基于 MPI 机群环境下的广义逆力法并行化初探

王开健[1]，刘西拉[2]，顾 雷[3]

(1. 清华大学 土木工程系，北京 100084；2. 上海交通大学 土木工程系，上海 200030；3. 清华大学 计算机系，北京 100084)

摘要：广义逆力法是一种以力法为力学概念基础，以广义逆矩阵理论为数学理论基础，以迭代求解为求解方式的新算法。该法主要针对材料非线性问题，由于无需像传统的基于位移法的逐步增量法那样逐步递进计算，所以也称特大增量步算法。广义逆力算法的迭代过程可以分为整体阶段和局部阶段，分别可以进行时间上的并行计算和空间上的并行计算。这种可并行性是算法本身所天然蕴含的，不同于结构并行计算领域内传统的基于子结构的并行计算，是一种全新的并行计算思路。机群系统是随着微处理器技术和计算机互连网络技术的迅速发展而出现的一种并行计算系统，它正以其易扩展、易维护、易升级、价格低等优势迅速地扩大着自身的应用范围。基于消息传递方式的并行环境，做为一种目前事实上最流行和通用的并行环境，正在逐渐成为这种编程模型的代表和事实上的标准。广义逆力法在基于 MPI 机群环境下的初步并行化中有着良好的表现，从实践上印证了该算法的可并行性。严格控制计算时间和通讯时间的比例，合理分配各个计算节点的负载是获得较好并行效率的关键点。

关键词：岩土力学；并行计算；广义逆力法；机群；MPI

中图分类号：O 246；TU 311.4 **文献标识码：**A **文章编号：**1000－6915(2005)01－0057－09

PARALLELIZATION OF THE FORCE METHOD BASED ON GENERALIZED INVERSE MATRIX IN THE CLUSTER SYSTEM WITH MPI

WANG Kai-jian[1]，LIU Xi-la[2]，GU Lei[3]

(1. *Department of Civil Engineering*，*Tsinghua University*，*Beijing* 100084，*China*；

2. *Department of Civil Engineering*，*Shanghai Jiaotong University*，*Shanghai* 200030，*China*；

3. *Department of Computer Science and Technology*，*Tsinghua University*，*Beijing* 100084，*China*)

Abstract：Generalized inverse matrix(GIM) is a new force method based on the generalized inverse matrix theory，which is an iteration method for solving material nonlinear problems. Unlike the classical force method，GIM does not need to consider the basic structure，so it is also feasible for computer to calculate. This method brings the force method a new light in the computer calculation field. GIM has some natural parallel-calculating characteristics，which are different from the classical substructural algorithm. In GIM，the system control equations are divided into a linear group，which includes the equilibrium equation and the compatibility equation，and into a nonlinear group，which includes the constitutive equation. The iteration procedure starts from a special solution of the equilibrium equation which is achieved by using the GIM theory. The whole iteration procedure can be divided into the global stage and the local stage. In the global stage，the linear group is used，while in the local stage，the nonlinear group must be considered. The calculation can be perform in a parallel way according to the time in the global stage and the space in the local stage. The parallelization process of GIM is present. Several different

收稿日期：2003－12－08；**修回日期：**2004－02－18

基金项目：国家重点基础研究发展规划(973)资助项目(2002CB412705，2002CB412707)

作者简介：王开健(1976－)，男，1998 年毕业于清华大学土木工程系结构工程专业，现为博士研究生，主要从事并行计算结构力学方面的研究工作。E－mail：wkj98@mails.tsinghua.edu.cn。

methods of parallelization are given and compared. The parallelization is considered in the cluster system with message passing interface(MPI). MPI is a kind of parallel environment which is widely used nowadays. Cluster system is a system of many computers linked by high-speed network. The parallelization of GIM in such an environment works well，which proves the parallel-calculating characteristics of GIM to have better performance. The ratio of the time between communication and calculation is an important key during the parallelization. And the parallel program must be compatible with both the hardware and software environment of parallelization.
Key words：rock and soil mechanics；parallel-calculating；GIM；cluster；MPI

1 引 言

近年来，随着科学技术和工程实践的发展，出现了越来越多的大型、超大型的复杂结构，也由此对力学计算提出了更高的要求。许多大规模的计算问题已经超出了单机所能承受的能力范围，这种现实的需求将促使我们去寻求高性能计算领域的帮助。当前，在计算力学领域内，正在逐渐形成一个以并行计算为特征的新的学科分支。但直到目前为止，在计算结构力学的并行计算研究中，研究较多也相对较成熟的是基于子结构的并行计算，其他形式的并行计算还较少见[1, 2]。

广义逆力法(force method based on generalized inverse matrix，GIM)是一种以力法为力学概念基础，以广义逆矩阵理论为数学理论基础，以迭代求解为求解方式的新算法[3]。该算法主要是针对材料非线性问题。由于无需像传统的基于位移法的逐步增量法那样逐步递进计算，所以也称特大增量步算法。同时 GIM 算法也是一种具有高度可并行性的算法，它不同于传统的基于子结构的结构并行计算，为高性能结构计算提供了新的思路[4]。作者介绍了整个算法在基于 MPI(message passing interface)的机群环境下从串行进行初步并行化的过程，并阐述了此过程中每一次改进的出发点、所遇到的问题、解决方案以及成败的原因。本文只考虑静力、小变形问题。

2 GIM 算法迭代过程描述

从程序计算的角度看，GIM 算法可分为 3 部分：初始化计算控制系统、迭代求解和结果输出。

系统控制方程为

$$\left.\begin{array}{ll} \boldsymbol{CF}=\boldsymbol{P} & \text{平衡方程} \\ \boldsymbol{C}^{\mathrm{T}}\boldsymbol{D}=\boldsymbol{\delta} & \text{协调方程} \\ \boldsymbol{\delta}^{\mathrm{e}}=\boldsymbol{\Phi}^{\mathrm{e}}(F^{\mathrm{e}},\ \varepsilon_{\mathrm{p}}^{\mathrm{e}}) & \text{本构关系} \end{array}\right\} \tag{1}$$

式中：Φ 为本构柔度函数(线性或非线性)，上标 e 表示该关系仅局限于各个单元内部。

对小变形、静力、材料非线性问题，平衡方程和协调方程均为线性方程，令其为一组，构成解空间Γ；本构关系为非线性方程，令其单独为组，构成解空间ξ。利用广义逆矩阵理论用平衡方程求出一组特解作为迭代初始值，然后在 2 个解空间之间迭代求得公共解(原问题的解)。迭代过程如图 1 所示。

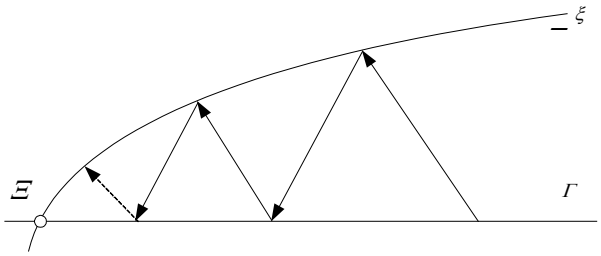

图 1 迭代过程图示

Fig.1 Illustration of the iterative process

加载历史函数为

$$P=P(t)，\ t\in[0，\ T] \tag{2}$$

在加载历史函数上取若干样本点：$P(t_0)$，$P(t_1)$，$P(t_2)$，…，$P(t_r)$，$(0=t_0<t_1<t_2<\cdots<t_r=T)$。

GIM 算法迭代过程如下：

(1) 首先，由广义逆矩阵理论[5]求得平衡方程的一个特解作为迭代初始值，令 $n=1$，即

$$F_1(t_j)=C^{+}P(t_j)=C^{\mathrm{T}}(CC^{\mathrm{T}})^{-1}P(t_j)\ (j=1，2，\cdots，r) \tag{3}$$

(2) 由 $F_n(t_j)$ 迭代求解 $F_{n+1}(t_j)$：

① 求解 F_n 时的变形为

$$\hat{\delta}_n^{\mathrm{e}}(t_j)=\Phi^{\mathrm{e}}(F_n^{\mathrm{e}}(t_j)，\ \varepsilon_{\mathrm{p}n}^{\mathrm{e}}(t_j)) \tag{4}$$

② 检验 $\hat{\delta}_n$ 的协调情况为

$$\varepsilon(\hat{\delta}_n(t_j))=\frac{\left\|\beta\hat{\delta}_n(t_j)\right\|}{\left\|\hat{\delta}_n(t_j)\right\|} \tag{5}$$

③ 确定搜索方向为

$$S_n(t_j) = -\beta K_n(t_j)\beta\hat{\delta}_n(t_j) \tag{6}$$

④ 确定搜索步长 $h_n(t_j)$ 为

$$h_n(t_j) = -\frac{\hat{\delta}_n^{\mathrm{T}}(t_j)S_n(t_j)}{S_n^{\mathrm{T}}(t_j)\overline{\Phi}_n(t_j)S_n(t_j)} \tag{7}$$

⑤ 求 $F_{n+1}(t_j)$ 为

$$F_{n+1}(t_j) = F_n(t_j) + h_n(t_j)S_n(t_j) \tag{8}$$

令 $n = n+1$，返回第①步循环迭代。

(3) 由迭代所得内力 $F_n(t_j)$ 求变形 $\delta_n(t_j)$ 和位移 $D_n(t_j)$ 为

$$\delta_n(t_j) = \alpha\hat{\delta}_n(t_j) \tag{9}$$

$$D_n(t_j) = (CC^{\mathrm{T}})^{-1}C\delta_n(t_j) \tag{10}$$

分析整个算法的迭代过程，其中涉及的数值计算主要有网格划分、计算广义逆矩阵、矩阵求逆、矩阵(向量)乘等几类。

3　并行环境描述

3.1　硬件环境

GIM 算法的并行化所使用的硬件条件为清华大学计算机系高性能计算研究所提供的机群系统。机群是指紧密连接的一组计算机，用来持续性地提供高性能的计算服务。机群系统使用高速通信网络将多台原本独立、完整的微机或工作站连接在一起，构成一个统一的整体，使之可作为一种单一的计算资源来使用。

机群系统的体系结构可由同构或异构方式组成，分布式存储，采用消息传递机制，适于粗粒度的并行[6]。机群系统的特点包括：

(1) 可扩展。机群的规模可根据需要随时扩大或缩小。

(2) 宜升级。随着 CPU 等硬件性能的提高，只需更换相应硬件就能迅速提高机群的计算速度，能够有效保护用户已有的投资。

(3) 高可用性和高可靠性。建立在网络上的分布式多节点计算系统相互间的关联度较小，因此，软件重构、负载迁移等高可用性和高可靠性技术可以得到很好应用。

(4) 易维护。由于每个节点的计算机都是独立的，因此，维护就非常方便，而且配件也很普及，维护费用低。

(5) 价格低。

作者先后使用过 2 套机群系统，最终的测试及本文的测试数据均采用了较新的第 2 套机群系统。

第 1 套机群系统名为 THNPSC-I/II(1998 年研制)，如图 2。I 型 CPU 是 PIII500(8 个双 CPU 的节点，内存是 512 M)，II 型 CPU 是 PIII700～1G(在 I 型的基础上增加了 12 个节点，CPU 主频从 700～1 G 不等，内存也是 512 M)。使用 100 M 以太网和 16 端口的 Myrinet 连接。

第 2 套机群系统名为云南机群系统(2001 年研制)，如图 3。共 8 个节点，每个节点有 2 个 AMD1900 的 CPU，内存是 512 M。使用 100 M 以太网和 32 端口的 Myrinet-2000 连接。

图 2　THNPSC-I/II
Fig.2　THNPSC-I/II

图 3　云南机群系统
Fig.3　Cluster system of Yunnan

3.2 软件环境

操作系统均采用 LINUX，并行环境采用 MPI 即基于消息传递方式的并行环境。消息传递方式是广泛应用于多类并行机的一种模式，特别是那些分布存储并行机[7]。许多组织对 MPI 标准付出了努力，主要来自美国和欧洲。MPI 的标准化开始于 1992 年。初始草案于 1992 年 11 月推出，在 1993 年 2 月完成了修订版即 MPI 1.0。目前已经发展到了 MPI-2[8]。

MPI 的定义包含以下几个方面的内涵：

(1) MPI 是一个库，不是一门语言。MPI 库可以被 FORTRAN 77/C/ FORTRAN 90/C^{++}调用，从语法上说，它遵守所有对库函数/过程的调用规则，和一般的函数/过程没有什么区别。

(2) MPI 是一种标准或规范的代表。迄今为止，所有的并行计算机制造商都提供对 MPI 的支持，可以在网上免费得到 MPI 并可在不同的并行计算机上实现。一个正确的 MPI 程序，可以不加修改地在所有的并行机上运行。

(3) MPI 是一种消息传递编程模型，并成为这种编程模型的代表和事实上的标准。MPI 虽然很庞大，但是，它的最终目的是服务于进程间的通信这一目标。

作为一种目前事实上最流行和通用的并行环境，MPI 可以为用户提供 3 个方面的优越性：较高的通信性能、较好的程序可移植性和强大的功能。

4 算法性能评测

首先根据问题编写出串行算法，然后使用 Profile 工具来评测算法的瓶颈，部分统计数据见表 1(迭代 100 次)。

表 1 中省略了占时很少的其他函数，从性能统计较易发现问题的瓶颈。表 1 显示，占用时间最多的调用函数是 CTCTI+INV，该函数是求广义逆矩阵的过程，它占用了约 72%的时间，而 ALFA 是迭代中调用的函数，调用了 203 次才占用了 22.5%的时间，也可以看作瓶颈之一。其他函数在整个迭代计算中占时甚少，可以不放在优化考虑范围之内。

通过对算法的复杂度分析可知，在上述的 2 个瓶颈中，CTCTI+INV 的算法复杂度是 O(n^3)，而 ALFA 的复杂度是 O(n^2)。随着问题规模的增大，ALFA 所占的比例将越来越小，因此，首要的优化目标选择为 CTCTI+INV 函数。此外，对于整个问题的分析过程可知，只有 CTCTI+INV 的复杂度是 O(n^3)，其余部分复杂度均不超过 O(n^2)。

通过瓶颈和复杂度的分析，避免了盲目优化全部迭代部分的策略，定位了真正需要优化的瓶颈。

5 并行化及优化过程

5.1 串行程序部分

以求逆部分为例，首先给出求逆的串行版本程序(主体部分)：

```
void INV(double* CT，int NCEX){
        int I1=NCEX；
        int I2=I1+1；
        double YY；
        double * line1 = new double[NCEX]；
        double * line2 = new double[NCEX]；
        for(int I=0；I<NCEX；I++){
                YY=1.0/CT[I*NCEX+I]；
            for(int J=0；J<NCEX；J++){
```

表 1　GIM 程序各函数统计数据

Table 1　Data of GIM functions

函数名称	函数功能	函数执行时间/μs	百分比/%	函数和子函数执行时间/μs	百分比/%	调用次数
CTCTI	求广义逆矩阵	16 615.760	60.8	19 690.556	72.0	1
ALFA	矩阵向量乘	6 156.593	22.5	6 156.593	22.5	203
INV	矩阵求逆	3 074.796	11.3	3 074.796	11.3	27
ZERO	矩阵赋零	94.436	0.3	94.436	0.3	27
INV1	矩阵求逆	78.893	0.3	78.893	0.3	40 800
EST	柔度/刚度矩阵	48.544	0.2	127.437	0.5	40 800
NEWFOR	求 F_{n+1}	30.616	0.1	30.616	0.1	101

```
                line1[J]=CT[I*NCEX+J]*YY；
                line2[J]=CT[J*NCEX+I]；
            }// for 20
            // 可以并行
            for(J=0；J<NCEX；J++){
                for(int K=0；K<NCEX；K++){
    CT[J*NCEX+K]=CT[J*NCEX+K]-line2[J]*line
1[K]；
                }
            }
            for(J=0；J<NCEX；J++){
                CT[J*NCEX+I]=-line2[J]*YY；
                CT[I*NCEX+J]=line1[J]；
            }
            CT[I*NCEX+I]=YY；
        }// end for 10
        delete [] line1；
        delete [] line2；
}
```

从数据的独立性角度来说，显然代码中黑斜体部分是最容易进行并行的，因为对于各行数据来说，计算是独立无关的。因此，最先对这一部分进行并行化。

5.2 并行程序部分

5.2.1 并行方案 1：最直观简单的并行

对于上述黑斜体部分，显然可以采用最直观简单的并行程序模型：SPMD(单程序多数据并行计算)，主进程先广播各进程需要计算的数据，然后各进程分别计算，最后主进程回收所有数据。并行程序如下：

```
        // 广播 line1 和 line2
        MPI_Bcast(line1，NCEX2，MPI_DOUBLE，
0，MPI_COMM_WORLD)；
        // 分条散播矩阵
        MPI_Scatter(buffer ， nDiv*nCol ，
MPI_DOUBLE，buffer，nDiv*nCol，MPI_DOUBLE，
0，MPI_COMM_WORLD )；
        // SPMD 并行计算
        for(J=0；J<nRow；J++){
            for(K=0；K<NCEX；K++){
    CT[J*NCEX+K]=CT[J*NCEX+K]-line2[J+nOffs
etOfJ]*line1[K]；
            }
        }
        // 主进程回收
        MPI_Gather(buffer ， nDiv*nCol ，
MPI_DOUBLE，buffer，nDiv*nCol，MPI_DOUBLE，
0，MPI_COMM_WORLD)；
```

这一并行化过程十分简单，但是根据实际测量运算结果，对于 $N = 1\ 000$ 的规模，时间是串行的 10 倍左右，而且每个进程的 CPU 占用率只有 30%左右。

图 4 是主进程与从进程的时间分布图。

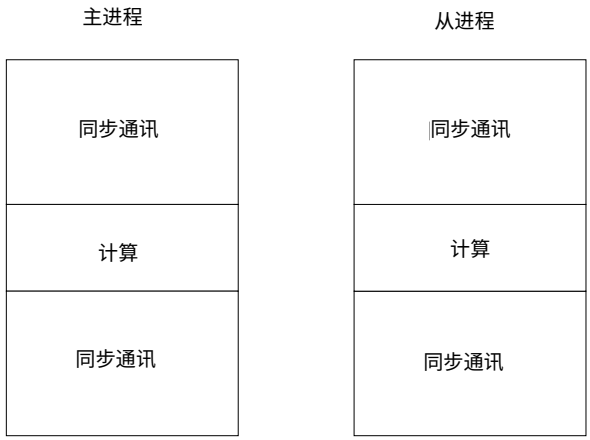

图 4　方案 1 的时间分布图

Fig.4　Time distribution of plan 1

5.2.2 并行方案 2：基于非阻塞通讯[7]的负载平衡方案

从图 4 可以看出通讯占用了大量的时间。因此，改进方案：考虑主进程是否可以采用异步通讯模式，每次使用非阻塞的发送和接收，同时将任务划分为更小的粒度，使得主进程和从进程的负载区域平衡。时间分布图如图 5 所示。

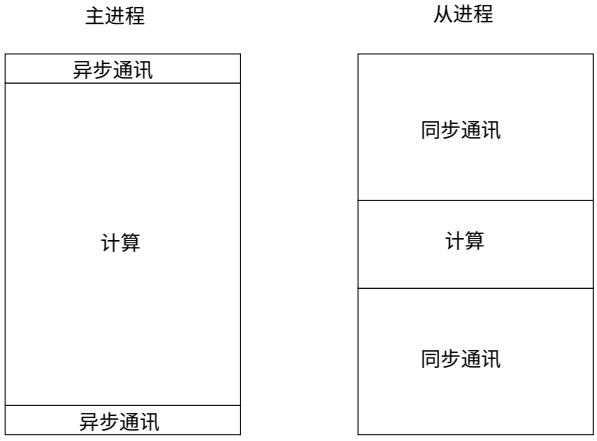

图 5　方案 2 的时间分布图

Fig.5　Time distribution of plan 2

实现代码如下：

```
// 广播 line1 和 line2
MPI_Bcast(line1，NCEX2，MPI_DOUBLE，
0，MPI_COMM_WORLD)；
MPI_Barrier(MPI_COMM_WORLD)；
// 用于非阻塞通讯
MPI_Request * requests =
new MPI_Request[nSize-1]；
MPI_Status  * statuss =
new MPI_Status[nSize-1]；
MPI_Status status；
if(nID == 0){
    // 主进程非阻塞发送每条数据
    for(int i=1；i<nSize；i++){
        MPI_Isend(buffer+i*nDiv*
nCol，nDiv*nCol，
MPI_DOUBLE，i，100，
MPI_COMM_WORLD，
&requests[i-1])；
    }
}else{
    // 其他进程阻塞接收主进程的数据
    MPI_Recv(buffer，nDiv*nCol，
MPI_DOUBLE，0，100，MPI_COMM_WORLD，
&status)；
}
// 并行计算一条任务
for(J=0；J<nRow；J++){
    for(K=0；K<NCEX；K++){
CT[J*NCEX+K]=CT[J*NCEX+K]-line2[J+nOffs
etOfJ]*line1[K]；
    }
}
int flag；
if(nID == 0){
    // 确定所有非阻塞发送完毕
    MPI_Waitall(nSize-1，requests，
statuss)；
}
// 接收所有从进程数据
if(nID == 0){
    for(int i=1；i<nSize；i++){
        MPI_Irecv(buffer+i*nDiv*nCol，
nDiv*nCol，MPI_DOUBLE，i，100，
MPI_COMM_WORLD，&requests[i-1])；
    }
}else{
    MPI_Isend(buffer，nDiv*nCol，
MPI_DOUBLE，0，100，MPI_COMM_WORLD，
&requests[0])；
}
if(nID == 0){
    // 等待所有非阻塞接收
    MPI_Waitall(nSize-1，requests，
statuss)；
}
else{
    MPI_Wait(&requests[0]，&status)；
    }
```

可以通过调整发送的数据来达到负载平衡。通过调整参数，在 2～8 个节点上最好的性能接近串行的 1.5～2 倍，但每个进程的 CPU 占用率只有 35% 左右，效果仍不理想。

5.2.3 并行方案 3：通讯优化方案

由于每次都需要广播矩阵的分条数据，其通讯量是十分可观的。初步认定前面 2 个方案效果不佳的原因是由于通讯数据量太大，统计计算时间与通讯时间如表 2 所示。

表 2 计算时间与通讯时间统计

Table 2 Time of calculation and communication

次数 N	内 2 层计算时间/s	总计算时间/s
1 000	0.018	18
4 000	0.300	1 200
8 000	1.200	9 600

当 N = 8 000 时，由于使用的机器是 512 M 的内存，而 double[8 000][8 000]的矩阵所需要的空间亦为 512 M，由于内存用尽，不断交换内存外存，实际每次计算内 2 层需要几十秒。

另外，对于异步算法中的非阻塞通讯函数的调用开销见表 3。

通过对通讯函数的开销与计算时间的比较可以看出，即使使用非阻塞通讯，由于每次函数的开销

与计算时间相当，甚至大于计算时间，因此，单纯通过使用非阻塞通讯是很难提高程序效率的。而 Bcast 函数耗用时间很少的原因是因为它每次只广播 2 行数据，相对于矩阵的每一条来说，通讯量很小。因此，现在考虑问题的关键就是降低每次循环内计算的通讯开销。

表 3　非阻塞通讯函数的调用开销统计

Table 3　Time of communication function

函数类型	时间/s	建议
Bcast	0.000 3	不需要优化，可以使用
Isend，Irecv	＜0.000 1	不需要优化，可以使用
Test	0.06～0.13	需要优化，无法使用
Wait	0.18～0.2	需要优化，无法使用

对求逆算法进行进一步分析得到如下结果：对于如图 6 所示的矩阵，每次每条计算需要的 line1 和 line2 实际上就是图 6 中带有阴影的行和列，而 line1 对应“ \\\\ ”行，该行由其所在进程计算得出，然后广播给其他各进程即可。而对于 line2 的使用，其实每个进程只需要自己包含的那一部分“ //// ”，没有必要通过广播来接收。这样每次外层循环内只需要广播 line1 即可。而对于矩阵的分条扩散过程和回收过程，只需要在外层循环外各进行 1 次即可。

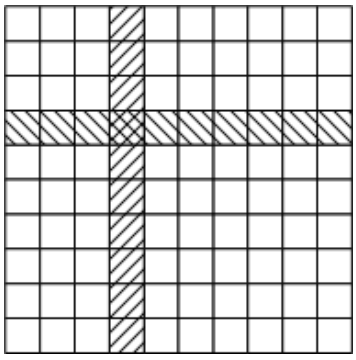

图 6　矩阵求逆示意图

Fig.6　Inverse procedure of matrix

改写后的求逆函数如下：

```
void INV(double* CT，int NCEX){
    // 系数
    double YY；
    // 循环变量
    int I，J，K；
    // 每一条的计算宽度
    int nWidth；
    if(nID == 0){
        nWidth = nMod+nDiv；
    }else{
        nWidth = nDiv；
    }
    double * line1 = new double[NCEX]；
    double * line2 = new double[nWidth]；
    // 辅助缓冲首地址，只用于 scatter 和 gather
    double * buffer；
    if(nID == 0){
        // root 进程
        //              buffer
        //               |
        //               V
        // CT[nMod][nDiv][nDiv]...[nDiv]
        buffer = CT+nMod*NCEX；
    }
    else{
        // root 进程
        // buffer
        //    |
        //    V
        // CT[nDiv]
        buffer = CT；
    }
    int nOffsetOfJ；
        // root 进程          1 进程          n 进程
        // offset0          offset1        offsetn
        //    |                 |              |
        //    V                 V              V
        // CT[ nMod ][ nDiv ][ nDiv ]...[ nDiv ]
    if(nID == 0){
        nOffsetOfJ = 0；
    }
    else{
        nOffsetOfJ = nDiv*nID+nMod；
    }
    // 计算之前，将矩阵散布出去
    MPI_Scatter(buffer，nDiv*nCol，MPI_DOUBLE，
buffer，nDiv*nCol，MPI_DOUBLE，0，
MPI_COMM_WORLD )；
```

```
MPI_Barrier(MPI_COMM_WORLD)；
    double begin，end；
// 中间每个矩阵计算自己的内部数据，交换边界数据
for(I=0；I<NCEX；I++){
        if(I % 100 == 0){
            begin = MPI_Wtime()；
        }
    // 计算谁含有对角元
    int   dia = I>=nMod+nDiv?(I-nMod-nDiv)/nDiv+1:0；
    // 计算对角元
    if(nID == dia){
        YY=1.0/CT[(I-nOffsetOfJ)*NCEX+I]；
    }
    // 广播对角元
    MPI_Bcast(&YY，1，MPI_DOUBLE，dia，MPI_COMM_WORLD)；
    // 计算 line1
    if(nID == dia){
        // 计算 line1
        for(J=0；J<NCEX；J++){
    line1[J]=CT[(I-nOffsetOfJ)*NCEX+J]*YY；
        }
        // 广播 line1
        MPI_Bcast(line1，NCEX，MPI_DOUBLE，dia，MPI_COMM_WORLD)；
    }else{
        // 接收 line1
        MPI_Bcast(line1，NCEX，MPI_DOUBLE，dia，MPI_COMM_WORLD)；
    }
    // line2 可以局部计算
    for(J=0；J<nWidth；J++){
        line2[J] = CT[J*NCEX+I]；
    }
    // 可以并行
    // 计算
    for(J=0；J<nWidth；J++){
        for(K=0；K<NCEX；K++){
    CT[J*NCEX+K]=CT[J*NCEX+K]-line2[J]*line1[K]；
        }
    }
    // 更新部分元素
    // 更新所有行的当前列元素
    for(J=0；J<nWidth；J++){
        CT[J*NCEX+I] = -line2[J]*YY；
    }
    // 更新当前行
    if(dia == nID){
        for(J=0；J<NCEX；J++){
            CT[(I-nOffsetOfJ)*NCEX+J] = line1[J]；
        }
        CT[(I-nOffsetOfJ)*NCEX+I] = YY；
    }
}
// 计算完毕之后汇总
MPI_Gather(buffer，nDiv*nCol，MPI_DOUBLE，buffer ， nDiv*nCol ， MPI_DOUBLE ， 0 ，MPI_COMM_WORLD)；
MPI_Barrier(MPI_COMM_WORLD)；
SAFE_DELETE(line1)；
}
```

对于此程序进行测试，在 8 个 CPU 上运行 4 000 与 8 000 规模的问题，实际加速比均接近 8，效果十分理想。各个节点的 CPU 占用率达到 95%以上。

对于串行程序来说，计算 8 000 规模的问题的迭代循环中每一步就需要几十秒，而采用并行程序，计算一步只需要 0.15 s 左右。通过并行计算，可计算的问题的规模大大提高了。由于每步通讯量都很小，随着问题求解规模的增大，通讯时间与计算时间的比例越来越小，因此，如果采用 n 个 CPU 来计算，加速比可以达到 $n-1$～n，同时计算的规模可以随着节点数的增加而增加。在内存允许的范围内，对于工作量一定的情况，性能加速规律符合 Amdahl 定律[9]，加速比接近线性。当各节点数据存储量超出单机内存范围之后，随着机器的增加，单位时间内可求解的计算量也随之增加，符合 Gustafson 定律[9]，对于固定负载的大规模计算可达到线性的加速比。

6 结 论

从定量分析的角度完成了 GIM 算法在基于 MPI 机群环境下的初步并行化和优化。从实际并行计算的结果可以看到，广义逆力法在理论上所预测的可并行性在计算实践中得到了良好的印证。这种并行

计算的模式是完全不同于传统的基于子结构的结构并行计算模式的。

在对广义逆力法的并行化进行的初步尝试中，可以得出 2 个在并行化过程中应当遵循并特别加以注意的原则：

(1) 严格控制计算时间和通讯时间的比例，这将直接影响到并行效率的高低。在采用非阻塞通讯来提高性能之前，由于没有对计算时间和通讯时间的比例做定量的分析，导致了先后采用了 2 种不佳的方案。最后，在对计算时间和通讯时间的比例进行测量分析之后，对算法本身从数据结构内部的关联关系出发，采取 SPMD 模型，把每个节点将要交换的数据作为所谓的“ 边界数据 ”单独开辟缓冲区用于更新交换，大大降低了通讯开销，最终使得性能大大提高。可见，对于采用消息传递方式的网络并行机群，降低通讯开销才是提高并行计算效率的关键[10]。

(2) 并行程序必须考虑到与实际并行环境(包括硬件环境和软件环境)相匹配的问题[10]。目前的并行解决方案适合于在完全同构的机群上运行。当每个节点的计算能力、存储能力、负载情况十分接近时，程序可以达到较高的效率。但是如果在差异较大的环境中运行时，还需要对程序进行改进，使得其可以动态调整计算任务以达到负载平衡。

参考文献(References)：

[1] 张汝清. 并行计算结构力学的发展和展望[J]. 力学进展，1994，24(4)：511－517.(Zhang Ruqing. The development of the parallel computational structural mechanics[J]. Advances in Mechanics，1994，24(4)：511－517.(in Chinese))

[2] 张汝清. 概说并行计算结构力学[J]. 计算结构力学及其应用，1995，12(4) ： 477 － 484.(Zhang Ruqing. Introduction to parallel computational structural mechanics[J]. Computational Structural Mechanics and Applications，1995，12(4)：477－484.(in Chinese))

[3] Zhang C J，Liu X L. A large increment method for material nonlinearity problems[J]. Advances in Structural Engineering，1997，11(2)：99－110.

[4] 刘西拉. 结构工程学科的现状与展望[M]. 北京：人民交通出版社，1997.(Liu Xila. The Status and Prospects of Structural Engineering[M]. Beijing：China Communications Press，1997.(in Chinese))

[5] 何旭初. 广义逆矩阵的基本理论和计算方法[M]. 上海：上海科学技术出版社，1985.(He Xuchu. Generalized Inverse Theory and Algorithms[M]. Shanghai：Shanghai Scientific and Technical Press，1985.(in Chinese))

[6] 陈　波，韩永国，刘志勤. 高性能并行计算的研究与分析[J]. 四川师范学院学报(自然科学版)，2003，24(2)：200－202.(Chen Bo，Han Yongguo，Liu Zhiqin. Study and analysis of high performance parallel computing[J]. Journal of Sichuan Teachers College (Natural Science)，2003，24(2)：200－202.(in Chinese))

[7] 秦忠国，姜弘道. 消息传递界面 PVM 和 MPI 的现状与发展趋势[J]. 计算机研究与发展，1998，35(6)：496－499.(Qin Zhongguo，Jiang Hongdao. PVM and MPI：current situation and development trend[J]. Computer Research and Development，1998，35(6)：496－499.(in Chinese))

[8] 都志辉. 高性能计算并行编程技术——MPI 并行程序设计[M]. 北京：清华大学出版社，2001.(Du Zhihui. Parallel Program Technology of High Performance Computation：Parallel Program Design of MPI[M]. Beijing：Tsinghua University Press，2001.(in Chinese))

[9] Wang Kaijian，Xu Zhiwei. Scalable Parallel Computing Technology，Architecture，Programming[M]. Boston：McGraw-Hill Press，1998.

[10] 孟　杰，孙　彤，李三立. MPI 网络并行计算系统通信性能及并行计算性能的研究[J]. 小型微型计算机系统，1997，18(1)：13－18.(Meng Jie，Sun Tong，Li Sanli. Communication performance and parallel performance research of networked parallel computing system[J]. Mini-Micro Computer System，1997，18(1)：13－18.(in Chinese))