息传递时，将这个大消息分解成 N 个消息包，依次进行发送。消息发送方依次向数据交换队列中填入消息包，同时，消息接收方依次从数据交换队列中取出消息包。当消息足够长时，共享内存中的一次数据拷贝的开销被隐藏在通信过程之中。

3.3 半用户级通信协议

远程消息通过高速互连网络进行传递。互连网络硬件由网络接口卡(NIC)和互连网组成。BCL-3 的半用户级通信协议是基于具有通信协处理器的网络接口卡而设计的。NIC 上的通信协处理器控制结点(端口)间的消息通信。结点(端口)间的消息传递由结点上的主机处理器发起，将消息发送/接收请求提交给 NIC 上的通信协处理器，然后由 NIC 上的通信协处理器进行消息传递。消息通信采用基于通道(Channel)的消息传递。两个进程间通过特定通道进行消息传递。在 BCL-3 的半用户级通信协议中，消息的传递分为两种情况，一种是采用完全异步的通信方式，通过缺省的系统消息通道(System Channel)进行消息传递；另一种是采用 Rendezvous 的通信方式，通过程序中指定的一般消息通道(Normal Channel)或开放消息通道(Open Channel)进行消息传递。一般地，在进行小消息传递时，由于所需的系统缓冲区较小，通常采用第一种方式。采用这种方式可以减少因同步消息发送与接收方而带来的开销，从而提高通信性能。在进行大消息传递时，由于同步开销相对于消息传递时间来说对通信性能影响较小，因此采用第二种方式。这种方式有助于减少因消息发送与接收方不匹配而造成的开销(如内存拷贝、丢包等)。

半用户级通信协议通过对系统数据结构的组织，使得通信过程中对重要数据结构的访问限制在系统核心环境下。同时，减少对重要数据结构的访问，以减少进入核心的次数。下面对半用户级通信协议中数据结构在内存中的分布和通信机制分别进行介绍。

3.3.1 内存分布

图 3.6给出了半用户级通信协议的数据结构在内存中的分布情况示意。进程 A 和进程 B 的通信控制结构分布在各自的内存空间中。两进程相互交换数据的

消息数据传输队列位于 NIC 的卡上内存中。

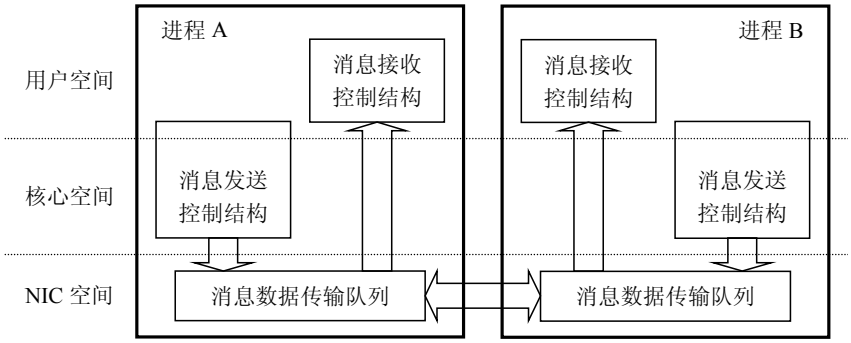

图3.6 半用户级通信协议数据结构分布

进程的通信控制结构分为两个部分，消息发送控制结构和消息接收控制结构。消息发送控制结构包括消息发送请求队列、消息发送缓冲区控制块和消息发送完成事件队列等。在这里，我们把消息发送控制结构中重要的部分放入核心空间，以保证其安全性。消息发送请求队列和消息发送缓冲缓冲区控制块涉及到与其它进程通信的关键数据，因此这部分数据结构处于核心空间中。不涉及其它进程、对系统安全影响较小的消息发送完成事件队列则被安排在系统的用户空间之中。由于消息接收过程均不涉及其它进程，因此这部分控制数据结构存放于用户空间之中。这样，在进行消息通信时，只有在消息发送的过程中需要进入核心空间，而在探询消息发送是否完成和接收消息时，无须进入核心空间。

消息数据传输队列是用来在进程间传递消息体数据的。任意两个相关进程之间有一组双向数据传输队列。数据传输队列由一个系统消息缓冲环和一组通道控制块组成。系统消息缓冲环是系统消息通道使用的缓冲区。它是一个环形的先进先出队列。通过系统消息通道传递的消息首先暂存在这个环形队列中，然后由消息接收函数接收到用户缓冲区。通道控制块用于一般消息通道和开放消息通道消息接收。每一个通道控制块控制一个通道。通道控制块中包含了使用该通道的两个进程的相关信息和用于接收消息的缓冲区信息。这类消息采用 Rendezvous 的通信方式进行传递，消息发送时其接收方缓冲区已准备好。

3.3.2 通信机制

在半用户级通信协议中，根据消息的长度采用两种不同的通信机制进行消息传递。对于长度较短的消息(小消息)采用系统消息通道的机制，这种通信机制可以减少通信关键路径上的开销，从而降低通信的延迟。对于长度较长的消息(大消息)采用一般消息通道的机制，这种通信机制可以一次传递较多的数据，从而有效地利用系统硬件的通信带宽。

系统消息通道的通信机制

系统消息通道使用系统消息缓冲环来进行数据传输。图 3.7给出了单向的系统消息通道通信结构。

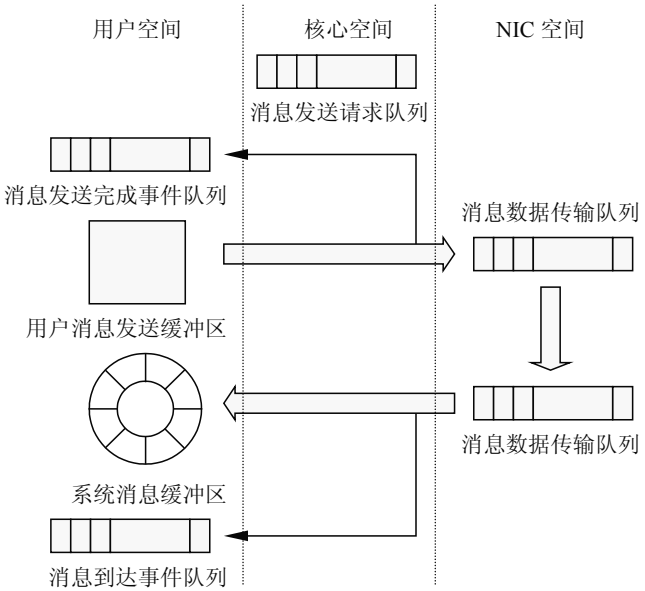

图3.7 半用户级通信协议中系统消息通道通信机制示意

在半用户级通信协议中，通过系统消息通道发送一个消息时，首先需要进入核心空间填写消息发送请求。NIC 上的通信协处理器将按照消息发送请求的内容将相应的数据通过 NIC 上的消息数据传输队列发送至相应的结点。当消息发送完成后，将会有一个消息发送完成事件送回用户空间的消息发送完成事件队列中，以通知相应的进程发送过程完毕。

从互连网络上接收到的系统消息通道消息通过在NIC上的消息数据传输队列依次传输到处于用户空间的环形系统消息缓冲区内。并且在相应的消息到达事件队列中填入消息到达记录，以通知用户进程接收消息。

一般消息通道和开放消息通道的通信机制

一般消息通道和开放消息通道直接使用用户缓冲区进行数据传输。图 3.8 给出了单向的一般消息通道和开放消息通道的通信结构。

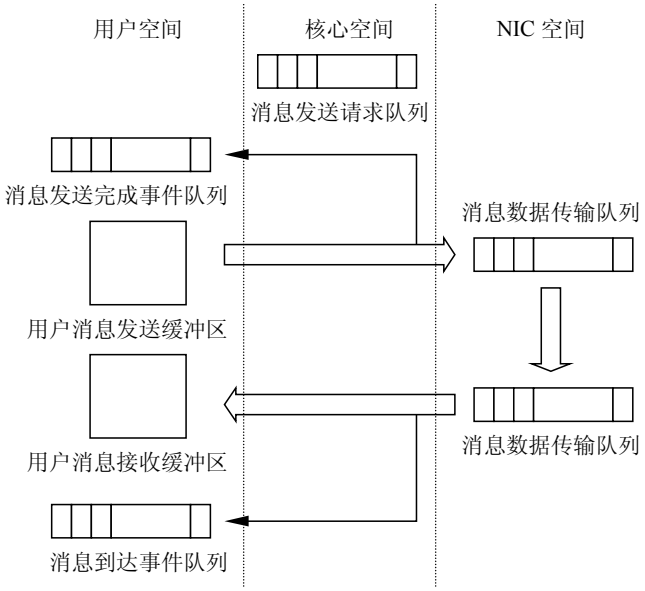

图3.8 半用户级通信协议中一般消息通道和开放消息通道的通信机制示意

与系统消息通道所使用的系统消息缓冲环不同，一般消息通道和开放消息通道直接使用用户的数据缓冲区进行消息传递。消息数据从用户的发送缓冲区中直接传递至 NIC 上的消息数据传输队列，通过高速互连网络的传输后再传输到接收方的消息接收缓冲区中。当消息传递完成后，系统会产生消息发送/接收完成事件，以通知相应进程操作完成。

3.3.3 半用户级通信协议的特点

传统的通信协议(如 TCP/IP 等)在消息传递的关键路径上需要进入操作系统的核心空间、进行中断处理等。所有这些通信路径上的开销将增大点到点消息传递的延迟。这类协议我们称之为核心级通信协议(Kernel-level Protocol)。用户

级通信协议(User-level Protocol)允许应用程序在通信的过程中越过操作系统直接访问网络接口卡(NIC)。消息数据在 NIC 的控制下直接在用户的缓冲区之间进行传递。通信过程中不再有传递通信协议中的保护区域。这样的消息通信机制能够有效地减少通信关键路径上的开销，从而降低消息传递的延迟。

然而，在商品化的系统中，安全性成为最重要的系统要求之一。由于用户级通信协议将全部的控制数据结构存放于用户空间，任何一个错误的或恶意的操作都有可能造成系统的崩溃。而核心级通信协议则能够有效地保护这些重要的数据结构。

为了能够同时保证系统的安全性和降低通信延迟，在半用户级通信协议中采用了将用户级通信协议和核心级通信协议相结合的方法。在半用户级通信协议中，采用将系统重要数据保存在核心空间中的方式来保证系统的安全性，同时采用减少非必要的核心操作的方式来减少通信关键路径上的开销。

采用半用户级通信的思想，不但可以保证系统的安全性、降低系统的通信开销，而且还能够增强应用程序的可移植性、支持异构网络环境(Heterogeneous Network Environment)等。

- **保证系统安全性、可靠性和并行任务的独立性**

一个安全的通信协议要求能在有错误或恶意操作的情况下保证系统能够正常运行，将破坏限制在最小的范围之内。采用一般的用户级通信协议则无法保证这一点。因为在用户级通信协议中，全部的操作都在用户空间完成，全部的数据结构都存放在用户空间。这样，只要有一般用户的权限就可以操作相关这一用户应用程序通信部分的全部数据结构。在并行系统中，当一个用户的能够控制相关通信的全部数据结构时，它就可以随意向系统中其它任何进程(包括其它用户的进程)发送任何形式的消息。这样，恶意的操作将有可能破坏系统中的全部并行任务。

采用核心级通信时，用户应用程序中相关通信的操作均在核心环境下完成。因此，恶意用户无法通过消息传递影响其它用户的并行任务。同时，有关流量控制等机制也在核心环境下处理，使得应用程序能在系统的完全控制之下，保证了系统的健壮性。不同应用任务也由于在系统核心环境下进行管理而能够保证其独立性。

半用户级通信协议吸收了上述核心级通信协议的优点，在进行通信时将能

够影响到其它进程的操作(消息的发送等操作)限制在核心环境下完成。同样地，有关流量控制等机制的主要操作也是限制在核心环境下。这样，恶意用户就无法通过消息传递影响其它用户的并行任务了。半用户级通信协议能够保证系统的安全性、可靠性和并行任务间的独立性。

- **降低通信关键路径上的通信开销**

传统的核心级通信协议在每次发送或接收消息的过程中都需要进入核心环境。这样，在一次消息传递的过程中至少要进入两次核心环境。在采用非阻塞(Non-blocking)的消息传递语义时，为了探询消息发送与接收是否完成，还有可能更多次地进入核心环境。这就在消息传递的关键路径上增加了很多额外的开销。

为了降低在消息传递的关键路径上进入核心环境的开销，国内外很多研究单位都设计了用户级通信协议。在用户级通信协议中，全部消息传递过程都在用户环境下完成，消息传递的关键路径上避免了进入核心环境的开销。

半用户级通信协议采用了用户级通信协议的思想，在进行消息传递的过程中尽可能地避免增加通信开销。为此，我们减少了进入核心的次数，仅在每次发送消息时进入核心环境。在接收消息及探询发送/接收是否完成的过程中不再进入核心环境。在半用户级通信协议中，我们不再简单地追求完全的用户级通信，而是采用这种思路去降低通信关键路径上的开销。除了采用减少进入核心次数来减少开销外，在半用户级通信协议中还采用了减少临界资源、减少锁操作、采用 Lock-Free 算法等来减少通信开销。通过对半用户级通信协议性能的测试和分析，我们可以看出采用这种思路能够有效地提高通信系统的性能。

- **二进制代码可移植**

在用户级通信协议中，为了能够实现不进入核心环境的通信过程，通信系统需要将通信硬件完全暴露给用户进程。这就导致了用户的应用程序中需要进行相关处理。当系统所用的硬件平台有所变化时，相应的用户级处理过程也需要有相应的变化。这样，应用程序在不同平台上运行时需要链接不同的用户级库函数，其代码只能达到源码级可移植，而无法实现二进制代码可移植。

在半用户级通信协议中，由于采用了将操作系统重要数据结构的过程屏蔽在核心环境之中的方法，系统可以将通信硬件完全屏蔽在核心扩展中。这样，

对于不同的硬件平台，可以保持其用户级库函数不变，而采用不同的核心扩展。通过这种方式，我们可以实现在不同通信硬件平台上(如采用 Myrinet 的通信平台或采用曙光设计的PMI-8/16的通信平台)用户应用程序的二进制代码可移植。

- **支持异构网络环境**

半用户级通信协议通过划分用户级处理过程和核心级处理过程将系统通信硬件的信息完全屏蔽在核心扩展中，就使得通信硬件对其上运行的并行任务透明。这样，采用不同通信硬件的结点互连的机群系统可以采用半用户级通信协议来屏蔽其硬件设备上的差异，为上层应用程序提供一个一致计算平台。半用户级通信协议提供对异构网络环境的支持。

3.3.4 半用户级通信协议对性能的影响

相对于用户级通信协议，半用户级通信协议在通信路径上增加了一次进入核心空间的过程。这就增加了通信路径上的开销。为了研究增加的通信开销对系统性能的影响，我们专门测量了进入核心空间的开销。测试是在曙光 3000 平台上进行的。测试平台配置 IBM 4-way Power3-II 375MHz 处理器，2GB 内存。计算结点使用 AIX 4.3.3 操作系统。在这一平台上所测得的进入核心空间的开销为2.05μs。对比最后系统实测的通信延迟性能，进入核心空间的开销大约占通信延迟的 1/10。

3.4 可扩展性分析

随着机群系统的规模的扩大，机群系统的互连也将成为影响可扩展性的一个问题。除了互连网络硬件需要保证整个网络性能的可扩展性外，通信软件也需要对可扩展性提供支持。这主要体现在通信软件对一些硬件资源的需求不应随系统规模的扩大而无限制地快速增长。由于硬件资源是有限的，对它需求的增长势必造成对系统扩展规模的限制。在 BCL-3 中，我们替换了原来基于 Credit 的流量控制，采用了基于 ACK/NAK 的流量控制。这使得通信软件的流量控制不再影响系统的可扩展性。软件系统具有较好的可扩展性。

第四章 多重通信协议的形式化模型

本章采用进程代数的方法形式化地描述了多重通信协议，并通过一些辅助工具对这一形式化模型进行验证。文章首先简要介绍了进程代数方面的背景知识，然后建立了协议的形式化模型，最后给出了通过辅助工具进行验证过程。

4.1 进程代数

4.1.1 进程代数概述

在协议的设计中需要对协议进行形式化的描述，以便进行正确性的验证。形式化的方法把一个系统的各个组成部件看作是数学对象，并采用数学模型来描述和预测这些对象的可观测属性及行为。采用形式化的方法在设计协议中有很多优点。首先，可以预先发现协议的初步需求中模糊、不确定和不完整的部分。其次，可以自动或采用计算机辅助的手段分析协议规范的正确性。最后，采用数学的方式可以降低初期开发的成本。

有很多方法可以形式化地表示一个通信协议，其中进程代数是较为适合于描述、分析有关通信、并发执行的系统。进程代数有很多种形式，包括CCS(Calculus of Communicating Systems)[73]、CSP(Communicating Sequential Processes)[74][75][76]、Acceptance Trees[77]和 ACP[78]等。这些代数方法都是基于并发和通信这两个最基本的理解复杂、动态系统的概念的形式化方法。进程代数中的一个最重要的方面是它支持模块化(Modular)的描述和验证一个系统。这都是源于代数定理系统是一个基于证明构造的系统，可以通过证明其每一部分来验证整个系统。进程代数被广泛地应用于描述和验证并发系统。

4.1.2 CSP

CSP 是在 1978 年首先由 C.A.R Hoare 在其文章[74]提出的。它的基本思想和符号在其 1985 年出版的[75]中进一步完善。1997 年 A.W. Roscoe 出版了[76]，

给出了 CSP 更完善论述。CSP 按其名称含义是采用一组相互独立运行、通过通道(Channel)进行通信的进程来描述一个系统。在采用面向对象的程序设计后，CSP 能更有效地描述此类系统，特别是那些并行的模块化的系统。

CSP 是一种描述通过消息传递进行通信的并发系统的抽象语言。它特别适用于描述和分析网络协议。在 CSP 中主要有事件(Event)、进程(Process)、踪迹(Trace)和规范(Specification)等概念。

事件(Event)

事件是构成一个系统模型的基本单元。所有可能发生的事件集合用α表示(如αBCL)。每一个事件可以是一个原子的单元，也可以由多个原子单元组成。如 *put.5* 由两部分组成：通道名 *put* 和数据 *5*。

进程(Process)

进程是构成一个系统的组成部分。它是用来表征动作过程的。一个进程可以涉及到 0 个或多个事件。例如：进程 *Stop* 表示结束，不包括任何事件，它表示一种停止的状态；进程 $x \rightarrow P$ 表示当发生事件 x 后进入进程 P；进程 $c!v \rightarrow P$ 表示从通道 c 中输出数据 v 后进入进程 P; $c?x \rightarrow P(x)$表示从通道 c 中输入数据 x 后进入进程 $P(x)$；进程 $P|Q$ 表示按照进程 P 执行或按照进程 Q 执行。为了描述一个并发系统，CSP 还引入了一些描述并发执行的符号。如：符号||是一种并发算子，$P||Q$ 表示进程 P 和进程 Q 并发执行，即进程 P 与进程 Q 之中共有的事件完全相同；符号□是一种非确定性算子，$P \square Q$ 表示若当前事件可以是 P 的第一个事件时，进程选择 P；若当前事件可以是 Q 的第一个事件时，进程选择 Q；若当前事件既可以是 P 的第一个事件又可以是 Q 的第一个事件时，在 P 与 Q 间的选是不确定的(Nondeterministic)。

踪迹(Trace)

踪迹 *trace(P)*是进程 P 的一种可能的运行过程。它是由一组事件构成的。它可以看作是事件的一种有序的排列。对于踪迹可以进行连接、筛选等操作。*traces(P)*表示进程 P 的全部踪迹的集合。

规范(Specification)

规范是踪迹应该遵循的规则。一个进程 P 满足规范 S 定义为该进程的所有

踪迹都满足 *S*：

$$P\ sat\ S \Leftrightarrow \forall tr \in traces(P).S$$

4.1.3 PROMELA 语言与协议验证工具 Spin

PROMELA 语言是 1983 年设计的 Argos 语言的一个扩展。它是一种用于描述通信协议或其它类似的分布系统的一种抽象语言。一个 PROMELA 程序由一组相互之间进行通信的进程构成。每一个进程是一个扩展的有限状态机(EFSM，Extened Finite State Machine)。在这种语言中只支持简单的数据类型，如各种范围的整型、记录类型(与 C 语言中的 typedef 类似)。

Spin[81][82]是使用 PROMELA 语言的一个辅助工具。它可以用来单步或者随机模拟 PROMELA 系统。它还可以用穷举法来验证一个用 PROMELA 语言描述的系统的正确性。Spin 通过穷举一个系统的全部可能状态，来验证这个系统是否会出现死锁、活锁等情况。这些验证工作依赖于用户在 PROMELA 代码中通过使用一些特殊的标号(如 *end-state-*，*acceptance-*和 *progress* 等)来指定系统的有效状态。此外，Spin 还可以验证一些时序逻辑(Temporal Logic)的公式[83]。只是在每次验证的过程中只能指定一个时序逻辑公式。所有的验证工作都是采用 on-the-fly 的方式，即在检查错误的时候无须产生系统整个的状态机。当然，在需要验证系统正确性时仍需要产生整个系统的状态机。

在 Spin 的验证部分采用了不同的状态空间生成算法。包括通常使用的深度优先搜索(Depth-First Search)、Holzmann 的 Supertrace bitspace 算法以及半序方法(Partial-order Method)等。其目的在于提高系统所能处理的问题的规模。

4.2 多重通信协议的模型

一个协议的规范(Specification)中包括了五个独立的部分。协议的这五个要素是：协议所提供的服务(Service)、协议所执行的环境的假设(Assumptions)、实现协议的消息词表(Vocabulary)、词表中每个消息的编码/格式(Encoding/Format)、保证协议一致性的过程规则(Procedure Rules)。其中第五个部分是一个协议中最难设计也是最难验证的部分。在本文中，我们首先建立通信协议的 CSP 模型，然后详细描述协议的规范。文中采用 PROMELA 语言描

述了系统的过程规则，并采用了 2000 年推出的 Spin 3.4.4 和 XSpin 3.4.2 来描述和验证过程规则。

4.2.1 基于共享内存的通信协议模型

基于共享内存的通信协议的 CSP 模型

为了建立基于共享内存的通信协议的 CSP 模型，我们需要分析相关的事件。对于一个通信协议，我们把它分成若干进程(Process)。首先分析与用户交互的表示层进程 $PRES_i$。在 $PRES_i$ 上发生的事件集定义为：

$$\begin{aligned}\alpha PRES_i = \{&send.i?dst(length),\ srq.i!dst(length),\\ &sevent.i?ack,\ spoll.i!ack,\\ &revent.i?ack,\ rpoll.i!ack\}\end{aligned}$$

其中 *send.i?dst(length)* 表示接收到用户进程的一个消息发送请求；*srq.i!dst(length)*表示开始发送消息；*sevent.i?ack* 表示等待消息发送完成事件；*spoll.i!ack* 表示向用户进程发出消息发送完成事件；*revent.i?ack* 表示等待消息接收完成事件；*rpoll.i!ack* 表示向用户进程发出消息接收完成事件。

表示层进程 $PRES_i$ 包括了消息发送、等待消息发送完成和消息接收(即等待消息接收完成)三个流程。消息发送从用户进程中接收一个消息发送请求，并启动消息发送(向共享内存进程 SM_i 发出消息发送请求)。等待消息发送完成的流程等待消息发送完成事件的产生，并向用户进程发出消息发送完成事件。消息接收等待消息接收完成事件的产生，并向用户进程发出消息接收完成事件。其形式化的表示如下：

$$\begin{aligned}PRES_i = &((send.i?dst(length) \rightarrow srq.i!dst(length)) \mid\\ &(sevent.i?ack \rightarrow spoll.i!ack) \mid\\ &(revent.i?ack \rightarrow rpoll.i!ack))\\ &\rightarrow PRES_i\end{aligned}$$

与表示层进程 $PRES_i$ 交互操作的是共享内存进程 SM_i。在 SM_i 上发生的事件集定义为：

$$\begin{aligned}\alpha SM_i = \{&srq.i?dst(data),\ ibq.dst!data(i,\ dst,\ length),\ sevent.i!ack,\\ &ibq.i?data(src,\ dst,\ length),\ revent.i!ack\}\end{aligned}$$

其中，*srq.i?dst(data)*表示从表示层 $PRES_i$ 接收到一个消息发送请求；*ibq.dst!data(i, dst, length)*表示向数据交换队列写入数据(消息发送)；*ibq.i?data(src, dst, length)*表示从数据交换队列中读出数据(消息接收)；*sevent.i!ack* 表示向表示层 $PRES_i$ 返回消息发送完成事件；*revent.i!ack* 表示向表示层 $PRES_i$ 返回消息接收完成事件。

共享内存 SM_i 包括了消息发送和消息接收两个流程。消息发送流程从表示层进程 $PRES_i$ 中接收一个消息发送请求，并向数据交换队列传输数据，然后向表示层 $PRES_i$ 返回消息发送完成事件。消息接收流程等从数据交换队列中接收数据，并向表示层 $PRES_i$ 发出消息接收完成事件。其形式化的表示如下：

$$SM_i = ((srq.i?dst(length) \rightarrow ibq.dst!data(i, dst, length) \rightarrow sevent.i!ack) \mid (ibq.i?data(src, dst, length) \rightarrow revent.i!ack)) \rightarrow SM_i$$

基于共享内存的消息通信协议表示为 *BCLINTRA*。它由多个结点上的进程 SMP_i 并发执行构成。每个结点进程 SMP_i 由其表示层 $PRES_i$ 和共享内存 SM_i 并发执行构成。其形式化的表示如下：

$$\alpha SMP_i = \{send.i?dst(length), srq.i!dst(length), sevent.i?ack, spoll.i!ack, revent.i?ack, rpoll.i!ack, srq.i?dst(length), ibq.dst!data(i, dst, length), sevent.i!ack, ibq.i?data(src, dst, length), revent.i!ack\}$$

$$\alpha BCLINTRA = \bigcup_{i=1}^{n} \alpha SMP_i$$

$$SMP_i = PRES_i \parallel SM_i$$

$$BCLINTRA = SMP_1 \parallel SMP_2 \parallel \ldots \parallel SMP_n$$

验证其无死锁的条件是：

$$\forall tr \in traces(BCLINTRA).(BCLINTRA / tr) \neq STOP \quad (4.1)$$

即

$$BCLINTRA\ sat\ \forall tr \in traces(BCLINTRA).(BCLINTRA / tr) \neq STOP \quad (4.2)$$

上式表示从 *BCLINTRA* 中去除其任意踪迹后该进程不等于停止进程 *STOP*。即 *BCLINTRA* 进程永远不会停止。

基于共享内存的通信协议的验证

为了验证协议的正确性，我们采用 PROMELA 语言描述了整个系统，并用 Spin 来进行验证。下面，我们将协议按照其五个组成部分分别进行描述。

- 服务说明(Service Specification)

基于共享内存的通信协议的目的是提供可靠有序的 SMP 结点内部进程间的消息传递。由于系统采用基于共享内存机制，可以认为在进行数据传递的过程中不会出现丢包、破坏包等情况。

- 环境假设(Assumptions About the Environment)

协议执行的环境包括2个或2个以上进程和连接这些进程的共享内存空间。每个进程可以假设不断地进行消息发送，并探询和接收到达的消息。消息在传输的过程中不会被损坏、丢失、复制、插入或乱序。

- 协议词表(Protocol Vocabulary)

由于共享内存保证消息传递的可靠性，在协议词表中只需定义一种消息：*data* 用来表示传输数据消息。协议词表定义如下：

```
V = {data}
```

- 消息格式(Message Format)

每一个消息由一个标识消息类型的控制域(Control Field)、一组地址域(Address Fields)、一个序号域(Sequential Number Field)和一个数据域(Data Field)组成。在这里，为了简化模型，我们可以认为消息的控制域和数据域都是定长的。每一个消息可以被看作如下结构：

```
{tag, src, dst, length, seqno, payload}
```

采用类 C 语言描述如下：

```
enum control {data};
struct message {
    enum control tag;
    byte src, dst;
```

```
        byte length;
        byte seqno;
        byte payload[MAX_PAYLOAD_LEN];
    };
```

- 过程规则(Procedure Rules)

按照协议的 CSP 模型，我们将协议分为几个层(Layer)。用户进程直接使通信协议的 API(表示层，Presentation Layer)提供的服务。在这一层中，用户进程向通信协议发出请求(消息发送/接收等)，并等待协议的反馈(消息发送/接收完成等)。在表示层之下是通信协议的具体实现(会话层，Session Layer)，在这一层上实现协议的具体过程。对于基于共享内存的通信协议来说，数据直接进行内存拷贝，不再通过网络通信设备进行传输。这样，就不需要数据链路层(Data-Link Layer)。使用共享内存进行数据传递具体过程将包含在会话层中。

✧ 表示层(Presentation Layer)

表示层提供与用户进程的接口。用户进程通过向表示层提交消息发送请求(SReq)来发送消息，并通过探询消息发送完成事件(SEvent)来确定消息发送的完成。用户进程通过探询消息接收完成事件(REvent)来接收消息。

表示层可以用以下的 PROMELA 语言描述：

```
do
:: SReq[n]?dst, length -> SRQ[n]!dst, length;
:: SEvent[n]?tag -> SPoll[n]!tag;
:: REvent[n]?tag -> RPoll[n]!tag;
od;
```

✧ 会话层(Session Layer)

会话层实现了基于共享内存通信协议的主体结构。在消息的发送方消息被打包后，送入数据交换队列(IBQ，Internal Buffer Queue)，并产生发送完成事件。用 PROMELA 语言描述如下：

```
:: SRQ[n]?dst, length ->
    IBQ[n]!data, n, dst, length;
    SEvent[n]!ack;
```

在消息的接收方，当从数据交换队列(IBQ)中接收到数据时，直接生成消息接收完成事件。用 PROMELA 语言描述如下：

```
:: IBQ[n]?data(src, dst, length) ->
   REvent[n]!ack;
```

4.2.2 半用户级通信协议模型

半用户级通信协议的 CSP 模型

为了建立半用户级通信协议的 CSP 模型，我们需要分析相关的事件。对于一个通信协议，我们把它分成若干进程(Process)。首先分析与用户交互的表示层进程 $PRES_i$。在 $PRES_i$ 上发生的事件集定义为：

$$\alpha PRES_i = \{send.i?dst(length),\ srq.i!dst(length),\ sevent.i?ack,\ spoll.i!ack,\ revent.i?ack,\ rpoll.i!ack\}$$

其中 *send.i?dst(length)* 表示接收到用户进程的一个消息发送请求；*srq.i!dst(length)*表示开始发送消息；*sevent.i?ack* 表示等待消息发送完成事件；*spoll.i!ack* 表示向用户进程发出消息发送完成事件；*revent.i?ack* 表示等待消息接收完成事件；*rpoll.i!ack* 表示向用户进程发出消息接收完成事件。

表示层进程 $PRES_i$ 包括了消息发送、等待消息发送完成和消息接收(即等待消息接收完成)三个流程。消息发送从用户进程中接收一个消息发送请求，并启动消息发送(向网络会话层进程 $SESS_i$ 发出消息发送请求)。等待消息发送完成的流程等待消息发送完成事件的产生，并向用户进程发出消息发送完成事件。消息接收等待消息接收完成事件的产生，并向用户进程发出消息接收完成事件。其形式化的表示如下：

$$PRES_i = ((send.i?dst(length) \rightarrow srq.i!dst(length)) \mid (sevent.i?ack \rightarrow spoll.i!ack) \mid (revent.i?ack \rightarrow rpoll.i!ack)) \rightarrow PRES_i$$

与表示层进程 $PRES_i$ 交互操作的是会话层进程 $SESS_i$。在 $SESS_i$ 上发生的事件集定义为：

$$\alpha SESS_i = \{seqno.i!v,\ seqno.i?v,\ expectedseqno.i!v,\ expectedseqno.i?v,$$
$$srq.i?dst(length),\ srec.i!dst(length, v),\ srec.i?dst(length, v),$$
$$OFIFO.i!(tag, src, dst, length, in\_seqno),$$
$$IFIFO.i?(tag, src, dst, length, in\_seqno),$$
$$sevent.i!ack,\ revent.i!ack,$$
$$syserr.i!(tag, src, dst, length, in\_seqno),$$
$$syserr.i?(tag, src, dst, length, in\_seqno)\}$$

其中，*seqno.i!v*、*seqno.i?v*、*expectedseqno.i!v* 和 *expectedseqno.i?v* 用于计算发送消息序号和接收消息预期序号；*srq.i?dst(length)*表示从表示层读出一个消息发送请求；*srec.i!dst(length, v)*和 *srec.i?dst(length, v)*用于记录正在进行的消息传输任务；*OFIFO.i!(tag, src, dst, length, in_seqno)*表示向消息输出队列发送数据；*IFIFO.i?(tag, src, dst, length, in_seqno)*表示从消息输入队列读出数据；*sevent.i!ack* 和 *revent.i!ack* 表示向表示层发出消息发送完成事件和消息接收完成事件；*syserr.i!(tag, src, dst, length, in_seqno)*和 *syserr.i?(tag, src, dst, length, in_seqno)*用于模拟消息包出错的情形。

会话层 $SESS_i$ 首先初始化发送消息的序号(seqno)和接收消息的预期序号(expectedseqno)。然后根据发生的事件分别按消息发送流程 $SSEND_i$、接收消息数据流程 $SRDATA_i$、接收消息应答流程 $SRACK_i$ 执行。为了模拟通信过程中的丢包现象，这里增加了用于通知发送方程序重发消息的通道 *syserr.i*。

消息发送流程 $SSEND_i$ 读出消息发送请求，然后加上消息序号后发送到消息数据输出队列(OFIFO)中；接收消息数据流程 $SRDATA_i$ 首先匹配消息序号，若序号匹配，则生成消息接收完成事件和应答消息。若序号不匹配，则直接丢弃该消息，并模拟出错情况给消息发送方发出通知；接收消息应答流程 $SRACK_i$ 接收到应答消息后，返回消息发送完成事件。此外，当会话层发现数据错误时(从 *syserr.i* 通道接收到事件)，则重发该消息包。

会话层的形式化的表示如下：

$$SESS_i = seqno.i!0 \rightarrow expectedseqno.i!0 \rightarrow$$
$$\mu X((srq.i?dst(length)$$
$$\rightarrow SSEND_i(dst, length)) \mid$$
$$(IFIFO.i?data(src, dst, length, in\_seqno)$$

$\rightarrow SRDATA_i(src, dst, length, in\_seqno))$ |

$(IFIFO.i?ack(src, dst, length, in\_seqno)$

$\rightarrow SRACK_i(src, dst, length, in\_seqno))$ |

$(syserr.i?(tag, src, dst, length, in\_seqno)$

$\rightarrow OFIFO.i!(tag, src, dst, length, in\_seqno)))$

$\rightarrow X)$

$SSEND_i(dst, length) = seqno.i?v$

$\rightarrow seqno.i!(v+1)$

$\rightarrow srec.i!dst(length, v)$

$\rightarrow OFIFO.i!data(i, dst, length, v)$

$SRDATA_i(src, dst, length, in\_seqno) =$

$expectedseqno.i?v$

$\rightarrow(if\ v = in\_seqno\ then\ (revent.i!ack$

$\rightarrow OFIFO.i!ack(i, src, 0, in\_seqno)$

$\rightarrow expectedseqno.i!(v+1))$

$else\ (expectedseqno.i!(v)$

$\rightarrow syserr.src!data(src, dst, length, in\_seqno)))$

$SRACK_i(src, dst, length, in\_seqno) = sevent.i!ack \rightarrow srec.i?dst(length, v)$

在会话层之下是数据链路层 DLL_i。在 DLL_i 上发生的事件集定义为：

$\alpha DLL_i = \{IFIFO.i!(tag, src, dst, length, seqno),$

$OFIFO.i?(tag, src, dst, length, seqno),$

$syserr.i!(tag, src, dst, length, seqno)$

$link.i!(tag, src, dst, length, seqno),$

$link.i?(tag, src, dst, length, seqno)\}$

其中，*IFIFO.i!(tag, src, dst, length, seqno)*表示向消息输入队列送入数据；*OFIFO.i?(tag, src, dst, length, seqno)*表示从消息输出队列中读出数据；*link.i!(tag, src, dst, length, seqno)*和 *link.i?(tag, src, dst, length, seqno)*表示消息包在网络上的传输。*syserr.i!(tag, src, dst, length, seqno)*表示出现错误包。

在数据链路层主要处理数据的传输过程。即从消息输出队列中读出数据后

送入互连网络进行传输；从互连网络上读出数据后转发至消息输入队列。其形式化的表示如下：

$$
\begin{aligned}
DLL0_i = ((&OFIFO.i?(tag, src, dst, length, seqno)\\
&\rightarrow link.i!(tag, src, dst, length, seqno)) \mid\\
&(link.i?(tag, src, dst, length, seqno)\\
&\rightarrow IFIFO.i!(tag, src, dst, length, seqno)))\\
&\rightarrow DLL_i
\end{aligned}
$$

$$
\begin{aligned}
DLL1_i = ((&OFIFO.i?(tag, src, dst, length, seqno)\\
&\rightarrow syserr.i!(tag, src, dst, length, seqno)) \mid\\
&(link.i?(tag, src, dst, length, seqno)\\
&\rightarrow syserr.src!(tag, src, dst, length, seqno)))\\
&\rightarrow DLL1_i
\end{aligned}
$$

$$DLL_i = DLL0 \,\square\, DLL1$$

其中，$DLL0_i$表示正常的物理链路，$DLL1_i$表示出错(数据未能正常传递)的情况。由 $DLL0_i$和 $DLL1_i$通过非确定算子操作构成的 DLL_i描述了一个不可靠(可能丢包)的数据链路。

半用户级消息通信协议表示为 *BCLINTER*。它由多个结点上的进程 $NODE_i$并发执行构成。每个结点进程 $NODE_i$由其表示层 $PRES_i$、会话层 $SESS_i$、数据链路层 DLL_i并发执行构成。其形式化的表示如下：

$$
\begin{aligned}
\alpha NODE_i = \{&send.i?dst(length), srq.i!dst(length), srq.i?dst(length),\\
&spoll.i!ack, rpoll.i!ack\\
&sevent.i!ack, sevent.i?ack, revent.i!ack, revent.i?ack,\\
&seqno.i!v, seqno.i?v, expectedseqno.i!v, expectedseqno.i?v,\\
&srec.i!dst(length, v), srec.i?dst(length, v),\\
&OFIFO.i!(tag, src, dst, length, in\_seqno),\\
&OFIFO.i?(tag, src, dst, length, seqno),\\
&IFIFO.i?(tag, src, dst, length, in\_seqno),\\
&IFIFO.i!(tag, src, dst, length, seqno),\\
&syserr.i!(tag, src, dst, length, in\_seqno),\\
&syserr.i?(tag, src, dst, length, in\_seqno)
\end{aligned}
$$

link.i!(tag, src, dst, length, seqno),
link.i?(tag, src, dst, length, seqno)}

$$\alpha BCLINTRA = \bigcup_{i=1}^{n} \alpha NODE_i$$

$$NODE_i = PRES_i \,||\, SESS_i \,||\, DLL_i$$

$$BCLINTRA = NODE_1 \,||\, NODE_2 \,||\, ... \,||\, NODE_n$$

验证其无死锁的条件是：

$$\forall tr \in traces(BCLINTER).(BCLINTER / tr) \neq STOP \tag{4.3}$$

即

$$BCLINTER\ sat\ \forall tr \in traces(BCLINTER).(BCLINTER / tr) \neq STOP \tag{4.4}$$

上式表示从 *BCLINTER* 中去除其任意踪迹后该进程不等于停止进程 *STOP*。即 *BCLINTER* 进程永远不会停止。

半用户级通信协议的验证

为了验证协议的正确性，我们采用 PROMELA 语言描述了整个系统，并用 Spin 来进行验证。下面，我们将协议按照其五个组成部分分别进行描述。

- 服务说明(Service Specification)

半用户级通信协议的目的是提供可靠有序的点到点的消息传递。协议需要提供对传输过程中出现的错误包(如 CRC 错等)进行纠正。

- 环境假设(Assumptions About the Environment)

协议执行的环境包括 2 个或 2 个以上结点和连接这些结点的物理通道。每个结点可以假设不断地进行消息发送，并探询和接收到达的消息。消息在传输的过程中可以被任意地损坏，但不能被丢失、复制、插入或乱序。传输过程中的所有的错误都可以被检测出来。控制消息在传输的过程中不会被损坏或丢失等。

- 协议词表(Protocol Vocabulary)

协议词表定义了三种不同的消息：*data* 表示数据消息；*ack* 表示正常接收的应答消息；*err* 表示出错的消息包。协议词表定义如下：

```
V = {data, ack, err}
```

每一个消息类型可以更进一步被细化为一组子类(Sub-Type)的低层的消息。

- 消息格式(Message Format)

每一个消息由一个标识消息类型的控制域(Control Field)、一组地址域(Address Fields)、一个序号域(Sequential Number Field)和一个数据域(Data Field)组成。在这里，为了简化模型，我们可以认为消息的控制域和数据域都是定长的。每一个消息可以被看作如下结构：

```
{tag, src, dst, length, seqno, payload}
```

采用类 C 语言描述如下：

```
enum control {data, ack, err};
struct message {
    enum control tag;
    byte src, dst;
    byte length;
    byte seqno;
    byte payload[MAX_PAYLOAD_LEN];
};
```

- 过程规则(Procedure Rules)

为了更好地描述整个协议，我们将协议分为几个层(Layer)。用户进程直接使通信协议的 API(表示层，Presentation Layer)提供的服务。在这一层中，用户进程向通信协议发出请求(消息发送/接收等)，并等待协议的反馈(消息发送/接收完成等)。在表示层之下是通信协议的具体实现(会话层，Session Layer)，在这一层上实现协议的具体过程，包括流量控制等功能。最底层是具体的硬件通信结构(数据链路层，Data-Link Layer)。在这一层主要描述硬件的特性。

✧ 表示层(Presentation Layer)

表示层提供与用户进程的接口。用户进程通过向表示层提交消息发送请求

(SReq)来发送消息，并通过探询消息发送完成事件(SEvent)来确定消息发送的完成。用户进程通过探询消息接收完成事件(REvent)来接收消息。

表示层可以用以下的 PROMELA 语言描述：

```
:: SReq[n]?dst, length -> SRQ[n]!dst, length;
:: SEvent[n]?tag -> SPoll[n]!tag;
:: REvent[n]?tag -> RPoll[n]!tag;
```

✧ 会话层(Session Layer)

会话层实现了通信协议的主体结构。在消息的发送方消息被打包、加上序号(seqno)标识后，送入消息输出队列(OFIFO)，并记录在消息发送记录(SRec)中等待消息接收方的应答消息。用 PROMELA 语言描述如下：

```
:: SRQ[n]?dst, length ->
   SRec[n]!dst, length, seqno[n];
   OFIFO[n]!data, n, dst, length, seqno[n];
   seqno[n] = (seqno[n] + 1) % MAX_SEQ_NUM;
```

在消息的接收方，当从消息输入队列(IFIFO)中接收到数据时，首先判断其序号(seqno)是否与预期序号(expected_seqno)相符。若序号相符，则接收消息，返回消息接收后的应答消息(ack)，并生成消息接收完成事件。否则，直接丢弃该消息。当从消息输入队列(IFIFO)中接收到应答消息时，产生相应的事件，并删除在消息发送记录(SRec)中对应的记录。用 PROMELA 语言描述如下：

```
:: IFIFO[n]?data(src, dst, length, in_seqno) ->
   if
   :: (in_seqno == expected_seqno[src]) ->
      REvent[n]!ack;
      OFIFO[n]!ack(n, src, 0, in_seqno);
      expected_seqno[src] =
          (expected_seqno[src] + 1) % MAX_SEQ_NUM;
   :: else -> skip;
   fi;
:: IFIFO[n]?ack(src, dst, length, in_seqno) ->
   SEvent[n]!ack;
```

```
SRec[n]?dst, length, in_seqno;
```

当在数据链路层发生数据错时，根据假设，系统将会检测到该错误。在这里，我们通过从 SYSERR 通道中读出出错包的信息，并进行处理。为此，我们在丢包的处理程序中加入如下代码来完成错误的通知：

```
SYSERR[src]!tag, src, dst, length, in_seqno;
```

当出现错误后，通信协议将重发该消息包。为此，我们在消息发送的过程中加入了重发机制：

```
:: SYSERR[n]?tag, src, dst, length, in_seqno ->
   OFIFO[n]!tag, src, dst, length, in_seqno;
```

✧ 数据链路层(Data-Link Layer)

数据链路层实现数据的物理传输。它从消息输出队列(OFIFO)中读出要发送的消息包，并将其发送到互连网络(link)上去。在接收方从互连网络上接收到消息包，再将消息包存储于消息输入队列(IFIFO)中。

数据链路层可以用以下的 PROMELA 语言描述：

```
do
:: OFIFO[n]?tag, src, dst, length, seqno ->
   link[dst]!tag, src, dst, length, seqno;
:: link[n]?tag, src, dst, length, seqno ->
   IFIFO[n]!tag, src, dst, length, seqno;
od;
```

为了模拟数据传输错误，我们增加了随机出错的程序：

```
:: OFIFO[n]?data(src, dst, length, seqno) ->
   if
   :: link[dst]!data(src, dst, length, seqno);
   :: SYSERR[src]!data(src, dst, length, seqno);
   /* lose */
   fi;
:: link[n]?data(src, dst, length, seqno) ->
   if
   :: IFIFO[n]!data(src, dst, length, seqno);
```

```
:: SYSERR[src]!data(src, dst, length, seqno);
/* lose */
fi;
```

4.2.3 多重通信协议的组合模型

多重通信协议的 CSP 模型

我们可以把基于共享内存的通信协议的 CSP 模型和半用户级通信协议的 CSP 模型结合起来，建立多重通信协议的 CSP 模型。从前文中可以看出，多重通信协议实际上就是用相同的表示层将两个不同的协议结合在一起，在通信的低层上仍然各自使用各自的通信路径。为此，我们首先进行换名，即将基于共享内存的通信协议模型中与半用户级通信协议模型中重名的变量全部换名。在基于共享内存的通信协议模型中对重名变量加后缀 *0*；在半用户级通信协议模型中对重名变量加后缀 *1*。定义多重通信协议的表示层如下：

$$\alpha PRES_i = \{send.i?dst(length),\ spoll.i!ack,\ rpoll.i!ack\} \cup \alpha PRES0_i \cup \alpha PRES1_i$$

$$\begin{aligned} PRES_i = ((&send.i?dst(length) \rightarrow \\ &(if\ (dst\ within\ one\ node)\ then\ (send0!dst(length) \rightarrow PRES0_i) \\ &\quad else\ (send1!dst(length) \rightarrow PRES1_i)) \mid \\ &(spoll0.i?ack \rightarrow spoll.i!ack) \mid \\ &(spoll1.i?ack \rightarrow spoll.i!ack) \mid \\ &(rpoll0.i?ack \rightarrow rpoll.i!ack)) \mid \\ &(rpoll1.i?ack \rightarrow rpoll.i!ack)) \\ &\rightarrow PRES_i \end{aligned}$$

这时，多重通信协议可以表示为：

$$\alpha SMPNODE_i = \alpha PRES_i \cup \alpha SMP_i \cup \alpha NODE_i$$

$$\alpha BCL = \bigcup_{i=1}^{n} \alpha SMPNODE_i$$

$$SMPNODE_i = PRES_i \,||\, SM_i \,||\, SESS_i \,||\, DLL_i$$

$$BCL = SMPNODE_1 \,||\, SMPNODE_2 \,||\, ... \,||\, SMPNODE_n$$

验证其无死锁的条件是：

$$\forall tr \in traces(BCL).(BCL / tr) \neq STOP \tag{4.5}$$

即

$$BCL\ sat\ \forall tr \in traces(BCL).(BCL / tr) \neq STOP \tag{4.6}$$

上式表示从 *BCL* 中去除其任意踪迹后该进程不等于停止进程 *STOP*。即 *BCL* 进程永远不会停止。

多重通信协议的验证

为了验证协议的正确性，我们采用 PROMELA 语言描述了整个系统，并用 Spin 来进行验证。下面，我们将协议按照其五个组成部分分别进行描述。

- 服务说明(Service Specification)

多重级通信协议的目的是提供可靠有序的点到点的消息传递和 SMP 结点内部的消息传递。对于 SMP 结点内的消息传递，由于系统采用基于共享内存机制，可以认为在进行数据传递的过程中不会出现丢包、破坏包等情况。对于点到点的消息传递，协议需要提供对传输过程中出现的错误包(如 CRC 错等)进行纠正(假设所有的错误都可以被检测出来)。

- 环境假设(Assumptions About the Environment)

协议执行的环境包括 2 个或 2 个以上结点和连接这些结点的物理通道及单结点内 2 个或 2 个以上进程和连接这些进程的共享内存空间。每个结点及进程可以假设不断地进行消息发送，并探询和接收到达的消息。消息在 SMP 结点内传输的过程中不会被损坏、丢失、复制、插入或乱序。消息在结点间传输的过程中可以被任意地损坏，但不能被丢失、复制、插入或乱序。控制消息在传输的过程中不会被损坏或丢失等。

- 协议词表(Protocol Vocabulary)

协议词表定义了三种不同的消息：*data* 表示数据消息；*ack* 表示正常接收的应答消息；*err* 表示出错的消息包。协议词表定义如下：

```
V = {data, ack, err}
```

每一个消息类型可以更进一步被细化为一组子类(Sub-Type)的低层的消息。

- 消息格式(Message Format)

每一个消息由一个标识消息类型的控制域(Control Field)、一组地址域(Address Fields)、一个序号域(Sequential Number Field)和一个数据域(Data Field)组成。在这里，为了简化模型，我们可以认为消息的控制域和数据域都是定长的。每一个消息可以被看作如下结构：

```
{tag, src, dst, length, seqno, payload}
```

采用类 C 语言描述如下：

```
enum control {data, ack, err};
struct message {
    enum control tag;
    byte src, dst;
    byte length;
    byte seqno;
    byte payload[MAX_PAYLOAD_LEN];
};
```

- 过程规则(Procedure Rules)

根据协议的组成，协议的过程规则可以基于共享内存的通信协议过程规则和半用户级通信协议的协和规则合并而成。在这里，为此，我们首先进行换名，即将基于共享内存的通信协议的过程规则中与半用户级通信协议的过程规则中重名的变量全部换名。在基于共享内存的通信协议的过程规则中对重名变量加后缀 0；在半用户级通信协议的过程规则中对重名变量加后缀 1。

表示层提供与用户进程的接口。用户进程通过向表示层提交消息发送请求(SReq)来发送消息，并通过探询消息发送完成事件(SEvent)来确定消息发送的完成。用户进程通过探询消息接收完成事件(REvent)来接收消息。表示层接收到消息请求后根据消息的目的地址确定通过共享内存进行 SMP 结点内部消息通信或通过互连网络进行 SMP 结点间消息通信。为了简化描述，我们定义了函数

SameNode(src，dst)来判断 src 与 dst 是否在同一个 SMP 结点。

多重通信协议的表示层可以用以下的 PROMELA 语言描述：

```
:: SReq[n]?dst, length ->
   if
   :: (SameNode(n, dst)) -> SReq0[n]!dst, length;
   :: else -> SReq1[n]!dst, length;
   fi;
:: SPoll0[n]?tag -> SPoll[n]!tag;
:: SPoll1[n]?tag -> SPoll[n]!tag;
:: RPoll0[n]?tag -> RPoll[n]!tag;
:: RPoll1[n]?tag -> RPoll[n]!tag;
```

第五章 多重通信协议的评价与性能分析

性能是评价一个通信协议的重要手段。本章给出了在曙光 3000 上实现的多重通信协议——BCL-3 的通信性能。同时，为了评价系统整体运行的性能，本章还给出了一组常用的集合操作(Collective Operation)的性能。此外，本章还结合 LogGP 模型给出了通信性能的分析，以及集合操作性能的理论分析。

5.1 机群通信系统评价标准与测试环境

5.1.1 机群通信系统评价标准

机群通信系统主要的功能就是在各处理结点上应用程序的进程之间提供高效、可靠和有序的通信服务。对于一个机群通信系统，我们应从以下四个方面进行评价：机群通信系统的点到点通信性能、机群通信系统的均衡通信性能、机群通信系统的可扩展性和机群通信系统的高可用性。其中前两点是机群通信系统的性能指标，后两点是机群通信系统的功能特性。

对于机群通信系统的点到点通信性能和均衡通信性能，我们采用了一组基准程序(Benchmark)进行测试。对于系统的可扩展性，这里着重从功能的角度进行分析。系统的高可用性则通过一组模拟的故障程序来验证。

点到点通信性能

机群通信系统的点到点通信性能是表征一个机群性能的最基本的指标。它包括点到点的通信延迟和通信带宽。点到点通信是机群通信系统的基础，好的点到点通信性能是一个高性能机群通信系统的必须具备的条件。为了测试 BCL-3 的点到点通信性能，我们采用了一组基准程序：

- 通信延迟(Ping-pong 测试)

测量通信延迟的基准程序采用 Ping-pong 测试，即开始计时后由进程 A 向进程 B 发送消息，当进程 B 接收到消息后；向进程 A 返回一个相同的消息；

当进程 A 接收到由进程 B 发来的消息后停止计时。所测得的时间为往返延迟(*RTT*，Round-Trip Time)。单向延迟为往返延迟的½。

- 通信带宽(单向带宽测试)

测量通信带宽的基准程序采用单向带宽测试，即开始计时后由进程 A 向进程 B 不断发送消息；发送一定量消息后由进程 B 返回应答消息；进程 A 接收到应答消息后停止计时。通信带宽为所发送消息量与时间的比值。

- LogGP 模型参数(分段延迟测试)

LogGP 是描述一个通信过程的一种模型。它的五个参数中除了处理器数(*P*)、大消息字节间隔(*G*)外，其它的三个参数都是用来描述通信路径上的开销的。处理器数(*P*)由系统的配置确定；大消息字节间隔(*G*)可由通信带宽确定；其它的三个参数：网络延迟(*L*)、通信开销(*o*)和通信间隔(*g*)则通过一组基准程序进行测试。具体测试方法及模型的含义参见第5.2.1节。

均衡通信性能

点到点通信性能只能反映一个通信系统的基本通信性能。而对于一个机群通信系统，整个机群通信网络的总体性能才是衡量其通信性能的主要标准。当通信网络上充满消息时，消息传输间的相互干扰有可能会增加整个网络的通信延迟和降低整个网络的通信带宽。一些较复杂的通信模式的性能的低下有时会导致实际应用程序的运行效率的降低。

为了测量系统在不同通信模式下的性能，我们采用了一组基于 BCL-3 的 MPI/PVM 集合操作(Collective Operation)测试程序来进行测试。

可扩展性

机群系统的可扩展性首先就体现在它的机群通信系统的可扩展性上，没有机群通信系统的可扩展就谈不上机群系统的可扩展。机群通信系统的可扩展性主要包括三个方面的含义：

- 通信网络硬件物理上可扩展

这要求互连网络硬件模块化。目前使用的可扩展高性能网络通常都是由若干个交换设备构成，通过改变互连交换设备的数量就可实现网络物理上的扩展。

- **通信网络性能可扩展**

具体含义是随着网络规模的扩展，通信的累积带宽相对结点机个数最好呈线性增长，而通信延时要保持缓慢而有限的增长。

- **通信软件可扩展**

这主要体现在通信软件对一些硬件资源的需求不要随系统规模的扩大而无限制地快速增长。由于硬件资源是有限的，对它需求的增长势必造成对系统扩展规模的限制。这是通信软件设计实现中要考虑的一个重要因素。

在曙光 3000 的实现中，我们采用了 Myricom 公司生产的网络互连硬件。在后续章节中我们将对其硬件的互连方式进行分析。对于互连网络性能的可扩展性我们通过通信网络的互连方式和互连网络的等分带宽(Bisection Bandwidth)的测量进行分析。通信软件则通过分析其结构来说明其可扩展性。

高可用性

机群通信系统的高可用性体现在，当系统中部分结点或网络因出现故障而无法正常工作时，不会影响通信在其它结点上的正常进行，并且在故障结点修复后能恢复正常的消息通信。这对机群通信系统提出了三点要求：一是保证残存在网络上的与故障结点有关的失效消息包会被硬件或软件丢弃，不会造成网络阻塞；二是提供必要的网络状态监控服务，以便能及时发现故障点，取消向故障点发送后续的消息包并实施必要的恢复和处理操作；三是能够方便地恢复故障结点或网络。

BCL-3 提供了一组结点及网络状态的监控函数(程序)。通过执行监控程序或调用监控函数可以获得(控制)结点的状况。在这里，我们通过一组故障模拟程序来验证系统在出现错误时的可用性。这些程序包括：网络故障模拟程序和结点故障模拟程序。

5.1.2 测试环境

本章中的测试采用了曙光 3000 超级服务器作为测试平台。曙光 3000 系统共有 64 个计算节点，配置 IBM 4-way Power3-II 375MHz 处理器，2GB 内存。计算结点使用 AIX 4.3.3 操作系统。

计算结点之间通过高速网络互连。互连网络采用 Myricom 生产的 M2M-

PCI64A-2 网络接口卡和 M2M-OCT-SW8 交换设备。其拓扑结构如图 5.1所示。

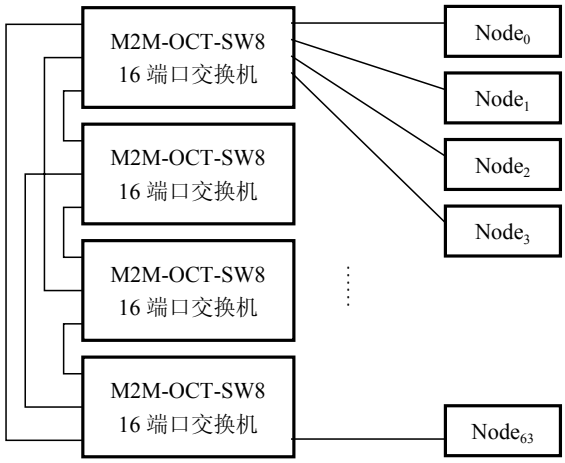

图5.1 曙光 3000 拓扑结构

表5.1 测试平台的结点性能

	性能指标
每结点上处理器数	4
处理器类型	Power3-II
处理器主频	375MHz
结点内存	2GB
结点内存拷贝带宽 (无 Cache 影响)	303.4MB/s
进入核心操作	2.05μs

表5.2 Myrinet 互连硬件性能

	性能指标
NIC 64 位 I/O 读时间	0.98μs
NIC 64 位 I/O 写时间	0.24μs
NIC DMA 读启动时间	2.21μs
NIC DMA 读带宽	232.28MB/s
NIC DMA 写启动时间	2.57μs
NIC DMA 写带宽	262.36MB/s
Switch 理论互连带宽	160MB/s

每一个 SMP 结点由 4 个处理器通过内存总线互连，处理器间采用共享内存机制。结点间通过连接在 I/O 总线上的 NIC 互连，系统通过 4 个 Myricom 的 16

端口交换机将 64 个结点互连在一起。

表 5.1给出了每个结点的性能参数。表 5.2给出了 Myrinet 互连硬件的性能参数。

在测试的过程中，数据长度以 1024 字节(Byte)作为 1KB；1024KB 作为 1MB。时间的测度均以 *gettimeofday()*函数获取，精确到 1μs(=10^{-6}s)。计时函数的开销通过重复执行被测程序来消除。

5.2 点到点通信性能

点到点通信性能是描述一个通信系统的关键指标之一。在这一节中，我们首先给出了通信系统的 LogGP 模型，然后给出了 BCL-3 的测试数据。

5.2.1 通信的 LogGP 模型及性能参数

最简单的定义通信性能的参数就是进行一次消息发送与接收所需要的时间。美国 California 大学 Berkeley 分校的 Culler 等人提出的 LogP 模型[85]和 California 大学的 Alexandrov 等人对 LogP 模型扩展而提出的 LogGP[87][88]模型能够很好的描述这样的通信过程。

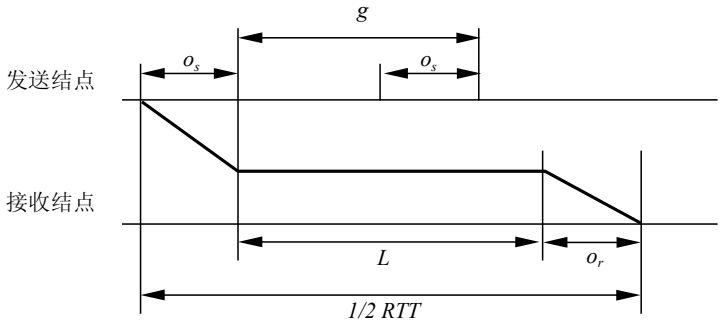

图5.2 通信系统的 LogP 模型

在进行定长的小消息通信时，LogP 模型用以下四个参数来评价一个机群系统的通信性能：

L(Latency)：消息在通信网络(通信物理线路)上传输时间的上限

o(Overhead)：处理器为发送和接收一个消息所用的开销

g(Gap)：一处理器连续两次的消息发送或接收之间的最小间隔时间

p(Processor)：互连网络连接的处理器个数

通常在描述一个通信系统时，我们会将 LogP 模型中的 *o* 分解为发送开销 o_s 和接收开销 o_r 两个部分(图 5.2)。发送开销 o_s 是在进行消息发送时为了将消息送入互连网络进行发送而需要的处理器处理时间；接收开销 o_r 是在进行消息接收时将消息从互连网络上接收下来所需要的处理器处理时间。LogGP 模型在 LogP 模型的基础上定义了参数 *G*，用来表示在长消息传递时连续两个字节之间的间隔时间，它等效于通信带宽的倒数。

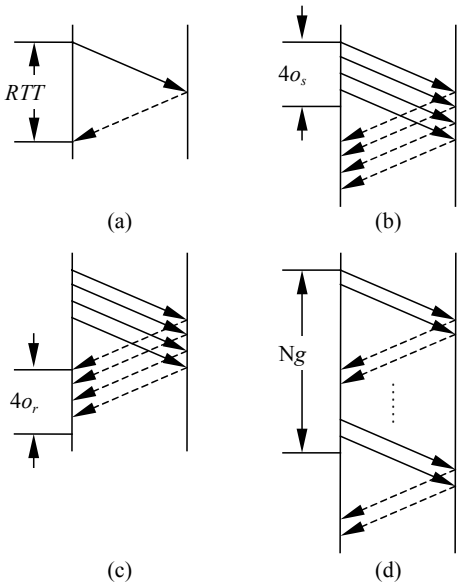

图5.3 LogP 模型中的性能参数的测量方法

在本章中我们测量了 BCL-3 的 LogP 模型的各个参数。图 5.3给出了测量这些参数的方法[89]。首先通过 Ping-pong 测试测量出通信的往返延迟(*RTT*)。然后，分段测出通信的发送开销 o_s 和接收开销 o_r 以及参数 *g*。最后，根据下式计算得出通信网络延迟 *L*。

$$L = \frac{RTT}{2} - o_s - o_r \tag{5.1}$$

发送开销 o_s 和参数 *g* 测量方法的不同在于，发送开销 o_s 只使用有限次(较少次数)的发送过程进行测量，从而避免计入因系统繁忙而导致的开销，只统计相关发送过程的处理器开销。而参数 *g* 重复较多次的发送过程，从而得出当发送大量消息时系统的开销，计算两次发送间的最小间隔。

5.2.2 性能数据

多重通信协议的点到点通信性能从四个方面进行测试：通信协议的 LogGP 模型参数、BCL-3 点到点的通信性能、基于 BCL-3 的 MPI 的点到点通信性能和基于 BCL-3 的 PVM 的点到点通信性能。其中，第一组数据建立了多重通信协议通信过程的理论模型，而后三组数据则给出了通信协议在实际应用中常用到的一些功能的性能指标。

为了减少测量过程中造成的误差，通信的点到点性能采用重复测量的方法。一方面通过在计时区间内重复同一操作来减少因计时函数的开销造成的误差。另一方面通过重复测量同一数据来减少因系统及网络负载的波动引起的偶然误差。对于重复测得的数据，保证其在 95%的可信度下，可以采用如下标准偏差形式。其计算公式如下：

算术平均值：

$$\bar{V} = \frac{1}{N} \sum_{i=1}^{N} V_i \tag{5.2}$$

算术平均值的标准偏差：

$$\Delta_V = \sqrt{\frac{\sum_{i=1}^{N} (\bar{V} - V_i)^2}{N(N-1)}} \tag{5.3}$$

我们将测量的数据表示为如下格式：

$$V = \bar{V} \pm \Delta_V \tag{5.4}$$

LogGP 模型参数

表 5.3给出了 BCL-3 在曙光 3000 上测试的性能参数。从通信延迟上看，结点内通信的速度高出结点间 6 倍左右。其性能指标为结点内通信延迟 2.726μs；结点间通信延迟 18.34μs。结点内通信的间隔时间也是结点间通信间隔的 1/4 左右。其性能指标为结点内通信间隔 3.120μs；结点间通信间隔 11.0911μs。通信带宽上结点内通信由于受到 Cache 的影响，其性能较表 5.1中给出的测试平台内存拷贝性能高，且测试数据误差较大。结点内通信带宽达到 391MB/s；结点间

通信带宽达到 146.412MB/s。

表5.3 BCL-3 的通信性能参数

性能参数	结点内通信 (基于共享内存的通信协议)	结点间通信 (半用户级通信协议)
网络延迟(L)	–0.14±0.03 μs	10.27±0.03 μs
发送开销(o_s)	1.85±0.03 μs	7.05±0.01 μs
接收开销(o_r)	1.0172±0.0002 μs	1.0172±0.0002 μs
间隔(g)	3.120±0.002 μs	11.0911±0.0006 μs
大消息字节间隔(G)	2.55±0.03 ns/B	6.8300±0.0002ns/B
带宽($1/G$)	391±5 MB/s	146.412±0.004 MB/s
单向延迟($RTT/2$)	2.726±0.001 μs	18.34±0.03 μs

结点内通信的延迟参数 L 的测量得出了负的结果，这表明了通信过程中发送和接收的重叠[90]。在这里，由于结点内部的通信实际上没有通信硬件(通信线路)上的延迟，因此测量值为负数。

BCL-3 的点到点通信性能

表 5.4给出了 BCL-3 的点到点通信的通信延迟数据。图 5.4和图 5.5分别给出了结点内通信延迟和结点间通信延迟的变化曲线。从图中可以看出结点间通信延迟在 64 字节长度消息时发生了跳变。这是因为 64 字节以内消息采用 I/O 方式进行主机与 NIC 的数据交换，而 64 字节以上消息采用 DMA 方式。DMA 方式的数据交换可以提高通信的带宽，但对通信延迟有一定的影响。对于结点内的通信过程，由于采用相同的内存拷贝机制，因此性能的变化曲线较平稳。

表5.4 BCL-3 的点到点通信延迟

消息长度(Byte)	结点内通信延迟(μs) (基于共享内存的通信协议)	结点间通信延迟(μs) (半用户级通信协议)
0	2.726±0.001	18.34±0.03
4	2.741±0.003	18.74±0.09
8	2.807±0.007	19.41±0.04
16	2.826±0.006	19.51±0.04
32	2.923±0.007	20.491±0.007
64	3.284±0.002	20.94±0.01
128	3.92±0.01	27.48±0.01
256	4.385±0.003	28.861±0.009

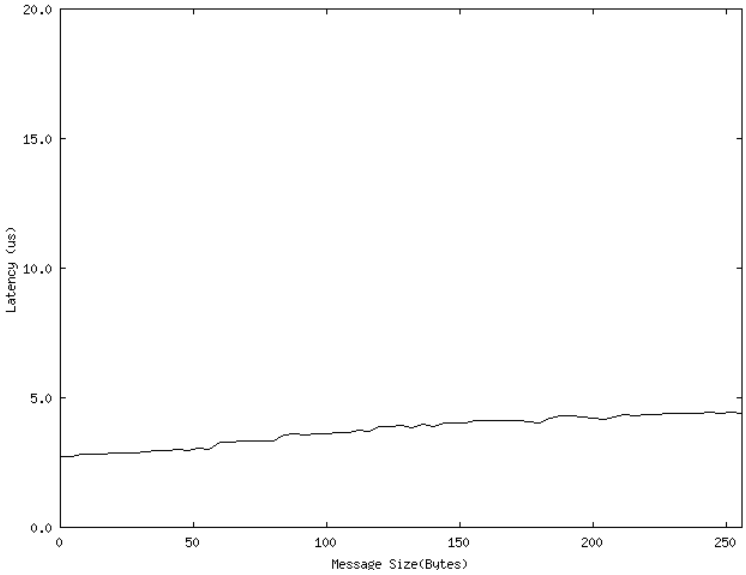

图5.4 BCL-3 结点内通信的单向延迟

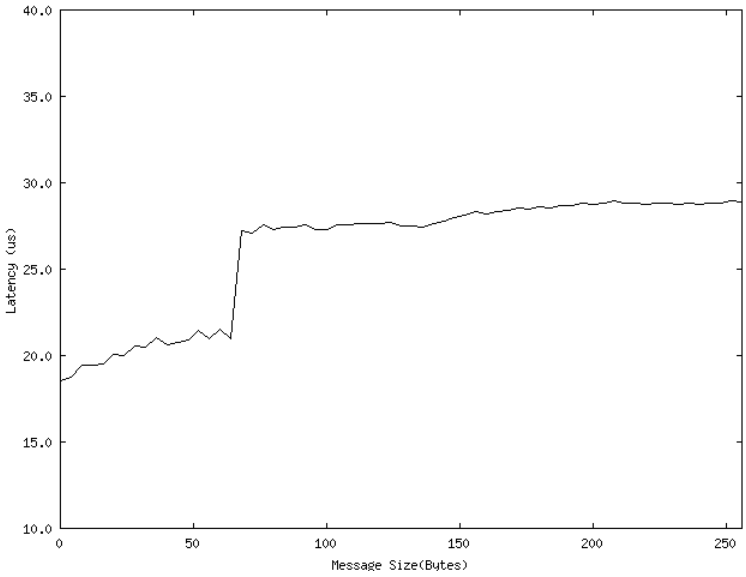

图5.5 BCL-3 结点间通信的单向延迟

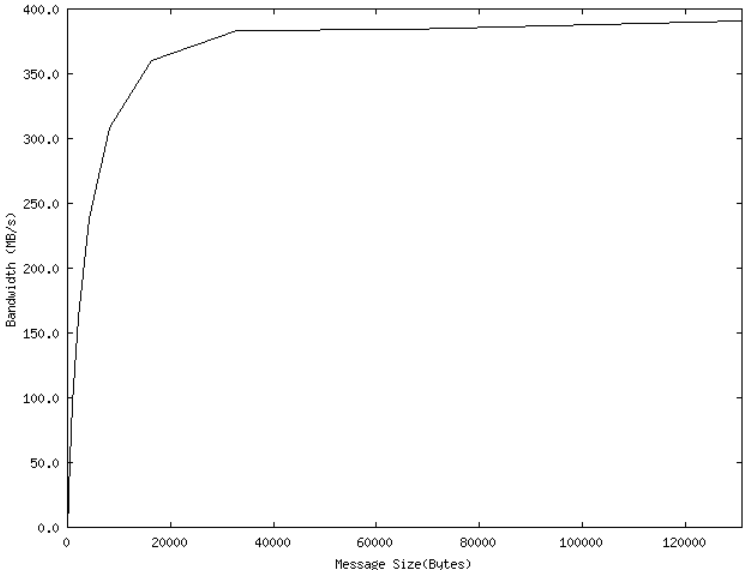

图5.6 BCL-3 结点内通信的带宽

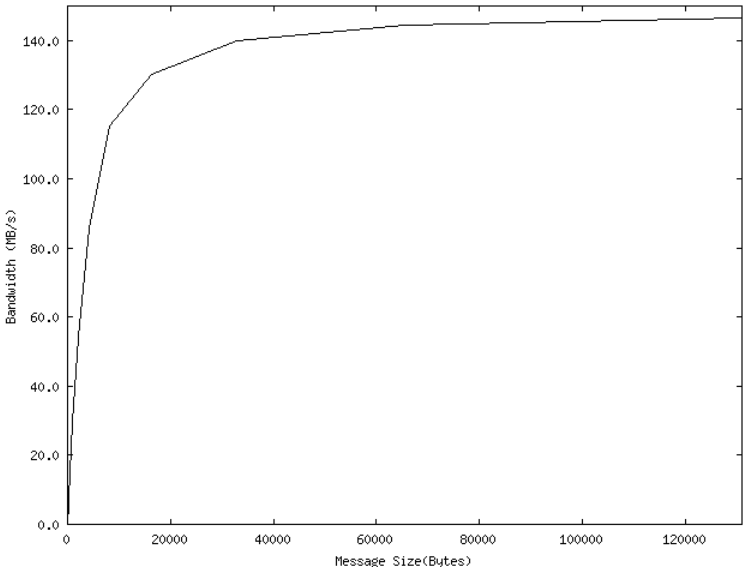

图5.7 BCL-3 结点间通信的带宽

表 5.5给出了 BCL-3 的点到点通信的通信带宽数据。图 5.6和图 5.7分别给出了结点内通信带宽和结点间通信带宽的变化曲线。由于通信过程中受到 Cache 的影响，结点内通信的带宽高于实测的内存拷贝带宽，且测量误差较大。通信带宽的性能曲线上升较快，结点内和结点间通信的半带宽均在 2KB 至 4KB 长度消息时达到，并保持了较平稳的上升趋势。

表5.5 BCL-3 的点到点通信带宽

消息长度(Byte)	结点内通信带宽(MB/s) (基于共享内存的通信协议)	结点间通信带宽(MB/s) (半用户级通信协议)
1024	95.2±0.3	29.51±0.01
2048	162.0±0.4	54.23±0.04
4096	237±2	85.6±0.2
8192	308±3	115.12±0.05
16384	360.0±0.2	130.38±0.09
32768	382.9±0.1	139.78±0.06
65536	384±5	144.47±0.03
131072	391±5	146.412±0.004

MPI 的点到点通信性能

表 5.6给出了基于 BCL-3 的 MPI 的点到点通信的通信延迟数据。图 5.8和图 5.9分别给出了结点内通信延迟和结点间通信延迟的变化曲线。从图中可以看出结点间通信延迟在 48 字节长度消息时发生了跳变。这是因为 BCL-3 在不同字节长度时采用的通信方式不同，增加了 MPI 的包头后通信延迟的跳变点变为 48 字节。对于结点内的通信过程，由于采用相同的内存拷贝机制，因此性能的变化曲线较平稳。

表5.6 基于 BCL-3 的 MPI 点到点通信延迟

消息长度(Byte)	结点内通信延迟(μs)	结点间通信延迟(μs)
0	6.33±0.01	23.70±0.04
4	6.44±0.02	23.68±0.05
8	6.48±0.02	24.2±0.1
16	6.52±0.02	24.66±0.05
32	6.66±0.02	25.25±0.02
64	7.75±0.02	31.06±0.06
128	8.073±0.005	31.93±0.01
256	9.10±0.02	33.57±0.06

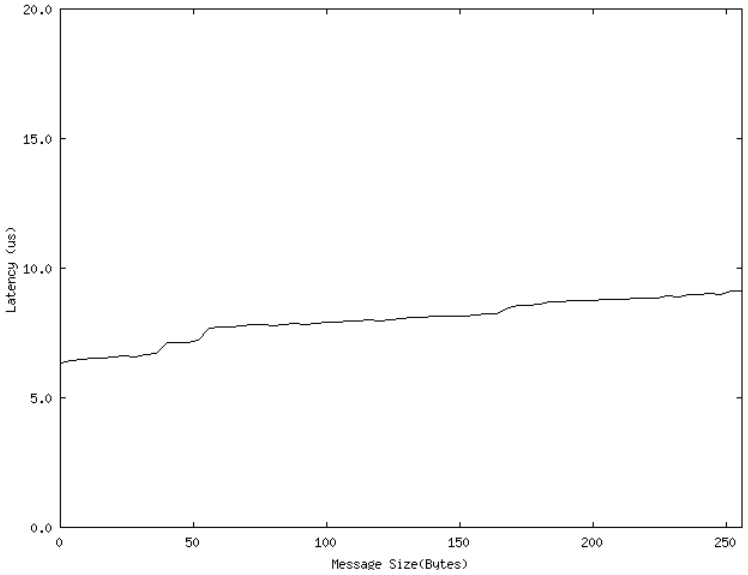

图5.8 基于 BCL-3 的 MPI 结点内通信的单向延迟

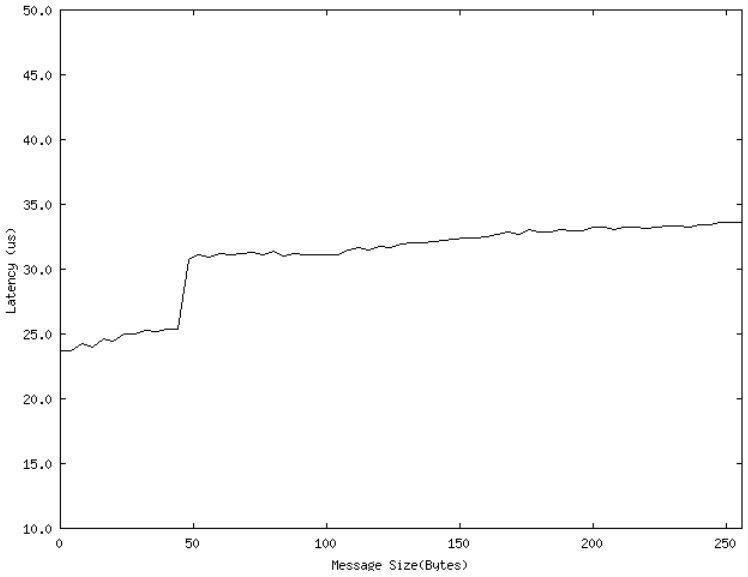

图5.9 基于 BCL-3 的 MPI 结点间通信的单向延迟

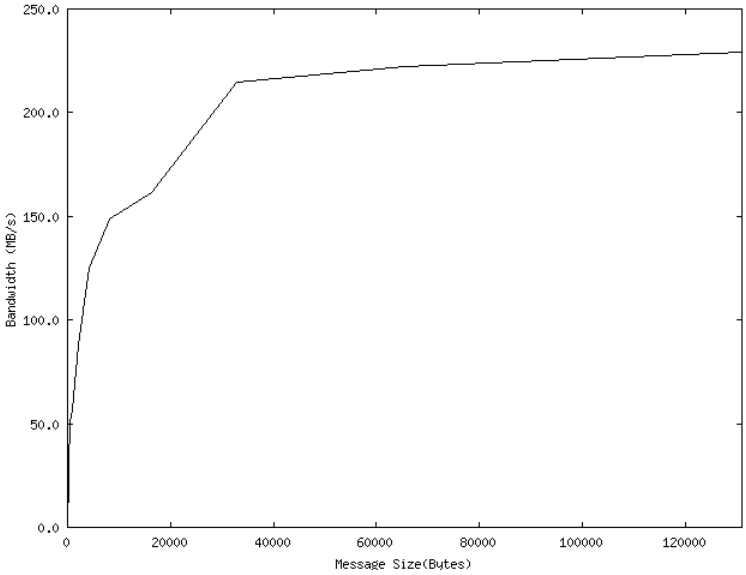

图5.10 基于 BCL-3 的 MPI 结点内通信的带宽

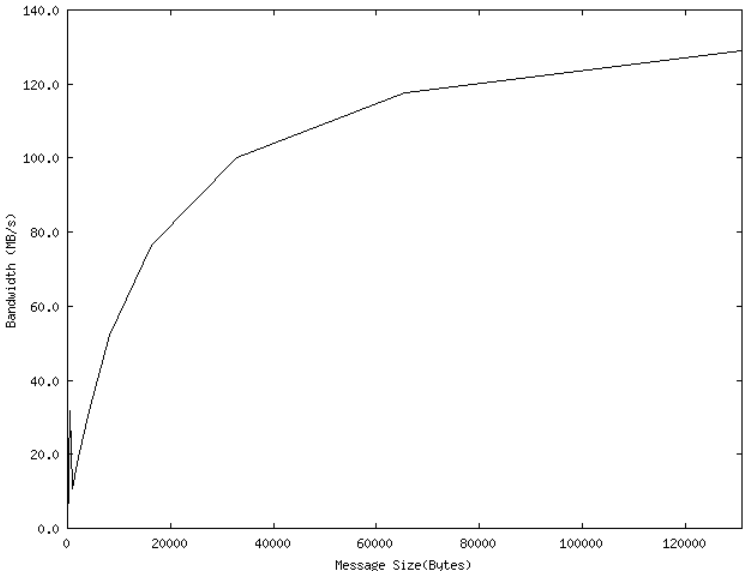

图5.11 基于 BCL-3 的 MPI 结点间通信的带宽

表 5.7给出了基于 BCL-3 的 MPI 的点到点通信的通信带宽数据。图 5.10和图 5.11分别给出了结点内通信带宽和结点间通信带宽的变化曲线。从图中可以看出在消息长度为 1KB 时通信带宽的性能指标出现跳变。这是由于在消息长度小于 1KB 时 MPI 的通信采用异步的通信模式，而当消息长度大于 1KB 时，MPI 采用 Rendezvous 语义，通信的双方需要通过握手才能进行消息通信。这样就增大了消息通信过程的开销，降低了通信的带宽。

表5.7 基于 BCL-3 的 MPI 的点到点通信带宽

消息长度(Byte)	结点内通信带宽(MB/s)	结点间通信带宽(MB/s)
1024	56.5±0.1	10.59±0.02
2048	88.2±0.2	19.10±0.04
4096	124.8±0.4	31.71±0.06
8192	149.0±0.6	52.23±0.08
16384	161.5±0.3	76.45±0.04
32768	214.5±0.5	100.28±0.09
65536	222±1	117.49±0.07
131072	229±1	128.90±0.04
262144	282±1	129.93±0.01
524288	317±1	130.40±0.02
1048576	338±1	130.50±0.04
2097152	328±2	130.68±0.03

PVM 的点到点通信性能

表 5.8给出了基于 BCL-3 的 PVM 的点到点通信的通信延迟数据。图 5.12和图 5.13分别给出了结点内通信延迟和结点间通信延迟的变化曲线。与 MPI 类似，结点间通信延迟在 52 字节长度消息处发生了跳变。

表5.8 基于 BCL-3 的 PVM 的点到点通信延迟

消息长度(Byte)	结点内通信延迟(μs)	结点间通信延迟(μs)
0	6.51±0.02	22.40±0.02
4	6.62±0.02	22.99±0.04
8	6.68±0.02	22.83±0.05
16	6.79±0.02	23.35±0.02
32	6.74±0.02	23.92±0.02
64	7.61±0.03	29.88±0.03
128	8.13±0.02	31.05±0.04
256	9.08±0.01	32.35±0.03

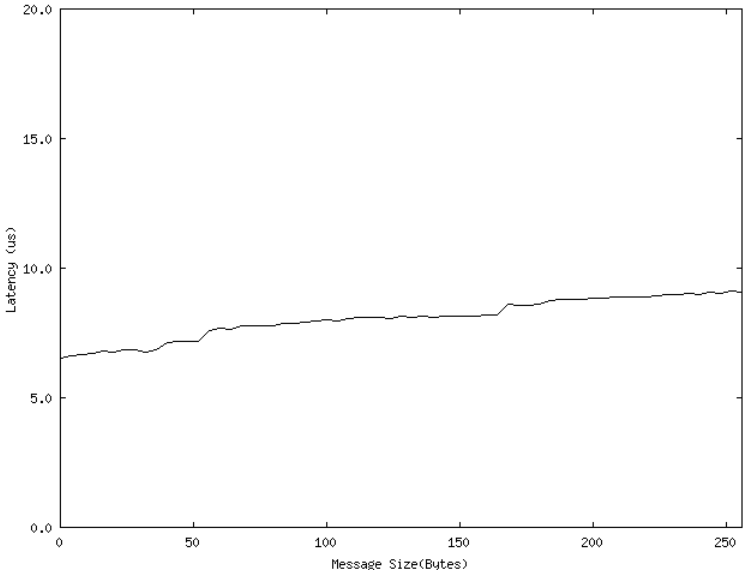

图5.12 基于 BCL-3 的 PVM 结点内通信的单向延迟

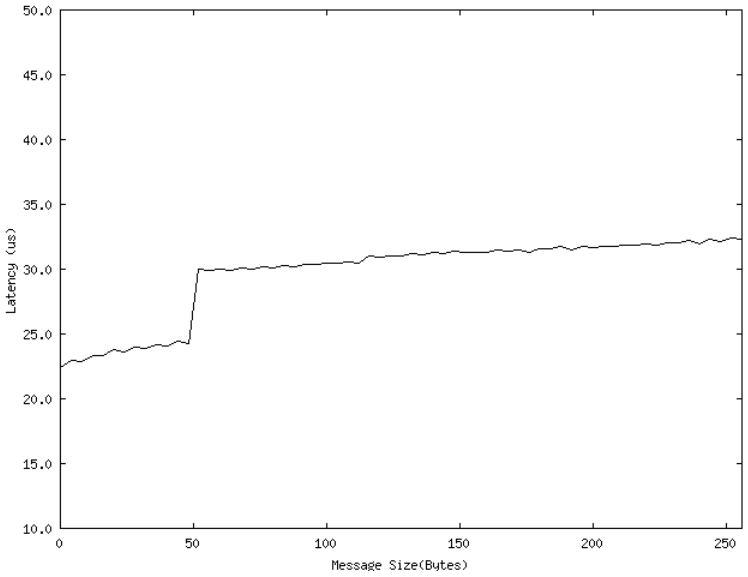

图5.13 基于 BCL-3 的 PVM 结点间通信的单向延迟

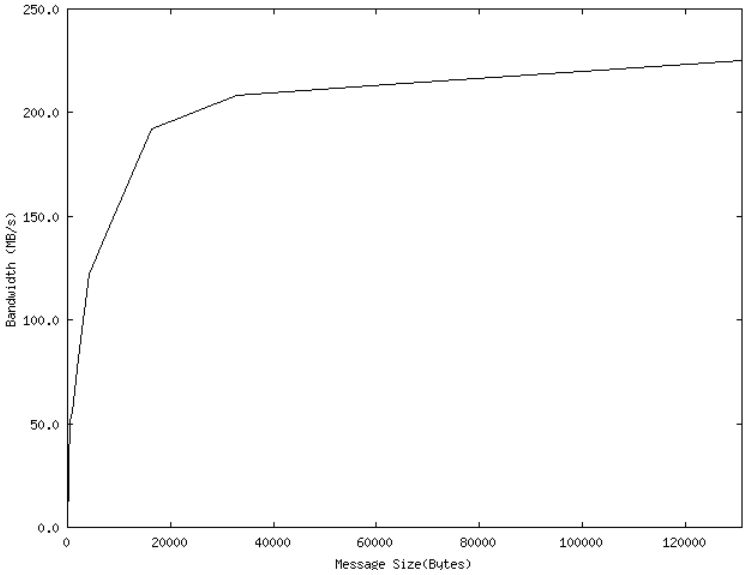

图5.14 基于 BCL-3 的 PVM 结点内通信的带宽

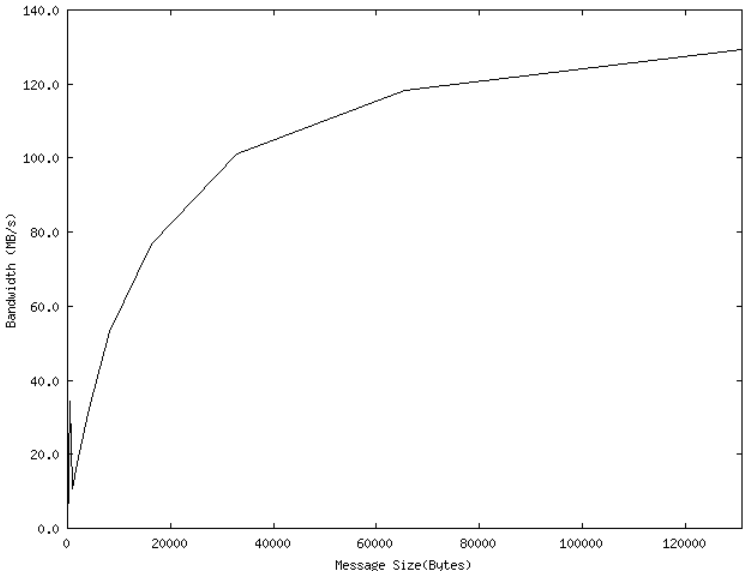

图5.15 基于 BCL-3 的 PVM 结点间通信的带宽

表 5.9给出了基于 BCL-3 的 PVM 的点到点通信的通信带宽数据。图 5.14和图 5.15分别给出了结点内通信带宽和结点间通信带宽的变化曲线。与 MPI 类似，通信带宽在 1KB 消息处有明显的变化。

表5.9 基于 BCL-3 的 PVM 的点到点通信带宽

消息长度(Byte)	结点内通信带宽(MB/s)	结点间通信带宽(MB/s)
1024	55.3±0.2	10.60±0.02
2048	81.6±0.1	19.02±0.02
4096	122.3±0.4	32.49±0.02
8192	145.4±0.3	53.05±0.06
16384	192.1±0.3	76.9±0.1
32768	208.1±0.4	101.03±0.09
65536	214.2±0.8	118.16±0.02
131072	225.2±0.5	129.27±0.02
262144	274.2±0.8	130.14±0.01
524288	308.2±0.8	130.60±0.01
1048576	327.0±0.7	130.80±0.02
2097152	313±1	130.848±0.005

5.3 均衡通信性能

5.3.1 机群系统整体性能的评价

点到点通信性能只能反映一个通信系统的基本通信性能。在一个实际的应用中，一次通信不可能不受到其它结点(进程)间通信的影响。因此，单纯从点到点的通信指标并不能说明一个通信系统的整体性能。

对于一个机群通信系统，整个机群通信网络的总体性能才是衡量其通信性能的主要标准。NAS 给出的基准程序从一定程度上能够测试出一个系统的性能[91][92][93]。在 NAS 的基准程序中采用了相当多的复杂的通信模式[94]，当通信网络上充满消息时，消息传输间的相互干扰有可能会增加整个网络的通信延迟和降低整个网络的通信带宽。一些较复杂的通信模式的性能的低下有时会导致实际应用程序的运行效率的降低，这就需要对通信进行优化[95]。

为了测量系统在不同通信模式下的性能，我们采用了一组基于 BCL-3 的 MPI 集合操作(Collective Operation)测试程序来进行测试[96][97][98][99]。这组程序包

括同步延迟(Barrier)、广播带宽(Broadcast)、归约带宽(Reduce)、散播带宽(Scatter)、聚集带宽(Gather)、右移带宽(Shift)和全交叉带宽(All to all)。

5.3.2 性能数据

同步延迟(Barrier)

同步延迟是指参与通信的各个进程同时进行同步操作(Barrier)的延迟。这个指标表征了一个系统通信延迟性能。它随着并行任务规模的增大而增大。其计算公式如下：

$$L_{Barrier} = T_{Total} / N \tag{5.5}$$

其中：T_{Total}为总运行时间，N 为重复运行次数。

表 5.10给出了 MPI 和 PVM 的同步延迟性能。

表5.10 同步延迟性能

结点数	MPI 性能(μs)		PVM 性能(μs)	
	每结点单进程	每结点 4 进程	每结点单进程	每结点 4 进程
1	—	32.0	—	54.0
2	37.5	78.2	56.3	95.4
4	84.2	135.2	97.3	142.1
8	128.5	184.2	136.9	198.6
16	182.7	256.6	185.0	273.7
32	228.5	341.3	287.2	416.0
64	432.4	764.0	393.3	684.1

广播带宽(Broadcast)

广播操作是指由单一进程向并行任务中的其它各进程发出相同的消息。为了提高广播带宽，在基于 BCL-3 的 MPI 和 PVM 中采用了二叉树的算法进行消息发送。即将并行任务中的各进程组织成二叉树的结构，由根进程(广播消息发起的进程)开始发出消息，并按树型结构将消息转发下去。这样，广播过程由未优化的(P-1)步减少为$\lceil \log_2 P \rceil$步(其中 P 为进程数)。广播带宽的计算公式如下：

$$B_{Bcast} = MsgLen \times N \times (P - 1) / T_{Total} \tag{5.6}$$

其中：T_{Total}为总运行时间，N 为重复运行次数，P 为总进程数。

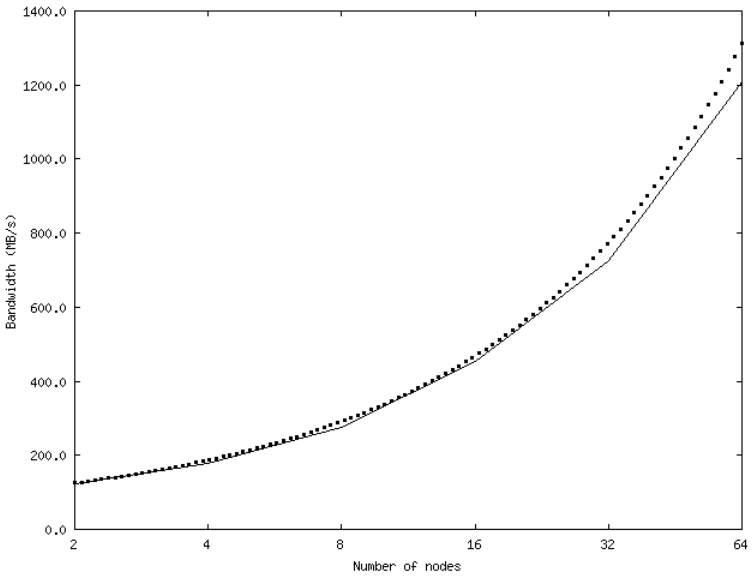

图5.16 MPI 广播带宽与理论值的比较

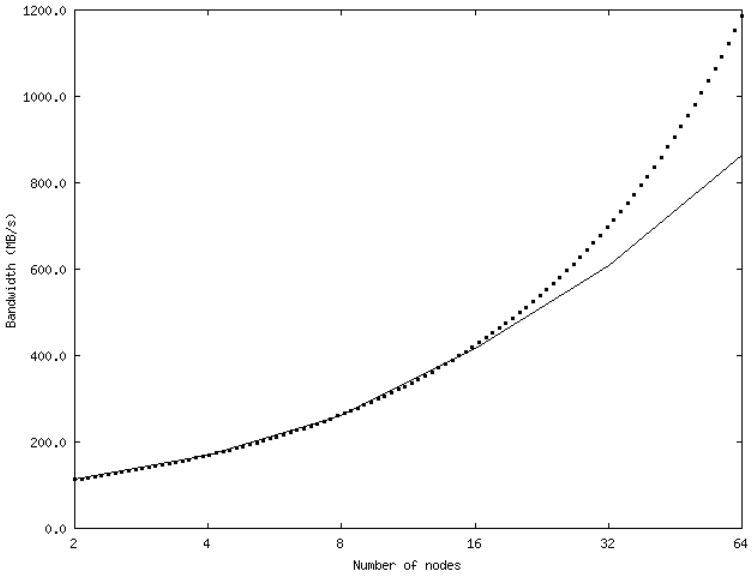

图5.17 PVM 广播带宽与理论值的比较

表 5.11给出了 MPI 和 PVM 的广播带宽性能。

表5.11 广播带宽性能

结点数	MPI 性能(MB/s)		PVM 性能(MB/s)	
	每结点单进程	每结点 4 进程	每结点单进程	每结点 4 进程
1	—	581.2	—	202.4
2	124.2	532.9	112.6	287.7
4	179.1	623.7	168.3	491.1
8	274.6	975.5	260.4	775.3
16	453.6	1549.0	416.7	1207.0
32	727.1	2489.0	607.0	1923.0
64	1209.0	4387.0	863.3	3223.0

采用二叉树的算法进行广播操作时，发送消息最多的结点将进行$\lceil \log_2 P \rceil$次消息发送。假设每个结点的点到点带宽为 B_{Node}，对于每结点上只有 1 个进程的情况，广播带宽的理论值为：

$$\begin{aligned} B0_{Bcast} &= MsgLen \times N \times (P-1) / T_{Total} \\ &= MsgLen \times N \times (P-1) / (\lceil log_2 P \rceil \times N \times T_1) \\ &= (P-1) \times B_{Node} / \lceil log_2 P \rceil \end{aligned} \tag{5.7}$$

其中 T_1 表示发送一个长度为 *MsgLen* 的消息所需要的时间。

图 5.17和图 5.17分别给出了 MPI 和 PVM 广播带宽的测量值与理论值的比较。两条曲线接近表明系统在进行广播模式的通信时可以提供较好的带宽性能。

归约带宽(Reduce)

归约操作是指由并行任务中的所有进程分别处理数据的一个分量，最后将数据的各个分量的处理结果归约为一个最后结果并放入一个特定的进程(根进程)内。为了提高归约带宽，在基于 BCL-3 的 MPI 中采用了二叉树的算法进行数据传递。即将并行任务中的各进程组织成二叉树的结构，数据由叶子结点一层一层向上级结点传递，最后传入根进程。这样，归约过程由未优化的(P-1)步减少为$\lceil \log_2 P \rceil$步(其中 P 为进程数)。归约带宽的计算公式如下：

$$B_{Reduce} = MsgLen \times N \times (P-1) / T_{Total} \tag{5.8}$$

其中：T_{Total} 为总运行时间，N 为重复运行次数，P 为总进程数。

表 5.12给出了 MPI 和 PVM 的归约带宽性能。

表5.12 归约带宽性能

结点数	MPI 性能(MB/s)		PVM 性能(MB/s)	
	每结点单进程	每结点 4 进程	每结点单进程	每结点 4 进程
1	—	241.9	—	160.2
2	96.0	429.6	110.2	120.1
4	160.5	586.8	133.1	113.6
8	255.5	920.6	126.0	106.8
16	390.1	1425.0	126.7	101.8
32	660.1	2427.0	114.3	101.6
64	1128.0	4159.0	107.3	99.2

采用二叉树的算法进行归约操作时，发送消息最多的结点将进行$\lceil \log_2 P \rceil$次消息发送。假设每个结点的点到点带宽为 B_{Node}，对于每结点上只有 1 个进程的情况，归约带宽的理论值为：

$$\begin{aligned} B0_{Reduce} &= MsgLen \times N \times (P-1) / T_{Total} \\ &= MsgLen \times N \times (P-1) / (\lceil \log_2 P \rceil \times N \times T_1) \\ &= (P-1) \times B_{Node} / \lceil \log_2 P \rceil \end{aligned} \tag{5.9}$$

其中 T_1 表示发送一个长度为 *MsgLen* 的消息所需要的时间。

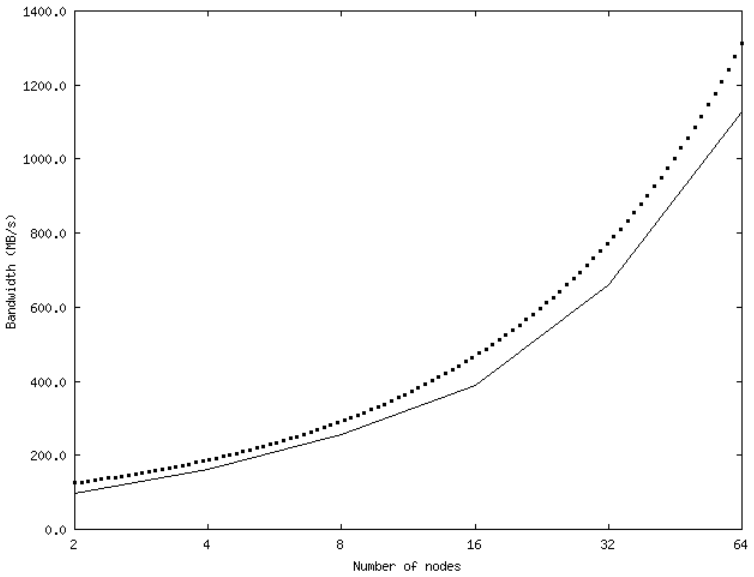

图5.18 MPI 归约带宽与理论值的比较

图 5.18给出了 MPI 归约带宽的测量值与理论值的比较。两条曲线接近表明系统在进行归约模式的通信时可以提供较好的带宽性能。对于 PVM，由于没有采用优化算法，其归约带宽受限于单结点的通信带宽 B_{Node}。因此，PVM 归约带宽的测量值保持在恒定的值附近。这一测量值与理论值吻合也表明了系统在进行未优化的归约模式通信时充分利用了网络硬件的性能，可以提供较好的带宽性能。

散播带宽(Scatter)

散播操作是指由单一进程向并行任务中的其它各进程发送一个私人化的消息。散播带宽的计算公式如下：

$$B_{Scatter} = MsgLen \times N \times (P - 1) / T_{Total} \tag{5.10}$$

其中：T_{Total} 为总运行时间，N 为重复运行次数，P 为总进程数。

表 5.13给出了 MPI 和 PVM 的散播带宽性能。

表5.13 散播带宽性能

结点数	MPI 性能(MB/s)		PVM 性能(MB/s)	
	每结点单进程	每结点 4 进程	每结点单进程	每结点 4 进程
1	—	307.9	—	320.3
2	97.4	159.7	94.9	154.8
4	113.5	137.9	104.5	125.0
8	118.3	121.1	107.0	115.2
16	116.3	117.1	108.4	110.9
32	109.2	110.5	107.0	105.5
64	106.2	103.1	106.9	106.2

散播带宽受限于发送消息结点的通信带宽 B_{Node}。因此，散播带宽的测量值保持在恒定的值附近。这一测量值与理论值吻合表明了系统在进行散播模式通信时充分利用了网络硬件的性能，可以提供较好的带宽性能。

聚集带宽(Gather)

与散播操作正好相反，聚集操作是指单一进程从并行任务中的其它每一进程接收一个消息。聚集带宽的计算公式如下：

$$B_{Gather} = MsgLen \times N \times (P - 1) / T_{Total} \tag{5.11}$$

其中：T_{Total} 为总运行时间，N 为重复运行次数，P 为总进程数。

表 5.14给出了 MPI 和 PVM 的聚集带宽性能。

表5.14 聚集带宽性能

结点数	MPI 性能(MB/s)		PVM 性能(MB/s)	
	每结点单进程	每结点 4 进程	每结点单进程	每结点 4 进程
1	—	153.2	—	149.2
2	97.6	87.1	96.0	88.8
4	120.9	90.9	123.2	90.9
8	124.4	83.8	112.2	82.1
16	98.8	108.8	102.1	79.9
32	96.9	113.9	91.7	79.8
64	114.7	114.1	83.8	76.7

聚集带宽受限于接收结点的通信带宽 B_{Node}。因此，聚集带宽的测量值保持在恒定的值附近。这一测量值与理论值吻合表明了系统在进行聚集模式通信时充分利用了网络硬件的性能，可以提供较好的带宽性能。

右移带宽(Shift)

右移操作是指将并行任务中的进程排成一个环，每一个进程向其右侧的进程发送一个消息，并从其左侧的进程中接收一个消息。右移带宽的计算公式如下：

$$B_{Shift} = MsgLen \times N \times P / T_{Total} \tag{5.12}$$

其中：T_{Total}为总运行时间，N 为重复运行次数，P 为总进程数。

表 5.15给出了 MPI 的右移带宽性能。

表5.15 右移带宽性能

结点数	MPI 性能(MB/s)	
	每结点单进程	每结点 4 进程
1	—	487.1
2	145.5	489.0
4	259.5	912.1
8	485.8	1816.0
16	946.2	4017.0
32	1982.0	6814.0
64	4026.0	14928.0

右移操作在每次通信时只涉及到其相邻的两个结点，每次右移操作进行 2

次消息通信。假设每个结点的点到点带宽为 B_{Node}，对于每结点上只有 1 个进程的情况，右移带宽的理论值为：

$$\begin{aligned} B0_{Shift} &= MsgLen \times N \times P / T_{Total} \\ &= MsgLen \times N \times P / (2 \times N \times T_1) \\ &= P \times B_{Node} / 2 \end{aligned} \tag{5.13}$$

其中 T_1 表示发送一个长度为 *MsgLen* 的消息所需要的时间。

图 5.19给出了 MPI 右移带宽的测量值与理论值的比较。由于右移操作仅涉及与自己相邻的进程，因此，它的性能基本上只依赖于点到点的通信性能，而受其它网络性能影响较小。两条曲线非常接近表明系统在进行右移模式的通信时充分利用了网络硬件的性能，可以提供较好的带宽性能。

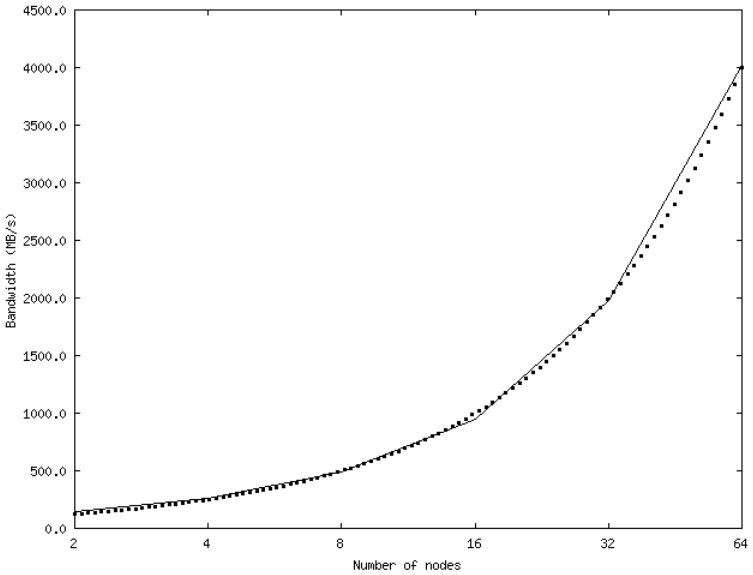

图5.19 MPI 右移带宽与理论值的比较

全交叉带宽(All to all)

全交叉操作是指将并行任务中的每一个进程都向并行任务中的所有进程(包括自身)发送一个私人化的消息，并从并行任务中的所有进程接收一个私人化的消息。全交叉带宽的计算公式如下：

$$B_{All2all} = MsgLen \times N \times (P - 1) \times P / T_{Total} \tag{5.14}$$

其中：T_{Total}为总运行时间，N为重复运行次数，P为总进程数。

表 5.16给出了 MPI 的全交叉带宽性能。

表5.16 全交叉带宽性能

结点数	MPI 性能(MB/s)	
	每结点单进程	每结点 4 进程
1	—	463.6
2	124.1	238.1
4	204.7	264.7
8	420.9	442.5
16	761.8	748.2
32	759.9	748.7
64	1501.0	1585.0

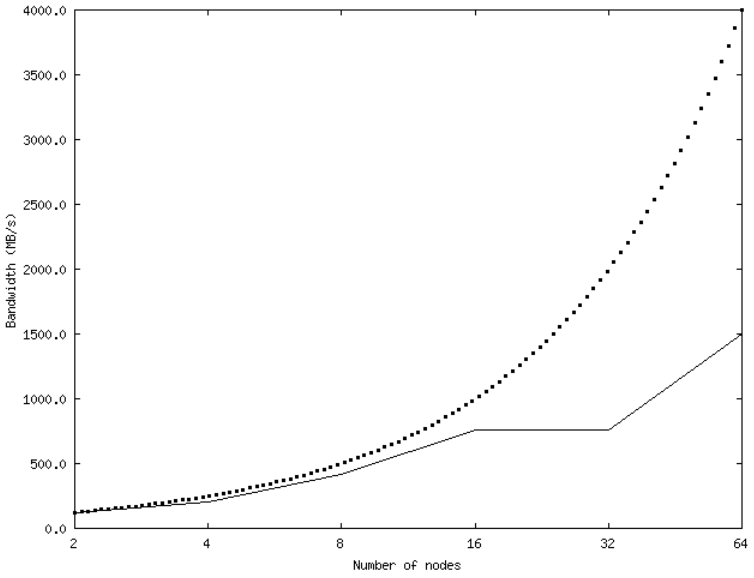

图5.20 MPI 全交叉带宽与理论值的比较

每次全交叉操作时进行 $2 \times (P - 1)$次消息通信。假设每个结点的点到点带宽为 B_{Node}，对于每结点上只有 1 个进程的情况，全交叉带宽的理论值为：

$$\begin{aligned} B0_{All2all} &= MsgLen \times N \times (P - 1) \times P / T_{Total} \\ &= MsgLen \times N \times (P - 1) \times P / (2 \times (P - 1) \times N \times T_l) \\ &= B_{Node} \times P / 2 \end{aligned} \tag{5.15}$$

其中 T_1 表示发送一个长度为 *MsgLen* 的消息所需要的时间。

图 5.20给出了 MPI 全交叉带宽的测量值与理论值的比较。两条曲线在 16 结点以内较接近，这表明系统在 16 结点内进行全交叉模式的通信时可以提供较好的带宽性能。超过 16 结点后由于网络互连结构的不对称，造成了跨交换机的结点间通信的瓶颈，系统的通信性能受到了影响。关于网络互连结构对系统可扩展性的影响将在下一节内进行详细讨论。

5.4 可扩展性

BCL-3 可以安装在基于 Myrinet 的互连网络上，也可以安装在基于 PMI-8/16 的互连网络上。这两种网络都采用交换的方式进行消息的传递，因此，交换网络的可扩展性问题就成为系统硬件可扩展性的关键问题。

从上一节中集合操作的性能可以看出，当消息传递带宽受限于通信软件的模式时，通信系统的带宽接近理论计算所得的数据；而当消息传递总带宽超出互连网络所提供的带宽时，通信系统的带宽就维持在网络总带宽附近，不随系统结点(进程)数的扩展而扩展。

表5.17 等分带宽性能

结点数	等分带宽(MB/s)	
	MPI	PVM
2	135.8	105.8
4	271.4	214.9
8	542.9	434.4
16	1083.8	857.1
32	579.0	632.1
	1111.6	930.1
64	1113.3	1116.3

表 5.17给出了通信互连网络的等分带宽。对于 16 结点以内的互连网络，在硬件的互连上局限于同一个交换机内。同一交换机内的结点间通信采用全交换的方式，其等分带宽随结点数的增加按照线性增长。对于 16 结点以上的互连网络，硬件的互连将要使用多个交换机的互连。交换机间的互连带宽限制了系统的等分带宽。对于 32 结点的等分带宽，表 5.17中给出了两组数据：第一组数据的测试环境与上一节中数据的测试环境相同，交换机间(16 结点到 16 结点)只采

用双线互连，其互连带宽理论值为 640MB/s(=160 × 4)。第二组数据调整了测试环境，增加了交换机间互连。每对交换机间采用四线互连，其等分带宽增加了一倍。这组数据也说明了上一节中全交叉带宽数据变化的情况。

随着机群系统的规模的扩大，机群系统的互连也将成为影响可扩展性的一个问题。互连网络需要保证整个网络的等分带宽性能的可扩展性，才能有效地保证通信系统性能的可扩展性。

除了硬件的可扩展性外，通信软件也需要对可扩展性提供支持。这主要体现在通信软件对一些硬件资源的需求不要随系统规模的扩大而无限制地快速增长。由于硬件资源是有限的，对它需求的增长势必造成对系统扩展规模的限制。在 BCL-3 中，我们替换了原来基于 Credit 的流量控制，采用了基于 ACK/NAK 的流量控制。这使得通信软件的流量控制不再影响系统的可扩展性。软件系统具有较好的可扩展性。

5.5 高可用性

BCL-3 的设计与实现中充分考虑了对高可用性的支持。当系统中部分节点由于某种原因在系统运行过程中崩溃或出现故障而不能正常工作时，系统管理员可将这部分节点从系统中暂时分离出去，系统可降级继续使用，系统中的其它节点仍可继续为用户提供服务。当故障节点修复后又可重新加入系统中。所有这些操作不会中断系统的正常运行。BCL-3 提供了基于 ACK/NAK 的通信控制。当互连网络出现故障时，通过 BCL-3 的超时重发机制可以有效地恢复丢失或损坏的消息。

我们从三个方面来评价机群通信系统的高可用性。第一，对系统网络硬件的监控手段；第二，网络硬件或结点故障对系统的影响；第三，对故障点的修复。

BCL-3 提供了一组结点及网络状态的监控函数(程序)。它包括一组状态监控与故障诊断命令(函数)及一个可视化的监控平台。通过使用监控命令或在程序中调用监控函数，应用程序可以获取或控制通信硬件信息并控制其运行状态。可视化的监控平台提供给机群系统管理员一个直观的管理平台。系统管理员可通过这一集成化平台来监控系统的负载状态和通信状态。

我们在 BCL-3 的通信层上增加了随机丢包的验证程序来模拟网络故障，并

采用重启结点的方式来模拟结点故障。通过程序验证，BCL-3 能有效地检测出系统的故障点。对于网络故障，BCL-3 能够通过重发机制修复故障对应用程序的影响；对于结点故障，BCL-3 需要系统管理员的介入来隔离故障结点并修复故障结点。

第六章 多网卡通信协议

本章着重研究了当前限制通信协议性能的因素，并给出了一个多网卡通信协议的结构模型。文章首先分析了点到点的通信性能，指出了限制通信性能提高的瓶颈。然后，文章给出了多网卡通信协议的层次结构和框架模型，并分析了在设计和实现多网卡通信协议中遇到的一些关键问题。最后，文章给出了一个多网卡通信协议的实例——双网卡通信协议的原型系统。

6.1 点到点通信性能分析

通信硬件的性能指标限制了通信协议的峰值性能。表 6.1给出了曙光 3000 所使用的平台中与通信相关硬件的理论峰值指标。

表6.1 通信硬件的性能指标

		性能指标
PCI 总线	数据宽度	64 位
	频率	50MHz
	理论峰值带宽	400MB/s
交换机理论峰值带宽		160MB/s (1.28Gbps)

从表中可以看出，当前通信路径上的瓶颈出现在交换机的带宽上。即使采用 Myricom 公司出生产的 Myrinet-2000 系列的交换机，其理论峰值带宽也仅为 2.0Gbps。因此，如何提高点到点数据交换的互连性能成为提高通信协议性能的关键问题。从硬件的角度，我们可以通过优化或重新设计交换芯片来提高交换机的性能。从软件方面，我们可以增加互连线数以提高点到点的交换带宽。

从上面的数据可以看出，每一条 50MHz 频率 64 位的 PCI 总线所能提供的带宽相当于 2--3 条交换线路的带宽。因此，对于一台有 2 条 PCI 总线的结点，可以支持 4 条以上的交换线路。为此，我们采用了多网卡互连的思想来设计机群系统的互连。

图 6.1给出了一个 4-网卡互连的机群系统的示意图。SMP 结点间通过 4 条

互连线路互连在一起。高速互连网络接口连接在 I/O 总线上。每个结点有多个处理器，多个存储模块。处理器与存储模块间采用存储总线互连。在结点内部各处理器以共享内存的方式访问各个存储模块。每个结点有各自的本地磁盘。结点与结点之间还可以通过低速的互连网络进行通信。高速互连网络主要处理计算任务(包括科学计算和信息等其它服务)中的通信，低速互连网络主要处理机群管理、程序编辑、编译等任务中的通信。

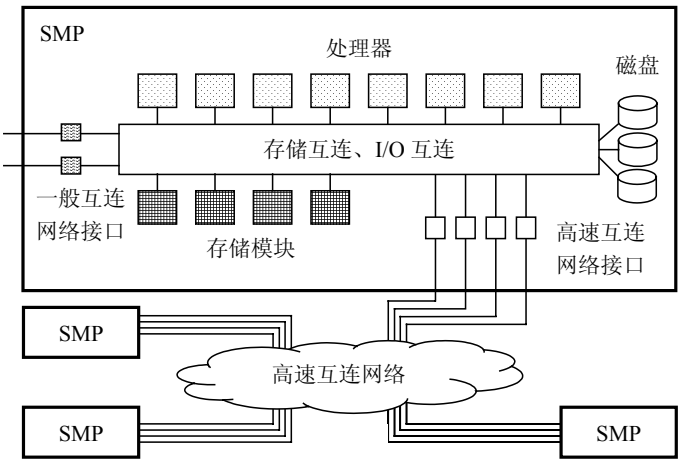

图6.1 多网卡互连的机群系统示意

采用图 6.1结构的机群系统若仍使用现行的实验平台，其网络互连性能达到结点到结点总带宽 640MB/s。实验所用的每个结点共有两条 PCI 总线，其结点 I/O 性能达到总带宽 800MB/s。这样，系统的通信能力的理论上限将提高四倍。这为通信协议的设计提供了较好的基础。

6.2 多网卡通信协议结构

与单网卡通信协议不同，多网卡通信协议需要着重考虑通信硬件的使用。在单网卡通信协议中，由于只有一个网络接口卡，通信过程中无须选择通信路径。然而，在多网卡通信协议中，任意两个结点间将存在多条通信路径。这为通信协议的设计提供了一定的灵活性，同时，也使得通信协议更加复杂。

6.2.1 多网卡通信的模式

为了研究多网卡的通信模式，我们首先假设结点间的互连网络结构采用全互连的交换方式。即当拥有 m 个网卡的结点 A 和 n 个网卡的结点 B 互连时，这 m+n 个网卡采用全交叉的互连方式。结点 A 的任何一个网卡均可以直接和结点 B 的任何一个网卡进行通信。这样，从结点 A 向结点 B 发出的任何一个消息(消息包)均有 m×n 种通信路径的选择。为了简化通信协议的复杂度，我们给出几种常用的通信模式。

- **基于路由的多网卡通信模式**

基于路由的通信模式采用根据通信的源结点和目的结点来选择所使用的通信路径。任意两结点间使用一条(或一对)固定的通信路径。

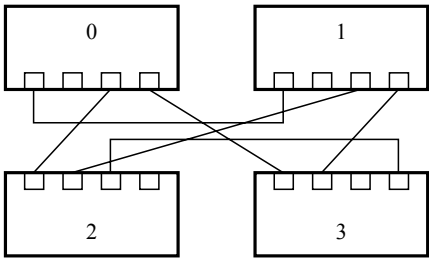

图6.2 基于路由的多网卡通信模式

图 6.2给出了一个由 4 个结点组成的基于路由的多网卡通信结构的模型。在这个图中可以看出，结点 i 与结点 j 通信时，使用结点 i 的网卡 j 进行消息发送，使用结点 j 的网卡 i 进行消息接收。这样，在大规模的机群系统中，一对多和多对一的消息传递性能将有明显的提高。

- **基于端口的多网卡通信模式**

基于端口的通信模式与基于路由的通信模式类似。它们的区别在于基于端口的通信模式用端口号(软件)来区分所使用的通信路径，而不根据路由(硬件)来区分所使用的通信路径。每一端口用一个特定的网卡与其它结点上的端口进行通信。由于每一个进程拥有一个特定的端口，因此不同的应用程序将使用不同的通信路径，每个并行任务的性能不会受到其它同时执行的并行任务的影响。

- **基于发送/接收的多网卡通信模式**

基于发送/接收的通信模式将网卡分为发送网卡和接收网卡两类。发送网卡只负责消息发送，接收网卡只负责消息接收。对于双网卡的系统，采用这种方式实现通信协议较为简单：一块网卡负责消息发送，一块网卡负责消息接收。这样，每块网卡的控制程序将得到简化，有利于提高通信的效率。

- **细粒度多网卡通信模式**

与前面几种通信模式不同，细粒度通信模式的每一次消息传递都由多块网卡协同完成。采用这种模式虽然会提高系统的复杂程度，但它能够有效地提高点到点的通信性能。

比较几种通信模式，不难看出前面的三种模式都有其局限性。基于路由的多网卡通信模式对于大规模的机群系统能提高其整体的通信性能，但是当两个结点间发生密集的通信时，其性能仍与单网卡系统相同。基于端口的多网卡通信模式在多任务的情况下能保持较高的性能，但在单任务的情况下与单网卡系统相同。基于发送/接收的多网卡通信模式虽然有效地将通信分为两个部分，但系统的点到点单向带宽没有改进。同时，它在双网卡系统上的实现较为容易，而在多网卡系统上实现仍有一定的困难。细粒度多网卡通信模式有效地提高了机群系统的点到点单向通信带宽。由于点到点单向通信带宽是其它通信方式带宽的基础，因此点到点带宽的提高将直接影响系统其它通信性能(集合操作等)。本文中将着重讨论这种多网卡通信模式。

6.2.2 细粒度多网卡通信协议设计中的关键问题

- **数据分片**

在细粒度多网卡通信协议中，消息的发送是通过不同的网卡进行的。每一块网卡发送不同的消息分片。这样，就需要对消息数据进行分片。数据分片的粒度直接影响着通信的性能。较细粒度的数据分片可以使得各个网卡所传递的数据量均衡，系统可以平衡地使用各个通信部件。但是，过于细的粒度的数据分片会使得每一次数据传输的长度(数据包的长度)变小，这样不利于提高通信的带宽。数据分片的粒度一般采用以页为单位(4KB)。这样，消息通信的过程将仍按照页的单位进行，保证了通信带宽不受影响。

数据分片可以把消息体细分为多个页面，每个网卡依次发出不同的页面。

数据分片也可以采用把消息体均分 n 个段(仍保证以页面为单位)，每个网卡发送一个数据段。

- **数据同步**

将消息数据以分片的方式通过不同的网卡进行传输就会产生数据同步的问题。数据同步是指消息体的不同分片如何保证同步到达消息接收方，并组装成完整的消息。当数据采用分片的方式传输时，同一个消息体的数据可能通过不同通信硬件通道从消息的发送方传递到消息的接收方。由于不同硬件通道的繁忙程度不同，虽然在消息的发送方按序进行发送，但消息的接收方可能不能按照预期的时间顺序接收到消息。这样，就为组装消息造成了困难。一个消息的到达不再能按照单网卡通信协议中按消息尾包的到达为标识，而要判定消息不同分片全部到达。

为了判定一个消息接收完成，在接收方需要判定该消息的全部分片到达。由于消息数据的每一个分片由一个网卡传输。在该网卡的传输过程中可以保证数据包的顺序。因此，要判定全部分片到达消息接收方只须判定在每一个网卡上传输的消息分片的尾包到达消息接收方。这将会增加消息接收的开销 o_r，从而降低系统的通信性能。此外，在进行小消息传输时，由于小消息(如小于 1 页的消息)的分片将无法使用全部的通信硬件，这使得在消息接收方不能简单地判断全部的硬件通道上消息分片的到达，而应结合接收消息的长度来进行判断。

- **消息保序**

采用多网卡的互连结构时，由于互连硬件不再是单一的通道，就会造成消息传输的无序性。相同源结点和目标结点间的消息传递若不通过同一硬件通道(相同的消息发送网卡和消息接收网卡)则不能保证消息传输的顺序，即先发出的消息先收到。这样，为了对上层通信软件提供可靠有序的消息传递，就必须控制消息传递的顺序。

提供有序消息传递的一种简单的实现方式是将每一个消息的某一分片(如尾分片)交由一个特定的网卡(如第 0 号网卡)进行传递。这样，在进行消息排序时只需按照该网卡上消息的到达顺序即可。但是，采用这一方法时小消息(只有一个分片的消息)通信性能与使用单网卡的系统相同。并且，由于排序网卡要处理每一个消息的数据包，造成了消息传递的瓶颈。

为了解决采取特定网卡排序所带来的问题，在多网卡系统中可以采用轮转使用法来使用系统中的多块网卡。即对于任一对结点间进行的通信，按顺序依次使用对应的网卡进行通信。这样，在接收消息的结点上就可以判定出从同一源结点上发出的消息顺序，从而进行排序。同样，采用消息序号标识的方法也可以确定同源同目的消息间的顺序。

- **网卡负载平衡**

使用多块网卡进行通信时，需要均衡每一块网卡上的通信负载。这样，才能更有效地利用通信硬件所提供的性能。在上面的讨论中可以看到，为了保证消息通信的有序性等，需要对某些网卡做特定的要求，如负责对消息进行排序等。这样，多块网卡在通信过程中所处的地位不再对称，其负载不再均衡。为了平衡每块网卡上的通信负载，在消息的分片过程中可以对一些特定的重负载网卡划分较小的数据分片。具体的划分可以根据实际应用中的通信负载进行设计。

此外，在消息传递的过程中，每块网卡上消息发送的负载较容易进行平衡。然而消息接收方的负载则难于进行控制。如何进行消息接收负载的均衡也是多网卡通信协议需要考虑的一个问题。

- **路由选择**

在多网卡系统中，结点上的每一块网卡都可以与其它结点上的任何一个网卡进行通信。这样，在通信过程中的路由选择也是设计多网卡通信协议时应考虑的问题。

在一般情况下，源结点和目标结点采用相同序号的网卡进行通信的方法可以简化通信协议的设计。但是，若要考虑到接收网卡的负载平衡问题时，就需要使用更复杂的互连模式了。

6.3 半用户级双网卡通信协议

半用户级双网卡通信协议是在半用户级通信协议的基础上设计的通信协议。在双网卡协议的原型系统中，协议接口和通信机制上没有变化，只是在进行物理通信的过程中使用了双通信通道机制。原型系统中的两块网卡的通信也采用固定路由的方法，即结点上的每块网卡只同其它结点上对应的网卡进行消

息传递，而不根据网卡的负载进行动态负载平衡。

6.3.1 内存分布

半用户级双网卡通信协议延用了半用户级通信协议中的内存分布模式，通信过程所使用的数据结构分为两个部分，分别存放于核心空间和用户空间中。消息发送请求队列和消息发送缓冲缓冲区控制块涉及到与其它进程通信的关键数据，因此这部分数据结构处于核心空间中，以保证其安全性。不涉及其它进程、对系统安全影响较小的消息发送完成事件队列放在系统的用户空间之中。这样，在进行消息通信时，只有在消息发送的过程中需要进入核心空间，而在探询消息发送是否完成和接收消息时，无须进入核心空间。

消息数据传输队列是用来在进程间传递消息体数据的。在半用户级双网卡通信协议中任意两个相关进程之间有两条双向数据传输队列，分别处理相关两个网卡的通信过程。数据传输队列由一个系统消息缓冲环和一组通道控制块组成。系统消息缓冲环是系统消息通道使用的缓冲区。它是一个环形的先进先出队列。系统消息通道消息首先暂存在这个环形队列中，然后由消息接收函数接收到用户缓冲区。通道控制块用于一般消息通道和开放消息通道消息接收。每一个通道控制块控制一个通道。通道控制块中包含了使用该通道的两个进程的相关信息和用于接收消息的缓冲区信息。

6.3.2 通信机制

在半用户级双网卡通信协议中，根据消息的长度采用两种不同的通信机制进行消息传递。对于长度较短的消息(小消息)采用系统消息通道的机制，对数据不分片。这种通信机制可以减少通信关键路径上的开销，从而降低通信的延迟。对于长度较长的消息(大消息)采用一般消息通道的机制，对数据进行分片。这种通信机制可以一次传递较多的数据，并且同时使用两块网卡，能够有效地利用系统硬件的通信带宽。

系统消息通道的通信机制

系统消息通道使用系统消息缓冲环来进行数据传输。在半用户级双网卡通信协议中采用与单网卡协议相同的系统消息通道的通信机制，整个通信过程只

使用 0 号网卡。通过系统消息通道发送一个消息时，首先需要进入核心空间填写消息发送请求。NIC 上的通信协处理器将按照消息发送请求的内容将相应的数据通过 NIC 上的消息数据传输队列发送的相应的结点。当消息发送完成后，将会有一个消息发送完成事件送回用户空间的消息发送完成事件队列中，以通知相应的进程发送过程完毕。

从互连网络上接收到的系统消息通道消息通过在NIC上的消息数据传输队列依次传输到处于用户空间的环形系统消息缓冲区内。并且在相应的消息到达事件队列中填入消息到达记录，以通知用户进程接收消息。

一般消息通道和开放消息通道的通信机制

一般消息通道和开放消息通道直接使用用户缓冲区进行数据传输。在进行传输之前，消息数据首先要进行分片。消息包被分为两个部分，每一部分分别通过一块网卡进行传输。从用户的发送缓冲区中直接传递至 NIC 上的消息数据传输队列，通过高速互连网络的传输后再传输到接收方的消息接收缓冲区中。当消息传递完成后，系统会产生消息发送/接收完成事件，以通知相应进程操作完成。消息发送和接收的完成需要分别判断消息数据的两个分片传输全部完成。

6.3.3 通信性能

为了测试半用户级双网卡通信协议的性能，我们在前面测试平台的基础上增加了通信硬件，建立了一个两个结点的双网卡通信平台。测试平台采用 IBM 4-way Power3-II 375MHz 处理器，2GB 内存，AIX 4.3.3 操作系统。每台计算结点通过 2 块 M2M-PCI64A-2 网络接口卡与其它结点互连，2 块网络接口卡与结点使用同一条 64 位 PCI 总线连接。互连硬件采用 Myricom 公司生产的 M2M-OCT-SW8 交换设备。

表6.2 双网卡系统性能测试

	点到点通信带宽(MB/s)
双网卡系统(使用 1 块网卡)	140.114±0.005
双网卡系统(使用 2 块网卡)	251.88±0.05
多重通信协议点到点通信带宽	146.412±0.004

由于两块网卡使用同一条 PCI 总线，这就导致了通信过程中的相互干扰，

从而影响通信性能。表 6.2给出了原型系统性能的测试结果。测试所得的点到点通信带宽达到 251.88MB/s。对比半用户级单网卡通信协议的性能数据(点到点通信带宽为146.412MB/s)，性能提高了 72%以上。在双网卡系统中每块网卡的利用率达到了 90%。

6.4 基于双网卡的多重通信协议

双网卡半用户级通信协议将机群结点间通信的硬件通道细节屏蔽在通信协议中，使上层通信软件能够透明地使用多路通信通道。多重通信协议则把机群处理器的拓扑结构屏蔽在通信协议中，使上层通信软件能够透明地使用非对称拓扑结构的各处理器。将这两点结合起来，我们就得到了基于双网卡的多重通信协议。

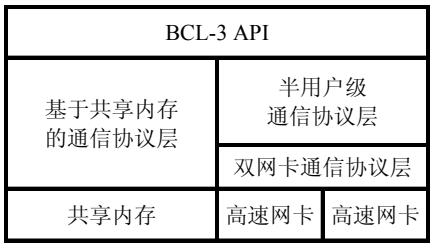

图6.3 基于双网卡的多重通信协议

图 6.3给出了基于双网卡的多重通信协议的框图。在这个框图中，采用两层通信协议来屏蔽处理器间的异构拓扑和互连网络的多路通道。结点内的通信仍采用单网卡的多重通信协议中的基于共享内存的结点内通信协议。结点间则采用半用户级双网卡通信协议充分利用硬件所提供的通信带宽。

总体上看，采用双网卡和 SMP 结点的机群系统按照处理器间互连性能可以看作是一种层次式的结构。第一个层次是 SMP 结点内部各处理器。它们采用共享存储的方式互连，具有较高的互连性能。第二个层次是通过高速互连网络连接在一起的结点间处理器。它们采用双网卡互连，两个物理通信通道的使用采用细粒度任务分配方式。扩展这一结构则可以设计多网卡互连及不对称多网卡互连(不同的结点间采用不同数量的互连网卡)。这样，根据其互连性能则可以把第二个层次再细分为若干个子层。根据每个子层的互连方式和性能采用不同的通信协议。

采用这种多层次的机群通信协议可以将硬件系统的特点屏蔽在软件协议之中，为上层通信协议和应用软件提供一个一致的通信服务。同时，针对不同性能的互连方式采用不同的通信模式以充分利用硬件性能，提供一个高效率的通信协议。扩展这个协议结构，我们可以在这个协议栈的顶端增加 MPI 及 PVM 协议。这样，通过多重通信协议中部分底层接口可以在高层的通信协议(MPI 及 PVM)中优化相关的集合操作通信算法，从而充分利用硬件拓扑结构的特点，提供最优的通信服务。

第七章 结 论

在论文的最后，本章总结了论文中所提出的问题及讨论。论文主要论述了机群通信系统的层次结构与多重通信协议；提出了用于实现结点内通信的基于共享内存的通信协议和用于实现结点间通信的半用户通信协议；对系统的总体性能评价进行了深入地讨论；提出了多网卡通信协议的模型及双网卡通信协议的原型系统。最后，本章给出了下一步在机群通信协议方面应进行研究的方向，对未来的研究进行了展望。

7.1 机群通信系统的层次结构与多重通信协议

随着机群系统结构的不断发展，机群结点从最初的单处理器发展为对称多处理器结构，机群的互连方式也从单通信通道向多重通信通道发展。本文中提出的机群通信系统的层次结构从互连性能的角度将基于 SMP 结点的机群系统划分为两个层次，这对其上的通信协议的设计实现具有一定的指导作用。根据这种层次结构，文中设计了多重通信协议。多重通信协议根据不同的互连性能采用不同的通信策略，从而最大限度地发挥了通信硬件的性能。从对上层协议和应用提供的服务来看，多重通信协议能够提供一致的通信服务，将系统硬件的差异屏蔽在协议层中。同时，上层通信协议和应用也可以根据系统硬件的拓扑结构进行一定的优化，以充分利用系统的性能。采用这种方法既可以为上层软件提供一个统一的界面，简化上层软件的设计开发难度，又可以使上层软件通过一定的方式了解到底层通信性能的差异，为其性能的优化提供空间。

机群通信系统的层次结构还可以推广到多网卡通信协议中。在多网卡机群系统中，网络互连可以采用多路通道。这样，根据不同的互连性能我们可以将机群的层次结构进一步细化。对于异构的互连方式，不同结点间的互连可以采用不同数量的互连通道。细化的层次结构对通信协议的设计具有一定的指导作用。同样地，对于基于机群的机群系统(Cluster of Clusters)这种层次结构则更为复杂。采用多重通信协议的模式就可以为上层协议及应用提供一个统一而又高效的通信协议层。

7.2 基于共享内存的通信协议

基于共享内存的通信协议是本文中组成多重通信协议的一个重要的组成部分。传统的机群通信协议主要是为互连组成机群的不同结点而设计的通信协议层。当采用 SMP 结构作为组成机群的结点时，结点内部的通信就成为提供系统总体性能的关键问题之一。为此，在本文中设计了基于共享内存的通信协议，它有效地利用了系统的硬件，使得 SMP 结点内部的通信性能大大提高。从而使得机群系统的总体性能得到提高。

7.3 半用户级通信协议

实现机群结点间通信协议时，为了降低通信路径上的开销，很多通信协议采用了用户级通信方式。但是，采用用户级通信方式在降低通信开销的同时，也将系统的重要数据结构和关键操作过程暴露给了用户进程。这对于一个机群系统的安全无疑是一个巨大的漏洞。作为一个需要推广应用的机群通信系统，我们需要保证系统的安全性、可靠性。为此，本文提出了半用户级通信协议。这种半用户级通信协议采用了用户级通信协议与核心级通信协议相结合的方式，采用核心级通信协议的进入核心空间的实现方式来保证系统的安全性与可靠性。同时，采用用户级通信协议的思路减少通信关键路径上进入核心空间的次数，降低通信路径上的开销。通过实际测量，我们在增加 2μs 的通信开销的范围内有效地保证了机群通信系统的安全性与可靠性。

7.4 机群系统均衡性能

传统评价机群通信系统性能的方法是测量机群通信系统的点到点通信延迟和通信带宽指标。点到点通信性能指标是衡量通信系统性能的一个侧面，这对于小规模的机群系统来说，基本能满足衡量系统性能的要求。但是，对于大规模的机群系统，单纯从点到点的通信性能已经不能表现一个系统的实际性能。在一个大规模的机群系统中，系统硬件的互连将会较为复杂，通信过程中的相互干扰将会影响实际的通信性能。在本文的性能分析中我们可以看到通信硬件互连的瓶颈会限制到整个系统在某些应用模式下的性能。在不同的通信模式下，

进程间的相互通信也会影响到其它进程间的通信性能。复杂通信模式下的通信性能不能简单地看作点到点通信性能的叠加。在本文中采用了一组集合操作的性能来衡量机群通信系统，通过测量集合操作的性能从另一个侧面反映了通信系统的性能。

评价一个机群系统性能的最根本的方法是实测应用程序在其上运行的性能。表 7.1和表 7.2分别给出了 Linpack 和矩阵乘在曙光 2000-II 和曙光 3000 上的性能对比。表 7.3给出一个应用实例的测试数据。从这里可以看出，当优化了结点间的通信过程，采用多重通信协议后，考虑计算结点性能的提高，系统性能仍得到了一定的提高。这从另一个侧面也验证了通过集合操作测试集对一个机群通信系统性能评价的有效性。

表7.1 Linpack 的性能数据

CPU 数	4	8	16	32	64	128	160	256
曙光 2000-II	58%	57%	53%	50%	44%	38%	36%	—
曙光 3000	86%	82%	79%	76%	71%	67%	—	61%

表7.2 矩阵乘的性能数据

CPU 数	4	8	16	32	64	128	160	256
曙光 2000-II	60%	58%	55%	52%	51%	49%	42%	—
曙光 3000	90%	87%	85%	83%	79%	76%	—	73%

表7.3 油藏模拟应用程序的性能数据

应用规模	平台	进程数	运行时间(小时)
33 年数据	曙光 2000-II	16	>27
	曙光 3000	16	13.42
135 年数据	曙光 2000-II	16	>100
	曙光 3000	16	21.34

7.5 进一步的工作

7.5.1 机群通信系统的层次结构与多重通信协议的扩展

机群通信系统的层次结构给出了基于 SMP 结点的机群系统的一种按照互

连性能进行划分的分层结构。随着机群系统结构的不断发展，逐渐会出现一些采用多互连通道(多网卡)结构。这样，还需要对这一层次结构进行进一步的改进和扩展。多互连通道的除了采用对称的多网卡通信互连结构外，还可以采用一些不对称的结构。如根据系统的拓扑结构采用一些胖树等互连结构。这样，分层的结构能够更有效地进行系统的设计。在这种异构/非对称的多通道结构上设计的通信协议以及上层的通信协议(软件)及应用等需要考虑有关网络负载平衡的问题。这对上层通信协议和应用软件的设计提出了更强的要求。同时，基于机群的机群系统(Cluster of Clusters)也同样可以用这种层次结构来描述。在其上的通信协议可以采用一种广义的多重通信协议将不同结构、不同性能的机群系统互连起来。

7.5.2 机群通信系统性能的评价及对应用的支持

在本文中，机群通信协议的性能采用了点到点通信性能、LogGP 模型参数和集合操作性能结合的评价方式。这是对机群通信系统整体性能评价的一个初步的研究。为了能对机群系统提供更好地支持，机群通信系统的性能不能仅通过一些简单的指标进行评价。对机群通信系统的性能评价以及机群系统设计和优化所应追求的目标是在以后机群通信系统设计实现中所需要考虑的一个重要的问题。

对于一个机群系统，其性能的最终评价标准是运行于其上的应用程序所表现出的性能。因此，机群通信系统如何提高对应用的支持是研究机群通信系统的又一个关键问题。

7.5.3 协议的形式化表示与验证

本文中尝试使用一些形式化的方式来描述一个通信系统，并使用一些验证工具对系统的正确性进行验证。数学是研究计算机的一个基础工具，采用逻辑的方法对协议进行验证和证明可以从数学的角度对一个协议的正确性给出严谨的理论推导。本文中只是给出了一些简单的描述，在以后的研究中还应在理论方面进行更进一步的研究。

参考文献

[1] Kai Hwang, Advanced Computer Architecture: Parallelism, Scalability, Programmability, McGraw-Hill, 1994.

[2] Kai Hwang, Zhiwei Xu, Scaleable Parallel Computing: Technology, Architecture, Programming, WCB/McGraw-Hill, 1998.

[3] 郑纬民，汤志忠，计算机系统结构，第二版，清华大学出版社，1998.

[4] M.Sato, COMPaS: a PC-based SMP cluster, IEEE Concurrency, 7(1):82-86, 1999.

[5] T. E. Anderson, D. E. Culler, D. A. Patterson, and the NOW Team, A Case for NOW (Networks of Workstations), IEEE Micro, 15(1):54-64, February 1995.

[6] B. S. Ang, D. Chiou, D. L. Rosenband, M. Ehrlich, L. Rudolph, and Arvind, StarT-Voyager: A Flexible Platform for Exploring Scalable SMP Issues, In Proceedings of SC98: High Performance Networking and Computing, Orlando, Florida, November 1998.

[7] S. L. Lumetta, Design and Evaluation of Multi-Protocol Communication on a Cluster of SMP's, Ph.D. Thesis, University of California, Berkeley, Berkeley, CA, 1998.

[8] J. Gray. Super-Servers: Commodity Computer Clusters Pose a Software Challenge, In G. Lausen, editor, BTW95, pp. 30-47. Springer-Verlag, 1995.

[9] Gordon Bell, Scalable, Parallel Computers: Alternatives, Issues, and Challenges, International Journal of Parallel Programming, Vol. 22, No.1, 1994.

[10] Matthias A. Blumrich, Kai Li, Richard Alpert, Cezary Dubnicki and Edward W. Felten, Virtual Memory Mapped Network Interface for the SHRIMP Multicomputer, the 21st International Symposium on Computer Architecture, April 1994.

[11] Matthias A.Blumrich, Cezary Dubnicki, Edward W. Felten, Kai Li and Malena R. Mesarina, Virtual-Memory-Mapped Network Interfaces, IEEE micro, February 1995.

[12] Cezary Dubnicki, Angelos Bilas, Yuqun Chen, Stefanos Damianakis, and Kai Li, VMMC-2: Efficient Support for Reliable, Connection-Oriented Communication, Computer Science Department, Princeton University, 1997.

[13] Cezary Dubnicki, Angelos Bilas, Kai Li, and James Philbin, Design and Implementation of Virtual Memory-Mapped Communication on Myrinet, IEEE micro, 1997.

[14] von Eiken, A. Basu, V. Buch, and W. Vogels, U-Net: A User-level Network Interface for Parallel and Distributed Computing, the 15th ACM Symposium on Operating Systems Principles, 1995.

[15] Matt Welsh, Anindya Basu, and von Eichen, Low-Latency Communicaiton over Fast Ethernet, http://www.cs.cornell.edu/Info/ Projects/U-Net, 1997.

[16] Matt Welsh, Anindya Basu, and von Eichen, Incorporating Memory Management into User-Level Network Interfaces, http://www2. cs.cornell.edu/U-Net, 1997.

[17] T. von Eicken, D.E. Culler, S.C. Goldstein, and K.E. Schauser. Active Messages: a Mechanism for Integrated Communication and Computation, In The 19th Annual International Symposium on Computer Architecture, pages 256--266, May 1992.

[18] Ellen Spertus and William J. Dally, Evaluating the Locality Benefits of Active Messages, Proc. of the 5th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, July 1995.

[19] AM-II specification. Active Messages Release 2.0. http://now.cs.berkeley.edu/AM/lamrelease.html.

[20] Xiao Limin and Zhu Mingfa, Active Message on the Dawning-1000, International Sysposium on Parallel Architecture, Algorithms and Networks, June 1996.

[21] N.J. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J.N. Seizovic, and W. Su. Myrinet: A Gigabit-per-second Local Area Network, IEEE Micro, 15(1):29--36, February 1995.

[22] Myrinet network products. Vendor homepage: http://www.myri.com/ .

[23] C. Dubnicki, A. Bilas, C. Yuqun, S. N. Damianakis, and L. Kai. Myrinet communication, IEEE Micro, 18(1):50--52, Jan.--Feb. 1998.

[24] Chun, B. N., A. M. Mainwaring, and D. E. Culler. Virtual Network Transport Protocols for Myrinet, IEEE Micro, 18(1):53-63, January 1998.

[25] R. Bhoedjang, T. Ruhl, and H. Bal. Design issues for user-level network interface protocols on Myrinet, IEEE Computer, 31(11):53-60, November 1998.

[26] 贺劲，基于 Myrinet 的高速通讯协议综述，第六届计算机科学与技术研究生学术讨论会论文集(下)，pp. 220-225，2000.07.

[27] Myricom. The gm api, http://www.myri.com/GM/doc/gm toc.html, February 2001.

[28] Myrinet, Inc. The GM Message Passing System, http://www.myri.com, February 2001.

[29] Prylli, L., and Tourancheau, B. BIP: A New Protocol designed for High-Performance Networking on Myrinet, In Proc. of PC-NOW IPPSSPDP98, March 1998.

[30] France Laboratory for High Performance Computing (LHPC) at Ecole Normale Superieure de Lyon (ENS-Lyon), Lyon. BIP Messages, http://lhpca.univ-lyon1.fr/bip.html, 2000.

[31] Basic Interface for Parallelism. http://lhpca.univ-lyon1.fr/bip.html, 2000.

[32] Loic Prylli. BIP User Reference Manual, http://www-bip.univlyon1.fr/software/bip-manual.ps, April 1997.

[33] MPICH-BIP/SMP, http://lhpca.univ-lyon1.fr/mpibip.html.

[34] 孙凝晖，徐志伟，曙光 2000 超级计算机系统软件的设计，计算机学报，Vol. 23 No.1, pp. 9--20，2000.

[35] Intel Compaq and Microsoft. Virtual interface architecture specification version 1.0. Technical report, http://www.viarch.org, December 1997.

[36] Technology Brief: Virtual Interface Architecture for SANs, Compaq Computer Corporation, 1997.

[37] S. Pakin, M. Lauria and A. Chien, High Performance Messaging on Workstations: Illinons Fast Messages (FM) for Myrinet, SC'95, San Diego, California, 1995.

[38] S.Paking, Karamcheti and A.Chien, Fast Message(FM): Efficient, Portable Communication for Workstation Clusters and Massively-Parallel Processors, IEEE Concurrency, 1997.

[39] Steven H. Rodrigues, Thomas E. Anderson and David E. Culler, High-Performance Local Area Communication with Fast Sockets, Technical reports, University of California at Berkeley, June 1996.

[40] S. S. Lumetta, A. M. Mainwaring, and D. E. Culler. Multi-Protocol Active Messages on a Cluster of SMP's, In Proceedings of Supercomputing '97, San Jose, CA, November 1997.

[41] Steven S. Lumetta and David E. Culler. Managing concurrent access for shared memory active messages, In International Parallel Processing Symposium, April 1998.

[42] Patrick Geoffray, Loc Prylli et Bernard Tourancheau, BIP-SMP : High Performance Message Passing over a Cluster of Commodity SMP's, SC'99, November 1999.

[43] Ma Jie, A Simplified Communication Protocol on Dawning 2000, HPC-Asia 2000, 2000.

[44] Ma Jie, He Jin, Meng Dan, Li Guojie, BCL-3: A High Performance Basic Communication Protocol for Commodity Superserver DAWNING-3000, Journal of Computer Science and Technology (Accepted).

[45] 孟丹，马捷，贺劲，BCL-3 总体设计，曙光 3000 技术报告，2000.

[46] 马捷，贺劲，BCL-3 详细设计，曙光 3000 技术报告，2000.

[47] 肖利民，机群高效通信系统研究，中国科学院计算技术研究所博士学位论文，1998.

[48] 毛永捷，用户级通信在软件分布式共享存储系统中的应用，中国科学院计算技术研究所博士学位论文，2000.

[49] M.Snir, P.Hochschild, D.D.Frye and K.J.Gildea, The Communication Software and Parallel Environment of the IBM SP-2, IBM Systems Journal, VOL 34, No. 2, 1995.

[50] ZhiWei Xu and Kai Hwuang, Modeling Communication Overhead: MPI and MPL Performance on the IBM SP2, IEEE Parallel and Distributed Technology, 1996.

[51] T. von Eicken, V. Avula, A. Basu, and V. Buch, Low-latency Communication over ATM Networks Using Active Messages, IEEE Micro, 15(1):46-53, February 1995.

[52] R. W. Wisniewski, L. I. Kontothanassis, and M. L. Scott, High Performance Synchronization Algorithms for Multiprogrammed Multiprocessors, In Proceedings of the 5th Symposium on Principles and Practice of Parallel Programming, pp. 199-206, Santa Barbara, California, July 1995.

[53] A. Singhal, D. Broniarczyk, F. Cerauskis, J. Price, L. Yuan, C. Cheng, D. Doblar, S. Fosth, N. Agarwal, K. Harvey, E. Hagersten, and B. Liencres, Gigaplane: A High Performance Bus for Large SMPs, In Proceedings of Hot Interconnects IV, pp. 41-52, August 1996.

[54] J. D. Valois, Implementing Lock-Free Queues, In Proceedings of 7th International Conference on Parallel and Distributed Computing Systems, pp. 64-9, October 1994.

[55] T. E. Anderson, The Performance of a Spin Lock Alternative for Shared-Memory Multiprocessors, IEEE Transactions on Parallel and Distributed Systems, 1(1):6-16, January 1990.

[56] M. Michael and M. Scott. Relative performance of preemption-safe locking and nonblocking synchronization on multiprogrammed shared memory multiprocessors, In Proceedings of the 11th International Parallel Processing Symposium, April 1997.

[57] M. Michael and M. Scott. Non-blocking Algorithms and Preemption-safe Locking on Multiprogrammed Shared Memory Multiprocessors, Journal of Parallel and Distributed Computing, 54(2), pp. 162--182, Feb. 1998.

[58] C. P. Kruskal, L. Rudolph, and M. Snir, Efficient Synchronization on Multiprocessors with Shared Memory, In Proceedings of the 5th Symposium on Principles of Distributed Computing, pp. 218-28, Calgary, Alberta, Canada, August 1986.

[59] M. M. Michael and M. L. Scott, Simple, Fast, and Practical Non-Blocking and Blocking Concurrent Queue Algorithms, In Proceedings of the 15th Symposium on Principles of Distributed Computing, pp. 267-75, Philadelphia, Pennsylvania, May 1996.

[60] S. K. Reinhardt, J. R. Larus, and D. A. Wood, Tempest and Typhoon: User-Level Shared Memory, In Proceedings of the 21st International Symposium on Computer Architecture, pp. 325-36, Chicago, Illinois, April 1994.

[61] B.H. Lim, P. Heidelberger, P. Pattnaik, and M. Snir, Message Proxies for Efficient, Protected Communication on SMP Clusters, In Proceedings of the 3rd International Symposium on High-Performance Computer Architecture, pp. 116-27, San Antonio, Texas, February 1997.

[62] N. S. Arora, R. D. Blumofe, and C. G. Plaxton, Thread Scheduling for Multiprogrammed Multiprocessors, In Proceedings of the 10th Symposium on Parallel Algorithms and Architectures, pp. 119-29, Puerto Vallarta, Mexico, June 1998.

[63] G. T. Byrd. Models of Communication Latency in Shared Memory Multiprocessors. Tech. Report CSL-TR-93-596, Stanford University, Dec. 1993.

[64] D. R. Cheriton and R. A. Kutter, Optimized Memory-Based Messaging: Leveraging the Memory System for High-Performance Communication, Computing Systems, 9(3):179-215, 1996.

[65] I. Foster, J. Geisler, C. Kesselman, and S. Tuecke, Managing Multiple Communication Methods in High-Performance Networked Computing Systems, In Journal of Parallel and Distributed Computing, Vol. 40, pp. 35-48, January 1997.

[66] I. Foster, C. Kesselman, and S. Tuecke, The Nexus Approach to Integrating Multithreading and Communication, Journal of Parallel and Distributed Computing, 37:70-82, August 1996.

[67] B. Falsafi and D. A. Wood, Scheduling Communication on an SMP Node Parallel Machine, In Proceedings of the 3rd International Symposium on High-Performance Computer Architecture, pp. 128-38, San Antonio, Texas, February 1997.

[68] Richard B. Gillett, Memory Channel Network for PCI, IEEE Micro, February, 1996.

[69] R. B. Gillett and R. Kaufmann, Using the Memory Channel Network, IEEE Micro, 17(1):19-25, February 1997.

[70] W. W. Gropp and E. L. Lusk, A Taxonomy of Programming Models for Symmetric Multiprocessors and SMP clusters, In Proceedings of Programming Models for Massively Parallel Computers 1995, pp. 2-7, Berlin, Germany, October 1995.

[71] J. Heinlein, K. Gharachorloo, S. Dresser, and A. Gupta, Integration of Message Passing and Shared Memory in the Stanford FLASH Multiprocessor, In Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 38-50, San Jose, California, November 1994.

[72] R. Martin, A. Vahdat, D. Culler, T. Anderson. Effects of Communication Latency, Overhead, and Bandwidth in a Cluster Architecture, International Symposium on Computer Architecture, Denver, CO, June 1997.

[73] R. Milner, Communication and Concurrency, Prentice-Hall International, 1989.

[74] C.A.R. Hoare, Communicating sequential processes, Communications of the ACM, 21(8):666--676, August 1978.

[75] C.A.R. Hoare, Communication Sequential Processes, Prentice-Hall International, 1985.

[76] A.W. Roscoe, The Theory and Practice of Concurrency, Prentice-Hall International, 1997.

[77] M. Hennessy, Algebraic Theory of Processes, MIT Press Series in the Foundations of Computing, MIT Press, 1988.

[78] J. A. Bergstra and J. W. Klop, Algebra of Communicating Processes with Abstraction, Journal of Theoretical Computer Science, 37:77--121, 1985.

[79] I. Lee, P. Br'emond-Gr'egoire, and R. Gerber. A Process Algebraic Approach to the Specification and Analysis of Resource-Bound Real-Time Systems, Proceedings of the

IEEE, pages 158--171, Jan 1994.

[80] K. Havelund and N. Shankar. Experiments in theorem proving and model checking for protocol verification, In M.C. Gaudel and J. Woodcock, editors, Third International Symposium of Formal Methods Europe (FME'96), LNCS 1051, pages 662-681, Springer-Verlag, 1996.

[81] Gerard J. Holzmann, The Model Checker Spin, IEEE Transactions on Software Engineering, Vol. 23, No. 5, May 1997.

[82] Gerard J. Holzmann, Design and Validation of Computer Protocols, Prentice-Hall International, 1991.

[83] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In Proc. 15th Work. Protocol Specification, Testing, and Verification, Warsaw, North-Holland, June 1995.

[84] Steve Schneider. Verifying authentication protocols with CSP, In Computer Security Foundations Workshop (14), pages 3--17.

[85] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus E.Schauser, Eunice Santos, Ramesh Subramonian and Thorsten von Eicken, LogP: Towards a Realistic Model of Parallel Compution, In Proceedings of the 4th Symposium on Principles and Practice of Parallel Programming, San Diego, California, May 1993.

[86] G. Iannello, M. Lauria, and S. Mercolino, LogP Performance Characterization of Fast Messages atop Myrinet, In Proceedings of the Sixth Euromicro Workshop on Parallel and Distributed Processing, pp. 395-401, Madrid, Spain, January 1998.

[87] A. Alexandrov, M. Ionescu, K. E. Schauser, and C. Scheiman, LogGP: Incorporating Long Messages into the LogP Model--One Step Closer towards a Realistic Model for Parallel Computation, In Proceedings of the 7th Symposium on Parallel Algorithms and Architectures, pp. 95-105, Santa Barbara, California, July 1995.

[88] Albert Alexandrov, Mihai F. Ionescu, Klaus E. Schauser, and Chris Scheiman, LogGP: Incorporating Long Messages into the LogP Model for Parallel Computation, Journal of Parallel and Distributed Computing 44, 71--79, 1997.

[89] D.E.Culler, L.T.Liu, R.P. Martin, and C.O. Yoshikawa, Assessing Fast Network Interfaces, IEEE Micro, 16(1):35--43, February 1996.

[90] L.T.Liu and D.E.Culler, Evaluation of the Intel Paragon on Active Message Communication, In Proceedings of the Intel Supercomputer Users Group Conference, June 1995.

[91] Bailey, D. et al. The NAS Parallel Benchmarks, RNR Technical Report, RNR-94-007, March, 1994.

[92] Franck Cappello and Daniel Etiemble, MPI versus MPI+OpenMP on the IBM SP for the

NAS Benchmarks, SC2000.

[93] Jenwei Hsieh, Tau Leng, Victor Mashayekhi and Reza Rooholamini, Architectural Performance Evaluation of GigaNet and Myrinet Interconnects on Clusters of Small-Scale SMP Servers, SC2000.

[94] Frederick Wong, Richard P. Martin, Remzi H. Arpaci-Dusseau, David Wu, and David E. Culler, Architectural Requirements and Scalability of the NAS Parallel Benchmarks, SC'99.

[95] Steve Sistare, Rolf vandeVaart and Eugene Loh, Optimization of MPI collectives on clusters of large-scale SMP's, SC99.

[96] Glenn R. Luecke and Ying Li, Performance Comparison of MPI, PGHPF/CRAFT and HPF Implementations of the Cholesky Factorization on the Cray T3E and IBM SP-2, The Journal of Performance Evaluation and Modelling for Computer Systems, January 1998.

[97] Glenn R. Luecke and James J. Coyle Iowa State University, Ames, Comparing the Performance of MPI on the Cray T3E-900, the Cray Origin2000 And The IBM P2SC, The Journal of Performance Evaluation and Modelling for Computer Systems, June 1998.

[98] Glenn R. Luecke, Bruno Raffin and James J. Coyle, Comparing the Communication Performance and Scalability of a SGI Origin 2000, a Cluster of Origin 2000's and a Cray T3E-1200 using SHMEM and MPI Routines, The Journal of Performance Evaluation and Modelling for Computer Systems, October 1999.

[99] Glenn R. Luecke, Bruno Raffin and James J. Coyle, The Communication Performance and Scalability of a Linux and an NT Cluster of PCs, a SGI Origin 2000, an IBM SP and a Cray T3E-600, The Journal of Performance Evaluation and Modelling for Computer Systems, March 2000.

[100] S. S. Lumetta, A. Krishnamurthy, and D. E. Culler, Towards Modeling the Performance of a Fast Connected Components Algorithm on Parallel Machines, In Proceedings of Supercomputing 1995, San Diego, California, December 1995.

[101] R. H. Saavedra, Micro Benchmark Analysis of the KSR1, In Proceedings of Supercomputing 1993, pp. 202-13, Portland, Oregon, November 1993.

[102] E. Speight and J. K. Bennett, Using Multicast and Multithreading to Reduce Communication in Software DSM Systems, In Proceedings of the 4th International Symposium on High-Performance Computer Architecture, pp. 312-22, Las Vegas, Nevada, February 1998.

[103] M. L. Scott and A. L. Cox, An Empirical Study of Message-Passing Overhead, In Proceedings of the 7th International Conference on Distributed Computing Systems, pp. 536-43, Berlin, West Germany, September 1987.

致　谢

历经三年的紧张艰苦的研究工作，在前人众多研究成果的基础上和整个研究团体的集体努力下，我得以完成这篇论文。在此，我特别感谢所有曾经帮助过我、为我的研究工作提出建议和指导的人们。

首先，我衷心感谢我的导师李国杰院士三年来对我的关心和指导。李老师渊博的学识、严谨的治学态度、孜孜不倦忘我工作的精神和灵活的思路都使我受益匪浅。这不但激励着我这三年来的博士研究工作，也同样将激励我在科学研究道路上的继续探索。

感谢计算技术研究所的徐志伟研究员。徐老师一直关心着曙光 2000 及曙光 3000 项目的研究工作，并提出了许多建设性的意见。感谢高性能计算机研究室的主任孙凝晖研究员和副主任孟丹副研究员。他们在曙光 2000 及曙光 3000 通信系统的研究工作中给我了巨大的帮助。他们的治学态度和进行科研工作的方法不但在博士研究工作中给我以启迪，也将对我将来的工作和学习起到重大的作用。

感谢高性能计算机研究室的副主任侯建如研究员和张佩珩高级工程师。每当我的研究工作中对硬件属性出现疑问和需要搭建实验平台时，都能够得到他们热心的帮助，使我的研究工作能够顺利进行。

感谢高性能计算机研究室系统软件组组长肖利民副研究员。在他的组织和帮助下我才能够在较少的时间内熟悉了前人的工作和当前的研究环境，能够以最快的速度投入研究工作。感谢与我合作进行研究工作的同学和同事，本文的工作都是在大家共同的努力下完成的。特别感谢先后与我合作的吴结生和贺劲博士。没有他们的通力合作我将无法完成我的研究工作。同时也感谢丁建峰、王哲和张剑峰在系统性能测试中对我的帮助。

感谢高性能计算机研究室中的同事和同学们，感谢参加博士生讨论班的博士生及老师们。他们在日常工作和讨论中的想法和思路为我的研究工作提供了很多启示。

最后，我希望能把这篇论文献给我的父亲母亲。在我小时候他们就期望我长大后能够在科研事业上有所作为。我愿这篇论文能够报答他们的养育之恩，不辜负他们对我的期望。

作者简介

个人资料

姓　　名：马　捷
性　　别：男
出生年月：1975 年 11 月
地　　址：中国科学院计算技术研究所高性能计算机研究室
电　　话：(010)62613792

学习经历

1991 年 9 月至 1995 年 7 月	西安交通大学 计算机科学与工程系 获工学学士学位
1995 年 9 月至 1998 年 7 月	西安交通大学 计算机科学与工程系 获工学硕士学位
1998 年 9 月至 2001 年 7 月	中国科学院计算技术研究所 高性能计算机研究室 攻读博士学位

科研经历

博士在读期间主要从事机群通信系统的研究。参加了曙光 2000-I 的测试工作、曙光 2000-II 和曙光 3000 超级服务器中机通信子系统的设计开发工作。工作集中于机群通信协议的研究。从用于曙光 2000-II 的 BCL-2 到曙光 3000 上的 BCL-3，研究了用户级通信协议、通信协议中的内存 0-拷贝技术、半用户级通信协议以及多重通信协议等。并在协议的形式化表示与验证方面进行了初步的研究。目前主要的研究方向包括高性能体系结构、机群操作系统、高性能通信协议和并行计算。

发表文章目录

1. Ma Jie, A Simplified Communication Protocol on Dawning 2000, HPC-Asia 2000, 2000.05.
2. 马捷，并行程序性能可视化分析系统的研究与实现，“网络时代的计算技术”第六届研究生学术研讨会，2000.07.
3. Ma Jie, He Jin, Meng Dan, Li Guojie, BCL-3: A High Performance Basic Communication Protocol for Commodity Superserver DAWNING-3000, Journal of Computer Science and Technology (Accepted).