Java 消息传递环境的研究*)

Study on the Java Message Passing Environment

曾志勇　陆鑫达　张　建

(上海交通大学计算机科学与工程系　上海 200030)

Abstract　A distributed parallel system is a heterogeneous system and Java is a powerful tool to solve heterogeneous problem. This paper introduces some Java message passing environments and points out that it is the natural way to use Java language to implement the portability in distributed parallel computing systems.

Keywords　Parallel computing, Heterogeneous computing, Message passing, Java

1　异构计算系统

异构计算(Heterogeneous Computing, HC)的概念源于 90 年代初期,指由高速网络连接的一系列计算单元(处理机)协同完成某特定任务,使系统开销最小。其中异构是相对同构而言的(同构实际是异构的一个特例),即处理机计算速度、系统结构、负载以及数据传输格式和网络类型可能互不相同。

异构并行计算孕育着无穷的机会,它有如下若干优点:

- 使用现有硬件,减少计算成本;
- 如果能够合理匹配和调度,可以实现超线性加速比;
- 可以利用计算问题本身的异构性实现性能优化,即将特定类型的运算安排到专用机上运行,提高运行速度;
- 虚拟机资源可以随时增减,这有助于采用最新的计算和网络技术。比如,可使用千兆以太网、FDDI(光纤分布数据接口)、HiPPI(高性能并行接口)、SONET(同步光纤网络)和 ATM(异步传输模式)等先进网络技术来提高传输速率;
- 程序员可以使用自己熟悉的环境进行程序开发和调试工作;
- 应用程序或基本操作系统中可以较容易地实现用户级或程序级的容错。
- 分布式计算可推进协同工作。

为了有效开发异构系统,不仅需要硬件的支持,更主要的是需要一个通用的网络编程环境,由此产生了 PVM 和 MPI。

2　PVM 和 MPI

PVM(Parallel Virtual Machine)创始人是 Emory 大学的 Vaidy Sunderam 和 Oak Ridge 国家实验室的 Al Geist,其最初目的是为当时新兴的网络异构计算开发一个研究框架,在 1991 年公开发布。MPI(Message Passing Interface)也是一种并行编程环境。为了统一互不兼容的用户界面,1992 年成立了 MPI 委员会(MPI Forum,简称 MPIF),负责制定消息传递界面的新标准,支持最佳的可移植平台。

PVM 和 MPI 的关键是虚拟机,即把整个异构计算系统看成一个大的虚拟机。它们提供必要的功能使任务在虚拟机上启动执行并支持任务之间的通信和同步。在系统中任务的概念和 UNIX 中进程类似,但不完全相同。由于 PVM 和 MPI 提供通信和同步机制,应用程序可以写成一系列子任务以并行执行,通过发送和接受消息,多个任务可以相互协作并行地解决整个问题。

PVM 和 MPI 支持应用程序、机器和网络等各个层次上的异构。换句话说,允许应用程序利用适合其解的最佳的系统结构组合。如果不同的计算机具有不同的整型、浮点型数的表示形式,PVM 和 MPI 能够处理所有的数据类型转换。另外,PVM 和 MPI 允许虚拟计算机内部由多种形式的网络互联。

文[2]详尽讨论了 PVM 和 MPI 在特征上的差异。

3　Java 消息传递环境

PVM 和 MPI 已被移植到多种计算机上,不同系统上的实现版本所提供的函数功能调用和支持工具的使用方法基本相同,这一优点是它们被广泛使用的主

*)本项目得到国家自然科学基金资助(69773014)。曾志勇　博士生,主要研究领域是并行处理,异构计算。陆鑫达　教授,博导,主要研究领域是并行处理,系统结构,指令级优化编译。

要原因之一。但是，它们也有自身不可克服的缺点。首先，透明性和灵活性差，编写起来较为困难；其次，由于PVM和MPI由C或C++实现，提供C、C++、FORTRAN接口，使得用PVM和MPI开发的并行程序不能在异构系统中做到无缝移植和运行[3]。

上述问题用传统的语言和方法是难以解决的，而近年来Java语言的兴起给这些问题的解决带来了希望。Java语言是一种平台无关语言，具有"一次写成、到处运行"的特殊优点，再加上分布性、多线程性、面向对象性等特点，奠定了Java将成为未来分布式并行计算的基础。

Java编程语言原先的解释执行方式导致Java程序的运行性能较差，但随着Just In Time(JIT)编译、自适应编译(Adaptive Compiler)和Java虚拟机优化等措施的采用和不断进步，使Java程序的运行性能得以持续提升，Java程序的执行速度估计可以达到传统C程序的68%左右[4]，而SUN公司最近发布的Microprocessor Architecture for Java Computing(MAJC)技术将使Java程序的执行性能得到根本的提高；另外，SUN公司已经着手制订实时处理Java API规范以使Java编程语言适用于实时处理领域。可以预见在不远的将来，Java程序将最终达到和C++程序相近的运行性能，成为主流编程语言。

为了把Java技术发展成为高性能计算最好的环境，1998年3月，Java Grande Forum[5]成立。它的下面有几个工作组：Java Grande Numerical Working Group(JGNWG)研究把Java用于数值计算相关的技术；Java Grande Concurrency and Applications Working Group(JGCAWG)则主要关心怎样提高Java程序(并行或串行)的性能和发展Java的基准测试程序以及提供Java消息传递的API。

鉴于PVM和MPI在并行计算上的成功和Java语言在异构网络计算方面的巨大潜力，把它们结合在一起就是非常自然的事了。事实上，在这方面已经有了一些有益的尝试，但都不是很成熟，相关的标准也在制定之中。下面，我们就简要地介绍几个已有的环境。

3.1 JPVM

PVM(Java Parallel Virtual Machine)[6]是一个纯Java语言实现的显式消息传递库。它提供类似于PVM提供的C和Fortran接口，而且由于Java语言的支持而在语法和语义上有所增强，也更好地符合Java编程风格。JPVM与PVM的这种类似性使有经验的PVM程序员能很快地熟悉它，降低了把现有并行程序移植到Java平台的难度。同时，JPVM又拥有PVM所不具有的一些特点，如线程安全、每个任务多个通信点和直接消息路由。JPVM由于完全由Java语言实现，因此具有高度的可移植性。但是，JPVM程序不能与标准的PVM程序进行互操作。

JPVM支持任务或线程作为并行的基本粒度。JPVM消息传递的实现是基于TCP Sockets的，如图1所示。

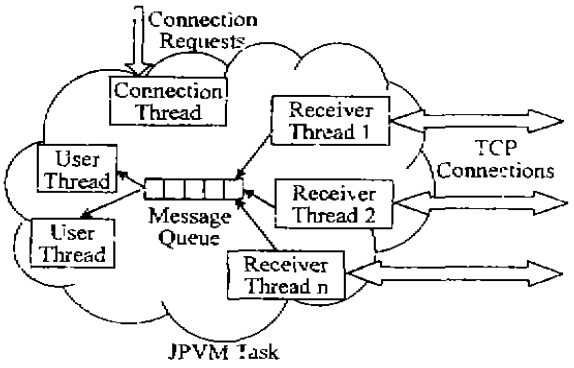

图1 JPVM的通信实现

使用这种任务与任务之间直接的TCP连接，性能超过了PVM中基于UDP的由守护进程路由的老式的消息传递设计。JPVM使用线程来管理连接和消息传递。每一个jpvmEnvironment(负责任务创建、发送消息和接收消息的实例)创建一个专门的线程用于监听和接受对等实体的连接。当这个专门管理连接的线程接受了一个连接后，它创建一个新的接收进程专门处理这个连接。当消息接收线程接收消息时，它把这个消息挂接到一个内部消息队列里。这样，当用户线程执行pvm_recv()操作时，它并不需要任务直接网络输入，而仅仅去查询内部消息队列就可以了。JPVM通信系统实现方案的一个重要特性就是线程安全。多个用户线程可以并行执行pvm_recv()操作，因为内部消息队列是个同步数据结构从而可以安全地被任意数目的读线程和写线程操作。同样，pvm_send操作也是同步而线程安全的。

3.2 JavaPVM

一个与JPVM在编程接口和编程模型上相类似的系统就是JavaPVM[7]。JavaPVM也对Java语言程序提供PVM风格的接口。JavaPVM与JPVM的主要差别在实现上，不象JPVM完全由纯Java语言实现，JavaPVM基于Java的本地方法调用机制，在已经有的标准的PVM函数上加了一层Java本地方法包装，接口是Java的接口，调用的实际上还是标准的PVM函数。虽然这样做有性能(至少在短期内)和互操作性上的好处，但是，可移植性就受到了限制——JavaPVM只能在拥有标准PVM的平台上运行，而且，因为仅仅是在标准PVM的外面加了一层包装，JavaPVM就受到了PVM语法和语义的限制。例如，JavaPVM就保留了PVM提供的非线程友好的缓冲区接口。

3.3 MPIJ

MPIJ 是 MPI 的纯 Java 实现[7]，它也是 Distributed Object Group Metacomputing Architecture（DOGMA）的一部份，MPIJ 参照 MPI-2 的 C++ 实现模型，实现了 MPI 的大部分函数。MPIJ 最重要的一点是，由于它的通信采用了 native marshalling（本地代码编组）技术，使 MPIJ 达到了与本地 MPI 实现相匹敌的通信速度。Native marshalling（本地代码编组）的思想是 Java 在发送非字节类型的数据时，要把数据整理成字节数组再发出去，这种数据整理用 Java 代码来实现是非常低效的，而用本地代码编组实现就快得多。图 2 是一个实验对比[7]。

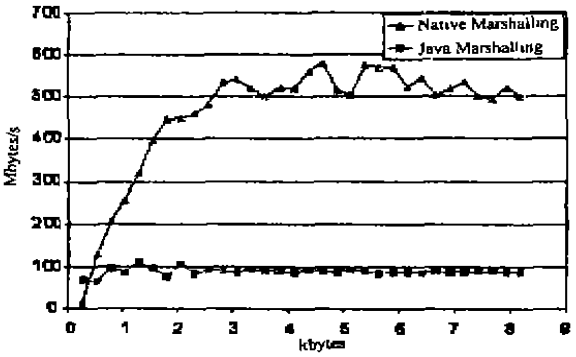

图 2 Native vs. Java Marshalling（JDK 1.2 on a Pentium II 266MHz machine running Windows NT）

3.4 其它

其它系统还有在本地 MPI 实现的基础上加上 Java 包装的 mpiJava[8]、JavaWMPI[9]、JavaMPI[10]，JavaWMPI 是建立在 Window 的 MPI 环境 WMPI 之上的，而 JavaMPI 的 Java 包装由一个叫 JCI（Java-to-C Interface generator）的工具自动产生。jmpi[11]是建立在 JPVM 上的 MPI 的纯 Java 实现。JMPI 是 MPI 软件公司的商业产品计划，但无相关报道。

JPI（Java Parallel Interface）[3]运用纯 Java 语言实现类似于 PVM 和 MPI 所提供的并行程序开发、运行支持环境，与前述软件包不同的是，它运用了 RMI 计算框架。

除了这些消息传递环境以外，还有一个 Java 的网络计算环境 IceT[12]，IceT 是个多用户的、协作的环境，允许用户共享资源并上载代码进行计算。这样做可以充分利用资源，但也带来了安全的问题。

结语 从文[5]、[4]中的测试结果表明：JPVM 原语的开销较大，但应用在合适的粗粒度时，仍表现出良好的加速比。与 JavaPVM 和 PVM 比起来，JPVM 的速度目前看起来最慢，但是，考虑到 JPVM 只是对消息传递的纯 Java 语言实现做了尝试，几乎没有考虑性能的问题。如果在内存分配、I/O 等方面做仔细的设计，性能将会有较大的提高。如 MPIJ[7]由于对 Java 的数据类型的传输采用了 native marshalling 技术，使 MPIJ 达到了与本地 MPI 实现相匹敌的通信速度。文[5]认为由 Java 语言实现的 JPVM 适于支持中粒度到粗粒度的应用，而不适于支持网络密集型的应用。基于 JPI 实现的线性方程组求解程序的性能测试结果表明，使用固定的全互连 Socket 网络的 JPI 能够为网络密集型程序的运行提供有效的支持[3]，另外，Java 语言本身的性能在不断地提高，每一个版本的 JDK 都比旧版本的速度快很多，如据 SUN 报告，JDK1.1.6 的性能比 JDK1.1 高四倍以上，比 JDK1.0.2 高九倍以上，加上 Java 语言带来的种种好处，从长远观点来看，纯 Java 实现方案具有最大的潜力。而 PVM 在 3.4 版本以后，其后继项目 HARNESS[13]已全面转向纯 Java 环境，可以预见，用 Java 语言处理分布式异构并行问题将成为一种趋势。

参考文献

1 Java Grande Forum http://www.javagrande.org/
2 Geist G A, Kohl J A, Papadopoulos P M. PVM and MPI: a Comparison of Features. Calculateurs Parallels, 1996, 8(2)
3 叶靖波，陆鑫达. JPI：基于纯 Java 编程语言的异构并行处理支持平台. 计算机学报，2000，7(7)
4 Yalamanchilli N, Cohen W. Communication Performance of Java based Parallel Virtual Machines. ACM 1998 Workshop on Java for High-Performance Network Computing, Feb. 1998
5 Ferrari A J. JPVM: Network Parallel Computing in Java ACM 1998 Workshop on Java for High-Performance Network Computing, Feb. 1998
6 Thurman D A. JavaPVM: The Java to PVM Interface, December 1996. URL: http. // www. isye. gatech. edu/chmsr/JavaPVM/
7 Judd G, Clement M, Snell Q. Design Issues for Efficient Implementation of MPI in Java. ACM 1999 Workshop on Java for High-Performance Network Computing, 1999
8 Baker M, et al. An object-oriented Java interface to MPI. In: Int. Workshop on Java for Parallel and Distributed Computing, IPPS/SPDP 1999, San Juan, Puerto Rico, April 1999
9 Martin P, Silva L M, Silva J G. A Java Interface to MPI. In: Proc. of the 5th European PVM/MPI Users' Group Meeting, Liverpool UK, September 1998 WMPI-http. // dsg. dei. uc. pt/w32mpi/
10 Mintchev S, Getov V. Towards portable message passing in Java: Binding, MPI. In: M. Bubak, J. Dongarra, J Wasniewski, eds. Recent Advances in PVM and MPI, LNCS, Springer, Nov. 1997. 135 ~ 142. http // perun. hscs. wmin. ac. nk/JavaMPI/
11 Dincer K. Ubiquitous message passing interface implementation in Java: Jmpi. In: Proc. 13th Int. Parallel Processing Symp. and 10th Symp. on Parallel and Distributed Processing, Institute of Electrical and Electronics Engineers, 1998
12 Gray P, Sunderam V. IceT: Distributed computing and Java. In: Proc. of ACM 1997 Workshop on Java for Science and Engineering, Las Vegas, Nevada, June 1997
13 http: // www csm. ornl. gov/harness/