基于构件和数据流的可视化元计算环境*)

Research on and Design of Meta-Computing system

李国东　王东奎　杨海荣　张德富

(南京大学计算机软件新技术国家重点实验室　南京210093)

Abstract Metacomputing technique is one of the main stream techniques in the future. In this paper we propose a component-dataflow-based metacomputing environment which provides integrated development platform and run-time environment to construct and execute parallel applications. A parallel computational model (dataflow model based on component)is presented, which enables programmers to use components to build parallel applications using the same way in building sequential applications. We also present a three-tier client/server architecture for metacomputing environment: front-end, middleware, resource tier. The front-end answers for visual parallel application development; the middleware is layered management system which manages tasks, computing resources and maps tasks to appropriate resources; and the resource tier is in charge of the management and scheduling of distributed resources. To avoid performing tedious and time-consuming low level operations, we use Globus toolkits to implement most of underlying functional components.

Keywords Metacomputing, Parallel computing model, Component, Dataflow, Visual environment

1. 介绍

1992年 Smarr 和 Catlett 首先提出了元计算的概念[1],他们认为,元计算就是把分布在广域网上的各种计算资源集合起来,并以一个整体的计算环境的形式提供给用户。元计算系统不但能提高系统整体的计算能力,而且它还集成了一些高性能专用设备,使得很多普通用户也有机会能够使用它们。然而,元计算并不仅仅指分布式计算。迄今为止,我们已经可以在很广范围内获得基于网络的元计算服务,从传统的基于局域网或 Internet 的客户—服务器应用程序、虚拟组织、远程工作,一直到 Internet 上的自动协商 Agent(如搜索引擎)。元计算的目标是将地理上分布的、结构上相异的各种高性能计算机、数据服务器、大型检索存储系统和可视化、虚拟现实系统等,通过高速互连网络连接并集成起来,共同完成一些缺乏有效研究办法的重大应用研究问题。

硬件、并行编程系统(如 PVM[2],MPI[3])和资源管理系统(如 Condor[4]、Codine[5]、LSF[6]等)这三方面的共同发展,推动了元计算系统真正得到实现并为人们所接受,其中的关键在于它成功地跨越了分布式系统异构性这个门槛。而我们面临的下一个门槛将是:不但要能够隐藏结点的异构性,而且还要能隐藏各结点资源管理策略的异构性,使得元计算环境能够以类似于单计算机环境的方式向用户显现。而在元计算研究中所遇到的最大的挑战,正是这种对透明性的要求所带来的。

现存的大部分元计算项目都是从对选定的几个方面入手,通过自底向上的方式,逐渐扩展成一个完整的元计算系统的。在过去几年里,人们已经设计并实现了许多元计算模型,其中并行编程模型是一个很流行的起始点,很多项目的定位就是把现有的编程模型扩展为完整的元计算环境。例如,俄亥俄超级计算机中心 LAM(Local Area Multicomputer)[7]使用 MPI 为互连的异构计算机构筑了用于一个可移植的系统开发和执行的环境;意大利的 WAMM(Wide-Area Metacomputer Manager)[8]用 PVM 来构建、提供了一个带有并行任务控制、远程编译和图形化用户界面的 PVM 编程环境,后来通过引入 PLUS[9]通信库来支持 MPI 程序。

还有一些项目以机群管理系统为前身,它们通常来说比较看重资源管理、任务分配、检查点以及进程迁移等方面。一些已经存在的机群管理系统(如 Condor、Condine、LSF)经过修改,已经能够管理大规模的、在地域上分布的由多个机群组成的系统。在衣阿华州立大学的 Batrun[10]项目、耶鲁大学的 Piranha[11]项目以及 Duke 的 Polder[12]项目的第一个版本中,都做过这方面的尝试,它们致力于利用闲置的工作站来进行大规模的计算以及高数据吞吐量计算。而 Nimrod[13]项目支持同一个串行程序在不同的参数集上计算的并行执行,已经开始运行某些复杂的模拟程序如空气污染和激光原子模拟程序等。而圣地亚哥的 AppLeS[14]项目则在异构环境里提供应用程序级的调度代理,它把各个资源的性能也作了考虑。Globus[15]项目是建立在 I-Way、I-Soft、AppLeS、Nexus 以及别的一些项目的研究结果上的一个可适应广域资源环境的框架,向用户提供一系列的工具以使得应用程序能与异构并且动态改变的元计算环境相适应。

在国内,元计算的研究工作开展得不是很多。中国科学院计算所、江南计算所、国防科技大学等单位,开展了网格系统软件的研制和开发,该软件系统具有单一的用户资源管理和单一的作业管理功能,支持远程开发和远程生产性使用,属于元计算前端的研究工作。在后端的资源管理以及编程模型方面,我们暂时还没发现比较成功的研究成果。

由于要想把那些计算资源构筑成元计算环境要作出很大

*)本课题得到国家863项目(编号863-306-ZT06-04-4)的资助。李国东　助教,硕士,主要研究领域为并行与分布计算。王东奎　硕士,主要研究领域为并行与分布计算。杨海荣　硕士,主要研究领域为并行与分布计算。张德富　教授,博士生导师,主要研究领域为并行处理技术和分布式系统。

的努力，并且由此得到的好处也往往只是对一些松耦合的应用才比较有效；另外，鉴于对象技术能够提供良好的封闭性和复用性，结合对象技术来构建元计算环境是一个重要尝试，而尽管CORBA[16]等技术为分布对象技术奠定了良好的基础，但它缺乏对异构分布的并行对象的充分支撑。为此，本文以数据流技术和构件技术为基础来构建一个可视化元计算环境，为并行应用程序的构造、执行提供了一个可视化的开发平台和运行环境。下面介绍基于构件/构架的软件开发技术；接着论述并行计算模型，具体描述一个适合并行应用程序开发的基于构件的数据流模型；我们提出的元计算环境体系结构是三层结构的客户/服务器模型；然后介绍客户端；最后给出结论。

2. 基于构件/构架的软件开发

在领域分析的基础上和构件/构架库的支持下进行应用开发，其主要特点是应用领域中各个系统的共同问题已在领域分析中得到了一般共识，并通过构件、构架的开发统一地进行描述与解决。因此该领域的应用系统开发主要是分析和解决具体系统中的特殊问题，系统开发可以看成一个以组装为主的软件开发过程，即：按照领域构架来确定本系统需要哪些可重用构件，根据本系统的特殊需求对构件和构架进行特殊化，并开发本系统的专用构件，然后将领域构架、领域构件和通用构件组装成一个完整的系统。图1说明了这样的开发过程。

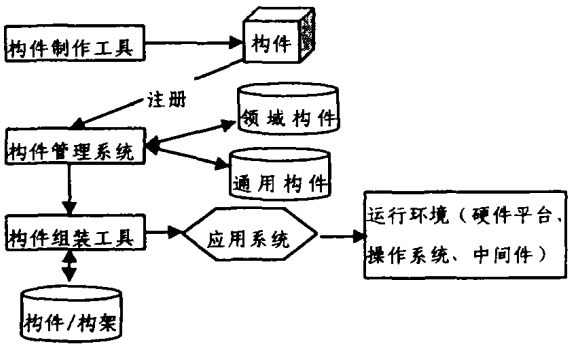

图1　基于构件/构架的软件开发

构件制作工具支持具体的构件制作。构件管理的功能主要包括构件注册/注销、构件存储、构件定位等，其核心是构件库及其管理系统。构件库存储构件，提供可重用的软件资源；构件库管理系统类似于数据库管理系统，是为构件库的建立、使用和维护而配置的应用软件，它建立在操作系统的基础上，对构件库进行统一的管理和控制，它提供一套构件库管理语言如构件定义语言(Component Definition Language--CDL)和构件操纵语言(Component Manipulation Language--CML)等来实现对构件库的一般管理和特殊管理。构件组装指的是将领域构件、领域构架和通用构件组装成一个应用系统。

分布式构件对象模型是为了开发者定义分布式处理软件而建立的体系结构和API集，使开发者可通过构件的动态组合来建立分布式应用系统。CORBA[16]是典型的分布式对象模型。

CORBA的构件体系结构包含四个彼此一致的对象模型：抽象构件模型、打包模型、配置模型和容器模型。它们相互协作，构成了完整的企业服务器计算体系结构。使用构件的第一步是，通过抽象构件模型创建和实现构件，然后通过打包模型把它们存到CORBA构件描述符(CCD)文件中。可以把一个CCD文件组合成一个完整的应用，也可以把它装配到构件集合描述符(CAD)文件中。在CAD文件中，每个构件通过打包模型和配置模型与一个目标名建立联系。配置工具理解CAD文件的含义，它通过配置模型把构件配置到已命名的配置目标上。每个结点上有一个安装对象，它负责安装和激活构件。构件根据容器模型在容器中运行。

封装的构件应支持软件开发周期的全过程。一个完整的构件体系如图2所示，应包括其分析设计文档体、源代码体、测试用例体等几个部分。具体说，分析设计文档体提供了构件实现时的各种文档，如图表(diagram)、说明(specification)等；源代码体是构件在分析设计基础上得到的语言级实现代码，也是构件最容易被重用的部分；测试用例体是构件黑盒、白盒测试时的典型测试用例的集合。为了保证构件的一致性，分析设计文档体、源代码体、测试用例体和计算环境也应保持描述上的一致性，对其中任一部分的修改也应同时修改其他相关部分。

构件标识信息	构件属性信息	分析设计文档体	源代码体	测试用例体	计算环境

图2　构件的封装

实际的构件在存储时，可能并不提供以上的所有部分，例如有时仅仅要求独立存储文档作为构件，而有时又限于存储空间仅保留代码而不需文档，这些都要求构件的封装保证一定的灵活性，标识部分信息中的“生命周期支持标识”正是出于这种考虑而设计的。

3. 并行程序设计模型

并行程序设计模型(Parallel Program Model)是一种程序抽象的集合，它给程序员提供了一幅计算机硬件/软件系统透明的简图，程序员利用这些模型就可以为多处理机、多计算机和工作站机群等设计并行程序。

目前的并行程序设计标准主要包括Fortran90[17]、HPF[18]、PVM、MPI、OpenMP[19]等。构件开发人员可以根据自己熟悉的开发平台开发构件，或者出于性能考虑而选择效率较高的开发平台，如HPF在开发大型数组、大型网络或网状数据的空间并行性时相当有效。

由于数据流的思想非常适于并行和模块级编程，也非常适于用图示的方式来描述和分析应用程序，我们采用了数据流方式的分布并行计算模型，子任务之间按数据驱动的方式决定并行执行的先后次序。结点是相对独立的具有数据变换功能的构件，有入端口和出端口，管道是结点间传递数据的通道。流图是由结点和管道组成的有向图，一个流图对应一个具体的可视化应用实例，表示各构件间的数据传递和依赖关系。流图可表示为有向图$G=\langle V,E,F,C,S\rangle$：V是结点集，包括控制结点在内的所有结点；E是边集，即流图中的管道；F是映射关系，$F:E\rightarrow V\times V$；C是约束条件集，它规定了G中各边的合法性，即用户在创建管道时应遵循的规则；S是状态集，表示每个结点的当前状态。若在图G中存在从结点V1到结点V2的路径，则称G中V1可达V2，记为access(V1,V2)。在G中V可达的所有结点集合称为V的下游，记为downstream$(V,G)=\{V'|access(V,V')\}$，其中V′称为V的下游结点。

在数据驱动方式下，当某结点所有入端口数据到达后，该结点就被自动点火执行，执行完后，其出端口数据由管道传递到所有依赖该结点的入端口。数据驱动方式下的流图运行：用户直接将某一结点 V1点火，系统运行此结点，并由此结点发出数据流向其下游结点传播，下游结点一旦成为可运行结点就立即运行。其请求运行结点集为 downstream(V1,G)∪{V1}。各结点的运行顺序完全由其何时成为可运行结点所决定。若两结点之间互不可达，即 access(V1,V2)与 access(V2,V1)均不成立，则 V1,V2可以并行。

功能分布与控制并行较直观地反映了数据流模型的特点，但并不能解决大计算量构件成为执行瓶颈、大型数据集限制应用规模这两大主要问题。为此，我们在流图一级引入数据分解与组合操作，实现了构件间的数据并行处理。

当实现构件内的并行时，数据分解限于构件内部，即构件本身根据并行计算或并行编程的需要，对数据进行适当划分，并在处理完成后重新组合，形成单个数据集传给下游构件，如图3所示。显然，构件间的数据传送仍是大粒度的数据集。

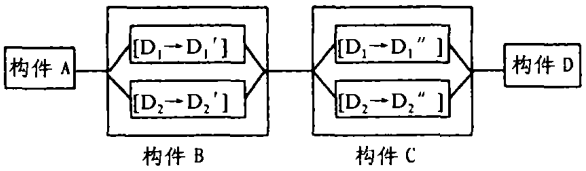

图3　构件内的数据并行

当实现构件间的并行时，数据分解与组合在流图一级由专门的构件来执行。实现大计算量构件并行处理的各个并行构件(即子任务)被提升到流图一级，成为新的并行构件。这些并行构件接收数据子集作为输入，同时输出新的数据子集，因而并行构件与流图中的其它构件之间已不再是大粒度数据流传送，而是各个数据子集。完成同一功能的各并行构件组成任务组，不同任务组的构件之间可实现并行数据通信，图4显示了构件间的数据并行。显然，构件间的数据传送已是细粒度的数据子集，而不再是整个数据集。

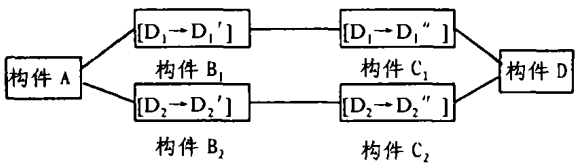

图4　构件间的数据并行

数据并行是提高大计算量构件执行速度的有效途径，但构件内的数据并行需要在各并行构件中对数据集进行分解与合成，不仅增加了开销，而且使构件之间存在通信瓶颈。构件间的数据并行在流图一级实现数据分解与组合，不仅可减少数据分解与组合的次数，而且可提供并行构件间的并行通信机制，消除了构件间的通信瓶颈；同时，细粒度数据流的形成提高了单个数据集的处理规模。

4. 元计算环境的体系结构

当前，Globus 等系统只提供了元计算系统内支持用户使用各种软硬件资源的工具和一些子系统，缺乏系统的统一视图，以牺牲用户的学习和开发为代价提高了系统的灵活性，没有从用户的思维模型出发，用户仍然感觉是在各个计算机系统内进行应用程序的开发，并且要了解相当一部分系统底层和网络技术的特性。

同时，很多应用程序需要在不同体系结构的机器上运行，而面向对象语言被认为可以用于减小这种应用程序的开发难度。比如：Charm[20]，Mentat[21]，都以面向对象的并行编程环境来支持可移植软件的开发。与此类似，Dome[22]项目利用 C++对象库使得我们能够在一个异构多用户环境下实现并行编程。它们的共同缺点就是都必须使用特定的编程语言。特别是对一些工业应用环境来说，它们原有的大量代码不能被移植到别的语言环境，这时这个缺点就显得尤为明显。

因此，我们以构件技术为基础，针对上述缺陷，从用户思维模型的角度出发，给出了元计算环境的框架结构。用户可以透明地提交并行任务，而无需知道执行这些并行任务的机器及通讯的具体现实和使用细节。简单地说，软件构件是具有确定的公共接口、能够独立地被调用的可执行的程序体。构件框架是由一组满足一定机制的构件组成，这些机制用于协调框架内各个构件的通讯来完成特定的任务。

图5是我们的元计算系统体系结构模型，其客户端用 Java Applets 实现，使用 CORBA 与远程服务器软件进行高效率交互，并协调和控制对计算资源的访问。用户透明地从 WWW 上下载所需的 Applets，Applets 依靠浏览器的 Java 运行时间库运行并呈现一个 GUI，用户通过 GUI 向服务器提交任务，服务器端的中间件为这些任务分配资源并控制任务在计算资源上的执行，任务执行完后由服务器把结果回送到客户端并通过 GUI 显示。

由于元计算环境中的各个计算资源是异构的，建立这样的环境需要中间件系统来支持和协调具有不同操作系统、不同计算平台下的任务之间的交互，下面我们就这样的中间层管理系统(采用类似于 Globus 的机制)进行详细的描述。

在广域元计算环境中 AS(Administrator System)负责连接各个独立的计算资源结点。客户向 AS 提交任务后，由 AS 负责为任务分派计算资源来实现负载共享。

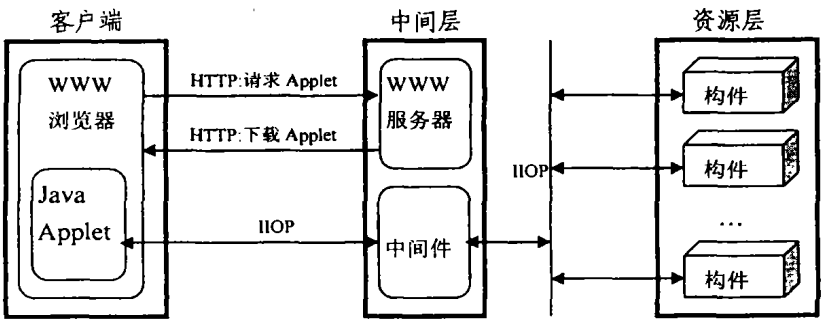

图5　元计算系统体系结构

AS的功能包括构件管理、任务管理、资源管理以及如何调度任务在计算资源上的执行。AS采用分层结构，包括GAS(Global Administrator System)和LAS(Local Administrator System)。GAS负责接收客户提交的任务请求并将这些任务分派给LAS，由LAS来负责任务的调度、资源的管理。分布的异构资源被分成相互独立的集合，每一集合中的资源由LAS来管理。另外，我们提供分布的GAS服务，元计算机可以被分成几个区域，每个区域都有一个GAS负责管理。每个GAS负责管理本区域的LAS们，并可以把自己看作其他的GAS的一个LAS。

AS管理有两个重要实体：计算资源和任务，计算资源为AS提供计算服务，任务向AS请求计算服务。

任务分为元任务和复合任务，元任务就是不包含任何其他子任务的任务，而复合任务是由其他一些子任务相互作用为实现一个完整功能而复合成的任务。任务被看作是由各子任务通过相互之间的依赖关系构成的无环有向图(任务图)。引入复合任务后，任务图不再是简单的平面结构图，而是一种层次结构图。

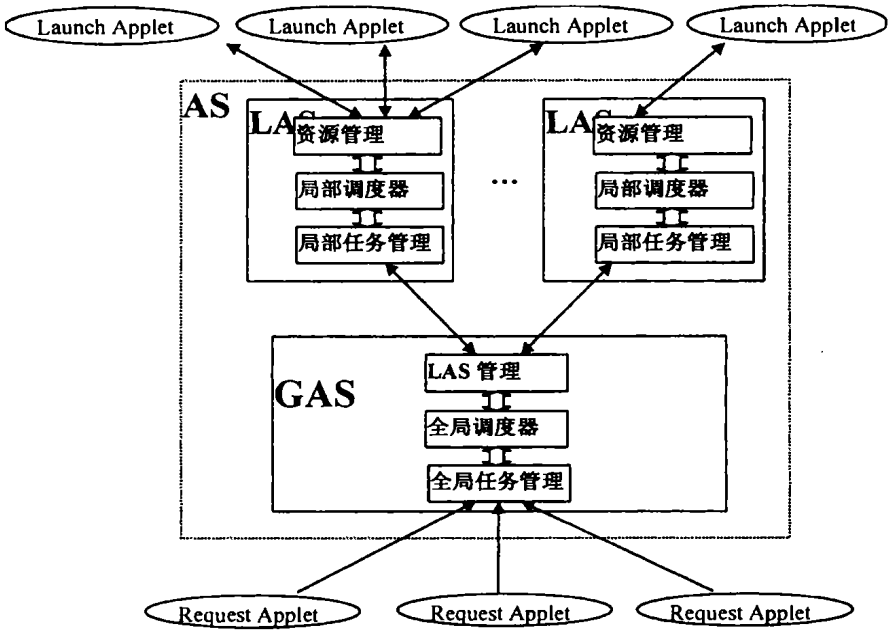

图6 管理系统构成

为了能有效地让应用程序构件在计算资源上运行，我们需要为应用程序构件匹配计算资源。我们在构件封装时，由构件开发人员为构件提供相应的执行环境需求(代码构件实现使用的编程语言、代码构件的操作系统平台、并行程序设计标准、节点计算能力、节点间的通信能力等)；计算资源在加入元计算系统时，也要提供类似的属性，为应用程序构件和计算资源的匹配提供了依据。为了实现上述功能，AS必须包括四个功能模块如图6：资源管理用于将分散的计算资源组织起来为客户通过集中的计算服务；任务管理用于管理用户提交的任务；构件管理用于管理任务的构成元素；调度管理用于为任务分配资源并控制任务的执行。

任务管理的功能是组织和管理整个系统中所有用户提交的任务。用户通过浏览器中需求Applet来提交任务，需求Applet负责把用户提交的任务发送给服务器端的任务管理器。任务管理器接到用户的请求时，首先根据用户身份验证其操作的合法性，若合法则注册该任务并返回一个任务句柄给用户。这里的任务管理分两步，首先由GAS的全局任务管理器根据LAS的负载情况将任务调度到合适的LAS，然后LAS再对任务进行动态调度。

具体的任务调度和管理情况请参考文[23]，而资源层和资源管理/调度的具体情况请参考文[24]。

5. 客户端

客户端采用数据流模型为体系结构，在数据流模型中引入了控制结点以实现循环和分支功能，引入了IF-THEN-ELSE，WHILE-LOOP等控制结构，并采用数据驱动的运行方式提供交互方便的可视编程界面。用户能够以"搭积木"的模块级编程方式从构件库中选取所需的可视化构件，直接构造其可视化应用实例，无须编写代码。

5.1 客户端的数据流模型

基本思想是将一个任务划分成相互独立的计算构件，每个构件看作是带有输入口和输出口的黑匣子，每个构件在接收到数据以后才开始执行，然后把结果向下传输，各构件之间依靠数据传递保持联系。任务的实现可以利用构件间的数据依赖关系图来表示，有时也称为数据流网络(dataflownetwork)或流图(flowgraph)。

客户端通过定义下面一些概念来描述和实现它的数据流控制结构：

·单元 它是一个相对独立的处理构件，是构造数据流图的基本实体。每个单元具有相应的数据输入/输出口，它们分别定义了相应构件的输入/输出参数和变量。客户端为用户提供了两大类单元：计算单元和控制单元。

·管道线 它把一个单元的输出口和另一个单元的输入口连接起来，负责把上一级单元处理后的数据传递到下一级单元去处理，从而实现单元间数据的有效流动。

·流图 单元和管道线是流图的最基本组成。它是一个数据处理的拓扑结构，整个流图就是一个具体的可视化应用。通过引入条件、循环等控制结构，平台支持用户实现流图的各种(从简单到复杂)拓扑结构。这是客户端相对于其它同类平台具有的特色之一。

·驱动方式 当构件的输入数据全部到达时，构件就被点火执行，其结果沿构件输出口依次向下传递，形成系列点火执行顺序。

客户端的数据流机制为用户提供了一种模块级的交互编程环境，它为平台的扩充提供了一种良好的结构，这也是现有其它一些封闭系统所不能比拟的。

5.2 可视编程平台

客户端将可视编程思想和数据流机制相结合，并引入IF-THEN-ELSE，WHILE-LOOP 等控制结构，形成平台的可视编程语言，其语法特征为：

·应用领域 为用户提供面向不同应用领域的构件，用户根据具体的应用需要选择领域构件。

·图形元素 为用户提供了一个可构造、编辑、执行流图的工作环境，称为流图编辑器。在该环境中，图符是表示流图的最基本实体，图符间的图形连线表示管道线。

·层次结构 为了应付大规模构造并行应用程序的需要，我们把完成一定功能的流图用复合构件来表示，这样简化了流图，需要时可以把复合构件再展开，通过复合构件支持流图的层次结构化。这样的层次结构模型的嵌套层次可达任意层。白色方框代表元构件，灰色方框代表复合构件，它可以分解，分解视图为虚线箭头所指。图7为两层结构示意图，在此基础上可进一步嵌套。

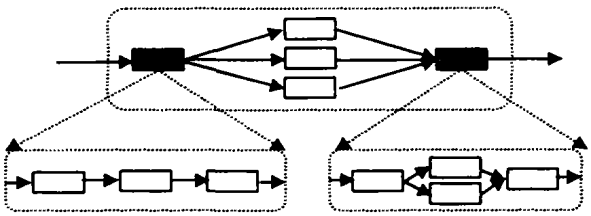

图7 复合构件支持流图的拓扑结构

5.3 构件库

客户端构件库包括应用构件库和控制构件库两部分。前者包括用于不同应用领域处理的各种构件；后者包括用于流图控制结构的构件。从用户的角度看，构件是一个图符界面，它主要包括四个层次：主框架模板、子模板、子模板对应的图符、从图符获取模块的控制面板。构件界面的层次化，可以有效地帮助用户快捷地使用构件，从而开发出可视化应用程序。从设计者的角度看，构件主要由三部分组成：1. 控制板，它是构件的图形界面，由标题说明、输入/输出口及 Widget 组成，每个 Widget 对应于构件的一个控制参数；2. 构件封装，它实现外部控制和构件内核间的数据传递；3. 构件内核，它是相应的程序实现，不同的构件对应于不同的算法。

结论 本文提出的基于构件的数据流模型中，对于一般的并行应用可以采用构件内的数据并行性和构件间的控制并行性相结合，对于大计算量的应用引进了构件间的数据并行处理技术。开发人员可直接使用构件象开发顺序程序一样来构造并行应用程序而不必涉及具体的并行实现细节，从而可简化并行程序开发过程，缩短并行应用软件开发周期。

在实现上，我们借助 Globus 工具箱来实现环境的资源管理、数据通信、安全认证等基本功能服务。Globus 是由美国多所大学合作研究的一个致力于构建计算网格的项目，它提供了让应用程序将分布式异构计算资源作为单个虚拟机器的软件底层结构，其中心元素是定义了构建计算网格的基本服务和功能的 Globus 元计算工具箱(GMT)，工具箱包括实现了安全、资源分配、资源管理、数据管理、资源保留和通信等基本服务的组件的集合，特定工具和应用程序的开发者可以根据各自的需要从中选择相应的服务。Globus 以层次结构构建，强调网格组件与其服务的层次合成，高层服务可以使用底层核心服务来开发。使用 Globus 服务的资源管理系统包括 Nimrod，NetSolve，AppLeS 和 Condor 等。Globus 工具箱在我们环境中的具体使用情况可参见[23,24]。

参 考 文 献

1 Smarr L, Catlett C E. Metacomputing, Commun. ACM, 1992, 35(6): 45～52

2 Geist G A, et al. PVM: Parallel Virtual Machine-A User's Guide and Tutorial for Networked Parallel Computing, The MIT Press, Cambridge, MA, 1994

3 MPI forum. Message-passing interface standard 1.0 and 2.0. Available at: WWW. mcs. anl. gov/mpi. index. html

4 Condor. Available at: WWW. cs. wisc. edu/condor

5 Condine. Available at: WWW. gridware. de/products/

6 LSF, Product Overview. Available at: WWW. platform. com/products/

7 Burns G, Daoud R, Vaigl J. LAM: An open cluster environment for MPI, in: Supercomputing Symposium'94, Toronto, Canada, June 1994

8 Baraglia R, et al. Experience with a wide area network metacomputing management tool using IBM SP-2 parallel systems, in: Concurrency: Practice and Experience, vol. 8, Wiley, New York, 1996

9 Brune M, Gehring J, Reomefeld A. Heterogeneous message passing and a link to resource management. Journal of Supercomputing, 1997, 11: 355～369

10 Tandiary F, et al. Utilizing idle workstations for largescale computing. IEEE Parallel and Distributed Technics, 1996. 41～48

11 Carriero N, et al. Adaptive parallelism and Piranha. IEEE Computer, 1995, 28 (1): 40～49

12 Polder. http://WWW. wins. uva. nl/projects/polder/

13 Abramson D, et al. A tool for performing parametised simulations using distributed workstations. The Fourth IEEE Symposium on High Performance Distributed Computing, Virginia, 1995

14 Berman F, et al, Application-level scheduling on distributed heterogeneous networks, Supercomputing, Nov. 1996

15 Foster I, et al. A Metacomputing Infrastructure Toolkit. Journal of Supercomputer Applications, 1997, 11: 115～128

16 Object Management Group. The Common Object Request Broker: Architecture and Specification, Revision 2.2, Feb. 1998

17 Adams J, et al. The Fortran 90 Handbook, McGraw-Hill, 1992

18 HIGH PERFORMANCE FORTRAN FORUM. High Performance Fortran language specification, version 1.0: [Tech. Rep. CRPCTR92225]. Center for Research on Parallel Computation, Rice University, May 1993

19 OpenMP Standards Board. OpenMP: A Proposed Industry Standard API for Shared Memory Programming, 1997

20 Kale L, Krishnan S. CHARM^{++}: A portable concurrent object oriented system based on C^{++}, in: Conference on Object Oriente Programming, Languages and Application (OOPSLA), 1993, 28: 91～108

21 Grimshaw A, et al. Metasystems: An approach combining parallel processing and heterogeneous distributed computing system. Journal of Parallel and Distributed Computing, 1994, 21: 257～270

22 Arabe J, et al. Parallel programming environment in a heterogeneous mutil-user environment, Supercomputing, 1995

23 王东奎. 基于构件技术的元计算环境框架 MefBC: [南京大学硕士毕业论文]. 2001

24 杨海荣. 元计算及其资源管理的研究: [南京大学硕士毕业论文]. 2001