基于 SMP 集群系统的并行编程模式研究与分析

宋 伟[1,2],宋 玉[1]

(1. 郑州大学 信息工程学院,河南 郑州 450052;
2. 郑州大学 高性能计算研究与发展中心,河南 郑州 450001)

摘 要:并行计算技术是计算机技术发展的重要方向之一,SMP 与集群是当前主流的并行体系结构。当前并行程序设计方法主要采用基于消息传递模型的 MPI 和基于共享存储模型的 OpenMP,两种编程模式各有特点和适用范围。对 SMP 集群以及 MPI 和 OpenMP 的特点进行了分析,介绍了在 SMP 集群系统中利用 MPI 和 OpenMP 混合编程的可行性方法。

关键词:并行计算;对称多处理器;集群;消息传递模型;共享存储模型;MPI;OpenMP;混合编程
中图分类号:TP311 **文献标识码**:A **文章编号**:1673-629X(2007)02-0164-04

Research and Analysis of Parallel Programming Model Based on SMP Cluster System

SONG Wei[1,2], SONG Yu[1]

(1. School of Information Engineering, Zhengzhou University, Zhengzhou 450052, China;
2. High Performance Computing Center of Zhengzhou University, Zhengzhou 450001, China)

Abstract: Parallel computing is one of the most important techniques of computing. SMP and cluster are mainly parallel architecture currently. MPI (which is based on the message-passing model) and OpenMP (which is based on the shared-memory model) are become mainly methods for parallel program design. Discussed and analyzed the characteristic of SMP, cluster, MPI and OpenMP, then introduced the feasible methods of parallel programming with hybrid MPI/OpenMP.

Key words: parallel computing; symmetrical multi processor; cluster; message-passing model; shared-memory model; MPI; OpenMP; hybrid programming

0 引 言

并行计算是提高计算机系统计算速度和处理能力的一种有效手段。当前,并行计算已经成为解决重大问题的关键,它的基本思想是用多个处理器来协同解决同一问题,即将被求解的问题分解成若干个部分,各部分均由一个独立的处理机来并行计算。与之相关的高性能计算 HPC(High Performance Computing)科学已经成为继理论科学和实验科学之后人类进行各种科学研究的第三大支柱。与此同时,集群系统以其良好的可扩展性和性能价格比,已迅速成为高性能计算领域的主流体系结构。随着 SMP(对称多处理器)系统和集群技术的发展,并行处理在科学研究、工程计算以及商业计算等领域得到了广泛的应用。并行程序设计是并行处理技术的核心问题,目前基于消息传递的 MPI (Message Passing Interface)[1] 编程是集群系统上主流的编程模型,而在集群系统上寻求共享存储编程乃至支持自动并行一直是并行计算的研究热点。OpenMP[2] 是共享存储体系结构的并行编程标准,其特点是易于编程且支持增量并行,但是不易扩展;MPI 消息传递编程具有很好的可扩展性,却不易编程调试。MPI 与 OpenMP 各有所长,将两者的优点融合在同一并行结构特别是 SMP 集群系统中,将会获得更好的性能。

1 并行体系结构与集群系统

1.1 并行体系结构分类

从 20 世纪 60 年代初开始,众多的科学家和工程师系统地研究了种类繁多的并行计算机体系结构,并形成了不同的分类方法,其中最为知名的是弗林(Flynn)分类法[3]。目前实际的硬件系统通常被抽象为 6 种机器模型:单指令流多数据流(SIMD)计算机、并行

收稿日期:2006-05-29
作者简介:宋 伟(1972-),男,河南周口人,硕士,讲师,CCF 会员,研究方向为并行计算技术、数据挖掘;宋 玉(1969-),男,河南南阳人,硕士,讲师,CCF 会员,研究方向为数据库。

向量处理(PVP)计算机、对称多处理器(SMP)计算机、大规模并行处理(MPP)计算机、工作站群(COW)和分布式共享存储(DSM)计算机。在这几种并行计算机模型中对称多处理器(SMP)计算机系统因价格低廉、编程模式简单而得到广泛应用,并同时被用来作为大型并行计算机系统的节点,成为构建大型、超大型系统的模块。

1.2　SMP 与集群系统

1.2.1　SMP 系统

当前,只使用单个处理器确实已经很难满足许多实际应用的需求,因而各服务器厂商纷纷通过采用对称多处理系统来解决这一矛盾。对称多处理器 SMP (Symmetrical Multi Processor)计算机中的各个处理器是对称的,所谓'对称",主要是指:

(1)各个处理器在硬件系统中的地位是相同的;

(2)处理器在内存存取上是平等的,每个处理器都能够通过总线共享全部内存,每个处理器存取所有内存空间时的开销是一致的;

(3)各个处理器通过总线进行通信。

在这种架构中,一台计算机不再由单个 CPU 组成,在国内市场上这类机型的处理器一般以 2 个、4 个或 8 个为主,有少数有 16 个以上处理器。多个处理器运行操作系统的单一复本,并共享内存和这台计算机的其他资源。虽然同时使用多个 CPU,但从管理的角度来看,它们的表现就像一个 CPU 一样,拥有一套操作系统,系统将任务队列对称地分布于多个 CPU 之上,从而极大地提高了整个系统的数据处理能力。同时,所有的处理器都可平等地访问内存、I/O 和外部中断。在 SMP 系统中,系统资源被系统中所有 CPU 共享,工作负载能够均匀地分配到所有可用的处理器上。

从 SMP 的系统架构来看,SMP 最大的优势在于共享内存,对内存统一编址。但也正是这一点为 SMP 的发展带来很大障碍,因为共享内存资源必然引起资源冲突。换言之,要保持 SMP 系统从 2 路到 64 路的线性或接近于线性增长,其 SMP 架构下的系统总线、高速总线等同样需要线性增长,而在短时期内要解决通信带宽的急速增长问题是极其困难的。目前大型 SMP 系统主要存在四个瓶颈:

(1)价格昂贵;

(2)系统扩展性较差;

(3)在技术方面,单独依靠 SMP 难以提高计算机整体性能;

(4)系统兼容性差。

因此,在现有条件下实现计算机系统性能的大幅度提升成为了一个焦点问题。

1.2.2　集群系统

近年来,集群系统得到了高速发展,成为提升计算机系统性能的一个较为有效的解决办法。所谓集群(Cluster)系统,就是指利用网络将计算机(主要是商用服务器)按照某种结构连接起来,在并行计算环境下支持统一调度的并行系统。

计算机集群系统允许使用低成本的商品化计算机来构造具有高可伸缩性和高可用性的高性能计算机网络系统,即今天所讲的集群系统。其性能通过数年的技术研究,已经逐渐接近甚至超越同时期同规模的大 SMP 系统。集群系统的主要特点和优点有:(1)性能价格比高;(2)可靠性高;(3)可扩展性好;(4)使用方便;(5)应用领域广泛。

1.2.3　SMP 与集群系统的融合

SMP 系统和集群各有其特点,目前采用 SMP 技术构造集群节点已成为一个趋势。计算机集群系统的发展,使得通过成熟的商业化产品组建的集群系统同时具备低价高性能成为可能。其组成部分主要包括节点机系统、网络系统、存储系统、控制系统、电源及散热系统等。

集群系统的节点机本身就是一个经济型的 SMP 系统,具备 SMP 所具备的优点,并且具有得天独厚的价格优势。由于构建集群系统核心的节点机系统是依托开放的商业计算机系统,使得集群系统具备良好的扩展能力。如可通过双路、四路 SMP 系统构造传统的集群系统,也可以通过 8 路及 8 路双核构造星群系统,通过并行系统的优化满足用户不断变化的应用需求。

2　并行编程模式与方法

随着并行计算技术的发展,为并行系统寻找和开发合适的编程模式与方法日益重要和突出。McGraw 和 Axelrod 于 1988 年提出了为并行计算机开发应用软件的 4 个途径[4]:

(1)扩展现有的编译器以将串行程序转化为并行程序;

(2)扩展现有语言,增加新的操作以允许用户表达并行性;

(3)在现有串行语言上增加一个并行语言层;

(4)定义全新的并行语言和编译系统。

这 4 种途径在实际研究和应用中各有优劣和特点,虽然目前在并行化编译器和高层次并行编程语言方面的工作仍在继续,但当前最流行的并行编程方法是扩展现有串行语言,通过增加函数调用或编译指导语句来表示低层语句,因为此方法只需要开发一个子程序库或函数库即可,现有的语言及其编译器仍然可

原样使用。采用 C 语言、Fortran 语言、MPI 和 OpenMP 是当前广泛使用的编程方式。低层并行编程可产生具有高性能和好的移植性的并行程序,与高层并行编程语言相比,其不足之处在于编程和调试比较困难。

2.1 消息传递编程与 MPI

2.1.1 消息传递模型

消息传递模型如图 1 所示,底层是一组处理器,每个处理器有自己的内存且只能直接访问本地的指令和数据。同时,一个互联网络支持各处理器之间进行消息传递。消息传递模型程序开始时,用户将指定并发的进程数,通常,在程序执行过程中,活动的进程数将保持不变。每个进程执行着同一个程序,但是,由于每一个进程都有一个唯一的 ID 号,在程序展开之后不同的进程可以执行不同的操作。一个进程或执行针对其局部变量的操作,或与其他进程及 I/O 设备进行通信,两个过程可交替进行,进程之间通过传递消息实现数据共享和进程同步。

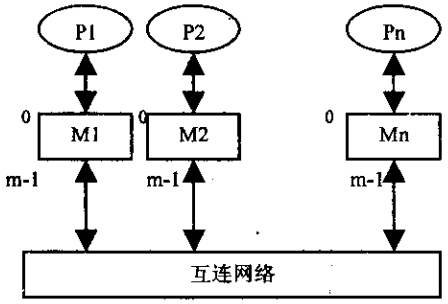

图 1 消息传递模型

2.1.2 消息传递库标准 MPI

20 世纪 80 年代末期,许多公司开始制造并出售多计算机系统。通常,此类系统的编程环境是由一种串行语言,一般是 C 或 Fortran 语言,以及一个能够使其支持进程间通信的消息传递库扩展所组成。由于每个供应商都有自己的函数调用接口,这样就带来了可移植性差的缺陷。于是几年后,出现了多个支持并行计算机的消息传递库的标准,如 PVM(Parallel Virtual Machine)[5]、MPI 等。MPI 吸收了现存的许多系统的最突出优点,成为了当今最为流行的用于并行编程的消息传递库标准。

2.2 共享存储编程与 OpenMP

2.2.1 共享存储模型

共享存储模型的底层硬件为一系列处理器,如图 2 所示,所有处理器可以访问同一个共享存储器的同一个单元,因此,它们可以通过共享变量进行交互和同步。

在共享存储的并行程序中,标准的并行模式为 fork/join。如图 3 所示,当程序开始执行时只有一个叫做主线程的线程存在,主线程执行算法的顺序部分。当遇到需要进行并行运算时,主线程就会派生创建或唤醒一些附加线程。在并行区域内,主线程和这些线程协同工作。在并行代码段结束时,附加线程退出或者挂起,同时控制流回到单独的主线程手中。

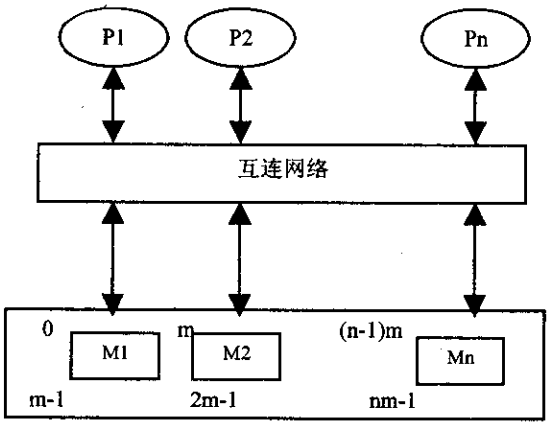

图 2 共享存储模型

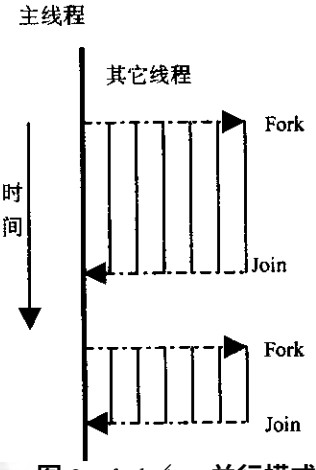

图 3 fork/join 并行模式

可以将顺序执行的程序看作是共享存储模型程序的一种特殊情况,即没有 fork/join 的形式[6]。无论是仅仅在一个循环并行执行中使用了一个 fork/join 的并行程序,还是那些大部分代码段都被并行执行的程序,它们都属于共享存储模型的并行程序。因此,共享存储模型支持增量并行化,就是说可以一次并行化程序中的一段代码,然后继续进行多次这样的操作,从而可以将整个顺序程序转化为并行程序。

2.2.2 OpenMP 的产生与发展

OpenMP 是由 OpenMP 标准委员会(OpenMP ARB, OpenMP Architecture Review Board)于 1997 年 10 月推出的支持共享存储并行编程的工业标准。OpenMP 标准通过定义编译制导、库例程和环境变量规范的方法,为程序员提供了支持 Fortran 和 C/C++ 语言的一组功能强大的高层并行结构,而且支持增量并行。该标准在不断地扩充和发展,目前的最新版本

为 OpenMP API 2.5 版[7]。

2.3　消息传递与共享存储编程的比较

通常认为,共享存储的可编程性比消息传递要好得多。但由于机群是一种典型的分布式存储系统,采用消息传递进行编程是很自然且有效的,因此消息传递系统是目前机群上主流的并行编程环境。然而,共享存储相对于消息传递更易于编程,这使得在机群上实现共享存储编程环境很有吸引力。

消息传递的优点是用户可以对并行性的开发、数据分布和通信实现完全控制,主要缺点是要求程序员显式地处理通信问题。对大多数科学计算程序来说,消息传递模型的真正困难还在于显式的域分解,即将对相应数据的操作限定在指定的处理器上进行。另一个问题是,消息传递程序无法以渐进的方式,通过逐步将串行代码转换成并行代码而开发出来[8]。

共享存储模型和消息传递模型的一个关键区别在于消息传递模型中的所有进程存活于整个程序的执行过程中,而在共享存储模型中,在程序的开始和结束时存活的线程数均为一,而在整个程序执行过程中线程数目会动态变化。

支持增量并行化是共享存储模型相对于消息传递模型而言最大的优势。可以通过分析得到顺序程序的轮廓,按照时间开销对程序的各个模块进行排序,然后从最费时的模块开始逐个进行分析,对适于并行执行的模块进行并行化,直到不能再获得更高的性能为止。

3　在 SMP 集群系统上融合 MPI 与 OpenMP

当今商业集群日益普及,很多集群是由双 CPU 甚至四 CPU 结点组成的 SMP 集群系统,所以由 MPI 与 OpenMP 混合编写的程序,可能更适合由多处理器计算机组成的多机集群系统,从而得到更好的性能。使用 MPI 与 OpenMP 混合编程在很多情况下比单纯用 MPI 编写的程序执行效率更高,主要原因和措施有以下几方面:

(1)更少的通信开销。例如,一个由 n 个结点组成的集群系统,每个结点有 p 个 CPU,如果要利用所有的 CPU 运行使用 MPI 编写的程序,必须要创建 np 个进程,在程序运行过程中,这些进程是要一直存活并相互通信的。如果采用混合编程模式,则只需创建 n 个进程,在代码的并行区,工作负载被分配到每个结点的 p 个线程中,这样,尽管每个 CPU 都被利用了,但由于在通信过程中只有 n 个进程,从而减少了通信开销,提高了加速比。

(2)采用轻量级线程将一些计算并行化。例如,串行程序 A 运行时间为 150s,其中用在可被完全并行化的部分 A1 上的时间为 120s(占总执行时间 80%),用在执行内部串行操作的部分 A2 的时间为 15s,剩余的 15s 是那些可以并行执行,但将会带来很大通信开销的部分 A3。由于对于 MPI 函数调用需要时间较多,所以对于 A3,并不将其并行化,而是在各个进程中均保留。下面来看一下加速比,假设使用了 8 个双 CPU 结点,在共 16 个 MPI 进程的情况下,根据 Amdahl 定律[9],最大加速比为:

$$\text{speedup} = \frac{1}{(q/p + (1-q))} = \frac{1}{(0.8/16 + 0.2)} = 4$$

如果在共享存储环境中,A3 部分的并行化将是可行的,假设这些操作在两个 CPU 上执行时,并行开销忽略不计,那么,在 8 个 MPI 进程上执行程序 A,每个进程内使用双线程,A1 部分由 16 个线程执行;A2 部分仍串行执行因而效率不变;而在各结点间复制的 A3 部分代码,由两个线程执行,从而可将速度提升至两倍。最大加速比为:

$$\text{speedup} = \frac{1}{(q/p + (1-q))} = \frac{1}{(0.8/16 + 0.1/2 + 0.1)} = 5$$

比较两种情况,混合编程模式比单纯 MPI 编程速度提高了 20%。

(3)派生线程来利用空闲 CPU。例如,某应用程序在多处理机集群上执行,在某个结点上,仅有一个进程忙碌而其它进程在等待消息传递,那么忙碌的进程如果派生出一些线程来利用空闲的 CPU 执行一些操作将会提高执行速度。

4　结束语

如何将消息传递和共享存储编程模式的优点融合在同一并行结构中,从而获得更好的性能,是并行计算领域当前和下一步所要解决的问题。MPI/OpenMP 混合编程模式对于现有的 SMP 集群系统具有重要的研究意义。目前 SMP 集群系统是大型计算机系统的一个发展趋势,每个计算节点是一个相对独立的 SMP 系统,适合采用共享内存的 OpenMP 编程,系统由使用 MPI 通信的计算节点构成。因此,大型并行程序设计的第一步往往是“OpenMP 程序并行化”,程序员首先通过 OpenMP 程序并行化使程序运行在 SMP 节点上,随着计算要求的提高和计算量的增大,再使用 MPI 编程模式把程序扩展到集群上。理论分析和测试结果都证明 MPI/OpenMP 混合编程模式是与 SMP 集群结构相匹配的最佳并行编程模式,并将成为今后一个时期研究和应用的重点。

(下转第 171 页)

析层解析,直到遇到<interface/>,Agent 据此通过模块接口层来判定哪个有效的模块负责处理这个请求,在发现有一个模块也叫 interface 后,请求被提交给这个模块。此时,interface 模块知道要把所有有效接口的属性值列表返回给 Agent(根据<get-config>标记)。随后 interface 模块通过一些系统调用获取当前的相关配置信息。

请求的回复则是一个相逆的过程:当模块获得所有请求的系统配置信息后,再把它们返回到 Agent 的接口模块层,XML 解析层为回复加上"头"和"尾"。回复的主要部分都是由模块自身产生的,Agent 模块接口仅仅负责向 XML 解析层通告新属性结点的加入:

```
<!-- 由 XML 解析层添加 -->
<?xml version="1.0"?>
<!DOCTYPE netconf.dtd>
<rpc-reply message-id="1">
  <config>
    <!-- 由 interface 模块接口传递而来 -->
    <interface>
      <iface>
        <name>lo</name>
        <address>127.0.0.1</address>
        <address6>::4624:240:41:240</address6>
        <broadcast>0.0.0.0</broadcast>
        <netmask>255.0.0.0</netmask>
      </iface>
      <iface>
        <name>eh0</name>
        <address>192.168.0.116</address>
        <address6>::48fc:408:41:240</address6>
        <broadcast>192.168.0.255</broadcast>
        <netmask>255.255.255.0</netmask>
      </iface>
    </interface>
    <!-- 由 XML 解析层添加 -->
  </conifg>
</rpc-reply>
```

回复的消息生成后,再递交给网络层,并发送给远端的 Manager。

3　结束语

使用 XML 技术代表着网络管理协议发展的方向,尤其在设备配置管理等应用上。文中以 IETF 正在研讨的 Netonf 协议为基础,具体设计了一个完全基于 XML 消息格式的网络配置管理方案,并提出了模块化实现的构架。利用 SSL 的传输协议,提高了系统的安全性和可靠性。在将来的工作中,需要进一步加强模块的功能,并根据 SNMP 所使用的 MIB 库的定义,扩充新的模块。通过对新协议的模拟实践,分析协议实际应用的可行性。

参考文献:

[1] Schonwalder J, Pars A, Martin-Flatin J P. On the Future of Internet Management Technologies[J]. IEEE Commun Mag, 2003, 41(10): 90-97.

[2] Choi M J, Hong J W, Ju H T. XML-Based Network Management for IP Networks[J]. ETRI J, 2003, 25(6): 445-463.

[3] Dissanaike S, Wijkman P, Wijkman M. Utilizing XML-RPC or SOAP on an embedded system[C]// Proceedings of the 24th International Conference on Distributed Computing Systems Workshops. [s.l.]: [s.n.], 2004: 438-440.

[4] Enns R. NETCONF Configuration Protocol draft-ietf-netconf-prot-10[S]. [s.l.]: IETF, 2005.

[5] Chou W. Inside SSL: accelerating secure transactions[J]. IT Professional, 2002, 4(5): 37-41.

(上接第 167 页)

参考文献:

[1] Snir M, Otto S, Huss-Lederman S, et al. MPI: The Complete Reference[M]. London: MIT Press, 1996.

[2] OpenMP Standards Board. OpenMP: a Proposed Industry Standard API for Shared Memory Programming [EB/OL]. 1997. http://www.openmp.org/openmp/mp-documents/paper/paper.Html.

[3] Flynn M J, Rudd K W. Parallel architectures[J]. ACM Computing Surveys, 1996, 28(1): 67-70.

[4] McGraw J R, Axelrod T S. Exploiting multiprocessors: Issues and options[C]// In Babb R G. Programming Parallel Processors. MA: Addison-Wesley, 1988: 7-25.

[5] PVM news group. PVM: Parallel Virtual Machine[EB/OL]. 2006-07-19. http://www.csm.ornl.gov/pvm/.

[6] Quinn M J. Parallel Programming in C with MPI and OpenMP[M]. [s.l.]: McGraw-Hill Companies, Inc, 2004.

[7] OpenMP Architecture Review Board. OpenMP 2.5 Draft Specification Released for Public Comments[EB/OL]. 2004-11-04. http://www.openmp.org/.

[8] AN Hong, CHEN Guo-liang. Parallel Programming Models and Languages[J]. Journal of Software, 2002, 13(1): 118-124.

[9] Pitman E B. AMDAHL'S LAW FOR PARALLEL SPEEDUP[EB/OL]. 2000-09-13. http://www.math.buffalo.edu/pitman/courses/cor501/HPC1/node11.html.