MPI语言绑定：MPI-Delphi，MPI-Java与MPI-Ruby*)

魏兵海

（华中科技大学计算机科学与技术学院　武汉430074）（国家外存储专业实验室　武汉430074）
（信息存储系统教育部重点实验室　武汉430074）

摘　要　MPI(消息传递接口)是最重要的主流并行计算模式之一，它既能应用于当今的分布式环境，也可用于未来的网格环境。本文对以下三种API语言绑定作了综合性分析：MPI-Delphi，MPI-Java和MPI-Ruby，并探讨了其体系架构、实现机制及相关的技术特征。MPI-Delphi基于DLL(动态语言连接)模式实现Delphi语言到MPI库的绑定。在MPI-Java绑定实现时，JVM(Java虚拟机)、JNI(Java本地接接口)、对象串行化范型和Java新I/O库Java.nio都是用作MPI性能优化的关键技术。MPI-Ruby能够提供给用户最易使用和最强大的接口。本文还对一些相关的绑定机制进行了介绍和分析。

关键词　MPI，语言绑定，MPI-Delphi，MPI-Java，MPI-Ruby

MPI Language Bindings: MPI-Delphi, MPI-Java and MPI-Ruby

WEI Bing-Hai

(School of Computer Sci. & Tech., Huazhong Univ. of Sci. & Tech., Wuhan 430074)

(National Storage System Laboratory, Wuhan 430074)　(Key Laboratory of Data Storage System, Ministry of Education, Wuhan 430074)

Abstract　MPI (Message Passing Interface) is one of the most important mainstream parallel computing models, which not only can be employed in present distributed environment but also in future grid applications. In this article, an overview and analysis on three kinds of MPI language Bindings: MPI-Delphi, MPI-Java and MPI-Ruby including which architectures, implementation mechanisms and correlated technologies are described. The MPI-Delphi provides MPI binding for Delphi based on DLL (Dynamic Language Link) mechanism. In MPI-Java binding implementations, JVM (Java Virtual Machine), JNI (Java Native Interface), object serialization paradigm and Java New I/O library - Java.nio are the key technologies that can be employed to optimize the MPI performance. MPI-Ruby, the last one, can provide user the most apt using and most powerful interface. Several related binding implementations are introduced and analyzed.

Keywords　MPI, Language binding, MPI-Delphi, MPI-Java, MPI-Ruby

1　引言

现代分布式并行应用主要有两种模式，即消息传递A模式MPI(Message Passing Interface)和共享存储模式DSM(Distributed Shared Memory)。由于消息传递规范的高标准化程度，消息传递模型应用的较高成熟特性，工业厂商对消息传递MPI的广泛支持事实，使得消息传递API成为当今并行应用领域最为重要的主流模式之一。

自从1994年正式发布MPI-1规范以来，诸多的MPI-1实现向用户提供了C/Fortran77语言的编程界面，在此基础上，1998年推出的MPI-2标准进一步规定了C++/Fortran90编程语言界面。然而，面对事实上存在着远较C/Fortran更多的编程语言类型，诸多应用人员的不同编程语言爱好和趋向，各种编程语言的不同应用优势和应用特征，有必要对MPI的多种语言绑定进行更进一步的深入探讨，以进一步提高消息传递MPI的应用性能、简用性和应用范围等。目前，这已经成为消息传递相关研究前沿中一个具有较大应用价值的分支。

本文拟分别介绍三种典型的MPI编程语言绑定：MPI-Delphi、MPI-Java和MPI-Ruby，分析探讨其各种实现的体系架构、实现机制、技术特征和功能特点等，为进一步的研究和应用提供参考和基础。

2　MPI-Delphi[1,2]

MPI-Delphi是指绑定了Delphi编程语言的MPI实现，其目标是面向Delphi偏好的应用开发人员。通过MPI-Delphi可使用Delphi可视化编程环境编写和调试并行应用程序或监控并行应用程序的运行。Delphi的优点之一是实时编译，Delphi实时编译器并不为特定的应用运行时分派额外文件，其生成的可执行代码具有非常高的效率。Delphi还拥有几个可数的主要可视化编程环境之一。由于通常并行程序的设计必须同时考虑算法的数学细节、并行性以及节点或处理器之间的通信处理等多方面的复杂性，利用可视化编程环境提供各种标准功能组件以简化并行应用开发自然引发了研究者的兴趣。面向Delphi语言的MPI-Delphi可提供高效、可移植的可视化编程界面，因此在许多特定应用领域具有无可比拟的优越性，比如：并行应用的可视化编程与调试、计算密集型并

*)华中科技大学博士后基金项目：基于Globus/Web Service的高性能CFD计算网络研究(AA183107)。**魏兵海**　博士后，主要研究方向为并行与分布式系统，高性能机群与网格计算等。

行图形模拟、多媒体影像的并行处理等。

目前有多个可用于 Windows 平台的免费 MPI 实现，如适用于 Win32平台的 WMPI[3]、可加载于 Win95/98/NT/2K 平台之上的 MPICH[4]等等，这类 MPI 实现的一个共同特征是提供了 C/C++语言绑定，因此此类 MPI 实现的使用环境可以是 VC++、BC++。而 Delphi 使用的本地编程语言则为 Object-Pascal，并不能直接访问 C/C++类的 MPI 库函数。为了实现 MPI-Delphi，必须提供一种从基于 Pascal 的 Delphi 环境到基于 C/C++的 MPI 库的使用访问机制。

一种可行的简易解决方案是利用 DLL 动态连接库模型，由于编程模式基于 Windows 的 DLL 是在运行时进行加载和连接的，因此，运用 DLL 模式可最大限度地减少对应用资源的请求与占用，并使代码的分布具有灵活性和柔韧性，而且，DLL 能够包含一些现有的 Windows 系统和 Windows 应用程序可调用的公共例程和资源，更重要的一点是，用一种语言如 C++编制的 DLL 代码可由其他语言如 Delphi 代码所直接任意访问，通过创建的 DLL 将 MPI 功能函数透明地映射到 Delphi 编译单元 DCU 中供 Delphi 用户调用，即能够实现 Delphi 环境到 MPI 的访问。如果在原 MPI 函数上添加一些标志字符构建 DCU 中的映射类 MPI 函数名，则熟悉原 MPI 和 Delphi 的应用编程人员几乎不需要学习更多的知识即可使用 MPI-Delphi。

如果用 C++编写一个 DLL 库如 dllmpi.dll，一端连接 MPI 的库功能函数，一端向外部如 Delphi 的 DCU 提供输出函数，则必须注意到 C/C++中某些数据类型与 Pascal 中数据类型的不一致性以及某些 Delphi 变量和 C/C++型 MPI 变量之间的转换必要性。C/C++的基本数据类型与 Delphi 的基本数据类型相同，如 single(float)、double、integer(int)、longint(long int)等，基于此种一致性，在 DLL 中定义一种中介数据类型，就可实现异构类型数据之上的信息转换传输。以 MPI 库函数 MPI_Test (MPI_Request * request, int * flag, MPI_Status * status)中变量类型 MPI_Status 为例，Delphi 中不存在相同的数据类型，为此，在 DLL 中用 C/C++类结构定义一个与 MPI_Status 相同结构内容的中介数据类型 DelphiStatus，构建 DLL 中的映射输出函数 FAR_export MPI_Test_beta (MPI_Request * request, int * flag, DelphiStatus * status)，供外部调用。

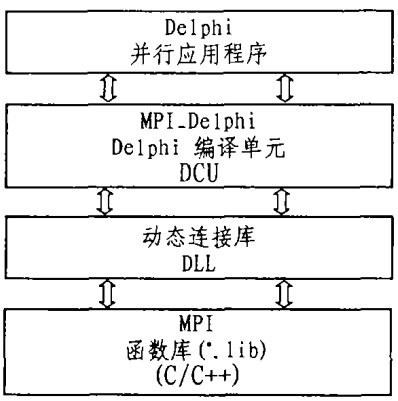

图1 MPI_Delphi 层次结构

DCU 的功能是提供一种屏蔽 DLL 存在的机制。DLL 的一个明显不足是缺乏检查机制。如果在 Delphi 中直接使用创建的 DLL，即在 Delphi 应用程序中以类似头函数文件的形式直接包含创建的 DLL 代码，在 Delphi 应用程序中调用映射输出函数，那么，Delphi 编译器无法检查输出函数中变量参数的个数和数据类型正确与否。通过创建 Delphi 编译单元 DCU，使之包含与 MPI 相关的数据类型和函数，则可解决这一问题。一个完整功能的 MPI_Delphi 层次结构如图1所示。

3 MPI-Java

MPI-Java 是指那些面向 Java 的 MPI 实现。以前，出于执行效率的考虑，多认为 Java 不适合用于做科学与工程计算，近年来，随着 Java 技术的发展和提升，特别是 Java Grande 论坛大力推动 Java 数值计算技术，Java 运行效率有了很大的提高。在 MPI 论坛发布没有包含有关 Java 界面规范内容的 MPI 标准之后，Java Grande 论坛消息传递工作组补充推出了 MPI 的 Java 公共 API 规范草案 MPJ[5]。此外，考虑到 Java 语言具有诸如安全性、健壮性、多线程、分布式、跨平台的互操作性、良好的可移植性等多种特征，使得 Java 语言极其适合于分布式网络编程应用，因此，将网络语言 Java 与高性能并行计算环境 MPI 加以结合自然成为一种很有吸引力的选择方案。

目前，根据 MPI 和 Java 耦合方式的不同，可将 MPI-Java 划分为两类：(1)通过 Java 封装接口绑定到本地 MPI 函数库(Fortran/C/C++)的混合实现[6~8]。(2)使用 Java 语言重写 MPI 库函数的纯 Java 实现[9,10]。

3.1 混合实现与纯实现的比较

在第一种 Java 封装调用 C/C++型 MPI 库函数的 MPI 实现中，MPI 进程仍然能够利用本地底层的高效 MPI 机制通信，因此通信效率较高，然而，在早期的实现中，本地 MPI 和 JVM 在调用系统资源时可能产生占用冲突，如 Java 运行时与 MPI 实现中的中断机制之间的底层冲突，Unix 信号问题等[11,12]，后期实现使用诸如 JDK1.2β 以上版本的本地多线程等机制改善了这一底层冲突[7]。第二种纯 Java 的实现方法基于 Java 的平台独立性实现了异构系统的可移植性，但由于 Java 操作比原本地 MPI 操作的层次高，因此在通常的实现中，MPI 通信性能一般相对较低，然而，最近，通过运用发展的 Java 新技术机制，纯 Java 实现的部分 MPI 通信性能相等于甚至超过 C/Fortran 实现的 MPI 通信性能[9]。

第一类混合实现相对于第二类纯 Java 实现而言实现起来较为容易，但大多数混合实现多少有些依赖于平台相关的本地 MPI 实现细节，并不能比第二类纯 Java 实现更能完美地体现 Java 语言的“一次书写，随处运行”的优秀可移植特性，而且，对用户而言，必须进行两级安装—本地 MPI 库安装后再安装匹配的 Java API 包[13]。因此，后期的 Java 实现多是第二类纯 Java 实现，而且，纯 Java 实现已经成为今后异构网络计算中间件架构技术中的一个发展方向。

3.2 混合实现

在第一种 Java 混合 MPI 实现中，用 Java 封装本地 MPI 函数库的机制通常是通过 JVM 加以实现的，Java 虚拟机提供一种称为 Java 本地接口 JNI[14]的机制，负责实现 Java 代码(Java 应用程序)与其他语言代码(本地 MPI 库函数 C 代码)之间的交互和转换。整个架构由上到下可分为 Java 层、JVM 层和 MPI 层。虽然这一架构能够保留高效的 MPI 进程间本地底层通信机制，但上部 Java 层的通信通常性能较低，同时，上下层 Java 与 C 代码之间的转换包括 Java heap 堆区与 C heap 堆区间的数据拷贝增加了总体开销，并且，JIT 即时编译器因 JNI 而必须做最大保守估计，这也降低了 JIT 可能作出的性能优化[9]。

JavaMPI[6]是一种 MPI 库 C 到 Java 用户界面的 C/Java

混合 MPI 实现。本地 MPI 库之 JNI 接口-Java 封装包由 JCI[15]从 C MPI 头函数中自动生成。JCI 是 Java 到 C 的界面生成器(Java-to-C interface generator),通过 JCI 可达到简化混合实现的目的[6,7]。

MpiJava[7]是另一种将 MPI-2 C++库绑定到 java API 的混合 MPI 实现。MpiJava 要求用户安装 MPI 库以及 Java 和 C/C++编译器,通过 Java 本地界面 JNI 将 mpiJava 类和 MPI 库进行连接,并在运行应用程序前将 Java 编译到 C/C++。在 mpiJava 的 API 函数实现过程中,利用 Java 语言的内在特性,mpiJava 对 API 函数变量列表进行了调整,mpiJava 函数变量列表比 MPI 库中对应的 C/C++函数变量列表更简洁,例如 mpiJava 函数基本上以数组形式返回结果值,省略了对数组元素个数进行计数的函数变量。另外,利用 Java 的自动垃圾收集特性,在大多数 mpiJava 的 API 类中可省略直接的资源释放语句,mpiJava 符合 MPI-2标准和 MPJ 标准[7,8,16]。这些特性都增加了 mpiJava 的易用性。

3.3 纯 Java 实现

第二种纯 Java 实现在用 Java 包函数实现 MPI 标准功能时,基于提高性能的考虑,多借用了一些其他系统机制和新发展起来的技术,如 PVM 机制,Java.nio 技术,Java 对象串行化技术,Java 类装载器技术,jini 技术等进行功能优化。

以应用较多的典型 Java.nio 新 I/O 库技术为例,由于在用 Java 函数实现 MPI 功能时,java.net 和 java.io 库在 WAN 环境中进行基于 C/S 的远程方法调用(RMI)时表现出可接受的性能,但用于 LAN 环境中与 C/Fortran 本地 MPI 实现相比则通信性能不足,因此,在 JSR 51《New I/O APIs for the Java Plateform》规范[17]以及 J2SE1.4推出新 I/O 非阻塞包 java.nio 技术之后,MPI 纯 Java 实现即利用这一种新的技术手段改善 MPI 通信性能。文[9,18]对相关细节作了详细描述。

MPJava 是 Maryland 大学采用 java.nio 新技术实现的一种纯 Java 的消息传递库包。MPJava 通过在相关的每一节点机上安装的 JVM,经由网络,各 JVM 之间通过 java.nio 库可无阻塞 TCP SocketChannel 互联进行高效组通信。将 MPJava 实现用在 NAS 共轭梯度 Benchmark 中,其测试性能超过了 Fortran/MPI 实现[9]

Jmpi 是一种建立在 JPVM 系统[11,12]之上的 MPI 纯 Java 实现,它借用了 JPVM 的一些功能特征如网络环境下的启动和通信机制、单任务多通信端点、线程安全等。Jmpi 一开始即在每一个节点主机上启动守护进程 daemon,完成相关环境初始化,如 MPI_Init( )进程创建、MPI Comm_world 通信体管理等。Jmpi 在 JPVM 之上实现了许多 PVM 没有的新特征,例如,Jmpi 任务到任务消息传递基于 TCP 套接字而非 PVM 环境中常用的 UDP 任务到任务通信机制,消息交付采用直接的任务到任务方式而非 PVM 环境中常用的经各主机节点上 daemon 路由间接到任务机制。Jmpi 实现包括其面向用户的接口函数都符合 Java Grande 的 MPJ 规范。性能测试结果表明:无论是长消息还是短消息,Jmpi 与 JPVM 的带宽十分接近,对长消息传输对象,Jmpi 带宽和启动延迟都极为接近,对短消息传输对象的带宽和启动延迟,Jmpi 和 JPVM 都较为接近,但两者性能都远低于 C/PVM[10]。

MPIJ 是一种纯 Java 的 MPI 实现。MPIJ 启动后首先寻找并优先利用本地机编组(Marshling)库,其次才是使用 java 库进行数据编组。这一机制的设计原因在于,虽然 C 和 Java 传输 byte 数组数据的底层通信机制相同因而网络通信效率相近,但进行非 byte 数据类型到 byte 数据类型转换的数据编组机制却大不相同,在 C 中用存储拷贝(memcpy)即可简单实现的数据编组在 Java 中则需要相当多个指令包括方法调用、移位操作、数据类型转换操作等复杂过程才能完成。在 MPIJ 的进一步改进升级中,除原设计中优先利用本地机编组库机制外,还考虑进一步使用 Java 即时编译技术 JIT 优化 Java 编组机制以及无存储拷贝的类型化数组通信技术以大幅提高 MPIJ 通信性能。在多处理器机上,MPIJ 还使用对类变量的共享访问进行高效组通信,允许用户在源进程缓冲器和目的进程缓冲器之间进行直接数据拷贝以提高通信效率[19]。MPIJ 符合 Java Grande 的 MPJ 规范,并已经作为一个整体模块置入分布式对象组元计算结构 DOGMA 系统[20]中。

3.4 功能扩展的 Java 实现

随着 MPI 标准中功能的扩展,从 MPI-1到 MPI-2增加了进程创建动态迁移容错等许多新特征,MPI 作为一种基础设施在网格环境中也得到推广应用,这对 MPI 的 Java 实现提出了更多的要求和课题,众多的研究小组对异构网络环境下的 MPI-Java 实现进行了探索。

M-JavaMPI 是一种支持进程间位置透明通信、进程迁移和动态负载平衡的混合实现,作为机群中间件,M-JavaMPI 首先设置了一预处理层(Pre-Processing Layer),在 JVM 执行之前往 Java 应用程序字节码文件(Java.Class)中插入 exception 异常句柄,异常句柄内容执行进程状态的恢复,当插入的恢复函数与迁移层共同作用时,即产生动作恢复 Java 堆栈和迁移进程的重新执行。预处理层下的 Java-MPI API 层提供 Java 应用程序调用 MPI 的接口,再下一层的迁移层(Migration Layer)负责使用 JVM 异常处理捕捉和使用对象串行化技术保存迁移进程在源节点上的执行状态信息,并在目的节点上恢复迁移进程的执行状态。迁移层还负责与底层的可恢复 MPI 层协作重构并行应用间的通信通道。为不对 JVM 做出改变,利用 Java 内建界面 JVMDI(Java Debugging Interface),在将可能的进程迁移点设定在代码行的开首以避免操作数堆栈开销后,由 JVMDI 捕捉所有进程的执行状态,将迁移进程执行状态交付到迁移层。紧邻 JVMDI 的之下是 JVM 层,在 JVM 层和本地 MPI 层之间设置一可恢复 MPI 层(Restorable MPI Layer),即设计添加的 MPI 组件层,通过 MPI daemon 守护进程支持通信进程迁移后自动重构通信通道,以恢复进程迁移后与其他进程的通信[21,22]。

基于代理的并行 Java 项目 ABPJ[23]通过在参与主机上安装 Agent 软件包,负责对多节点上的并行进程实施协作管理和调度,并提供动态负载平衡和容错等多种功能机制。ABPJ 提供给用户的 Java 对象传递界面 JOPI 非常类似于标准 MPI 和 MPJ 界面,JOPI 允许进程进行对象粒度的消息传递,并使用 Java Socket 套接字实现比 RMI 更快的通信速度和更高效的性能[24]。文[16]对部分分布式并行 Java 消息传递项目及其技术细节作了比较和总结。

综上所述,Java 是一种比其他任何一种语言都能够更好地支持并行机制的编程语言,Java 具有优秀的通信库,并且处于不断的升级和优化之中,表现出强大的生命力。Java-MPI 可能是将来最有发展前途的语言绑定[25]。

4 MPI-Ruby[26~28]

MPI-Ruby 是将脚本语言 Ruby[27]绑定到 MPI 库的消息传递实现。解释型语言 Ruby 继 Perl、Python、Tcl、Guile 等语言之后于1993年由 Yukihiro Matsumoto 发明并逐渐在日本和美国流行,由于具有严格的面向对象性、可嵌入动态特性、良好的可移植性、规则直观的语义等多种优秀特征,Ruby 成

为迄今为止最易于编写、最易于阅读、最完全面向对象、功能最强大超过 Perl、Python 等任何一种编程语言的脚本语言。Argonne 实验室曾尝试将各种脚本语言如 Python 绑定到 MPI[29]，但是由于其他脚本语言包括 Python 本身固有的限制而只能部分实现标准 MPI 功能函数。而 Ruby 良好的相容性为完整地实现所有标准 MPI 库函数提供了完全的可能性[26]。

MPI Ruby 是 Berkeley 大学正在进行的一个 MPI-Ruby 实现项目。MPI Ruby 集成了 MPI 和 Ruby 的优势并进一步作了扩展。曾参与 Argonne 实验室 MPICH 项目的 Emil Ong 在2002年下旬推出 MPI_Ruby 时，其开放源码的 MPI_Ruby 初版已经基本实现了 MPI 对 Ruby 脚本语言的绑定，实现的 MPI Ruby 因而兼具 MPI 丰富的通信功能特性、Ruby 的强大语法功能特性、彻底的面向对象性和极端的易用性，不仅如此，优于 MPI 对 C/C++/Fortran 等语言绑定的是，MPI Ruby 在实现 MPI 对 Ruby 语言绑定的同时，更重要的是利用 Ruby 语言的特性对 MPI 实施了简化，极大地提高了易用性。

MPI Ruby 包括解释器 mpi_ruby 和 ruby 语法格式的 MPI 功能模块两部分。MPI 功能模块用 Ruby 语言实现 MPI 标准函数库功能，解释器 mpi_ruby 执行对 Ruby 解释器的调用，它本质上是一个简单 MPI 模块，集成实现 MPI_Init (&argc, &argv)初始化、MPI_Finalize()结束以及调用 ruby 三种功能。通过省略原 MPI 中的 MPI_Init(&argc, &argv)和 MPI_Finalize()两条语句，以及在大规模计算的源程序中省略冗长的变量声明，MPI Ruby 简化了并行源程序的读写。另外，在 C 或 Fortran 绑定的 MPI 中通常还非常注重缓冲(buffers)、数据类型(data types)、字节数 (byte counts)等概念，为防止差错，编程时必须对这一类型的参数小心谨慎。而在 MPI Ruby 中，由于其彻底的面向对象特性，包括数据、消息等一切都可视为对象，对象可用来代替函数，通过调用对象的方法实现原函数特定的功能，在用户编程实现数据发送/接受功能时，无需缓冲、数据类型和字节数等参数概念。以接收操作为例，在 C/MPI 中为：MPI_RECV(void * buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status * Status)，其中，buf：接收缓冲区起始地址，count：接收消息对象个数，datatype：接收消息数据类型，source：发送消息进程标号，tag：消息标志，comm：发送/接收进程所在通信体，status：返回状态。在 MPI Ruby 中为：MPI::Comm::WORLD.recv (task, source, tag)其中，task：接收的消息对象，rank：发送消息进程标号，tag：消息标志。语句的简化使程序人员的注意力可以从语句细节本身转移到程序算法结构上来，极大地增强了语言的易用性。

解释型脚本语言优于编译型语言的一个特点是开发时间上的快速性和可嵌入动态特征。类似于 JSP，Ruby 语言语句可嵌入到 MPI Ruby 等程序语言的语句中，解释器能够从一个动态构建的字符串中或从一个读入的文件中获取和执行 Ruby 代码。利用 Ruby 的嵌入动态机制，Ruby 执行守护程序包 RED 缩短了许多并行应用的进程启动时间，首先一个 MPI Ruby 进程从一个套接字或文件中读取 RED 并广播到相关节点，远程节点上的解释器将 RED 作为字符串加以执行启动相关进程，如此，一个远程进程的启动延迟缩短为利用 MPI 传递 RED 文本的时间。

MPI-Ruby 正处在发展中，类似于初期的 Java-MPI，Ruby 的解释性本质限制了程序执行的速度，然而，Ruby 的多种内在特征赋予了 Ruby 语言强大的生命力，相信随着 Ruby 技术和其他相关技术的进步，MPI-Ruby 将会逐步成为最为流行的并行语言绑定之一[30]。

总结 MPI-Delphi、MPI-Java 和 MPI-Ruby 是三种具有代表意义的 MPI 编程语言绑定，本文对多个相关绑定实现的架构和机制进行了研讨和分析。

MPI-Delphi 基于 DLL 动态连接模式实现 Delphi 语言到 MPI 库的绑定，面向 Delphi 用户提供具有可视化和图形处理优势的并行 MPI/Delphi 编程界面。

MPI-Java 则包括 Java MPI 混合实现和 MPI 的纯 Java 实现两大类型，随着 Java 技术的进步，相关研究人员在不同时期逐渐发展出了各具特色的不同实现，Java 虚拟机机制、对象串行化范型、Java 本地接口 JNI，Java 新 I/O 库 Java.nio 等，都是相关实现用以进行性能优化的关键技术，Java Grande 论坛在推动 MPI-Java 等并行大规模 Java 应用方面发挥着重要作用。

MPI-Ruby 提供了将新型编程语言 Ruby 绑定到 MPI 库的实现。虽然受限于 Ruby 语言的年轻性特征，但 MPI-Ruby 表现出强大的应用功能、广阔的发展前景和实用特性。

相对于实际应用中较多的 MPI 编程语言绑定实现，限于篇幅，本文介绍的仅仅只是其中的一部分。读者可以进一步参考相关文献资料。在未来，随着 MPI-2等规范对 MPI 特性的延伸和网格技术浪潮的出现，MPI 与不同语言技术的绑定集成将更为复杂，并表现出更多的高级功能。

参 考 文 献

1 Acacio M, Canovas O, Garcia J M, et al. MPI-Delphi: an MPI implementation for visual programming environments and heterogeneous computing. Future Generation Computer Systems, 2002, 18:317～333

2 Acacio M, Garcia J M, LopezdeTeruel P E. The MPI-Delphi interface: a visual programming environment for clusters of workstations. In: proc. of the PDPTA'99, CSREA Press, 1999. 1730～1736

3 Marinho J, Silva J G. WMPI-message passing interface for Win32 clusters. In: proc. of the Fifth Euro PVM/MPI, Lecture Notes in Computer Science, Springer, Berlin, 1998. 113～121

4 Gropp W, Lusk E. User's Guide for mpich, A portable implementation of MPI, 1996

5 Carpenter B, Getov V, Judd G, et al. MPI for Java: Position Document and Draft API Specification, Java Grande Forum, JGF-TR-3, 1998. http://www.javagrande.org

6 Mintchev S, Getov V. Towards portable message passing in Java: Binding MPI, Report TR-CSPE-07, University of Westminster, School of Computer Science, London, 1997

7 Baker M, Carpenter B, Fox G, et al. mpiJava: An Object-Oriented Java interface to MPI. In: Intl. Workshop on Java for Parallel and Distributed Computing, IPPS/SPDP 1999

8 Baker M, Carpenter B, Ko S, et al. mpiJava: A Java interface to MPI, 1st UK Workshop on Java for HKCN, 1998

9 Pugh W, Spacco J. MPJava: High-Performance Message Passing in Java using Java.nio. In: Proc. of Mid-Atlantic Student Workshop on Programming Languages and Systems, MASPLAS'03, Haverford College, 2003

10 Dincer K. A Ubiquitous Message Passing Interface Implementation in Java: jmpi. In: Proc. of 13th Intl. and 10th Symposium on Parallel and Distributed processing, 1999

11 Thurman D. Javapvm: [Technical report]. http://homer.isye.gatech.edu/chmsr/JavaPVM.html/, 1997

12 Ferrari A J. JPVM: Network Parallel Computing in Java. In: Proc. of ACM 1998 Workshop On Java for HPCN, 1998

13 Baker M,Carpenter B. A Proposed Jini Infrastructure to Suppot a Java Message Passing Implementation. 2000
14 Liang S. The Java Native Interface: Programmer's Guide and Specification. Addison Wesley, 1999
15 Getov V, Flynn-Hummel S,Mintchev S. High-Performance parallel programming in Java: Exploiting native libraries, Concurrency: Practice and Experience,1998
16 Al-Jaroodi J,Mohamed N,Jang H,et al. A Comparative Study of Parallel and Distributed Java Projects for Heterogeneous Systems. Workshop on Java for Parallel and Destributing, IPDPS, 2002
17 Reinhold M. New I/O APIs for the JavaTM Platform, JSR 51. http://www.jcp.org/jsr/ detail,2002
18 Welsh M,Jaguar D C. enabling efficient communication and I/O in Java. Concurrency: Practice and Experience,2000,12(7):519～538
19 Judd G,Clement M,Snell Q,et al. Design Issues for Efficient Implementation of MPI in Java. In: ACM 1999 Java Grande Conf. ACM Press, June 1999
20 Judd G, Clement M J, Snell Q. DOGMA: Distributed Object Group Metacomputing Architecture. Concurrency-Practice and Experience,1998,10(11-13): 977～983
21 Ma R K K,Wang C L,Lau F C M. M-JavaMPI: A Java-MPI Binding with Process Migration Support The Second IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2002), Berlin, Germany
22 Ma M J M,Wang C L,Lau F C M. JESSICA: Java-Enabled Single-System- Image Computing Architecture. Journal of Parallel and Distributed Computing,2000,69(10)
23 Al-Jaroodi J, Mohamed N,Jang H,et al. An Agent-Based Infrastructure for Parallel Java on Heterogeneous Clusters. In:4th IEEE Intl. Conf. on Cluster Computing (CLUSTER 2002)
24 Al-Jaroodi J,Mohamed N,Jang H,et al. Agent- Based Parallel Computing in Java - Proof of Concept: [Technical Report TR-UNL-CSE-2001 -1004]. http://cse.unl.edu /～jaljaroo/ projects_1.htm
25 Fox G. Grande Applications and Java. Short Review on 2002 Joint ACM Java Grande- ISCOPE Conf. 2002
26 Emil ong, MPI Ruby: Scripting in a Parallel Environment. IEEE Computing in Science & Engineering,2002,4(4):78～82
27 Thomas D, Hunt A. Programming Ruby: The Pragmatic Programmer's Guide. Addison Wesley Longman, New york, 2000
28 Ruby Home Page. http://www.ruby-lang.org/
29 Miller P. PyMPI: Parallel, Distributed Python:[Technical Report: UCRL-PRES-150673]. Oct. 2002
30 Thomson P. TaskMaster: Distributed computing with ruby. Second international ruby conference 2002. http://www.zenspider.com/dl/rubyconf2002/TMpres.tar.gz

(上接第155页)

的相互影响减小到最小,而独立连接件思想将软件的计算功能和交互(集成功能)相分离,进一步提高了软件的重用性和扩展性。

SCBSA 采用策略与机制分离的原则、功能与实现隔离的思想以及构件技术,将连接件进一步分成了由 Switch 构件、Router 构件和 Bridge 构件,形成了彻底构件化的软件体系结构。其中 Router 构件负责连接策略,Switch 构件负责连接机制的实现,Bridge 构件负责 SCS 之间的交互。

SCBSA 的实现环境主要由构件基础设施层、SCBSA 组装器和 SCBSA 运行支持器构成。构件基础设施层主要为生产构件和构件运行服务;SCBSA 组装器是利用已经生产好的构件组装出一个符合 SCBSA 体系结构风格的软件;SCBSA 运行支持器在构件基础设施层的支持下,把组装结果实例化成可运行的软件系统。构件基础设施层和 SCBSA 运行支持器可以由一般的面向对象编程语言或 COM、CORBA 等公共组件编程规范来实现。

通过一个应用实例和对相关工作的比较,SCBSA 体系结构具有以下重要特点:首先,通过构件更替的方式来代替一般的软件修改方式,利用策略与机制分离及功能与实现相分离的思想进一步减小软件各部分之间的影响,使其为适应变化而改动的地方更少、更集中,改动的方式更灵活,改动的负作用更小,提高了软件的维护性;其次,提高了软件系统中客户机(集成)部分的重用性和灵活性,从而提高了整个软件的重用性、可扩展性并具有较高的效率,同时灵活支持各种体系结构模式的动态演化;最后,SCBSA 在概念上可以看成是普通程序设计语言的推广,设计者能够很容易理解和使用 SCBSA。总之,从 CORBA、COM 等公共组件对象模型规范一直到 SCBSA 是一个体系结构构件化程度不断加深和重用性不断提高的过程。

下一步的工作是将 SCBSA 这种体系结构应用到更多的软件设计中,特别是把它应用于实时软件中提高实时软件的开发效率。为了方便 SCBSA 的应用,需要开发相应的 CASE 工具支持 SCBSA 的设计应用,特别是开发图形化的 SCBSA 组装器来设计应用软件。还要开发能在各种平台上运行的并有高效率的 SCBSA 运行支持器。

参 考 文 献

1 Mehta N R,et al. Towards a Taxonomy of Software Connectors [A]. In:Proc. of the 22nd intl. conf. on Software engineering[C]. New York:ACM Press,2000. 178～187
2 Henning M,Vinoski S. 基于 C++ CORBAR 高级编程[M]. 北京:清华大学出版社,2000
3 Rogerson D. COM 技术内幕[M]. 北京:清华大学出版社,2002
4 Jeong C,Lee S. Implementing Software Connectors through First-Class Methods[A]. Systems,Man,and Cybernetics. In:2000 IEEE Intl. Conf. on[C]. Nashville,TN:IEEE,2000. 92～96
5 张家晨,冯铁,陈伟,金淳兆. 基于主动连接件的软件体系结构及其描述方法[J]. 软件学报,2000,11(8):1047～1052
6 Taylor R N,et al. A Component- and Message-Based Architectural Style for GUI Software[J]. IEEE Trans. Software Engineering,1996,22(6):390～406
7 Medvidovic N,et al. Using Object-Oriented Typing to Support Architectural Design in the C2 Style[A]. In:Proc. of the 4th ACM SIGSOFT symposium on Foundations of softwareengineering[C]. New York:ACM Press,1996. 24～32
8 李保建,曾广周,林宗楷. 一种基于 TriBus 的软件集成框架[J]. 计算机研究与发展,1999,36(9):1116～1120
9 Beach B W ,et al. Connecting Software Components with Declarative Glue[A]. In:Proc. of the 14th intl. conf. on Software engineering[C]. New York:ACM Press,1992. 120～137