摘要

并行计算模型实质是从不同的并行计算机体系结构模型中抽象出来，为并行算法设计与分析提供具有指导意义的理论界面和模型框架。当前并行计算模型很多，其中的 BSP 模型由于定义了程序的结构，能灵活引导用户进行程序设计而逐渐被广为接受。传统的 BSP 模型都是基于大型机的中央处理方式、分布式计算环境等并行计算机体系结构建立的，严格按照该模型设计的并行程序容易产生通信拥挤，并由此影响程序的运行效率等问题。目前并行计算环境发展到了网络并行计算环境，原有的 BSP 模型显然已经不能很好的适应网络这个新型的并行计算环境，本文正是在基于现代网络的并行计算环境下对 BSP 模型作了深入研究并对其进行相应扩展，从而使扩展后的模型能适应网络并行计算环境。

目前关于网络并行计算环境的文献很多（如异构并行计算环境等[1][4][6]），文章总结归纳出了几种典型的并行计算环境，并在分析这些并行环境特性的基础上，描述了一种网络并行计算环境，并对该环境的异构性、非独占性和透明性等主要特征作了研究分析。

论文重点研究了网络环境下的 BSP 模型的扩展。由于 BSP 模型在现有并行环境下的局限性以及网络并行计算环境的要求，BSP 模型面临必然的扩展。由于网络并行环境有其专门的网络管理系统，能合理调度网络系统中各节点间的通信，所以在 BSP 模型中引进关键消息算法[10]，提出了扩展的 BSP 模型（E-BSP）。该模型用优先级通信机制替代传统的 FIFO 机制，其中的关键消息算法能根据任务的紧急程度，赋予各节点不同的优先级，高优先级的节点都存储在关键路径中，NMS 负责从关键路径中调度各节点进行通信，从而改善节点间的通信效率，提高整个并行程序的运行效率。最后就 E-BSP 模型的开销计算和代价公式作了详细的分析和推导。

论文最后，给出了扩展的 BSP 模型在网络环境下可扩展性的详细分析，给出了常用的可扩展性度量方法，并就进一步完善和改进提出了意见。

关键词： 网络并行计算环境，关键消息算法，E-BSP 模型，可扩展性

Abstract

Parallel computing model is materially abstracted from parallel computer architecture, and is an offer of instructional theory interface and model framework for parallel arithmetic. Nowadays there are many parallel computing model, the BSP is generally accepted by researcher duing to it has defined the program structure and inducted user programming easily. The old BSP is based on central processing in mainframe computer and distributed parallel computing environment. Parallel programs that found on this model easily cause communications bottleneck and the low run efficiency. Now the parallel computing environment is running to network parallel computing environment, then the old BSP model doesn' t fit the new environment. So this paper will extend the BSP model in network parallel computing environment to suit with this environment.

After reading many literatures on parallel computing, such as "isomerous parallel computing environment" e.g. This paper will sum up several typical parallel computing environments. Then the paper' ll describe a parallel computing environments, and analyze the characteristic of this environments.

This paper will emphasize the extensive BSP model in network parallel computing environment. Because of the model' s short-coming in network environment and this environment self-demand, lead to extend the old BSP model necessarily. As this network environment has special network management system (NMS) that can distribute communications reasonablely among those nodes in network environment, so this paper will induct key message arithmetic in the BSP model, and then define the extensive BSP

(E-BSP). We make use of priority rather than FIFO communication principle, the arithmetic put different priority according the tasks' urgency, the NMS then takes out the high priority nodes that waited in the key message queue. So improve on the communication efficie -ncy among nodes and advance the whole performance of parallel programms. We also full analyze the spending and cost of this E-BSP.

At last, this paper' ll analyze elaborately the expansibility of this E-BSP model in the network parallel computing environment, and describe the measurement formula. Moreover, put forward advice for the next improvement work.

Keywords: network parallel computing environment, key message arithmetic, E-BSP model, extensibility

独创性声明

本人声明，所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽本人所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京交通大学或其他教学机构的学位或证书而使用过的材料。与我一起工作的同志对本研究所做的任何贡献已在论文中作了明确的说明并表示了谢意。

本人签名：_____

日期：_____年___月___日

关于论文使用授权的说明

本人完全了解北京交通大学有关保留、使用学位论文的规定，即：学校有权保留送交论文的复印件，允许论文被查阅和借阅；学校可以公布论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存论文。论文中所有创新和成果归北京交通大学计算机与信息技术学院所有。未经许可，任何单位和个人不得拷贝。版权所有，违者必究。

本人签名：_____

日期：_____年___月___日

1 综述

1.1 课题背景和意义

1.1.1 课题背景

当前，大规模高性能计算正受到前所未有的关注。一方面，由于在诸如预测模型的构造和模拟、工程设计和自动化、能源勘探、医学、军事以及基础理论研究等诸多领域中都对计算提出了及高的具有挑战性的要求，以期通过快速而精确的计算来完成自身的需求；另一方面，高性能计算研究领域的不断突破及所取得的长足进步（计算机的理论峰值在 20 年里提高了 4 个数量级，从一亿次增加到一亿万次浮点运算/秒）使得这些任务的顺利完成有了可靠的保证。未来随着人们对高性能计算继续永无止境的追求，可以设想，并行计算机和并行计算将是不可抵挡的发展潮流。

近年来，由于微机技术和网络技术的不断发展，并行计算平台也开始利用工作站集群系统，与以往的并行计算环境相比，该环境具备一定的异质性和非独占性，即组成系统的工作站的计算能力各不相同，并且每个工作站上可能有其他用户计算在执行。但是，早期这种分布式并行计算环境中，因为没有统一管理，各工作站之间的计算、尤其通信容易发生拥挤，故运行效率低下，系统可靠性差。所以，为了得到分布式并行计算环境所具有的优点，同时使其实现较容易，出现了一种折衷的办法，即在当前的并行计算环境之上，构筑一个可以屏蔽系统的结构特征、提供透明访问和协同合作的系统—网络并行计

算环境[10]。

但是由于网络并行计算环境具有异质性、非独占性和透明性等新特性，将传统的并行计算模型简单移植后并没有体现新系统的这些特性，由此就导致了并行计算模型对于这种网络并行计算模式显得有点苍白，为了保证我们需要的并行计算模型是一个能在比较容易实现的同时又在适合网络计算模式，就必须对原有的并行计算模型进行改进。

1.1.2 研究的必要性及意义

纵观当前高性能计算领域技术的发展， 关于并行计算模型的概念已经越来越清晰和统一。从总体上看，对并行计算模型的研究经历了较有影响的 PRAM->BSP->LogP->LogGP[1]的几个阶段，由于其中的 BSP 模型定义了程序的结构，能够灵活引导用户进行程序设计，使得基于该模型的并行算法设计更加简单、高效，故该模型倍受研究人员的推崇和青睐。所以研究人员对并行计算模型的研究又重新回到 BSP，同时对 BSP 的概念和优缺点有了更深刻的认识和了解。

尽管 BSP 并行计算模型指导下的并行程序设计具有编程简单、独立于目标结构、执行性能可预测性等优点。BSP 程序由于其超步的结构使得消息传递不会出现死锁，同时在正确性、性能分析和编程方法上都简单易行，很适合异构环境下的消息传递编程。但由于模型中每个超步中都是必须先各自完成计算后再进行通信，这样不仅计算和通信重叠度不高而且容易发生通信阻塞。且仍存在超步间通信满足 h-relation，g 的测算不够精确，在异构环境下不再合适等问题。因此如何使得计算和通信同步，同时又尽可能减少通信所带来的系统瓶

颈、使得 BSP 模型在异构环境下有较好的适应性是一个非常难驾驭的问题。

为解决这个问题，本文基于网络并行计算环境对 BSP 模型进行扩展。由于该环境中网络管理系统可对各种资源进行统一管理，我们引进关键消息算法，对 BSP 模型进行扩展，使得在超步中计算和通信可以异步执行，且需通信的消息在等待队列中不再以 FIFO（First in First out）方式而代之以优先级的方式等待调度。这样，超步中计算和通信重叠度增加，超步间的通信效率也大大提高了，消除了系统在通信瓶颈的限制，使并行程序设计性能和用户响应速度都得到很大的改善。

1.2 BSP 模型研究现状

国外过去对 BSP 模型的研究，主要是从理论分析的角度，来研究它之上的一些典型的并行算法的分析与设计，而近期的研究热点却从在模型上的算法研究转向模型的编程研究，即从理论研究转向实际应用，最近的一些研究可以参考文献[14][17][22][25]，因为任何并行算法的应用最终都要落实到具体的编程上面，所以这种转变是顺应应用要求的。如文献[3]提出了异步 BSP 模型，该模型取消了同步限制，程序可以异步执行，这种转变提高了程序的执行效率和吞吐率。文献[4]提出的 NHBSP 模型，该模型将通信操作看成一个整体，使得我们的并行程序设计更加简单易行。

目前，对并行计算模型的研究受到越来越多国内学者的关注。且已经不再是仅仅受到算法研究者的密切关注，它也受到越来越多的软件设计者和硬件设计者的关注。这是因为在诸如设计问题的解决方案、软件编码、将算法转换为高效的机器指令执行序列、设计高效的

执行硬件等一系列过程中，人们都对建造计算模型提出了自己的要求。尤其随着因特网的飞速发展，如何利用高速低廉网络成为并行研究者的工作重点。如文献[5]中提出适合基于工作站集群系统的 NHBL 计算模型。本文也将建立全新的高速网络并行计算系统结构，并基于此对当前的 BSP 模型进行扩展。

1.3 论文主要研究内容

本文的工作分为以下几部分：

1. 第一章简要介绍了并行计算模型的重要性及发展现状。
2. 第二章首先分析了传统并行计算环境的不足，然后描述了网络并行计算环境，并对其特性进行分析。
3. 第三章首先引入了关键消息算法，并通过将该算法应用到 BSP 模型中对其进行网络环境下的扩展，接着，给出该扩展的 BSP 模型的性能评价和代价公式，最后对改进后的模型进行性能评价及应用分析。
4. 第四章对扩展的 BSP 模型在网络并行计算环境下的适应性进行分析，并给出适应性的度量方法。
5. 第五章是本文的总结，包括文章的结论和后续工作。

2 网络并行计算环境

2.1 几种典型并行计算环境

2.1.1 并行计算综述

并行计算机是由多个处理器单元组成的计算机系统，这些处理单元相互通信和协作，能快速、高效地求解大型复杂问题。在并行机上求解问题，首先要写出求解问题的并行算法，而并行算法是在并行计算模型上设计出来的，而并行计算模型是从不同的并行计算机体系结构模型中抽象出来的。当前流行的并行计算机体系结构有 SIMD 计算机，PVP，SMP[6]，COW，DSM[7]等。

并行计算模型是从不同的并行计算机体系结构模型中抽象出来的，是一个将高层次特点与低层次特性分离开来的界面，提供在其上进行编程的特定操作，对其下所有结构中每个操作的实现做出要求。它用来把软件设计与高效并行执行分离开来，提供抽象性和稳定性。抽象性是由于模型提供的操作比底层结构所提供的操作高级得多，简化了软件结构，减少了软件设计的困难。稳定性是由于软件设计可以基于一个长时间保持稳定不变的标准接口，不用考虑并行计算机结构的变化。同时，模型形成了每个并行机实现时所基于的固定起点（程序转换系统、编译和运行系统）。

2.1.2 大型机的中央处理方式

大型机的中央处理方式[15]以管理系统为核心统一管理和协调所有的 CPU，管理系统与 CPU 的通信都是通过本地的总线实现的。这种模式的控制、管理功能都集中在管理系统上，管理系统将一个程序分割成多个程序段分别在不同的主机上执行，从而获得高的处理性能，这种模式中各个处理机之间不能相互通信，只能通过管理系统进行信息的交换。

在这种管理方式下，管理效率和数据传输速度非常高，这相应地使得整个并行程序的运行效率很高，而且系统的可靠性很高。但是，这种方式由于以大型机为硬件环境，故购置、维护、管理费用昂贵，再加上扩展性差，移植性差使得原本卓越的性能也变得黯然失色。

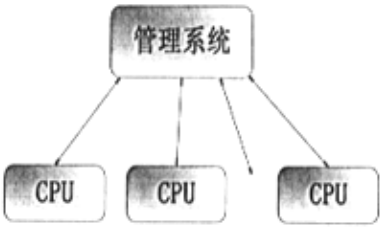

图 2.1 大型的中央处理方式

2.1.3 分布式计算环境

为了解决大型中央处理方式的昂贵的维护管理费用，及改进扩展性移植性，有些文献采用另一种新型的计算环境-分布式计算环境[15]。可以看出，在分布式计算环境中，没有一个像大型机中的中央管理系统，各个主机之间通过网络连接，自由耦合，并没有一种固定的协作

关系。这种模式将控制、管理功能放到每个主机节点上，依据现场智能设备实现基本控制功能，完成相应的计算任务。但是这些节点主机并不是孤立的，为了完成某个计算任务，这些节点主机之间必须进行信息交换，接受监控计算机的调度、管理，协同地完成一项更大型、更复杂的任务。在一定程度上，这些节点可以并行的、独立的完成一些计算任务，因而具有一定的并行特征。所以，这种计算模式是由基于网络互连的一组异构节点构成的具有超强计算能力，用以完成复杂的控制/管理活动的分布式协同计算模式，其计算模式是一种典型的松耦合的分布式并行计算模式。

这种环境要实现的主要目标[7]是系统互联，资源共享，协同工作，任务并行和容错与负载均衡。该计算环境有如下特性[8]：自治性，透明性，并行性，可扩展性。相对于大型机来说，这种方式的优越性相当突出，即价格低廉、可扩展性强、可移植性好。但是，这种计算环境中，因为没有统一管理，故运行效率低下，系统可靠性差。

图 2.2 分布式计算环境

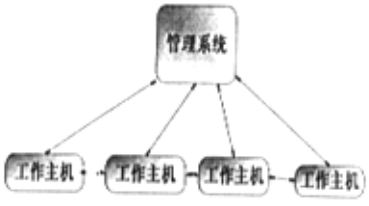

图 2.3 网络计算环境

2.2 网络并行计算环境

2.2.1 网络并行计算环境定义

为了得到分布式计算环境所具有的优点，同时又避免其运行效率低下，系统可靠性差的缺点，使这种松耦合的计算环境实现较容易，出现了一种折衷的办法，即在流行的分布式计算环境之上，构筑一个可以屏蔽系统的结构特征、提供透明访问和协同合作的系统-网络并行计算环境[18]。

定义 2.1 我们称满足以下约束条件的并行计算模式为网络并行计算环境。

1. 系统中有专门的网络管理系统（NMS），负责对多个处理主机进行统一管理。
2. 体系结构/处理其透明性：用户不知道系统中有多少机器及其类型。
3. 异构性：系统中的机器类型，计算能力可以各不相同。
4. 并行透明性：用户提交的作业自动可分配到处理器并取得最大限度的并行执行。
5. 非独占性：每个节点上可能有其他节点的计算在进行。
6. 任务位置透明性：用户并不知道作业在多少处理机及在哪些处理机运行。
7. 错误透明性：在某处理机出现问题时，系统能及时发现并自动把它调度到其它处理器运行。
8. 对内部资源的使用和编程是透明的。
9. 系统是可以方便灵活扩展的，易于使用的，必须支持传

统的高级语言。

10. 包括对异构性的支持，解决更为突出的安全性问题。

综上所述，网络并行计算环境是通过商用网络互联一组的独立计算单元和相关资源组成的具有单一映像的计算环境，该系统中具有一个类似大型机中的中央管理系统，负责将一个程序分割成多个程序段分别在不同的主机上执行，从而获得高的处理性能，并可以对多个处理主机进行统一管理。除此之外，管理系统与工作主机之间以及各个工作主机间以网络互连，彼此之间通过网络通信。

2.2.2 网络并行计算环境组成

上一节从概念上详细描述了网络并行计算环境，并通过与传统的计算环境对比，横向给出了网络并行计算环境的特性及组成，本节将从纵向详细介绍网络并行计算环境的组成。

a. 底层结构

底层结构是由局域网络和操作系统构成，是整个并行计算环境的基础。这里的局域网络可以是当前的各种局域网技术，不过为了提高网络并行计算环境的性能，应优先采用高速局域网络技术。操作系统可以是任何一种多任务的操作系统。也就是说网络并行计算环境的底层结构应该是一个应用目前的各种局域网技术，比如 Ethernet、FDDI、ATM 等将运行于各种多任务操作系统的主机连接在一起的局域网络。

基于网络的并行计算环境中，各个处理机都有自己的操作系统，对本地资源有很高的自治性，同时又通过网络并行计算环境核心层提供对网络应用系统的支持。这样用户既可以在原有的平台上工作，使用本地资源，又是该网络并行计算环境中的一分子，可以通过网络并

行计算环境为他提供其它的服务。

b. 核心层

如上所述，在底层结构的上一层是网络管理系统（中间核心层），这一层是整个计算环境的核心，它赋予该环境并行计算能力，该系统的功能是屏蔽底层平台相关细节并向上一层的用户并行程序提供并行计算服务。该计算环境的整体描述如下图 2.4：

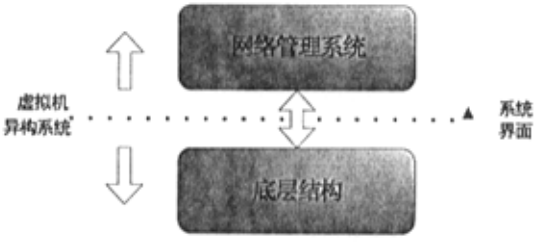

图 2.4 网络并行计算环境的构成

由上图可以看出，网络并行计算环境的两个主要模块，即网络管理系统和底层结构，以图中所示的模式交互并在模块间形成一个系统界面（或系统服务界面），如图中虚线所示。以系统界面为基点观察相关的两个系统模块会产生两个视图，为了深入地分析系统特性，下面的两节将对这两视图分别展开讨论。对网络并行计算环境观察所形成的视图是一个网络并行虚拟机，从而为用户提供平台无关的服务。而对底层结构观察所形成的视图则是一个异构模型，该模型是一个实际系统描述模型，它包含了构成网络并行计算环境的底层结构中所有细节的描述。

c. 应用层

这是该环境的最高一层，直接面向最终用户。并且由核心层屏蔽了并行系统的结构特征，提供透明的访问，用户并不知道作业在多少

处理机及在哪些处理机运行，在某处理机出现问题时，网络管理系统能及时发现并自动把它调度到其它处理器运行。

应用层和核心层之间的通信是有其专门的领域接口来实现的，这些接口与特定的领域有关，例如，制造业、金融业、电信行业等。不同领域服务提供不同的接口，该接口技术即基于核心层提供的一组更高层的函数，这些函数包括用户界面、信息管理等方面的一些通用设施，为最终用户提供一组共享服务接口，例如组合文档、系统管理和电子邮件服务等，也称为水平公共设施。该领域接口是由销售商提供的、可控制其应用接口（用户与应用层之间的接口）的产品。

2.3 网络并行计算环境特征

2.3.1 异构性

在一个实际的网络并行计算环境中，从网络硬件到主机，再到操作系统都可能来自不同的生产厂商，而这些不同的系统之间无论是从网络的拓扑结构到网络数据传输模式，还是从主机的硬件体系结构到其中的操作指令，以及从操作系统中的数据格式到用户编程及操作模式都不尽相同，除此之外，不同类型网络的数据传输速率以及不同类型主机的处理性能都有相当的差异，这样的网络并行计算环境构成了一个异构环境。在分析这样一个异构环境的时候，总是希望将它模型化，这样既便于分析、理解，也使得问题简单化，而且具有代表性。那么，可以基于这样一种思想，可以从上面所述的异构环境中抽象出一个异构模型。

在上一节提出的网络并行计算环境中，以系统界面为基点对底层

结构观察所形成的视图是一个异构模型。该模型的研究对象是一个一般化的网络并行计算系统，该系统中具有任意类型的网络、任何可能的主机设备以及可以运行于其上的任何操作系统。则该模型在以下三个层次上体现了异构特性。异构性是网络并行计算环境最本质的性质，该特性决定了其其他特性，所以本节将对其进行详细的描述。

a. 操作系统层

操作系统是与主机硬件直接相关的，不同类型的主机上运行着不同的操作系统。而不同的操作系统提供给用户的应用程序、编程界面、处理性能（包括 CPU 调度效率、进程管理以及文件系统的性能）以及数据表示格式等都不尽相同，这样，他们提供给用户的各种操作和编程界面也因系统而异。而应用系统直接建立在操作系统层上，这样对应用系统来说底层结构的异构性就直接体现在操作系统层的异构性上。

b. 主机硬件层

来自不同厂商的主机内存，决定了以下两种结果：首先是主机类型（硬件结构）的差异由操作系统所屏蔽，并决定了在其上运行的操作系统的异构性（在上面的操作系统层已经描述）。其次，应用系统可见的是由主机硬件决定的主机性能的不同。因此，底层结构的异构性体现在主机硬件层上，主要是不同的主机处理性能的差异，这也是造成整个系统异构性的重要因素，并需要在网络并行计算系统中给与特殊的处理以实现良好的并行计算性能。

c. 网络层

如上所述，一个异构环境允许不同类型的网络共存，比如说 Ethernet、FDDI、Token Ring、ATM 等。就像主机硬件层中硬件结构

的差异可以由操作系统屏蔽一样，不同的网络中数据的传输格式、传输模式的差异对应用系统来说是透明的。因此，影响应用系统的只是数据传输速率的不同，这在网络层上体现出了网络并行计算环境的异构性。

2.3.2 透明性

基于网络并行计算环境的这些底层处理机一方面都有自己的操作系统，对本地资源有一定的自治性，另一方面又可以通过网络提供对其上应用系统的支持。这样用户既可以在使用自己的原有平台下工作，又可以通过网络提供或享受其它服务，且并行计算网络屏蔽了底层的异构操作系统，隐藏了很多实现细节，包括物理位置、并发控制、错误处理，对系统的用户提供了系统细节的透明性。所以系统扩展非常方便。该系统的目标是互连或者集成已经存在的独立的、或者异构的系统或设备，负责给用户提供一组统一的编程语言、编程工具以及运行时的环境。

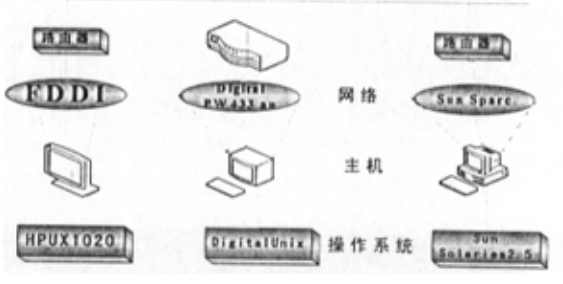

图 2.5 网络并行计算环境中底层结构异构典型实例

图 2.5 所示为一个异构模型的典型实例。其中，网络类型包含 FDDI、Ethernet 和 ATM 网络等等，而挂在这些网络上的主机类型则有

HP 的 HP9000/712/80、DEC 的 Digital PW433au 以及 Sun 的 Sun Sparc 等，相应的操作系统是 HPUX10.20、Digital UNIX4.0b 和 Sun Solaries2.5。该实例在网络、主机和操作系统三个层次上都集中体现了本节所述的异构模型中所蕴含的网络并行计算环境的异构和透明特性。

2.4 本章总结

网络并行计算环境的体系结构看上去和前两种都有些类似之处。这使得该环境继承了以上两种环境的优越性。首先，由于中央管理系统的并行计算环境以及管理模式，使得整个系统的性能、运行效率、可靠性和资源利用率提高。另外，由于各个工作主机以网络互连，故系统的扩展性好，灵活性强。而且，该环境中的工作主机的性价比远比大型机中的单个 CPU 要吸引人，同时该环境对工作主机的选用没有软硬件平台的限制，故移植性超强。然而，由于该环境有 NMS 统一对全局管理，对每个处理单元的并行计算自治控制有一定要求，所以呈相对弱自治性。

从以上的分析可以知道，网络并行计算环境充分继承了大型机的中央处理方式和分布式计算环境的优越性，并摒弃了前面两种环境的弱点，成为一种性能卓越、价格低廉的计算环境。

3 扩展的 BSP 模型

3.1 BSP 模型

3.1.1 BSP 模型

BSP(块同步并行)模型[16]定义一个并行结构由以下三个部分组成：

① 一组处理器/存储器模块对；

② 实现处理器/存储器模块对之间点到点传递信息的选路器；

③ 执行同步所有处理器/存储器模块对的全局通信机制。

根据上述情况，BSP 计算模型设定了三个定量参数：p 表示处理器数量；g 表示选路器吞吐率，也称带宽因子；L 表示全局路障同步之间的时间间隔。

BSP 计算模型的一个超步里，每个计算操作只使用它自身局部存储器中的数据。这些数据或在程序启动时，或由以前超步中的通信操作将其存放到局部存储器中。所以一个进程的计算操作与其它进程无关。通信总是以点对点方式进行，因此不允许多个进程在同一周期对同一存储单元进行读写。

3.1.2 BSP 模型的局限性

由前面的分析可知，由于网络并行计算环境具有异构性、非独占性和透明性等特性，由前面的分析可知，传统的并行计算模型不能体

现这些特点，使得原有的并行计算模型对于这种网络并行计算环境显得无能为力，因此，我们需要一个既适合网络计算环境又容易实现的并行计算模型，而 BSP 模型由于易于编程易于扩展等特性，再次受到我们的关注。针对 BSP 模型中计算和通信同步和在异构环境下的不适应性等问题。我们引进关键消息算法[11]，并将其应用到 BSP 模型中，对 BSP 模型进行了扩展。

3.2 关键消息算法

3.2.1 相关定义

a. 背景知识

网络环境下的并行计算， TCP/IP 协议栈在通信中是主要的瓶颈，并引起并行应用消息传递中的主要通信延时，且在目前的 TCP/IP 协议栈中，消息都是以 FIFO 方式处理的。因而，在通信网络负载很大的情况下，需通信的消息在消息队列中要等待很长时间，因为不得不等待前面的消息都被处理，这样很容易发生通信阻塞。

为了解决这个问题，我们提出关键消息模型（以下简称 KMM）。在介绍该算法之前，我们先定义如下相关概念。

b. 相关概念

关键消息算法：根据消息通信量、通信紧急程度给各消息赋予不同的优先级，并自动搜索高优先级的消息，放入关键消息路径。

关键消息路径：由关键消息算法生成的优化路径。其实质是存储高优先级的消息的数据结构。

关键消息：由并行应用的关键消息路径生成的高优先级消息。

MPI：消息传递界面（Message Passing Interface）的简称。它将并行编程环境分解成两个部分，第一是构成该环境的所有消息传递的标准函数的标准接口说明，它们是根据并行应用程序对消息传递功能的不同要求而制定的，不考虑函数具体能否实现。第二是并行机厂商提供的对这些函数的具体的实现。这样用户只需要学习 MPI 库函数的标准接口，设计 MPI 并行程序，便可在支持 MPI 并行环境的具体并机上执行该程序。

BSP 模型[10]：块同步并行计算模型由一组处理器/存储器模块对、实现处理器/存储器模块对之间点到点传递信息的选路器、执行同步所有处理器/存储器模块对的全局通信机制组成。

c. 关键消息算法（KMA）

我们的关键消息模型（以下简称 KMM）优化通信方案中提出了 TCP/IP 协议层中的关键消息路径 KMP 概念（即高优先级的消息队列）。关键消息模型中通信优化算法根据消息通信量、通信紧急程度给各消息赋予不同的优先级，并自动搜索高优先级的消息，放入 KMP。因为在等待队列中高优先级的消息将比低优先级的先处理，而由关键消息路径生成的关键消息（包含那些本地串行应用）放在 KMP 中。所以它们在队列中的等待时间就会变短，从而关键消息的处理时间越短，关键消息路径的处理速度也越快，这样整个并行应用的处理时间就将低了。

3.2.2 关键消息模型

网络并行计算环境是并行计算的成本-效率平台，已经研发出一些软件包为网络并行计算提供环境，如 MPI、PVM，还有[6]中，活动消息

和快速消息等。Kale[7]等人曾经提出利用操作系统提供的基于优先级的进程进度表，利用该进度表加快并行任务的处理速度。尽管该高水平的进度表在并行应用计算中增加了优先级机制，但在接下来的通信任务中并没有优先级概念。绝大多数的 DAG 前期工作[8]主要集中在任务进度表中的算法研究上，很少有研究把重点放在为降低进程间的通信时间提供有效的技术指导上。Dong 虽然在 NOW 中研究了通信优先级[9]，但在他们的研究中，设置并行应用中的通信进程优先级高于串行应用中的通信进程，并且只涉及到 SPMD 的并行应用，并且只是在模拟分析中阐述了使用优先级的最优化效率。

我们的关键消息模型不同于 Dong 的最优化技术在于并行应用中的消息被赋予不同的优先级，我们的最优化目标是通过识别需要被优先传递的关键消息来减少并行应用中的执行时间。除此之外，我们的方法不仅适应 SPMD 而且适应 MPMD 并行应用，在关键消息方法中，只有在关键路径中的消息才有优先权。下面的章节我们将详细讲述关键消息模型及相关算法。

a. 系统结构

首先我们介绍一下关键消息模型的系统结构，如图 3.1。首先并行应用经由任务图翻译器处理生成任务图；接下来由关键消息算法根据网络负载的不同给任务图指定优先级，得到具有不同优先级的任务图；接下来，利用代码生成器生成基于关键消息 API 和优化后任务图的修改后的并行应用；最后，修改后的并行应用与关键消息库连接，经编译后生成可执行代码，该可执行代码将在关键消息运行系统最上层被执行。

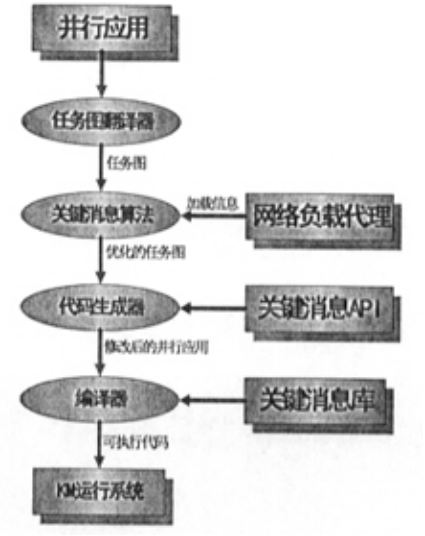

图 3.1 KM 系统结构模型

b. 关键消息算法

接下来，我们将详细介绍关键消息算法（以下皆简称 KMA）。KMA 的基本思想是给并行应用关键路径中的消息指定优先级。任务图用来抽象并行计算，在每一个最优化步骤中，KMA 从任务图中查找出一条关键路径并且使沿着该关键路径下的消息传递进行最优化处理。任务图中边的权重由总的延时公式根据优先-抢占策略[1]调节。优先级分为两个层次，指定要被最优化的消息被置为高的优先级，其它的消息（包括串行进程需要通信的消息）则被置为低的优先级。算法在没有消息要优化的时候终止。

不同节点间并行任务的消息传递被模型化为消息队列，如 FIFO 优先级策略的 M/M/1 队列系统。如果通信网络是共享媒介质的话，在关键消息模型中，每个信息包都会经历网络中的队列延时的变化。

假设 1 在一个节点中，高优先级的并行任务没有队列等待延时；

唯一的等待延时是服务时间。

（1）算法输入

①包括一系列路径的任务图，每个路径包括一系列的任务（消息）和权重，任务权重是完成该任务需要的时间，边的权重是每个包中消息间的数据发送次数。

②有优先级的队列延时需求到达率，用（λ_1,λ_2）来表示；服务率，由网络负载代理提供的数据来计算，用（μ）表示；网络负载代理动态的从网络中获取数据。λ_1主要是基于任务图，λ_2，μ主要是基于网络负载信息(关于怎么从网络中获得数据不是本文的内容)。

算法输入后，生成优化后的任务图，该任务图将在下面的运行系统中用到。该算法分为三步，算法描述在[1]中有详细描述。

（2）变量定义

①任务编号：任务图中的任务（消息）编号为 0 到 N-1，其中 N 为任务图中任务总数。

②边：任务 i 和 j 之间的边用有序对（i，j）表示。

③路径：任务图中路径用一系列编表示。如[（i_1，i_5），（i_5，i_8），……，（i_{k-1}，i_k）]是一条从任务 i 开始，到任务 k 结束的路径。

④连接矩阵：任务图表存在矩阵中，该矩阵是对称的并且有如下三个属性，用 TG[i，j]表示。

⑤数据大小：如果（i，j）是任务图中的一条边，则 TG[i，j].DataSize 则表示从节点 i 发送到节点 j 的信息包

数量；否则，TG[i，j].DataSize 为无穷大。

⑥权重：TG[i，j].Weight 则表示边（i，j）负载的通信量。TG[i，i].Weight 则等于节点 i 负载的通信量。

⑦标记：TG[i，j].Flag 则表示边的通信是否被最优化。

结论 1 对在任务图 DAG 路径集 PS 中任意一非关键路径 P_y，而任意一关键路径被定义为 P_x，则 P_x，P_y 满足：

$$\forall \text{path } P_y \in PS,$$
$$\text{PathCompTime}(P_x) \geq \text{PathCompTime}(P_y)$$

其中 PathCompTime(P) 是路径 P 中所有任务和边的权重的和。

则关键路径计算通信完成时间为：

$$\sum_{j=1}^{k-1}(TG[i_j,i_j].\text{Weight} + TG[i_j,i_{j+1}].\text{Weight}) + TG[i_k,i_k]。$$

其中，（i_j，i_{j+1}）是任务图中第 j 条边，在路径 P 中共有 k-1 条边，$TG[i_j,i_j].\text{Weight}$ 是任务 i_j 计算权重和，$TG[i_j,i_{j+1}].\text{Weight}$ 是任务 i_j 通信权重和。

在关键消息算法中，函数 AllPath（TG）返回任务图 TG 中所有路径；函数 CriticalPath（PS）返回关键路径；函数 CompTime（P）返回的是路径 P 中所有任务和边的权重之和。

c. 算法性能分析

现在分析关键消息算法最优化的事件复杂度。性能评测的标准依赖于算法的内部结构，通过分析算法的设计了解算法的内部结构，包括模拟比较次数，迭代次数及 KM 算法中最优化操作次数等等。

（1）变量定义

c：某个任务平均输出的次数。

l：任务图最长路径中任务的总数。

s：最短路径中任务的总数。

a：一般情况下的任务总数。

（2）算法时间复杂度分析

在关键路径算法中，为了得到关键路径要进行 n-1 次比较。在最好的情况下，沿着关键路径有（l-1）×c 次最优化操作，这种情况发生在路径计算通信完成时间很长并且只有一条路径被最优化时；在最坏的情况，（a-1）×c×（n-1）将会有次最优化操作，这种情况发生在路径计算通信完成时间很短并且只有所有路径都要被最优化时。

最坏情况下，整个比较和最优化次数之和为（n-1）+（a-1）×c×（n-1）；最好情况下，整个比较和最优化次数之和为（n-1）+（l-1）×c；则关键消息算法的计算时间复杂度为：

O（Max{（n），（l×c）}），　　最好情况下

O（Max{（n），（a×c×n）}）=O（a×c×n），　最坏情况下

进一步的模型的开销计算和算法代价分析在本文 3.3.3 节中介绍，算法、模型的模拟研究将在下一节中进行，模拟实验方法及数据都由关键消息算法实现。

3.2.3 基于 MPI 的关键消息算法

```
/*   (1)初始化 */
/*   (1.1) 修改任务图 TG 中任务权重 */
for (i=0; i<N; i++) do
     for(j=i+1; j <N; j ++) do
```

```
        if( TG[i, j].DataSize ≠ infinity) then
            TG[i, j].Weight ← 1/( μ − λ 1− λ 2)+ TG[i, j] .DataSize/ μ ;
/* (1.2)初始化路径 */
Uoptp ← AllPath(TG) ;
Optp ← Φ;
/*  (1.3)初始化原始的和优化后的通信时间 */
CP ← CriticalPath(Uoptp);
OriCompTime ← PathCompTime (CP);
OptCompTime ← 0;
/*  (2) 生成优化的关键路径 */
while(TRUE)
    if(PathCompTime (CP)> OptCompTime ) then {
/*  (2.1)优化 CP 中的权重 */
    for ∀(i,j)∈CP  do
     {
      if (~TG(i,j).Flag) then
        {
          TG(i,j).Weight ← 1/( μ − λ 1)+ TG[i, j] .DataSize/ μ ;
          TG(i,j).Flag ← true;
          AllFlag ← true;
          }
        }
/*  (2.2) 修改其他相关边的权重*/
    for ∀(i,j)∈CP  do
    {
        for ∀k, k≠j do
            if ((TG[i, k].DataSize ≠ infinity) ∧(~TG(i,k).Flag))
            then
                TG[i,k].Weight ← μ /(( μ − λ 1) ( μ − λ 1− λ 2))+ TG[i,
                k] .DataSize/ μ ;
/*  (2.3)修改优化的路径集合 */
    UoptP ← UoptP – {TP};
    OptP ← OptP ∪ {CP} ;
/*  (2.4 )修改优化的路径的完成时间 */
    OptCompTime← Max(PathCompTime(CP),OptCompTime) ;
/*  (2.5) 修改 CP*/
    CP  ← CriticalPath (UOptP);
```

```
    }
    else
    break;
}
/* (3) 结论 */
/*  (3.1) 修改所有未被优化的路径的权重（即非关键路径的任务权
    重）*/
if (AllFlag) then
{
    for ∀(i, j) AllPath ∈ (UoptP) do
    {
        for ∀k, k≠j do
        {
            if ((TG[i, k].DataSize ≠ infinity) ∧(~TG(i,k).Flag)) then
                TG[i,k].Weight ← μ /(( μ - λ 1) ( μ - λ 1- λ 2))+ TG[i,
                k] .DataSize/ μ ;
        }
    }
}
/* (3.2) 输出性能参数*/
OutPut (OriCompTime);
OutPut (OptCompTime);
OutPut (OptP);
OutPut (UOtpP);
```

3.3 扩展的 BSP 模型

3.3.1 E-BSP 模型的定义

通过以上对关键消息算法分析，提出定义基于该算法的的 E-BSP 模型如下：

（1） E-BSP 模型是基于现代网络的并行计算模式的 BSP 模型的扩展。

（2） E-BSP 具有所有 CSA-BSP 模型[13]的性质，参数 L，g，h 的含义也和 CSA-BSP 模型相同；E-BSP 程序也是由一系列的超步组成，每个超步都是前面由发送段-通信段组成，后面由接受语句组成；同步不明显出现在超步中。

（3） 与 CSA-BSP 模型最重要的区别，也是该模型最关键的特性是，每个消息段（发送段-通信段）在消息队列中不再是 FIFO 方式处理，而是通过关键消息算法通过通信量不同赋予不同的优先级，用优先级方式处理。

（4） 粗粒度并行，尽量减少通信。由于通信速度相对较慢，过多的通信将大大增加计算代价，这就决定了程序设计时必须尽量减少各节点间的通信，也就是说，最好采用粗粒度的并行算法。通俗地说，就是在分配每个并行任务中时，尽量分得粗一些。

（5） 各个子任务（消息段）保持相对独立。SPMD 的结构和限制通信的要求决定各节点上的程序必须保持一定的独立性和完整性。各节点间的信息交换主要通过消息传递的方式完成，应尽量减少共享变量的使用。

（6） 与并行结构无关。因为网络并行系统是可扩展的，相应的，软件也必须具有良好的可扩展性，在网路中节点增加或减少时，仍然能正常进行。

（7） 参数 Dpct 表示关键路径完成时间（包括计算时间和通信时间）度；Ir 表示改进率；Or 表示最优化率；CCr 表示计算通信比例。

在 E-BSP 模型中，一方面细分程序段为发送段-通信段，增加了

内部/局部通信次数，但是减少了进程之间的同步；另一方面，全局通信要求尽可能粗粒度的并行。这就要求在实际应用中，根据不同的应用需要，选择不同的应用模式。新增的参数用于评价模型的通信、计算性能改善情况。如果 Ir>1，表示采用 E-BSP 模型后程序的通信时间没有得到改善；Ir<1，则表示得到改进；Ir=1，表示没有任何改变。关键消息算法通过 CCr 来确定各消息段的优先级；通过 Or 可知并行任务的粒度粗细；Dpct 用来衡量该 KM 算法的最优化程度，下面的章节中我们将通过实验得到相应数据来验证该模型的性能。

3.3.2 E-BSP 模型的特性

结论 2 按照 E-BSP 模型，E-BSP 采用优先级通信机制程序通信性能一定比 CSA-BSP 模型采用 FIFO 通信机制有所改进，即参数 Ir≤1。

在这里，假设优先级的设定原则是短通信时间的消息段优先级高（实际上各消息段的优先级是通过关键消息算法计算出），给出下面实例，说明优先级通信机制比 FIFO 通信机制性能有所改进。表 3.1 种有五个需进行通信的进程 A、B、C、D 和 E，它们到达等待队列的时间分别是 0、1、2、3 和 4，需要通信的时间分别是 4、3、5、2 和 4，由表 3.1 还可以看出，A、B、C、D 和 E 的完成时间分别是 4、7、12、14 和 18。从每个进程的完成时间减去进程的到达时间，即得到每个进程的通信时间，进而可以算出每个进程的带权的通信时间。

作业情况 / 通信机制	进程名	A	B	C	D	E	平均
	到达时间	0	1	2	3	4	
	通信时间	4	3	5	2	4	
FIFO	完成时间	4	7	12	14	18	
	通信时间	4	6	10	11	14	9
	带权通信时间	1	2	2	5.5	3.5	2.8
优先级	完成时间	4	9	18	6	13	
	通信时间	4	8	16	3	9	8
	带权通信时间	1	2.67	3.1	1.5	2.25	2.1

表 3.1 进程通信时间列表

由上表中的 A 和 B 可以看出，采用优先级通信机制，不管是平均通信时间还是平均带权的通信时间都有明显的改善。尤其是通信时间较短的进程 D，其通信时间从 11 降至 3；而平均带权通信时间也从 5.5 降至 1.5。这就说明优先级通信机制能减少进程的通信等待时间并且可提高系统的吞吐量。从而证明了上述推理 2。

如果说 CSA-BSP 模型满足面向系统的准则，使得系统吞吐量高，处理机缓存利用率好，各类资源能够平衡使用，通信和计算时间最大可能的重叠；则 E-BSP 模型就偏向面向用户的准则，使得关键路径完成时间短，通信完成时间快，最优化率尽量小改进率尽量大，提高用户的响应速度。

3.4 扩展的 BSP 模型的性能分析

3.4.1 E-BSP 模型的性能评价

a. 定义测试参数

前面已经详细讲述关键消息模型及基于该模型的 E-BSP 模型，本

节将着重于该模型的性能评价上。首先，我们根据模型的性质定义以下测试参数；然后编写基于 MPI 环境的测试算法，并进行算法模拟实验；然后分析实验结果。

定义的测试参数以及定义如下：

网络通信因子：　定义为 g，含义同上。

路径完成时间度：　定义为任务图中最大的路径完成时间与最小的路径完成时间之比。

改进率：　定义为原始最大的路径完成时间与最优化后最大的路径完成时间之比。

最优化率：　定义为任务图表中关键路径数与图表中所有路径数之比。

计算-通信率：　定义为任务权重（计算权重）与通信权重之比。

程序总体效率：　定义为评价程序运行效率是否达到最优的目标函数。其成员是启动时间、通信队列等待时间等因素。

b. 测试算法

(1) 关键消息的测试算法

为测试上节中的各参数性能，编写测试程序，见文[1]，实验及模拟数据。其算法如下：

```
Begin
/* Step 1  初始化*/
    Assign a priority to all tasks in the task graph;
    Init_PriQue(Ready _tasks);
    Insert_tasks (unimmediate_predecessors);
    Sort_ priorities (decreeing_order);
/* Step 2  调度  */
    While (priority_queue != empty)
        Obtain_task(prio_queue);
```

```
        Select_idle_proc(task);
        If (! Pri_queue) , then
              Insert_successor(priority_queue);
        End If;
    End While;
End

Begin
/* 计算完成时间路径的度 */
    DegreeTG ← the degree of path completion time;
    If DegreeTG ≤ 2 then
    {
/* 定义路径完成时间的极限 */
        ThresH ← thresdhold-value;
        For (all paths in a task graph)
        {
            if ThresH ≤ PathCompTime(P) then
            {
                /* 优化路径 P */
                EXEC optimization process in the original key
                 message algorithm;
            }
        }
      }
      else
        EXEC the original key message algorithm;
      End;
```

(2)通信代价及启动代价 g 的测试算法

我们对于该模型的测试主要关心的是各个通信函数的时间特性，即时间延迟和带宽及启动代价等。我们的测试装有 linux 和 MPICH2.0 版本的有 8 个节点的小型机上进行的[参考文献一刘的程序] 。

① 动态创建新的进程

```
Begin
/* Step 1 初始化*/
MPI_INIT ();
```

```
/* Step 2 动态创建一个新的进程 */
MPI_COMM_SPAWN();
/* Step 3 Get the parent process */
MPI_COMM_GET_PARENT();
/* Step 4 Create several new processes dynamically */
MPI_COMM_SPAWN_MULTIBLE();
End;
```

② 独立进程之间的通信

```
Begin
/* Step 1 初始化*/
MPI_INIT ();
/* Step 2 打开一个服务端口 */
MPI_OPEN_PORT();
/* Step 3 等待端口连接*/
MPI_COMM_ACCEPT();
/* Step 4 连接进程间的服务*/
MPI_COMM_CONNECT();
/* Step5 连接服务名和端口 ID*/
MPI_PUBLISH_NAME();
/* Step6 连接服务名和端口名 */
MPI_LOOKUP_NAME();
/* Step7 改变进程间的 Socket 通信*/
MPI_COMM_JOIN();
/* Step8 关闭端口连接*/
MPI_CLOSE_PORT();
/* Step9 断开进程与服务器间的连接*/
MPI_COMM_DISCONNECT();
/* Step10 取消服务名和端口 ID 的连接*/
MPI_UNPUBLIC_NAME();
/* Step11 取消服务名和端口名的连接*/
MPI_UNLOOKUP_NAME();
End;
```

c. 编写测试程序

测试程序见附录。

d. 测试结果分析

节点个数和程序性能改进率的关系变化图如下：

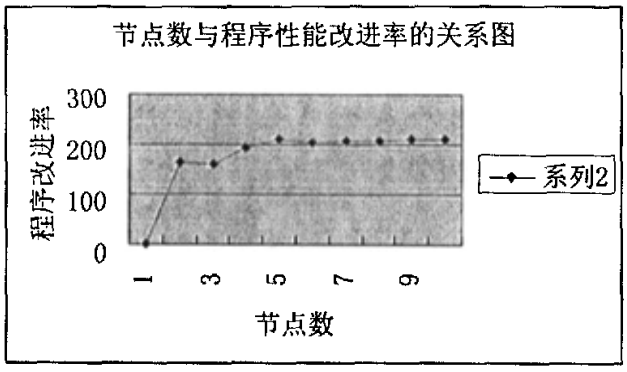

图 3.2 节点个数和程序性能改进率的关系变化图

路径完成时间的度和最优化率的变化关系图如下：

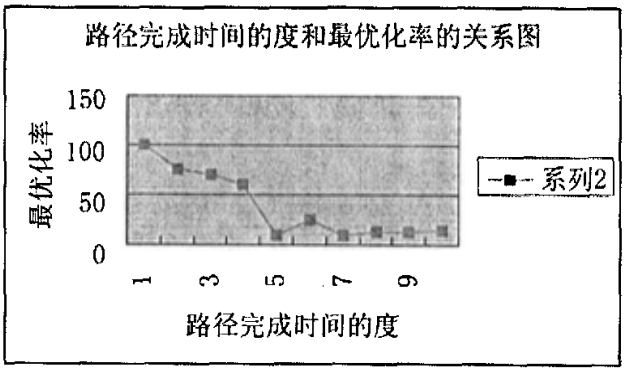

图 3.3 路径完成时间的度和最优化率的变化关系图

计算-通信率和程序性能改进率的关系变化图如下：

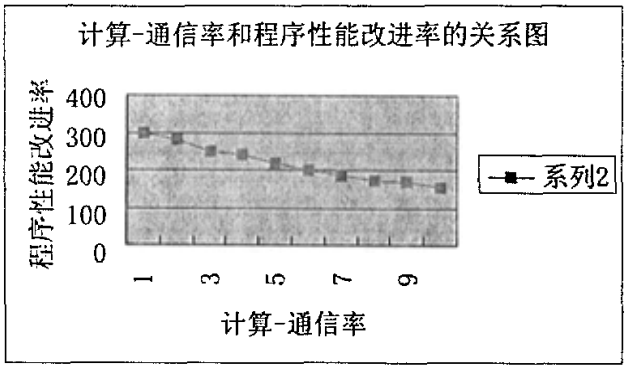

图 3.4 计算-通信率和程序性能改进率的关系变化图

路径完成时间的度和程序性能改进率的关系变化图如下：

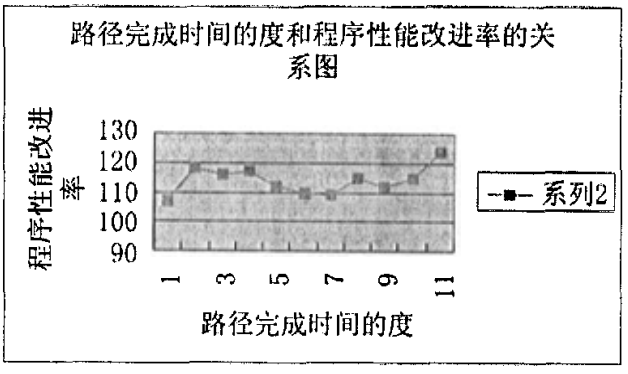

图 3.5 路径完成时间的度和程序性能改进率的关系变化图

网络负载到达率和程序性能改进率的关系变化图如下：

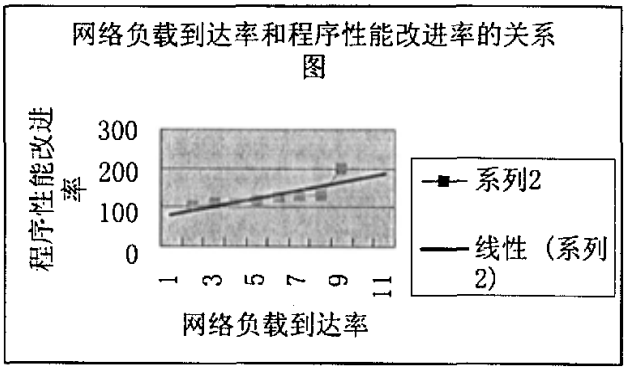

图 3.6 网络负载到达率和程序性能改进率的关系变化图

路径完成时间的度和迭代次数的关系变化图如下：

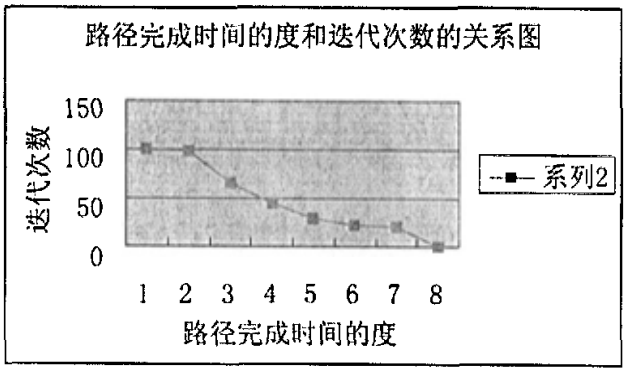

图 3.7 路径完成时间的度和迭代次数的关系变化图

任务节点比率和迭代次数的关系变化图如下：

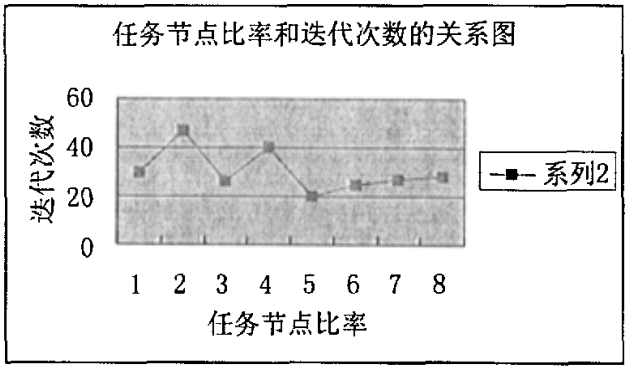

图 3.8 任务节点比率和迭代次数的关系变化图

从以上各图表中的各测试参数间的关系图及相关趋势线不难看出：当节点数越多时，利用关键消息模型改进的通信率越高；最优化率越高时，路径完成时间的度就越低；而通信改进率越高时，路径完成时间的度就越低，计算-通信比率也越低，但网络负载到达率越高；迭代次数增加时，路径完成时间的度越低，而节点中的任务数目与其的关系呈正弦曲线，且迭代次数越多，路径完成时间的度的变化趋于直线。

总的来说，当并行应用中的整个任务完成时间最小且在网络系统高负荷的情况下系统利用率最大的时候，该关键消息算法是高效的。并且，因为可以通过在现有的通信系统的缓冲管理区域中提供优先级机制来建立关键消息模型，因而，关键消息模型在并行应用中具有很强的通用性。

另一方面，由[28]可得，节点个数与系统的启动时间的变化关系如图，由图中不难得出，各个通信函数在通信量较小的时候，曲线基本平直，这说明在这种情况下消息传递所占的时间比例较小，启动时间是主要的消耗时间。随着通信量的增大，曲线斜率增大，这说明启动

时间所占时间比例减小，传送数据的时间开始成为影响时间消耗的主要因素。在图上也可以清楚地看到节点数不同，它们的平均启动时间有很大不同，随着节点数得增加它们的平均启动时间在明显的增长。而且结点数不同它们曲线的增长也不同，结点多的通信时间的增长要明显快于结点少的通信时间的增长。

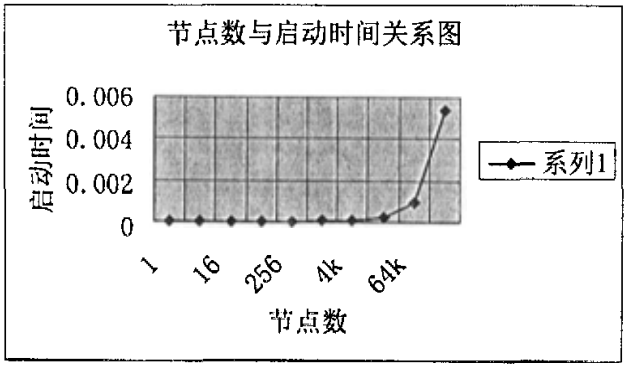

图 3.9 节点个数与系统的启动时间的变化关系图

综上所述，易得如下结论：在利用该模型进行高性能计算时，应使并行应用中的整个任务完成时间尽可能最小且在网络系统高负荷的情况下系统利用率尽可能最大，这样能够通过细分消息、减少通信队列等待时间，尽可能的提高节点间的通信效率；但是另外一方面，随着系统中的节点数不断增加，系统计算能力不断进步的同时，系统得启动代价也随之增加；所以，在实际应用中，用户需根据具体的网络软、硬件条件（节点数、网络条件等），选择相应的最佳方案，获得最优的通信性能。未来的工作可以将该关键消息模型进行详细设计及实现，这样在实际使用时可以给用户提供更为具体的选择方案。下面一节我们将详细叙述该模型在网络并行环境下的适应性。

3.4.2 影响网络通信因子 g 的因素

参数 g 表示通信网络传递数据的能力，它衡量网络连续地向多个地址传递数据的能力，多个地址服从均匀随机分布[16]。处理器上随机过程的产生时间和各种技术如自适应的路由选择等都有助于使 h-关系所产生的通信载荷接近由随机序列产生的通信载荷。这样参数 g 就能恰当衡量网络上的实际载荷了。在参数 g 的定义中，加入一些小的限制条件后，hg 就是对大的体系结构范围内通信性能的精确度量。g 值大小已经按照体系中的时钟频率进行了归一化的处理，以便于指令序列执行时有统一的时间单位。

参数 g 与通信网络的带宽有关，但二者的作用并不相等。g 还取决于其他一些因素如：

1. 通信网络接口所采用的协议；
2. 处理器和通信网络二者的缓存管理；
3. 通信网络采用何种路由选择策略；
4. E-BSP 运行系统。

参数 g 也考虑到了通信的启动代价[28]。BSP 模型中一个超步中所有的通信，被视为在超步的最后发生；在 CSA-BSP 模型中，一个超步被分为一系列消息-发送段，计算和通信有重叠；在 E-BSP 模型中，消息-发送段被赋予不同的优先级，通信性能进一步优化。这样的话，使得我们不用等到系统中每个超步的计算部分结束后才开始进行必要的通信行为，在每个进程完成所需 C-S 段的计算后就可进行通信行为，则通信的启动代价即为每个 C-S 段启动的代价之和。

参数 g 还受到消息大小的影响。因为 BSP 模型不区分一个进程发

送大小为 h 的消息和发送 h 个大小为 1 的消息的代价，两者的通信代价均为 hg。但是由于消息传递的启动代价的影响，通信总量非常小的超步仍然可能与 E-BSP 模型指出的代价不一致。所以，我们在计算 g 属性时，要综合考虑到启动代价和消息大小。

[4]中提出了为改进标准代价模型，将消息粒度对通信代价的影响纳入了新的模型。在改进的模型中，参数 g 定义为消息尺度 x 的函数：

$$g(x) \approx \{n_{1/2}/x + 1\}g_\infty$$

其中 g_∞是指极大消息尺度下通信代价的渐近值；$n_{1/2}$指占有机器最大带宽的 1/2 的消息尺度，故 $g(n_{1/2}) \approx 2g_\infty$。则有以下推理成立：

结论 3 在并行程序中，要通信的消息长度不能太小，以避免 g 的急剧上升，通信代价太大；这样就要求我们在分 CS 段时要综合考虑启动代价和通信代价。

结论 4 对于一个并行程序，当问题规模不变时，程序的性能不会随着系统规模的增加而增加，其效率总会有一个最优值。

结论 5 当并行任务中的通信和计算重叠度越大时（即把消息分为计算-发送段，且允许只要所在 CS 段完成计算就可进行通信），则系统的启动代价越大。

3.4.3 栅栏同步参数 l 的代价分析

对于每一种并行结构，参数 l 反映了栅栏同步的代价。该代价分为以下两个方面：

1. 各个运算步骤分担整个运算任务，由于各步的完成时间不同而引入代价；

2. 在所有的处理器间达到一种全局组合状态所付出的代价。

当然，这不仅取决于通信网络的性能，而且也有赖于是否采用了满足特殊要求的硬件以实现同步，以及每个处理器中断的方式。

栅栏同步对于今天的并行结构来说，常常是昂贵的。因而我们尽可能的减少采用栅栏同步。而另外一个方面，未来并行结构的发展将使之变得更加便宜。栅栏同步是并行程序设计的一个重要工具，它减少了程序的状态空间。通常情况下，有共享内存的并行机中的栅栏同步代价会比较小，而分布式和网络系统的代价会比较大，因为后两者不提供硬件的低级访问。但是我们可以利用被传输消息的重组来有效地减少同步代价。

E-BSP 结构化模型相对非结构化模型，有一个显著的优点，就是，E-BSP 程序中的同步都发生在局部或全局通信和局部计算过程之间；而 BSP 程序中的同步是发生在全局通信和局部计算过程之间。这使得我们可以很容易的利用局部或全局通信将栅栏同步进行整合，从而减少两者的代价。具体整合方法如下：

1. 执行一个全局信息交换，信息包括：信息的数量、大小和目的节点。然后，将消息标注，发送给远程节点。全局信息交换，将预传输的消息直接拷贝到远程节点的内存中，并在超步中起栅栏同步的作用。
2. 保证每个节点完成自己的数据输入输出。这是通过消息计数来完成的，而且可以将输入输出交叉进行，以节省缓冲区空间。

可以看出，这种结构中，栅栏同步的代价大大降低，因为全局通信与之同时进行。即通信与同步重叠。则有以下推理成立：

结论 6 为了尽量减少栅栏同步的代价，在设计并行算法的时候，

应当尽可能地均匀分配计算任务，以减少节点的等待时间；

结论 7 尽可能的减少通信的次数，即减少同步的次数，因为一次通信，就对应一次同步，通信的数据在同步后才能生效。

3.5 本章总结

本章主要讲述在网络环境下的 E-BSP 模型的定义组成及其性能和应用，在给出精确定义和进行详细性能分析后，我们可以得出下面的结论：采用该模型进行并行程序设计时，通信和计算时间重叠度增加，且关键路径完成时间、通信完成时间短，从而使得最优化率、改进率增大，提高用户的响应速度。当处理器个数增问题规模增大时，加速比也随之增大，且近似为线性增长，算法效率也趋于 1，与[22]中的加速比定律的理论分析完全符合。所以说，扩展后的 BSP 模型不仅适用于网络并行计算环境，易于实现，而且提高了程序设计的效率及用户的响应速度。

4 扩展的 BSP 模型在网络环境下可扩展性初步分析

4.1 并行系统可扩展性基础知识

4.1.1 可扩展性性定义

可扩展性[17]是设计高性能并行机和并行算法所追求的另一个重要目标。在设计并行算法和高性能计算机时，人们注意其计算性能是否随计算机台数增加而增加，即可扩展性如何，只有具有适应性的并行机和并行模型才有生命力。由于并行系统的性能和系统规模、问题规模、算法内在并行性以及由于通信、同步和进程创建等引起的系统开销等因素有关，它们对并行系统的结构和应用程序的可扩展性都有影响，因此本节将讨论并行计算模型在网络环境下的可扩展性/适应性。试图找出一种合适的度量方法，来评价系统规模和问题规模的变化对并行机和并行算法（如关键消息算法）性能的影响。

定义 1 如果能加以扩展（即增加其资源）以满足不断增长的对性能和功能的要求，或是能够缩减（即减少其资源）以降低成本，则称包括硬件和软件资源的计算机系统是可扩展的。

定义 2 一个并行系统是可扩展的，若它具有定义 1 中的特性，即如果随着并行系统的规模增大，通过适当增加问题规模，使并行系统的功能与其规模成线性比例增长，则这个并行系统是可扩的，适应性很好。

由上述定义可得：如果一个并行系统的规模无法增大，或是随着并行系统的规模增大，无论怎样增加问题的规模，都不能使并行系统

的性能与其规模成线性比例关系，则这个系统是不可扩的。如当前流行的 BT 下载工具[19]，就是有扩展性的应用工具。

一个系统的可扩展性主要是指以下三个方面：资源可扩展性，应用可扩展性、技术可扩展性。资源可扩展性是指通过增加机器规模（即处理器数）、投入更多存储部件（高速缓存、主存、磁盘）以及增加软件等方法，使系统具有更高性能或更多功能。应用可扩展性是指应用程序也必须是可扩展的。这就是说，相同程序在一个可扩展系统上运行时，其性能随着规模扩大成比例改进。技术可展性是指系统能否适应技术的改变。

4.1.2 并行系统可扩展性及其基本特征

并行系统的可扩展性指标[17]有以下三个内容：

1. 可扩展性的基本指标包括系统规模和问题规模；
2. 可扩展性的硬件指标主要包括互联网性能、I/O 需求、存储器需求、时钟速率和计算机价格；
3. 可扩展性的软件指标主要包括操作系统、并行环境、程序设计开销、系统开销以及运行时间等等。见图 4-1：

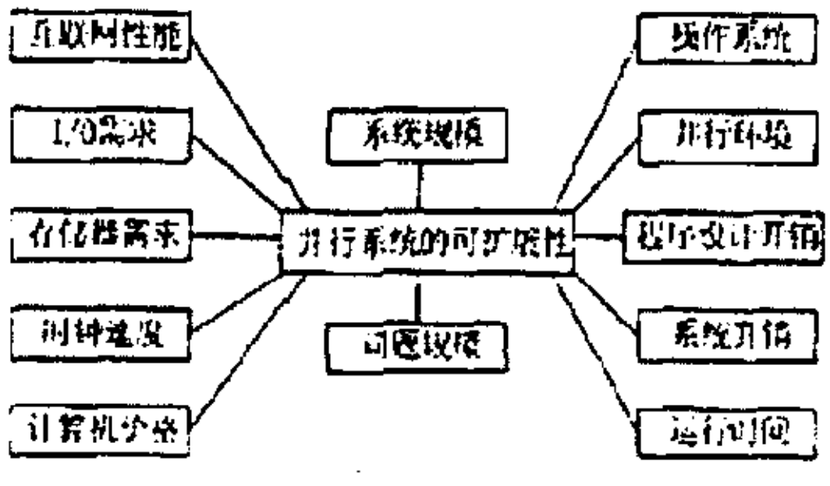

图 4.1 并行系统可扩展性指标

图中各指标意义如下：

1. 系统规模　并行系统所用的处理器数目，系统规模增大是指并行系统资源增多和计算能力增强；
2. 问题规模　即在串行计算机上求解问题的最好算法所需的计算总量；
3. 互联网性能　网络在增加机器资源时应具有模块化可扩展的能力，即要求有高速率的数据传输、低延时和宽通信频带等能力；
4. I/O 需求　传送与应用程序运行有关的程序、数据和相应结果的输入/输出要求，即提供并行 I/O 操作以及与 CPU 操作重叠；
5. 存储容量　程序运行时使用的主存储器的最大容量，它应随着系统规模增加而增大；
6. 时钟频率　决定机器的基本周期。可扩展性要求其部件能随着技术的更新而不断提高它的时钟频率；
7. 计算机价格 实现执行程序所需的硬件和软件资源的总价格；
8. 操作系统　应扩展为支持并行操作系统和能管理并行工作的资源；
9. 并行环境　为在不同粒度级别上用智能更强的编译器来描述并行性或便于监测并行性，就必须扩展语言或开发新的语言结构，以使用高层并行的概念或结构来描述并行性；
10. 程序设计开销　与应用程序有关的研制开销；
11. 运行时间　指串行、并行的运行时间；
12. 系统开销　由并行处理引起的所有处理器额外开销的总和时间，它包括同步、通信、进程创建和算法的串行部分引起的空

闲时间总和。

4.1.3 常用的可扩展性度量方法

定义 3 所谓等效率是指系统规模增加时，测量增加多少运算量会保持效率不变。

如果问题规模 W 保持不变，只增加系统的规模 P，将会引起并行开销的增大，使效率降低，而当系统规模 P 不变时，将会引起并行开销的增大，使效率降低，而当系统规模 P 不变时，只增加问题规模 W，由于并行开销增大得比 W 慢，从而效率增加。因此，我们可以通过同时增大 W 和 P，使效率保持不变。此时有：

$$E_p = \frac{t_c W}{PT_p} \frac{t_c W}{t_c W + T_0(P,W)} = \frac{1}{1 + T_0(P,W)/t_c W}$$

式中，T_p-算法的并行计算时间；

$T_0(P,W)$-并行计算系统的总开销；

t_c-并行机上每个基本操作的平均时间；

$t_c W$-串行运行时间。

由上式可得到，

$$W = \frac{1}{t_c}\left(\frac{E_p}{1-E_p}\right)T_0(P,W) = K \times T_0(P,W)$$

式中 $K = \frac{1}{t_c}\left(\frac{E_p}{1-E_p}\right)$ 则由第二个公式可得：

（1） E_p 为常数时，第二个公式可表示为 W=f（P），此时该公式成为等效率函数。f（P）则表示保持效率不

变时所需运算量；

（2） W=f（P）也反映了效率不变时，问题规模与系统规模之间匹配关系；

（3） 固定问题规模时，由第一个公式看出，效率随系统规模P增大而降低，由于系统开销随P增大而增加，处理器的利用率因此而降低；

（4） 对于一定规模的并行系统，效率随问题规模W增加而提高，所以随系统规模的增加而适当地增加问题规模，就可保持效率不变。

然而，有实验表明[22][25]，为了维持效率不变，随着系统规模的增加，问题规模往往以多项式甚至指数规律增长，这可能超过并行机有限存储容量，使得有限 I/O 能力成为瓶颈。因此，等效率函数的度量方法的缺陷是：没有反映出系统的规模增加时，体系结构的计算和通信能力的变化对算法内在并行性与通信需求的影响，也没有提供不可扩系统的不可扩信息。

定义 4 平均单位速度定义为：$V_p=W/PT_p$ 。

式中 T_p-使用 P 台处理器时的处理时间。假设 W’ 是使用 P’ 台处理器时维持平均速度不变时所增加的问题规模。平均速度不变，意味着

$\frac{W}{PT_p}=\frac{W'}{P'T_p}$ 这时系统规模由 P 变到 P’，并行算法-机器组合的可扩展性定义为：

$Scal(P, P')=\frac{W/P}{W'/P'}=\frac{T_p}{T_p'}$ 该式说明：系统规模由 P 变化到

P′ 时，在保持平均速度不变时，每个处理器平均增长浮点数操作数的多少。

等速度度量方法的优点是：（1）包含了问题规模和执行时间两个重要因素，问题规模描述了应用程序的一些性质，执行时间反映了机器性能和程序效率；（2）速度对不同机器来说是公平的，执行时间包括计算时间和开销时间两部分；（3）由于问题规模出计算中的浮点操作数决定，所以容易估计。但是在以下几方面有所不足：（1）由于用浮点操作数来估计问题规模，忽略了非浮点操作所引起的性能变化，因此，这种度量方法不能清楚地估计出机器的影响和程序的通信延时；（2）开销包含在整个执行时间中，而在总的浮点运算 W 却没有包含；（3）没有反映出并行算法与体系结构的匹配程度。

4.2 E-BSP 模型在网络系统下的可扩展性分析

本文中的并行环境是现代网络，所以我们使用等计算时间/通信开销比率的概念，用来度量并行算法/体系结构组合可扩展性。

由于一个应用问题内在的计算时间/通信开销比率决定了这个应用问题在任何体系结构上的性能。该比率越高，说明每个处理器的使用率越高，但它常常会随系统规模增加而下降，故可通过适当增加问题规模来提高这个比率。因此，随着系统规模的增加，适当增加问题规模来提高这个比率，可使得计算时间/通信开销比率保持常量。

定义 5 假设 W 为使用 P 台处理器的问题规模，W′ 为使用 P′ 台处理器时维持计算时间/通信开销比率不变时增加的问题规模，则并行算法-体系结构组合的可扩展性定义为：

$Scal(P, P') = \frac{\overline{V_p}}{\overline{V_{p'}}}$　　式中 $\overline{V_p}, \overline{V_{p'}}$ - 分别表示系统规模为 P 和 P' 时并行系统得平均速度。

计算时间/通信开销比率是指并行计算时间与系统开销的比率，当系统规模增加时，通过保持计算时间/通信开销比率不变来获得所需要增加的问题规模，然后利用可扩展性公式评价并行系统的可扩展性。

由式 $T_0=PT_p-T_1$ 得到 $Tp=(T_0+T_1)/P$，若计 $T_{comp}=T_1/P$，$T_{00}=T_0/P$，则定义 5 可以推导出以下式子：

$$Scal(P, P') = \frac{W}{W'} \times \frac{T_{p'}}{T_p} \times \frac{P'}{P}$$

$$Scal(P, P') = \frac{W}{W'} \times \frac{T_{comp'}}{T_{comp}} \times \frac{P'}{P}$$

$$Scal(P, P') = \frac{W}{W'} \times \frac{T'_{00}}{T_{00}} \times \frac{P'}{P}$$

上述三式均定量给出了并行计算机规模和问题规模影响系统性能的信息，同时也反映了体系结构与应用问题算法之间的匹配程度，此外，第二个公式可用来预测并行算法的可扩展性，第三个式子可用于预测体系结构的可扩展性。

在这种并行算法-体系结构组合可扩展性定义下，可推导出以下结论：

结论 1　在一个并行体系结构上，如果随着系统规模 P（P>1）的增加，一个算法所获得的加速比 S_p 与系统规模 P 有如下线性关系：存在一个与 P 无关的常数 c（0<c<P），使得 $S_p=P-c$，则称这个并行计算-

体系结构组合是可扩展的，且可扩展为：

Scal（P，P’）= P’（P-c)/P(P’ - c)

结论 2 在一个规模为 P 的并行系统上，如果一个并行算法-体系结构组合是可扩展的，则当系统规模由 P 增加至 P’（P> P’）时，加速比 S_p 和 S_p’ 有如下关系：

$(S_p' - S_p)/(P' - P)=1$

结论 3 在一个规模为 P 的并行系统上，若一个并行算法-体系结构组合是可扩展的，则存在一个与 P 无关的正常数 c，使得 $T_p=\frac{cT_1}{cP-1}$

结论 4 在一个规模为 P 的并行系统上，t_c是体系结构上每个基本运算的平均运行时间（通常称为常数），W 是问题的规模，如果一个并行算法-体系结构组合是可扩展的，则存在一个与 P 无关的正常数 α，使得并行运行时间 T_p和理想并行运行时间 Tp^*（串行运行时间与系统规模的比率）之差为：$T_p-T_p^*=\frac{t_c W}{P(\partial P-1)}$。该式给出了并行系统运行时间与理想并行运行时间的差距；

结论 5 在推理 4 的前提下，则存在一个与 P 无关的正常数 α，使得 $\frac{T_p}{T_p - T^*_p}=\alpha P$。该式反映了并行运行时间与其所含的通信时间的比率仅与系统规模有关。

因为可扩展性已广泛用来描述系统规模和问题规模将如何影响并行机和并行算法的性能，它不但能够度量并行机体系结构在不同规模下的并行处理能力和并行算法内在的并行性，而且能够利用系统规模和问题规模已知的并行系统得性能来预测系统规模和问题规模增

大后的系统的性能。目前它已成为并行计算研究的一个热点课题，人们正在从不同角度对其进行深入研究。

对照以上关于可扩展性的说明，考察网络环境下并行计算机的体系结构，显然网络环境下并行计算机系统采用商品化的计算机组件，而且在通信上基本采用成熟的局域网技术，因此，其硬件系统可以用较少的代价实现更新，其节点的数目可以从几个到几十个乃至几百个不等，而且可以方便的增加或减少节点的数目，也就是说网络环境下的并行计算机系统不难实现上面所说的资源可扩展性和技术可扩展性。关于应用程序的可扩展性，是与具体系统的组成，特别是系统硬件资源的配置是否均衡有很大关系。通信网络的性能对系统整体性能有很大的影响。

另外，在网络上开发的软件具有很好的可移植性。现有的操作系统一般是以自由软件 linux 作为操作系统的，并行程序可以在标准的并行支持软件 MPI、BSPlib 之上开发，具有很好的通用性。不像以前的大型并行计算机那样，需要编程人员掌握针对特定系统的并行编程技术，而且开发出的软件难以移植到其它系统上。这一切，都使网络并行系统得到了越来越广泛的应用。

4.3 本章总结

通过上面的分析易得，基于该扩展 BSP 模型进行的并行程序设计，并行系统运行时间与其所含的通信时间的比率仅与系统规模有关，也就是说，该模型在网络并行环境下扩展性很好。

5 结束语

5.1 论文总结

由前面的分析可知，尽管 BSP 模型指导下的并行程序设计具有编程简单、独立于目标结构和执行性能可预测性等特点，且 BSP 程序由于其超步的结构使消息传递操作不会出现死锁，同时在正确性、性能分析和编程方法上都简单易行，很适合异构环境下的消息传递编程。但仍存在 g-relation、h 参数预测不精确、在网络环境下不适应等问题。所以本文研究了网络并行计算环境下的扩展的 BSP 模型（Extensived-BSP），基于扩展后的 BSP 模型进行程序设计的程序性能更好，用户的响应速度更快，并且给出了该模型的可扩展性详细的度量方法。

从对测试结果分析，可以得出如下结论：当节点数越多时，利用关键消息模型改进的通信率越高；最优化率越高时，路径完成时间的度就越低；而通信改进率越高时，路径完成时间的度就越低，计算-通信比率也越低，但网络负载到达率越高；迭代次数增加时，路径完成时间的度越低，而节点中的任务数目与其的关系呈正弦曲线，且迭代次数越多，路径完成时间的度的变化趋于直线。也就是说，当并行应用中的整个任务完成时间最小且在网络系统高负荷的情况下系统利用率最大的时候，该关键消息算法是高效的。并且，因为可以通过在现有的通信系统的缓冲管理区域中提供优先级机制来建立关键消息模型，因而，关键消息模型在并行应用中具有很强的通用性。

另一方面，由[28]可得，节点个数与系统的启动时间的变化关系如

图，由图中不难得出，各个通信函数在通信量较小的时候，曲线基本平直，这说明在这种情况下消息传递所占的时间比例较小，启动时间是主要的消耗时间。随着通信量的增大，曲线斜率增大，这说明启动时间所占时间比例减小，传送数据的时间开始成为影响时间消耗的主要因素。在图上也可以清楚地看到节点数不同，它们的平均启动时间有很大不同，随着节点数得增加它们的平均启动时间在明显的增长。而且结点数不同它们曲线的增长也不同，结点多的通信时间的增长要明显快于结点少的通信时间的增长。

综上所述，扩展后的 BSP 模型不仅适用于网络并行计算环境，易于实现，而且解决了并行应用中的通信拥挤问题，提高了程序设计的效率及用户的响应速度。

5.2 尚待研究问题

但未来的工作将会包括：将该关键消息模型进行详细设计及实现，这样可以给用户提供更为具体的选择方案，根据所测得的性能来改进和优化 KM 算法中的各个函数功能和性能特征，使得关键消息算法可以最大程度的改进消息的通信时间；另外，在本文一些细节方面，如关于消息在消息队列中的处理，也可以采用其他的方法，如可以采用时间片轮转法、短通信优先法等替代优先级算法；并可以在其它的并行环境（如 PVM）的通信函数进行测试，然后对它们作一些比较性研究；最后，对于网络并行环境下的该模型的存储性能方面的研究也是很必要的。

致谢

在本篇论文完成之际，我在北京交通大学的研究生生活也即将结束。在此，我要向那些对我完成此论文有过帮助的人们表示由衷的感谢。

两年半的研究生学习生活使我受益匪浅，我所取得的每一个成绩中都凝结着我的导师--于老师的辛勤汗水。感谢于老师长期以来工作和生活上对我的关心和指导，是她为我的课题指明了方向，给我提供了良好的研究和实践环境，使得我能掌握国内外的发展动态和获取最新的技术资料，并在实践中得到不断的提高。于老师缜密的思维，严肃认真的科学态度使我受益终身，严谨的治学作风更使我在具体实现阶段对自己严格要求，毫不松懈。值此论文完成之际，谨向她致以最诚挚的感谢和崇高的敬意。

特别感谢学院其他老师在毕业设计期间和论文最终定稿上的谆谆教诲，他们的渊博学识、严谨求实的治学态度、丰富的学术研究经验使我受益匪浅。

一直以来我的家人都给予了我持续地关心和支持，没有他们就没有今天的我，借此机会向他们表达最衷心的谢意。

最后向所有其他关心和帮助过我的同学、朋友致以衷心的感谢。

吴春燕

二〇〇五年三月

参考文献

[1] S.Parkin, V.Karamcheti and A.Chein, " Fast-message(FM): Efficient, Portable Communica tion for Workstation Clusters anf Massively-Parallel Processors" [J]. IEEE parallel processing, Microprocessor Operating Systems, April -june, 1999, pp60-73.

[2] Wang, Han, Wang, Hao, Shen, Jinmei, Performance modeling and practical parallel algorithms of cluster computing with applications on distributed platforms. IEEE Parallel co mputing ,2003.1, 208-214.

[3] Fournier, Aime, Taylor, Mark A., Tribbia, Joseph J., The Spectral Element Atmosphere Model (SEAM): High-resolution parallel computation and localized resolution of regional dynamics[J]. IEEE parallel compute environment 132(3), 2004.3, 726-748.

[4] Goldman, Alfrede, A model for parallel job scheduling on dynamical computer Grids, Concurrency Computation Practice and Experience[J]. v 16, n 5, Apr 25, 2004, Middleware for Grip Computing, p 461-468.

[5] Lan Foster, Designing and Building Parallel Programs: concepts and Tools for Parallel software Engeering[J]. Addison-Wesley, New York, 1998.

[6] P.Chung, Y.Hunag, S Yajnik, D. liang, J.Shih, C.Wang, Y.M. Wang, " [J]. DCOM and CORBA side by side, step by step, and layer by layer", C++report, vol.10, no.1, January 1998, pp.18-29, 40.

[7] Geist, G. A, J. A. Kohl, P. M. Papadopoulos, " PVM and mpi:A Comparison of features " [J]. Calculateurs Paralleles, 8(2), pp. 137-150, june, 1999.

[8] M. Y. Wu AND D. Gajski. Hypertool:A Programming Aid for Messege-passing Systems[J]. IEEE Transactions on Parallel and Distributed Systems, 1(3):330-343, 1999.

[9] 刘方爱，高效并行计算系统中的计算模型与通信网络[D]. 中国科学院计算技术研究所博士学位论文，2001.4.

[10]田媛，集群系统上基于 BSP 并行计算模型的算法设计[D]. 西安交通大学硕士学位论文，2002.4.

[11] 朱光明，微机网络并行计算的研究和应用[D]. 中国科学院计算技术研究所硕士学位论文，2001.4.

[12]余华山，支持 SMP 集群的并行技术[D]. 北京大学博士学位论文，2001.5.

[13]岑明，基于 CORBA 的分布式控制系统及其集成研究[D]. 四川工业学院硕士学位论文，2001.4.

[14]孙再强，基于 CORBA 的网络并行计算环境的设计与实现[D]. 西安电子科技大学硕士学位论文，2001.4.

[15]胡凯，胡建平，王强. 分布式并行计算网络体系结构研究[J]，小型微型计算机系统. 2000.2：113-115.

[16]任晓明，杨大鉴，刘国权. 网络并行计算系统模型[J]，计算机工程与应用. 2001.15：118-120，156.

[17]宋安军，彭勤科，胡保生. 并行计算模型在集群环境下的适应性[J]计算机工程，2003.10，29（18）：4-6.

[18]陈鑫达，黄伟民，并行计算模型在异构环境中的研究[J]. 计算机应用, 2002，11：1-3，44.

[19]陈国良，并行计算：结构、算法、编程[M]. 高等教育出版社，1999.

[20]陈国良，并行算法的设计与分析（修订版）[M]，高等教育出版社，2002.

[21]黄铠，徐志伟，可扩展并行计算--技术、结构与编程[J]，机械工业出版社，2000.

[22]黄伟民，陆鑫达，异构环境 HBSP 模型及其在 FFT 算法中的应用[J]，上海交通大学学报，2000，34（6）：796-799.

[23]李晓梅，莫则尧，胡庆丰，罗晓广，曾泳泓，迟利华，可扩展并行算法的设计与分析[J]，国防工业出版社，2000.

[24]刘方爱，刘志勇，乔香珍，一种异步 BSP 模型及其程序优化技术[J]，计算机学报，2002，25（4）：373-380.

[25]彭勤科，许宏斌，谭煜东，胡保生，集群计算机上基于 BSP 模型的并行算法及其程序设计[J]，微电子学与计算机，2002，03：1-4.

[26]曙光信息产业有限公司，曙光 TC1700 超级服务器用户手册 V1.0[N]. 2001.

[27]王鼎兴，郑纬民，沈美明. 并行集群的若干关键技术[J]. 清华大学学报（自然科学版），1998，38(51)：15-22.

[28]刘纯熙，MPI 并行环境下的性能评测. 北京交通大学学士论文[D]，2004.7.

附录

启动代价测试程序代码

PINGPONG

```
#include “mpi.h”
#include “stdio.h”
#define max 1000000
Main(int argc, char ** argv)
{
    int rank, size, n;                              /* n 用来指示消息的大小 */
    Char a[max], b[max];          /* 定义的接收缓冲区和发送缓冲区数组 */
    Double t1, t2, t, temp;
    MPI_Stuatus status;
    MPI_Init(&argc, &argv);                         /*并行编程的初始化*/
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);           /* 获取进程编号 */
    MPI_Comm_size(MPI_COMM_WORLD, &size);           /*获取进程总数*/
If(rank==0)
    {
        temp=MPI_Wtime();                           /*获取墙上时间*/
        t1=MPI_Wtime();
        MPI_Send(a, n, MPI_CHAR, 1, 100, MPI_COMM_WORLD);  /*发送消息*/
        MPI_Recv(a, max, MPI_CHAR, 1, 100, MPI_COMM_WORLD, &status);
                                                    /*接收消息*/
```

```
        t2=MPI_Wtime();
        t=t2+temp-2*t1;
        printf("the elapse time is:%f",t/2);
                                        /*得到所要测得的时间*/
    }
Else if(rank==1)
        {
          MPI_Recv(b,max,MPI_CHAR,0,100,MPI_COMM_WORLD,&status);
          MPI_Send(b,n,MPI_CHAR,0,100,MPI_COMM_WORLD);
        }
    MPI_Finalize();                                    /*并行程序结束*/
  }
```

MPI_Bcast()的测量

```
#include "mpi.h"
#include "stdio.h"
#define max 1000000
Main(int argc,char **argv)
    {
        int rank,size,n,i
        Char a[max],b[max];
        Double t1,t2;
        MPI_Init(&argv,&argv);
```

```
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    MPI_Comm_size(MPI_COMM_WORLD,&size);
    For(i=0;i<100;i++)
  MPI_Barrier(MPI_COMM_WORLD);
  t1=MPI_Wtime();
 for(i=0;i<100;i++)
        MPI_Bcast(a,n,MPI_CHAR,0,MPI_COMM_WORLD);
      t2=MPI_Wtime();
      printf("the elapse time is:%f\n",(t2-t1)/100);
      MPI_Finalize();
  }
```

MPI_Reduce()的测量

```
#include "mpi.h"
#include "stdio.h"
#define max 1000000
    Main(int argc,char **argv)
      {
          int rank,size,n,i
          Char a[max],b[max];
          Double t1,t2;
          MPI_Init(&argv,&argv);
          MPI_Comm_rank(MPI_COMM_WORLD,&rank);
          MPI_Comm_size(MPI_COMM_WORLD,&size);
          for(i=0;i<100;i++)
```

```
        MPI_Barrier(MPI_COMM_WORLD);
t1=MPI_Wtime();
for(i=0;i<100;i++)
        MPI_Reduce(a, b, n, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
t2=MPI_Wtime();
printf( "the elapse time is:%f\n" ,(t2-t1)/100);
        MPI_Finalize();
    }
```

MPI_Gather()的测量

```
#include  "mpi.h"
#include  "stdio.h"
#define max 1000000
Main(int argc,char **argv)
{
        int rank,size,n,i
        Char a[max],b[max];
        Double t1,t2;
        MPI_Init(&argv,&argv);
        MPI_Comm_rank(MPI_COMM_WORLD,&rank);
        MPI_Comm_size(MPI_COMM_WORLD,&size);
        for(i=0;i<100;i++)
                MPI_Barrier(MPI_COMM_WORLD);
t1=MPI_Wtime();
for(i=0;i<100;i++)
```

```
                MPI_Gather(a, n, MPI_CHAR, b, n, MPI_CHAR, 0, MPI_COMM_WORL
                        D);
    t2=MPI_Wtime();
    printf( "the elapse time is:%f\n" , (t2-t1)/100);

    MPI_Finalize();
        }
```

MPI_Alltotall()的测量

```
#include "mpi.h"
#include "stdio.h"
#define max 1000000
Main(int argc,char **argv)
    {     int rank,size,n,i
          Char a[max],b[max];
          Double t1,t2;
      MPI_Init(&argv,&argv);
      MPI_Comm_rank(MPI_COMM_WORLD,&rank);
      MPI_Comm_size(MPI_COMM_WORLD,&size);
      For(i=0;i<100;i++)
                MPI_Barrier(MPI_COMM_WORLD);
    t1=MPI_Wtime();
    for(i=0;i<100;i++)
              MPI_Alltotall(a, n, MPI_CHAR, b, n, MPI_CHAR, MPI_COMM_W
              ORLD);
```

```
        t2=MPI_Wtime();
        printf("the elapse time is:%f\n",(t2-t1)/100);
        MPI_Finalize();
    }
```

MPI_Barrier()的测量

```
#include "mpi.h"
#include "stdio.h"
#define max 1000000
Main(int argc,char **argv)
    {
        int rank,size,n,i
        Char a[max],b[max];
        Double t1,t2;
        MPI_Init(&argv,&argv);
        MPI_Comm_rank(MPI_COMM_WORLD,&rank);
        MPI_Comm_size(MPI_COMM_WORLD,&size);
        For(i=0;i<100;i++)
            MPI_Barrier(MPI_COMM_WORLD);
      t1=MPI_Wtime();
      for(i=0;i<100;i++)
            MPI_Barriert(MPI_COMM_WORLD);
    t2=MPI_Wtime();
    printf("the elapse time is:%f\n",(t2-t1)/100);
```

```
    MPI_Finalize();

}
```