基于 MPI 的并行有限元计算集群的构建*1)

李海江

(大连海事大学道路与桥梁工程研究所 116026)

MPI BASED PC CLUSTER DISTRIBUTED FINITE ELEMENT ANALYSIS

Li Haijiang

(*Department of Engineering Mechanics Dalian University of Technology, Dalian* 116024 *P.R.China*)

Abstract

With the emerging of huge challenging problems, single processor computer will not satisfy this requirement for the fundamental limitations of the silicon technology. Cluster based architecture, with its good cost/performance ratio, is becoming more and more popular in universities and research branches as an alternative to expensive parallel machines. Whilst MPI, one of the successful high performance message passing models, is considered the future standard. This paper describes a MPI based PC cluster distributed FEA system with emphasis on the building procedure and frame components. Such a system can make full use of kinds of advanced parallel and distributed computing tools nowadays and serve as a good base for the future grid computing.

Key words: Cluster, MPI, Parallel, FEA, OOP

§1. 引　　言

随着工程数值计算所分析问题规模的不断扩大及工程复杂性的日益提高，对计算机计算能力的要求也越来越高，而硅芯片单个 CPU 计算机因其较低量级的物理极限的存在也越来越无法满足需要. 在这种背景下，并行计算机也就应运而生. 另一方面，计算机计算能力的飞速发展，也促使更多更有效的数值分析算法的出现，并直接推动数值分析向更加复杂的方向发展. 串行的有限元分析方法对于高度复杂、海量数据的系统分析越来越力不从心，因而并行有限元分析方法也就随着并行机的发展而逐渐走向前台. 早期的并行有限元分析是非常困难和复杂的，并行算法的研究只能局限于某种特定的并行机，这样研究人员就需要花费太多的精力在与硬件相关的底层问题上，例如可移植性、通信效率等，这是影响并行有限元分析方法推广的主要瓶颈所在. 近年来，基于消息传递的并行编程模型为此提供了一个崭新的

* 2003 年 1 月 27 日收到.

1) 本课题得到国家 973 项目“大规模科学计算研究”的部分资助.

解决方案，其中，MPI 和 PVM 是两种最成功的范例，几乎所有类型的计算机，包括各种并行机，工作站及 PC 等，都有实现它们的版本. 新生的 MPI 更具有上百个传递函数和丰富的点对点通信机制，并被认为是将来的行业标准. 并行分析的平台可以是高性能并行计算机或集群，前者因为昂贵的价格、复杂的日常维护及高度硬件相关性致使无法大规模的应用，而集群计算则以优越的性价比越来越成为高校、科研院所甚至是企业界进行大规模科学计算的首选. 例如 NASA 的 Beowulf 项目 [1]，其成功的应用证明了运行着自由软件、通过快速连接设备连接的 PC 集群完全可以达到与高性能并行机相媲美的计算性能. 构建 PC 集群来进行并行有限元分析，不仅能以很少的投入充分利用现今各种先进的并行计算工具，同时也能为未来有限元网格计算的应用奠定一定的基础.

§2. Beowulf 集群的飞速发展

在众多的集群类型中，建立在自由软件基础之上的 Beowulf 集群的发展速度是相当快的. 1994 年，世界上第一个 Beowulf 集群 Wiglaf 被建立起来，它是由 16 个 66MHZ (很快被替换成 100MHZ)Intel 80486 处理器构成，对于某种特定计算问题，Wiglaf 系统每个节点的稳定运算速度达到 4.6Mflops，总体的运算速度达到 74Mflops. 第二个 Beowulf 集群是 1995 年完成的 Hrothgar，这个基于 Pentium 系列 CPU 和快速 Ethernet 网卡的 16 节点集群系统，最终达到 280Mflops 的运算速度. 到 1996 年底，第三代 Beowulf 集群完成，它们是建立在 PentiumPro 处理器的基础之上，其总体运行速度达到 Gflops 级别. 在 1996 年度的超机计算演示中，一个 32 节点的 Beowulf 计算集群在没有做任何特定优化的前提下，系统的总体运行速度达到 2Gflops，正因为如此出色的性价比一举获得 1997 年度的 Gordon Bell Prize. 在随后的 1997 年中，更多的计算节点被集成进这样的集群系统，当系统集成了上百个处理器时其稳定的运算速度超过 10Gflops，这样优异的高速计算性能逐渐引起各并行计算机制造商的密切关注，于是在 1998 年，基于 Beowulf 集群技术的 DEC Alpha 系列多处理机达到了 48Gflops 的稳定运算速度. 在 1998 年 11 月份的世界 Top500[2] 计算机排行中名列第 113 位. 从那以后到 2002 年短短 4 年的时间，随着新技术、新产品的层出不穷，集群技术得到了空前飞速的发展，在世界高性能计算机排行榜上所占据的份额越来越大. 世界各国的许多研究人员很快发觉这是一个很现实的实现超级计算的方案，一个 Beowulf 团体很快在全世界范围内自发地形成. 在美国、英国、澳大利亚、意大利、墨西哥、瑞士、捷克、德国、泰国、芬兰、挪威以及中国等国家中，以 Beowulf 思想构成的系统相继建成. 同时越来越多的商业企业争相踏足此领域又为集群的发展注入更充足的资金，在未来几年，集群的发展势头会依然强劲，传统的并行机将逐渐淡出 [3].

我国的并行计算研究尽管起步晚于国外，但在随后的并行算法研究中发展很快曾一度不断的缩小了与国外的差距 [4]. 不过早期的并行计算研究必须结合于特定的并行机，而并行机昂贵的价格令国内的大学及科研单位望而却步，而且超高性能的并行计算机一直都属于国外的高科技禁运产品，这样逐渐导致我国的并行计算研究没有依托平台，研究的人员越来越少，与国外的差距也越来越大。近几年来，随着我国经济实力特别是科研实力的增长，并行计算机的硬件研制突飞猛进，银河、神威以及曙光系列，特别是联想 2002 年的深腾 1800 集

群，其运算速度超过万亿次每秒 (1Teraflops)，已经昂首挺进世界超机计算机 Top500 中的第 23 位. 计算机集群无疑会是普及并行计算技术的有力工具，尤其在我国，许多从事科学工程计算的人员对并行处理还是陌生的，而许多搞计算机并行处理研究的人也没有并行计算的实际经验. 集群的出现，使一个部门甚至一个课题组都有可能拥有一个并行计算系统，所以说，基于自由软件的 Beowulf 集群第一次使国内的科研工作者在科研平台上有了与国外同行相同的起跑线. 同时要注意到，集群研究在我国还刚刚起步并且多集中在计算机技术领域 [5]，对于用其来进行并行有限元分析并应用到土木工程领域内的相关报道国内还不多见. 另外，对于像联想深腾 (集成了 256 个计算节点) 这样大型的、尖端的计算集群相对而言还是不能被多数研究机构广泛的应用，但集群的概念非常灵活，可大可小，对于一个小型的 8 节点或 16 节点的计算集群可以非常方便的在许多大学或科研院所进行集成并应用. 尤其令人注目的是，构成计算节点的单个计算机不必是当前最新的计算机，多个稍低档次的计算节点完全可以形成令人羡慕的高质、高速的集群并行计算机. 利用集群来进行并行计算，可以将注意力集中在应用问题上，按应用需求考虑系统配置 (处理器、主板、 Cache 、内存、磁盘、网络)，开发专门的并行应用程序，为该应用程序的优化尽一切努力. 在这个意义上，一台 Beowulf 基本上就是一台专用机.

§3. 特定并行有限元计算集群的构建

Beowulf 系统往往都是直接针对一个或一类应用问题求解，因而依托其开发并行程序设计模型通常是很有针对性的. 特定并行有限元计算集群的构建过程是有其内在规律的，如何根据应用需求配置系统，例如 CPU 、主板、快存和主存、硬盘、网络互联方式等，并且在允许的经费范围内进行权衡来构造 PCs 集群. 如果没有一套系统的方法，其最终的程序执行效率会非常低下，特别随着研究问题的扩大集群的规模也扩大时，集群组集的复杂程度也会成倍增加 [6]. 根据 ParaFEA 的程序框架，结合作者构建集群的经验将集群具体的实现过程分为系统分析、总体设计、节点安装和系统测试等几个阶段. 每个阶段都是设计与修改不断交互进行的循环过程，当发现效率或稳定性不尽如人意时，就要根据具体的组集报告来修改配置，之后再测试直到符合要求为止.

3.1 系统分析

计算机集群研究的一个侧重点在于如何减少结点机间的通信开销 [7]. 目前主要是从以下两方面着手：一是使用新的高速网；二是设计新的精简通信协议，减少传统通信协议的层次，以减小通信开销. 集群系统一般使用通用局域网连接，目前常用的局域网技术大体可以分成两类：一类是共享介质网络，另一类是开关网络. 对于共享介质网络，由于其聚合网络频带与单独链路频带是一样的，其性能会随网络负载的增加而下降，特别是对于某些负载比较集中的应用程序，这种影响会更明显，但是售价便宜，组成系统也相对容易，是组成中低档集群系统的一种较好的选择. 开关网络则相反，其聚合网络频带比单独的链路频率带要高得多，理论上讲是 N 倍. 除开关的交换延迟影响外，性能不会随网络负载的增加而降低很多. 开关网络的另一个优点是其可扩展性较好，交换延迟已经很低，与发送接收端的开销相比要小得多. 但是交换开关及相应接口卡的售价要高得多，组成集群系统的价格相对也比较高，

对系统的普及会受到一定影响. 新的网络技术大幅度地提高了传输速度, 目前在尖端科学计算领域中, 有许多专为计算集群设计的网络接口卡, 如 Myrinet, cLAN 和 SCI. 这些卡不但在集群的节点之间提供高带宽, 而且还减少延迟. Myrinet 单向互连速度最高可达到 1.28 Gbps, 两个直接连接的节点之间的平均延迟是 5 到 18 微秒, 比以太网快得多. cLAN 当前可以在节点之间提供 1 Gbps 单向通信, 最小延迟为 7 微秒. IEEE 标准 SCI 的延迟更少 (低于 2.5 微秒), 并且其单向速度可达到 400 MB/ 秒 (3.2 Gbps). 以此看来, 以往集群计算低带宽、高延迟的硬件瓶颈已经被突破, 但同时应注意到, 目前数据通信的限制因素并不是操作系统或网络接口, 而是服务器的内部 PCI 总线系统. 台式 PC 通常有基本 32 位 33-MHz PCI, 大多低端服务器也只能提供 133 MB/ 秒 (1Gbps) 的通信速度. 计算机存储结构是分级的, 最顶级是寄存器, 也是数据运行最快的部分, 下一级是多级缓存 (集成在主板上或在 CPU 中), 接着是局部内存、其他单元的内存, 硬盘等. 在不同的级别上数据传输的开销远远超过同级运行, 所以在算法的设计上 (越粗越好), 在域分解的过程中应考虑这些因素. 图 1 描述了目前几种网络连接设备的性能比较.

	Gigabit Ethernet	Giganet	Myrinet	Qsnet	SCI	ServerNet 2
MPI Bandwidth (MBytes/s)	35–50	105	140	208	80	65
MPI Latency (μs)	100–200	20–40	–18	5	6	20.2
List price per port	$ 1.5K	$ 1.5K	$ 1.5K	$ 3.5K	$ 1.5K	$ 1.5K
Maximum # nodes	1000's	1000's	1000's	1000's	1000's	64K
VIA support	NT/Linux	NT/Linux	Over GM	None	Software	In hardware
MPI support	MPICH over MVIA TCP	3rd party	3rd party	Quadrics/ Compaq	3rd party	Compaq/ 3rd party

图 1　几种网络连接设备的性能比较

有效地管理系统中的所有资源是集群系统研究的另一个重要方面, 常用的并行编程环境 PVM, MPI 等对这方面的支持都比较弱, 仅提供统一的虚拟机. 主要原因是结点的操作系统是单机系统, 不提供全局服务支持, 同时也缺少有效的全局共享方法. 考虑到 ParaFEA 具体的并行有限元实现是从域分解 [8] 开始的, 这一部分工作很繁琐目前不易集成进并行系统中, 另外还有一些初始化工作在整个分析过程中只占用很少一部分 CPU 时间, 这些工作可以利用较成熟的串行分析方法. 为了对系统资源统一管理, 在集群启动时, 应该由一主节点负责启动其它各从节点上的守护进程, 利用这些守护进程提供的信息动态地生成系统资源列表并负责整个系统的监控, 所以该集群从物理构成上讲应该具有不同功能的主、从节点. 为了降低系统的复杂度, 各对等从节点的计算能力最好相同, 即软、硬件的配置应该相同. 由于系统追求的目标并不是谋求具有分析巨大挑战性问题的能力, 这样的集群在性能上有所折衷, 因而并不要求具有尖端性能的硬件组件 (由此带来预算成本的大幅增加). 考虑到系统

面向应用的特性，应充分集成现有的各种成熟的计算及管理工具，在此基础之上再进行特定目标的创造性开发. 对于基于消息传递的并行有限元分析，不管算法的粗细，节点之间的通信速度会是影响系统性能的重要因素 (特别是对于一些粒度较细的分析算法)，所以对于以太网卡和交换机应该在经济条件许可的前提下尽可能越快越好，对于节点本身的计算能力则可以适当放松. 在系统分析结束时，应同时形成思路清晰的需求报告.

3.2 总体设计

个人电脑市场在商业化的全面推动下，在过去十年内的飞速发展 (摩尔定律) 使得低价位高性能的软硬件组件越来越容易获得，以台式 PC 为节点来构建集群不失为一种行之有效的方法. 计算机如此之快的发展速度，使集群的系统设计及维护具有一定的繁琐性及复杂度. 例如在设计初期，哪些组件应该怎样被集成进集群中，维护期间，由于系统需要不断的更新和升级，如何维持系统列表才能使系统不会被很快淘汰，如何组织系统结构才能使系统具有持续不断的升级能力. 在操作系统技术方面，如何决定 Swap 区的大小、网卡 Driver 的选择和开发，以及怎么能够让 BIOS 在查不到显示卡的情形下也能正常工作，是否用 NFS, NIS 等等一系列的问题都要求有一种全面的、系统的方法.

面向对象作为一种软件设计方法 [9]，过去的一段时间内，在降低软件系统设计的复杂度，维持软件系统的稳定性等方面已经显示出巨大的优越性. 利用面向对象的技巧来设计集群，将面向对象的概念引入软、硬件系统的构造中，主要的出发点是将所有的节点计算机及相关的硬件设施，还有各种欲组集和开发的软件系统都当成一系列对象. 将原来纷繁复杂的各种软、硬件之间的关系描述为一系列独立的对象之间的相互关系. 对象有自己封装的数据，而独立的计算节点拥有一系列独有的硬件，这正是数据封装的自然体现. 不同系统模块之间的数据必须通过一定的手段 (MPI 通讯函数) 才可以存取. 利用继承的方法，由顶向下通过一系列相同的子集群来构造较大集群. 对于软件系统是采用 C++, Java 等面向对象的语言来进行开发. 在概念级和抽象层中引入面向对象的方法，可以极大地帮助开发者去正确理解集群系统的硬件构成，准确把握系统各构件之间的组成关系，另外还可以把整个系统用多幅虚拟的拓扑网格图表示出来. 例如系统结构图、硬件分类图、软件系统图等，并辅之以完备的系统组集和调试报告. 这里的调试报告和软件系统设计的程序文档同样重要，清晰、详尽会起到事半功倍的效果，这样集群的各个相关硬件、软件，系统结构，各个对象之间的联系都一目了然. 集群的设计应该按照循序渐进的原则，就像软件设计提倡通用易读性、系统性和层次性而摒弃个体技巧性一样，这也是现代生产的大规模化所提出的现实要求.

并行有限元计算不管采用主从模式还是对等模式，当系统的容错能力不甚完美时，各计算节点的计算稳定性则必须满足更为严格的要求. 对于 Windows 操作系统，虽然具有方便的界面操作，但对于系统通信而言，负载较大而影响效率. 现时的 Windows2000 和 WindowsXP 在稳定性上比其前期产品有很大提高，但仍然无法与 Linux 系列一较高下， Linux 操作系统可以很轻松的正常持续运行一年以上. Linux 系统是开放的系统，可以根据有限元分析的具体要求来修改操作系统内核，以强化某些通信功能，弱化某些与有限元分析无关的选项，从而提供给一个短小精悍、方便快捷、专门用途的高度定制的系统. Windows 系列从来都是暗箱操作，用户所做的可能会使系统越来越庞大，越来越复杂. 在封闭网络集群中 (信任

集合，可以方便利用 r 族协议进行远程通信)，与计算、计算通信相关的节点采用 Linux 系统，同时考虑到以往很多成熟的工具程序在 Windows 下开发，在不影响系统计算性能的前提下，也利用 Windows 系统来做一些辅助工作．根据以上的功能性分析并结合资金条件，采用四台 PIII500(由于 Pentium 系列 CPU 具有很好的性能，为谋求最大的性价比，将其超频至 600MHz, 测试结果表明系统仍具有令人满意的稳定性)，512M 内存的 PC 构成系统的四个对等从节点，安装 Linux 操作系统，用来执行并行计算．在操作系统安装完毕之后，对于键盘及监视器并无保留的必要，因为 Linux 系统具有优秀的远程管理功能，可以利用诸如 telnet, ssh, vnc 等一系列软件远程实现各种软硬件的修改配置．一台 PIII500, 1G 内存的 PC 作为物理主节点，采用 Windows 2000 操作系统，用来执行数据准备及系统控制．配备 100M 快速以太网卡和交换机用来互联，利用 Linux2.4.18 作为系统内核并根据需要重新编译，以允许结合几个以太网接口、高性能网络驱动器等，利用 MPICH, SSH, RSH(通信协议)，NFS(文件系统，用来建立无盘计算节点) 建立 MPI 消息传递系统，并利用 Java 开发监控软件分别运行于主节点和几个从结点．利用提供的信息在每次新任务启动之初自动形成 MPIRUN 所需要的硬件配置文件，因其自动将出错节点排除在外从而实现系统的部分容错能力．由于此部分通信在计算通信之前，从而并不影响系统的计算性能．对使用者来说，用户终端就是前述的主节点，用户完全感觉不到这是一个多计算节点的集群系统，而计算能力却已经数倍于原来的单机操作．在主节点提出问题，利用域分解工具将求解问题划分为一系列大约相同的子问题，利用通信软件将这些子问题的数据分布到各计算节点上，从而实现有限元问题的并行求解．这样的集群系统对于并行算法的可行性研究，并行有限元分析系统的构建都有很重要的现实意义，当然对于大规模的实际应用，则应在此基础上进行适当的扩充，图 2 是系统构成简图．

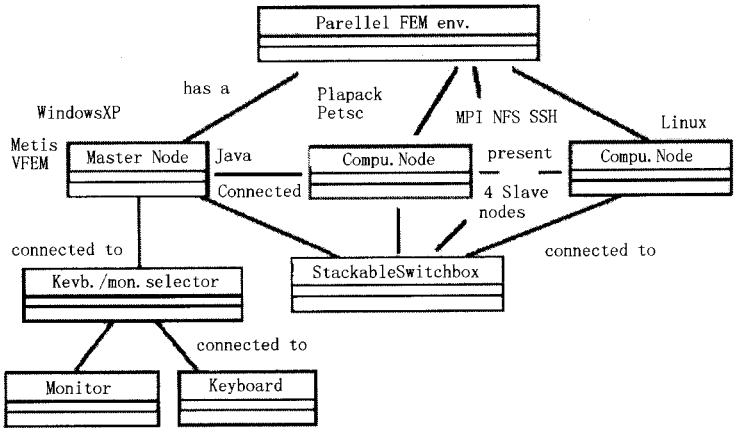

图 2　系统结构简图

3.3 节点安装

对于每个计算节点系统都利用 Lilo 实现了 Linux 与 Windows 的双重启动，系统的运行主要以 Linux[10] 为主．本系统选用 RedHat Linux7.2, 对于具体的 redhat 安装，请参考

相应文献 [11]. 安装后重新启动, 首先以 root 身份登陆并选择各种服务: 包括 crond, gpm, inet, keytable, kudzu, linuxconf, netfs, network, nfs, random, syslog 等. 之后需要修改一系列的相关文件以设置网络连接参数: /etc/hosts, /etc/resolv.conf, /etc/hosts.equiv, /etc/sysconfig/network, /etc/sysconfig/network-scripts/ifcfg-etho 等. 进而要设置网络文件系统 nfs, 修改相应文件 etc/exports 或直接利用工具软件 linuxconf, 利用 nfs 可以将主节点上的公共目录映射到每一个子计算节点. 还要设置用户账号, 安装 ssh 或者 rsh 作为系统相互通信的协议. 最后需要安装 MPICH 作为并行有限元分析的底层通讯模块, 以及 Petsc 作为并行有限元系统方程组的并行求解器. 为了系统管理的方便, 本系统还集成了一个 Beowulf 集群管理工具 SCMS, 其中有实时的网络监控软件, 并行的 Linux-shell 命令, 警报系统, 系列图形用户界面用来对系统进行管理, 而且还具有网络 WEB/VRML 操作界面, 使系统远程互联网控制成为可能. 图 3 是一些运行界面.

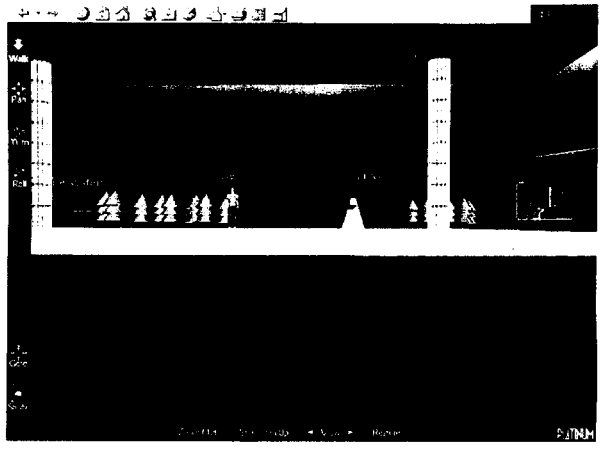

图 3 SCMS 的一些运行界面

3.4 消息传递系统 MPI 和 PVM

消息传递库 MPI[12] 是一种消息传递的标准, 而不是一种专门的计算机语言, 包括一个丰富的 MPI 函数库, 提供了上百个函数. MPI 定义了通信域的概念, 把消息上下文和任务组结合在一起来提供消息的安全性, 内部通信域允许一个组内的任务进行安全的消息传递, 外部通信域允许两个组的任务之间进行安全的消息传递. MPI 还提供了许多不同风格的阻塞和非阻塞的点对点通信机制, 并且提供了对结构化的缓冲区和派生的数据结构的支持. 并行虚拟机 PVM[13] 允许把一个有异构计算机构成的网络当作一个单一计算资源来使用, 包括监控程序、函数库、控制台三个部分. PVM 最大的优点在于它的可移植性和互操作性. PVM 还可动态改变虚拟机, 派生任务, 还提供了对容错和负载平衡的支持, 但 PVM 的性能不如 MPI. 总的说来, MPI 和 PVM 两种消息通信模型各有千秋, 前者为新生力量被誉为未来的行业标准, 后者则是开拓者并被认为是事实上的行业标准. 两个消息传递环境都经

历过几番更改，它们正在逐渐融合且之间的区别正在缩小. 考虑到手头的参考资料，同时认为性能对有限元分析来说更为重要，并且 MPI 具有更为丰富的点对点通信机制，最终选用 MPICH[14] 作为具体的实现版本.

3.5 系统测试

Linpack[15] 是一个用 Fortran 语言编写的线性代数软件包，主要用于求解线性方程和线性最小平方问题. 尽管其最初只是想提供一些常用计算方法的实现，但推出后的广泛采用，逐渐就为高性能并行机的性能比较提供了一个工具. Linpack 测试标准具体包括三个不同内容的测试：n=100 测试，n=1000 测试和对 n 没有限制的高度并行化计算测试. 由于本系统是面向并行有限元分析的特定集群，对于其运行效率可参考下述有限元算例分析.

§4. 并行有限元分析的系统组织结构

4.1 系统的数据准备

整个系统可以分为数据准备、有限元并行计算和结果处理三个部分. 当今的个人计算机速度已经达到 Gflops 级别，这样的运算速度对于域分解来说是很快的，对于百万节点的域分解工作，利用 Metis (Metis 是由 Minnesota 大学开发成功的优秀域分解软件包，用户可利用它将任意求解域分解为一系列子域) 可以在四十多秒内完成. 数据准备部分完全可以串行，这样就可以利用很多已有的优秀前处理工具. 由于以往的很多工作都是在 Windows 下开发的，这部分选用的操作系统是 Windows2000，即前述的物理主节点，并利用自主开发的可视化程序 VFEM 作为此部分的核心. 计算任务开始时，VFEM 首先将整个求解域进行有限单元网格剖分，而后利用 Metis 将求解域进行子域分解，Metis 输出的结果文件，再加上节点、单元数据一起作为 VFEM 核心计算的输入文件. VFEM 在整个系统启动之时，就利用驻留在各节点的 Java 通信程序提供的信息动态的生成服务节点队列，并为其统一编号 (此编号与 MPI 程序的进程号相对应). 运算之后，VFEM 形成一系列以进程号为文件名字的子域数据文件，并将它们分布到各个处理器作为各个处理器进行并行分析的输入文件.

4.2 核心并行有限元分析及算例测试

ParaFEA 总体上是利用面向对象方法编写而成的一个并行有限元分析模块，采用 Linux 系统进行互联，利用 MPI 进行消息传递. 在 ParaFEA 中同时留有多种开发方式的接口，一方面尝试将已有的串行程序进行并行化，以使最终形成的系统可以具有一定的实用性，所以 ParaFEA 集成了并行算法求解器 Petsc (Petsc 是美国阿岗国家实验室推出的免费的、具有源码的方程组并行求解器，已经在许多领域内得到应用)；另一方面系统还可以独自开发方程组并行计算模块，用来检验特定算法的效率以及系统的可行性，以下只以系统中集成的并行 CG 算法模块为例.

考虑二阶线性椭圆形偏微分方程的 Dirichlet 问题 [16]

$$-\underline{\nabla}\cdot(a(\underline{x})\underline{\nabla}u)=f(\underline{x}),\quad \text{在 } \Omega \text{ 内}, \tag{1}$$

$$u|_{\partial\Omega}=g(\underline{x}),\quad \text{在边界 } \Gamma \text{ 上}, \tag{2}$$

其中，$\Omega\subset R^2$ 为一有界开区域，并满足锥形条件，边界 Γ 逐段光滑. 将 Ω 分解为一系列不

重叠的三角形子域 T (I 个内部点， B 个边界点), 并在此子域上寻求 u 的分段有限元近似替代值 v

$$v = \sum_{m=1}^{I} v_m N_m(\underline{x}) + \sum_{m=I+1}^{I+B} v_m N_m(\underline{x}). \tag{3}$$

T 满足 $\bigcup_{m=1}^{p} T_m = T, |T_m| \approx |T|/p$(假定各处理器的处理能力相同), 并且 $T_m \bigcap T_n = \{\}$. 当 $m \neq n$, p 为处理器数， $N_m(x)$ 是 Ω 内的线性独立的连续函数序列, 称为试函数， v_m 是待定参数. v 满足 $\int_\Omega a\underline{\nabla} v \cdot \underline{\nabla} N_n d\underline{x} = \int_\Omega f N_n d\underline{x}$, 其矩阵表示为 $K\underline{v} = \underline{b}$, 其中 $\underline{v} = (v_1, \cdots, v_I)^T$, $K_{nm} = \int_\Omega a\underline{\nabla} N_n \cdot \underline{\nabla} N_m d\underline{x}$, $b_n = \int_\Omega f N_n d\underline{x} - \sum_{m=I+1}^{I+B} v_m \int_\Omega a\underline{\nabla} N_n \cdot \underline{\nabla} N_m d\underline{x}$.

假定结构的节点编号规则为内部节点先编号, 界面节点后编号, 则整体方程的系数矩阵就会形成为箭头状矩阵, 如图 4.

$$\begin{bmatrix} K_{11} & & & & & K_{1m} \\ & \ddots & & & & \vdots \\ & & K_{ii} & & & K_{im} \\ & & & \ddots & & \vdots \\ & & & & K_{nn} & K_{nm} \\ K_{m1} & \cdots & K_{mi} & \cdots & K_{mn} & K_{mm} \end{bmatrix} \begin{Bmatrix} \underline{v}_1 \\ \vdots \\ \underline{v}_i \\ \vdots \\ \underline{v}_n \\ \underline{v}_m \end{Bmatrix} = \begin{Bmatrix} \underline{b}_1 \\ \vdots \\ \underline{b}_i \\ \vdots \\ \underline{b}_n \\ \underline{b}_m \end{Bmatrix} \tag{4}$$

图 4 箭头状整体方程的系数矩阵

图 4 中， $\underline{v}_i$ 为内部节点位移 (i = 1,2,...,n), $\underline{v}_m$ 为子域边界节点位移， K_{ii}, $\underline{b}_i$, $\underline{v}_i$ 都可以同时、独立的在处理器 m 中进行运算并分布存储. 为了进一步提高效率, 还应对各子域进行静凝聚. 例如第 j 个子域的控制方程可写成

$$\begin{bmatrix} K^j_{mm} & K^j_{mi} \\ K^j_{im} & K^j_{ii} \end{bmatrix} \begin{Bmatrix} \underline{v}^j_m \\ \underline{v}^j_i \end{Bmatrix} = \begin{Bmatrix} \underline{b}^j_m \\ \underline{b}^j_i \end{Bmatrix}, \tag{5}$$

其中 m 下标代表界面点， i 下标代表内部点. 对每个子域做静凝聚可得

$$\begin{bmatrix} I & (K^j_{ii})^{-1}K^j_{im} \\ & K^j_{mm} - K_{mi}(K^j_{ii})^{-1}K_{im} \end{bmatrix} \begin{Bmatrix} \underline{v}^j_i \\ \underline{v}^j_m \end{Bmatrix} = \begin{Bmatrix} (K^j_{ii})^{-1}\underline{b}^j_i \\ \underline{b}^j_m - K^j_{mi}(K^j_{ii})^{-1}b^j_i \end{Bmatrix}. \tag{6}$$

再将静凝聚后各子域对总界面点的贡献组集可得

$$\left(\sum_{j=1}^{n} K^j_{mm} - \sum_{j=1}^{n} K^j_{mj}(K^j_{jj})^{-1}K^j_{jm}\right)\{\underline{v}_m\} = \sum_{j=1}^{n} b^j_m - \sum_{j=1}^{n} K^j_{mj}(K^j_{jj})^{-1}\underline{b}^j_j. \tag{7}$$

求解上述方程可得 $\underline{v}_m$ 并回带到方程 (6) 中即可求出各子域内部点的值 $\underline{v}^j_{jj}, j = 1, 2, \cdots n$

$$\underline{v}^j_{jj} = (K^j_{jj})^{-1}(\underline{b}^j_j - K^j_{jm}\underline{v}^j_m). \tag{8}$$

图 5 是程序中用到的并行 CG 算法 [17]:

1. $\underline{v}_i^0 = \underline{0}$;　$\underline{v}_{si}^0 = \underline{0}$	8. $r^k = Dot\,\mathrm{Pr}od\,(\underline{p}_i^k, \underline{p}_{si}^k; \underline{q}_i^k, \underline{q}_{si}^k)$
2. $\underline{b}_{si} = Fresh\,(\underline{b}_{si})$	9. $\alpha^k = \gamma^k / r^k$
3. $\underline{r}_i^0 = \underline{b}_i$;　$\underline{r}_{si}^0 = \underline{b}_{si}$	10. $\underline{v}_i^{k+1} = \underline{v}_i^k + \alpha^k \underline{p}_i^k$;　$\underline{v}_{si}^{k+1} = \underline{v}_{si}^k + \alpha^k \underline{p}_{si}^k$
4. $\underline{p}_i^0 = \underline{b}_i$;　$\underline{p}_{si}^0 = \underline{b}_{si}$	11. $\underline{r}_i^{k+1} = \underline{r}_i^k - \alpha^k \underline{q}_i^k$;　$\underline{r}_{si}^{k+1} = \underline{r}_{si}^k - \alpha^k \underline{q}_{si}^k$
5. $\gamma^0 = Dot\,\mathrm{Pr}od\,(\underline{r}_i^0, \underline{r}_{si}^0; \underline{r}_i^0, \underline{r}_{si}^0)$	12. $\gamma^{k+1} = Dot\,\mathrm{Pr}od\,(\underline{r}_i^{k+1}, \underline{r}_{si}^{k+1}, \underline{r}_i^{k+1}, \underline{r}_{si}^{k+1})$
$For\quad k = 0, 1, \ldots\quad Repeat\quad steps\quad 6-15$	13. $if\ (\sqrt{\gamma^{k+1}} \le tol)\quad END$
6. $\underline{q}_i^k = A_i \underline{p}_i^k + B_i \underline{p}_{si}^k$	14. $\beta^k = \gamma^{k+1} / \gamma^k$
7. $\underline{q}_{si}^k = Fresh\,(B_i^T \underline{p}_i^k + A_{si} \underline{p}_{si}^k)$	15. $\underline{p}_i^{k+1} = \underline{r}_i^{k+1} + \beta^k \underline{p}_i^k$;　$\underline{p}_{si}^{k+1} = \underline{r}_{si}^{k+1} + \beta^k \underline{p}_{si}^k$

图 5　并行 CG 算法

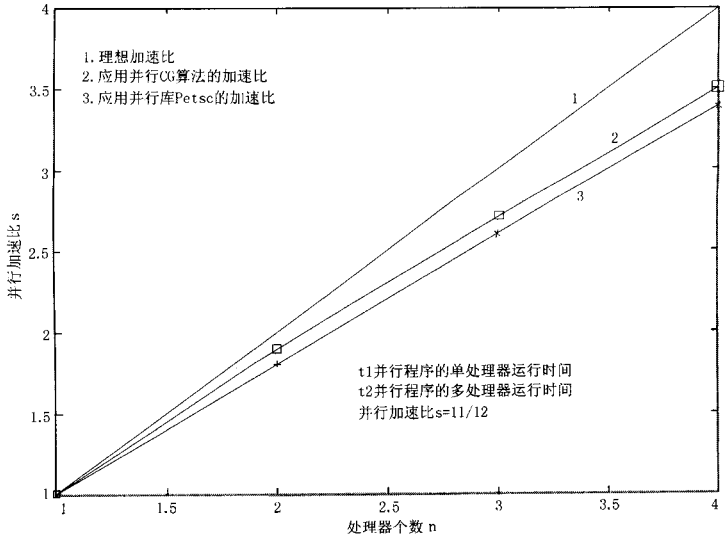

图 6　系统并行有限元分析算例

ParaFEA 中独立 CG 算法模块用到的两个主要数据结构是 NodeInf 和 ElementInf, 分别用来存储节点和单元的信息. 节点信息包括坐标 (x, y), 节点所属子域数 (0: 表示位 Dirichlet 边界条件, 1: 表示域内节点, >1: 表示多域共享节点), 局部坐标节点标号, 单元信息包括单元的三个节点坐标 (本例采用三节点三角形单元) 等. 程序结构采用对等模式, 在每个对等从节点上运行的程序都是相同的, 程序中利用条件语句来区分不同任务. 并行执行时, 各程序分别打开各自处理器上的数据输入文件, 读入节点、单元数据, 并分配好内存空间, 求解各自的 $K_{ii}, \underline{b}_i, \underline{b}_{im}$ 等. 而后直接调用 CG, CG 中包含两个函数, Fresh$(\cdots)$ 和 DotProd$(\cdots)$, 前者利用两个点对点的通信函数 MPI_Send$(\cdots)$, MPI_Recv$(\cdots)$ 和相邻的进程交换数据, 从而组集边界节点的各相关单元的贡献. 后者用来并行计算两个分布存储的

向量的内积，利用 MPI_Allreduce(···) 全局通信函数收集各进程的贡献求和并将此值散发到每一个进程， CG 结束之后，剩下的工作就是选择较好的方式将结果输出. 图 6 是利用上述并行 CG 算法以及方程组并行求解库 Petsc 实现的算例及其计算结果，将平面区域利用四节点等参单元划分为 10000 个单元，并依次分成 1, 2, 3, 4 个子域分配到各个计算节点并行求解. 计算结果表明系统的实现思路是完全可行的，同时较高的系统实现效率也很好的满足了性价比的要求.

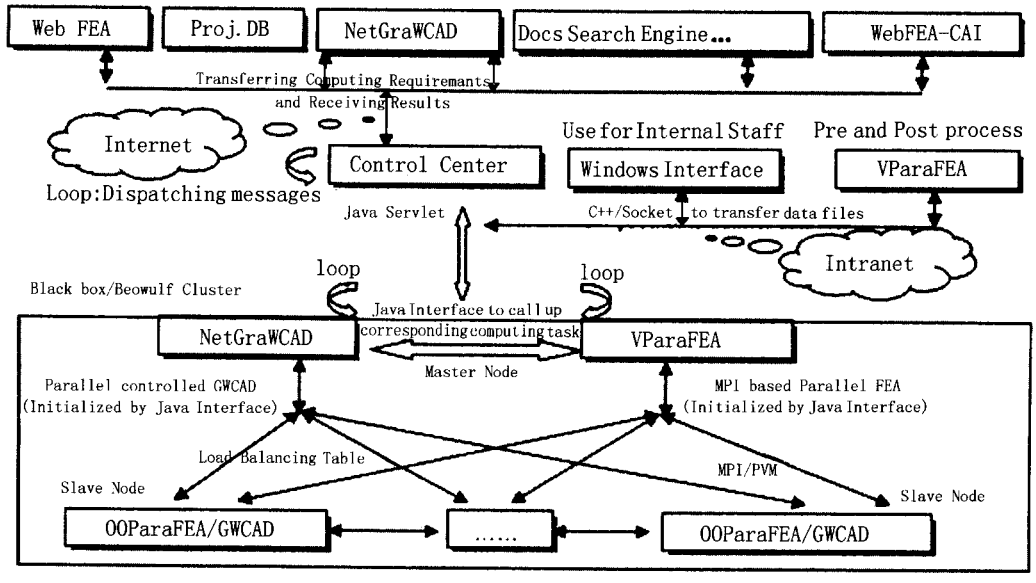

图 7 系统整体结构及计算流程

4.3 数据整理以及 Web 计算的集成

这部分用来整理数据并将数据可视化给用户，由于目前还没有形成并行可视化系统，仍然利用串行程序 VFEM 的后处理功能. 从结构上讲，此部分仍位于物理主节点上，所以除了可视化结果文件之外，还负责及时更新系统服务节点列表. 每当某一计算任务完成之后，VFEM 就会主动与各计算结点通信，及时去除崩溃节点、添加新的节点，这样一前一后动态生成服务器列表为系统实现了部分的动态配置及容错功能. 系统中还利用 Java 小程序开发了互联网应用客户端，以此将集群的计算能力推向互联网，从而将 Java/web 计算和后台的集群计算紧密的结合在一起. 互联网上的 Java 小程序提供了一种全新的软件使用方法. 当用户登陆到某一特定网站时， Java 小程序就会自动下载到使用者的电脑中，而且仅嵌入在网络浏览器中从而不会对用户的电脑做出任何破坏性行为，并自动地建立起与远程服务网站的通信联系. 计算资源是用户自己的，这样随着用户硬件资源的升级其计算能力也得到充分的利用. 同时，小程序是属于服务网站的，这样程序的升级再也不需要用户作任何事情，开发者负责更新软件，用户使用的总是最新的程序. 建立起拥有很强计算能力的集群之后，考虑到其并行调度能力，系统还将重力式码头结构 CAD 集成系统 GWCAD 移植到互联网，

在互联网上利用 Java 小程序来同步进行设计，这将给工程师带来前所未有的快捷性与方便性，并对未来工程设计软件的网络化发展提供一个思路. 对于整个系统的运行可用图 7 来表示.

§5. 结　　语

通过算例测试，此特定并行有限元计算集群系统具有很好的稳定性及效率，从而为日后的并行有限元分析提供了一个廉价而有力的基础计算平台. 同时未来的工作仍然很多，包括各部分的深入细化工作，比如充分利用各种并行库 (Petsc 等) 来加深、加大分析问题的难度并提高效率. 对系统的扩充，例如如何利用 Java 及 Internet 技术将系统整合成适合网格计算的新系统，如何充分利用现有的成功的串行有限元程序从而避免大规模的重复工作，如何利用面向对象的手段来设计计算核心，以减少新模块添加的复杂度. 采用何种方式如何将此系统推向工程实用等. 另外，集群这种高效并行计算平台的出现无疑给我国的高性能计算的快速发展注入充足的活力，在此之前，尽管我国也可以研制较高性能的并行机，但与国外的差距仍然很大，基于集群技术的联想深腾 1800 使国外对我国的高性能计算机的禁运行同虚设. 我国的众多并行有限元研究工作者也将因为集群技术而受益匪浅，尽管目前这方面的应用还不多，但随着时间的推移，最新集群技术在并行有限元研究中的应用会越来越广泛.

参 考 文 献

[1] The Beowulf Project, http://beowulf.gsfc.nasa.gov/

[2] TOP 500 Home Page. http://www.top500.org/

[3] Mark Baker. Cluster Computing White Paper. University of Portsmouth, UK. 2000.

[4] 张汝清，并行计算结构力学，重庆大学出版社，1993.

[5] 都志辉，高性能计算并行编程技术 -MPI 并行程序设计，清华大学出版社，2001.

[6] Thomas L. Sterling, etc. How to build a Beowulf, The MIT Press, London, 1999.

[7] Rajkumar Buyya. High Performance Cluster Computing Programming and Applications, Volume2. Prentice-Hall, Inc. 1999.

[8] 王希诚，结构优化设计的并行计算方法，吉林大学出版社，2001.

[9] Bruce Eckel, Thinking in C++. Prentice Hall Inc. 1998.

[10] David Tansley. Linux and Unix Shell Programming. Addsion Wesley Longman, Inc. 2000.

[11] Mohammed J. Kabir. RedHat Linux 7 Server. IDG Books Worldwide, Inc. 2001.

[12] Gropp, Lusk, and Skjellum, Using MPI: Portable Parallel Programming with the Message-Passing Interface, MIT. Press, 1999.

[13] PVM(Parallel Virtual Machine) http://www.csm.ornl.gov/pvm.

[14] MPICH http://www.mcs.anl.gov/mpi/mpich.

[15] J.J. Dongarra. The LINPACK Benchmark: An Explanation, Lecutre Notes in Computer Science, vol. 297, pp.456-474. Berlin: Springer, 1988.

[16] C. Johnson, Numerical Solution of Partial Differential Equations by the Finite Element Method, Cambridge University Press, 1987.

[17] D.C. Hodgson and P.K. Jimack. Efficient Mesh Partitioning for Parallel Elliptic Differential Equation Solvers, Comput. Sys. in Eng., 6 (1995), 1-12.