

MPICH 并行程序设计环境简介

安竹林

(合肥工业大学 计算机与信息学院 可视化与协同计算研究室)

anzhulin@sohu.com

[摘要] MPI(Message Passing Interface)是目前一种比较著名的应用于并行环境的信息传递标准。MPICH 是 MPI1.2 标准的一个完全实现,也是应用范围最广的一种并行及分布式环境。MPICH 除包含 MPI 函数库之外,还包含了一套程序设计以及运行环境。本文将简要介绍如何应用 MPICH 的 Windows 版本,建立一个基于 Windows 的并行程序设计及运行环境。并给出一个 C+MPI 程序设计实例。

1 MPICH for Microsoft Windows 的安装与配置

1.1 系统要求

安装 MPICH for Microsoft Windows 对系统有如下要求:

- Windows NT4/2000/XP 的 Professional 或 Server 版(不支持 Windows 95/98)
- 所有主机必须能够建立 TCP/IP 连接

MPICH支持的编译器有:MS VC++ 6.x, MS VC++.NET, Compaq Visual Fortran 6.x, Intel Fortran, gcc, 以及g77。安装MPICH,必须以管理员的身份登录。

1.2 下载

MPICH 的下载地址是:<http://www-unix.mcs.anl.gov/mpi/mpich/download.html> Windows 版本的 mpich.nt.1.2.5.exe 的下载地址是:<http://www-unix.mcs.anl.gov/~ashton/mpich.nt/>

1.3 安装

以管理员的身份登录每台主机,在所有主机上建立一个同样的账户(当然也可以每个机器使用不同的用户名和账户,然后建立一个配置文件,使用命令行的方式运行程序),然后,运行下载的安装文件,将 MPICH 安装到每台主机上。

打开“任务管理器”中的“进程”选项卡,查看是否有一个 mpd.exe 的进程。如果有的话说明安装成功。以后每次启动系统,该进程将自动运行。

1.4 注册与配置

安装好 MPICH 之后还必须对每台计算机进行注册和配置才能使用。其中注册必须每台计算机都要进行,配置只要在主控的计算机执行就行了。

注册的目的是,将先前在每台计算机上申请的账号与密码注册到 MPICH 中去,这样 MPICH 才能在网络环境中访问每台主机。配置方法:运行“mpich\mpd\bin\MPIRegister.exe”首先会提示输入用户账号,然后会提示输入两边密码,之后会问你是否保持上面的设定。如果选择是,则上面的信息将写入硬盘,否则保存在内存中,再重新启动之后就不存在了。

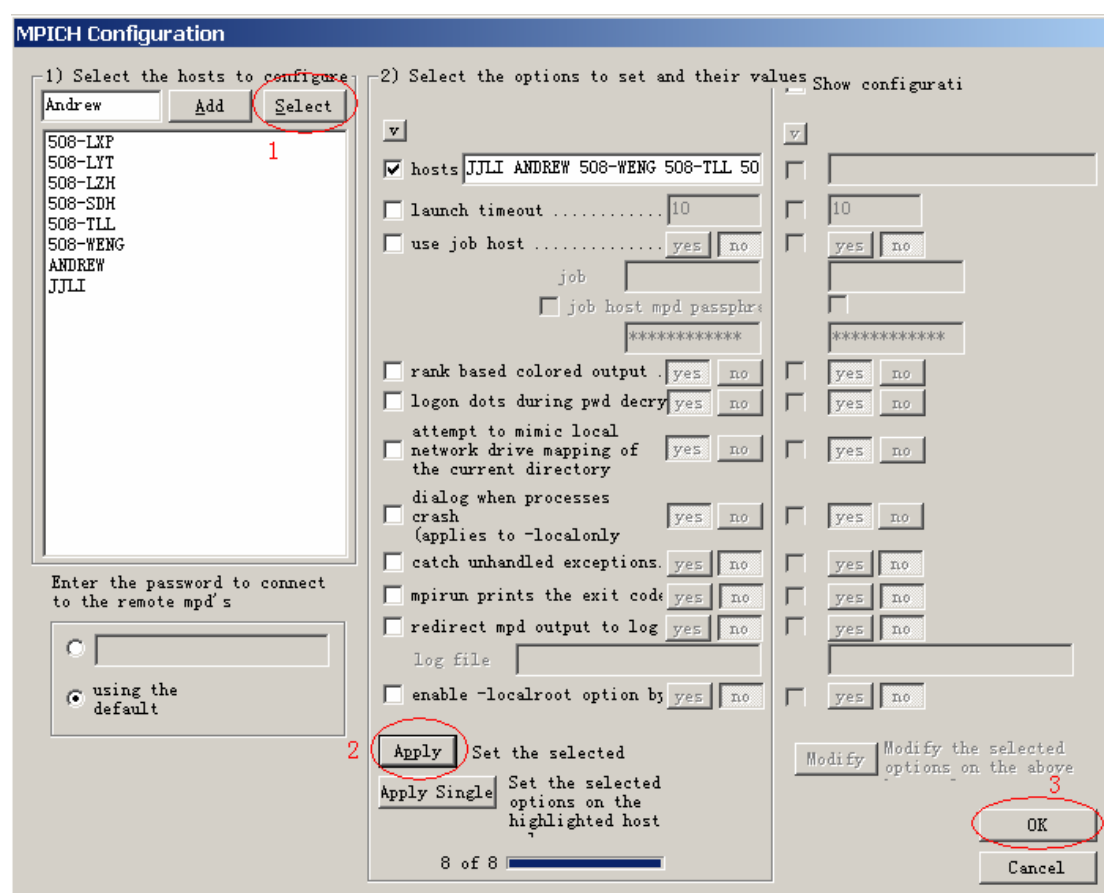
```

C:\MPICH\mpd\bin>mpiregister
account: andrew
password:
confirm:
Do you want this action to be persistent (y/n)? y
Password encrypted into the Registry.

```

图一 MPIRegister

为了让程序在许多主机上执行，而不需建立配置文件来给出相应的各个主机的信息，主控机必须直到当前可用的主机的信息。这时就需要运行MPICH的配置程序来进行配置了。MPICH提供的配置程序是一个图形界面的程序，可以从“开始->程序->MPICH->mpd->MPICH Configuration tool”启动。启动之后的界面如下图所示：



图二 MPICH Configuration tool

整个界面分为三栏，在第一栏中点击Select（1号按钮），然后在跳出的对话框中选择安装了MPICH的主机名。之后在第一栏的编辑框中会显示出所有选择的主机。检查无误后，点击第二栏的Apply（2号按钮），这时下方的进度条会显示对各主机核查的情况，如果没问题整个进度条会变为蓝色。最后点击OK（3号按钮）。整个配置就完成了。

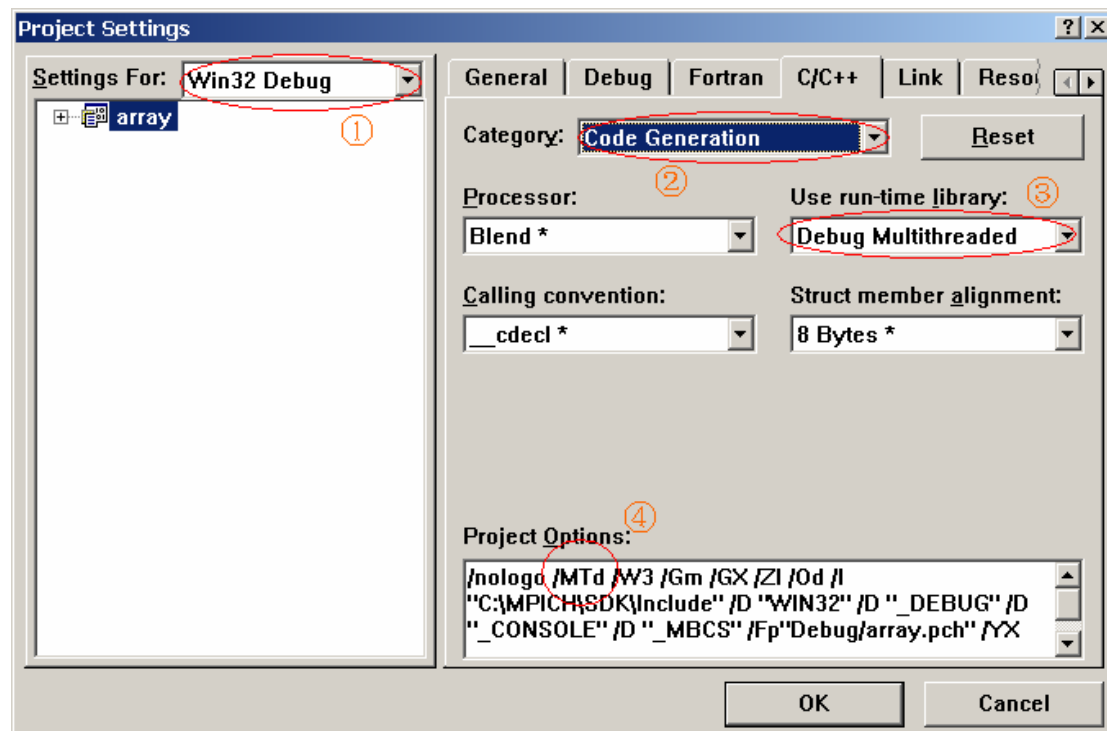
1.5 MPICH 与编译环境的整合

MPICH提供了C语言和Fortran语言的接口。要编译一个MPI+C或MPI+Fortran的程序必须对编译器进行设置。下面分别对Visual C++ 6.0、Compaq Visual Fortran 6.5、和Visual Fortran 5.0进行说明。

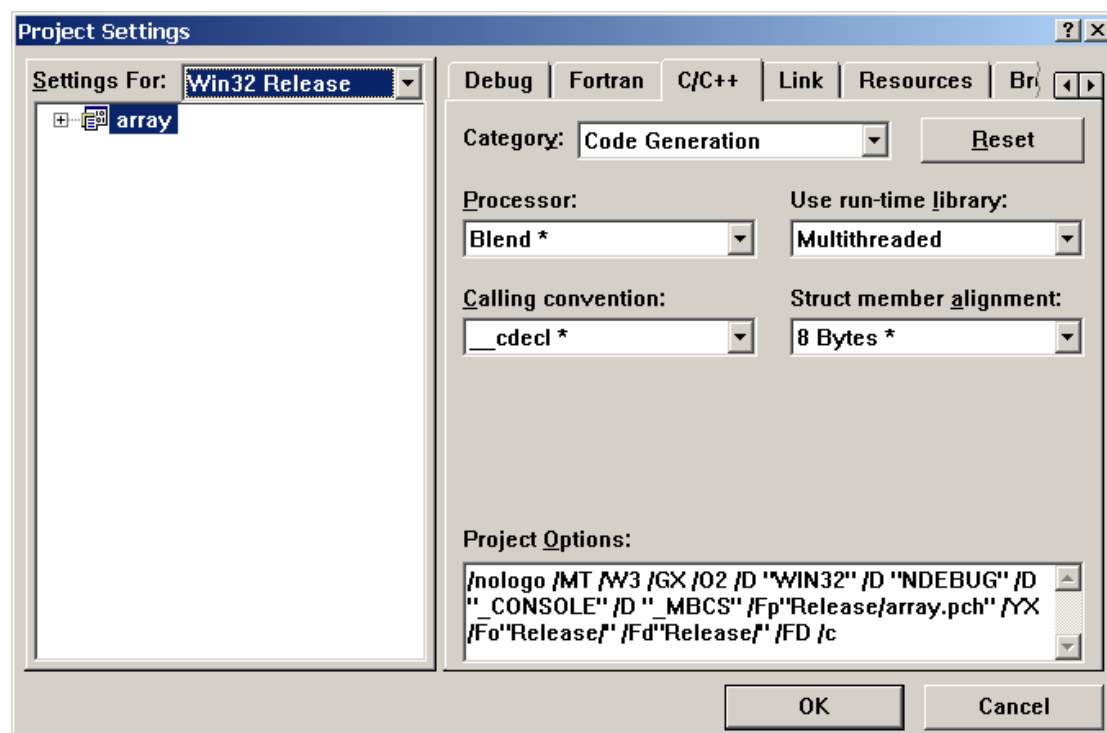
1.5.1 Visual C++ 6.0

在Visual C++ 6中编译一个MPI+C的程序步骤如下：

1. 打开Visual C++ 6的Develop Studio。
2. 新建一个工程，通常为Win32 Console Application。
3. 在新的工程的编辑界面下，按Alt+F7打开工程设置对话框。



图三 Visual C++设置 (1)

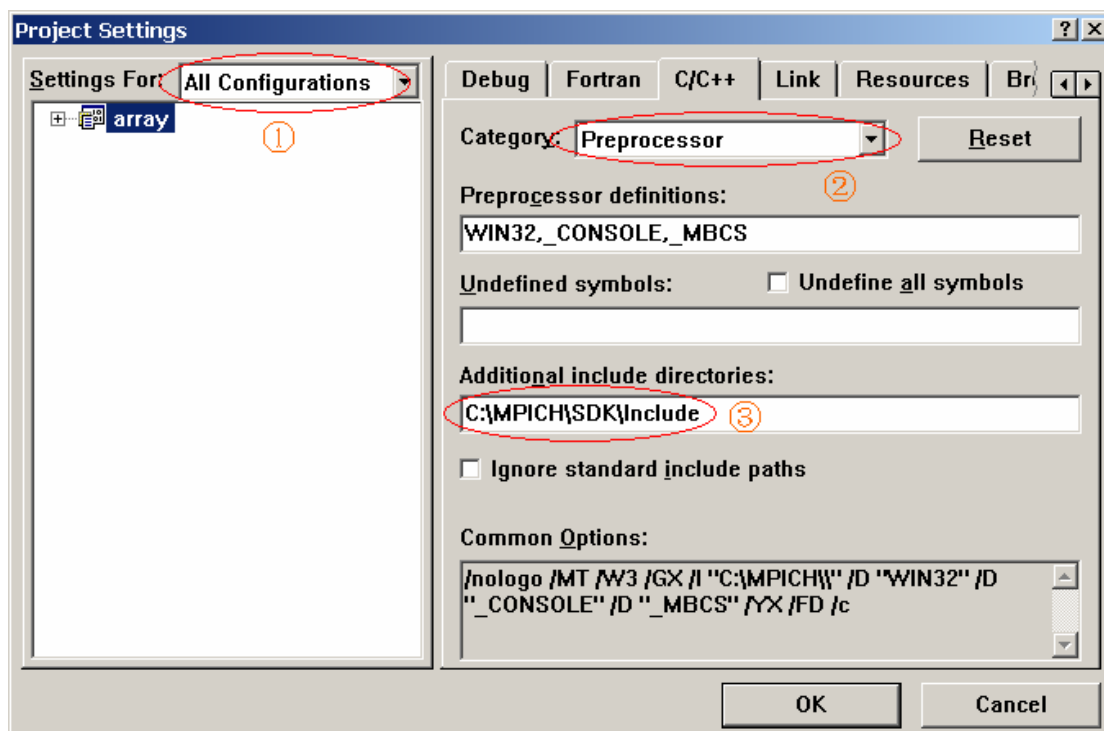


图四 Visual C++设置 (2)

4. 切换到C/C++选项卡。(如图三) 首先选择“Win32 Debug”(的下拉框), 再选择“Code Generation”(的下拉框), 再选择“Debug Multithreaded”(的下拉框)。这时在“Project Options”的文本框中显示“/MT”表示设置成功。然后选择“Win32

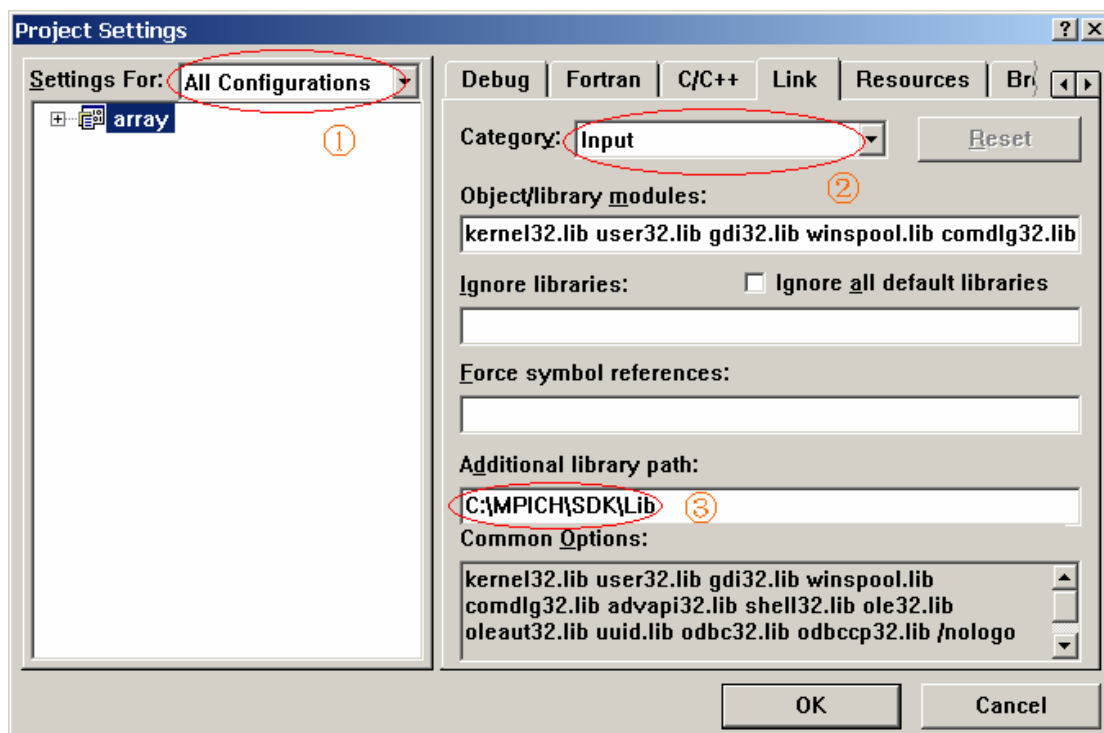
Release”(的下拉框)重复上述步骤。(图四)

5. 在C/C++选项卡中,选择“ All Configurations”(的下拉框)选择“ Preprocessor”(的下拉框),在“ Additional include directories”的文本框中输入MPICH所附带的头文件的目录。(如图五)



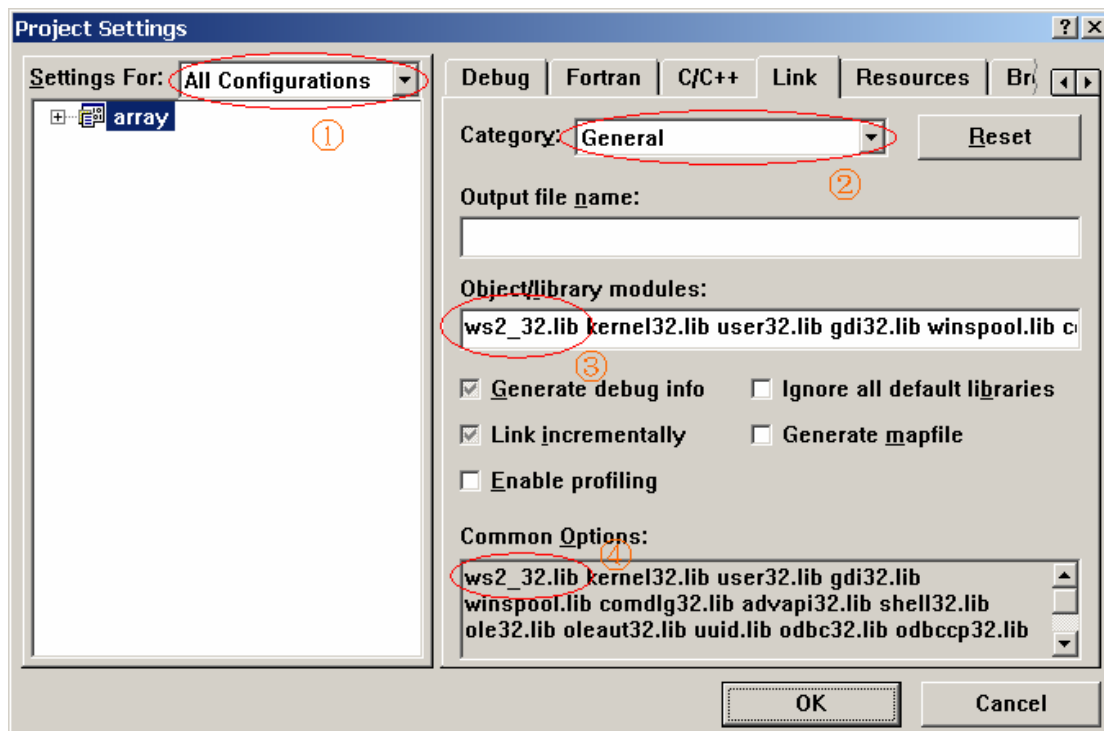
图五 Visual C++设置 (3)

6. 在Link选项卡中,选择“ All Configurations”(的下拉框)然后再选择“ Input”(的下拉框),在“ Additional library path”的文本框中输入MPICH所附带的库文件的目录。(如图六)



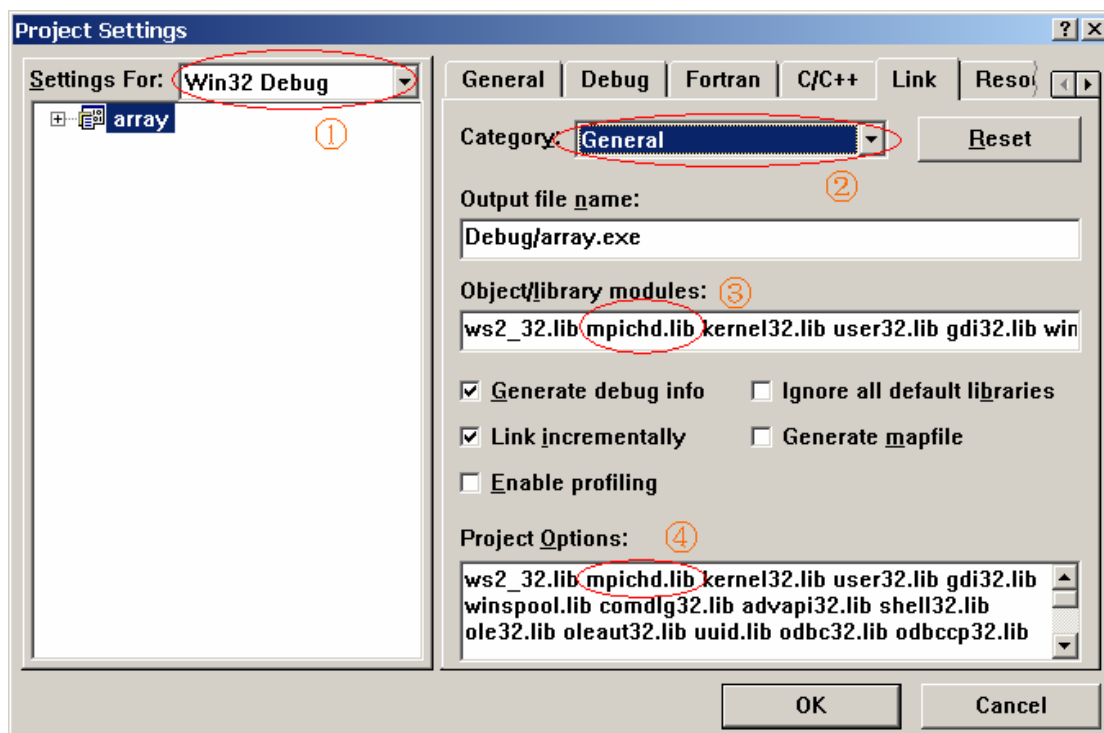
图六 Visual C++设置 (4)

7. 在Link选项卡中,选择“ All Configurations ”(的下拉框)然后再选择“ General ”(的下拉框),然后在“ Object/library modules ”()的文本框中添加“ ws2_32.lib ”。点击“ OK ”。这时在“ Common Options ”()中会出现“ ws2_32.lib ”。(如图七)



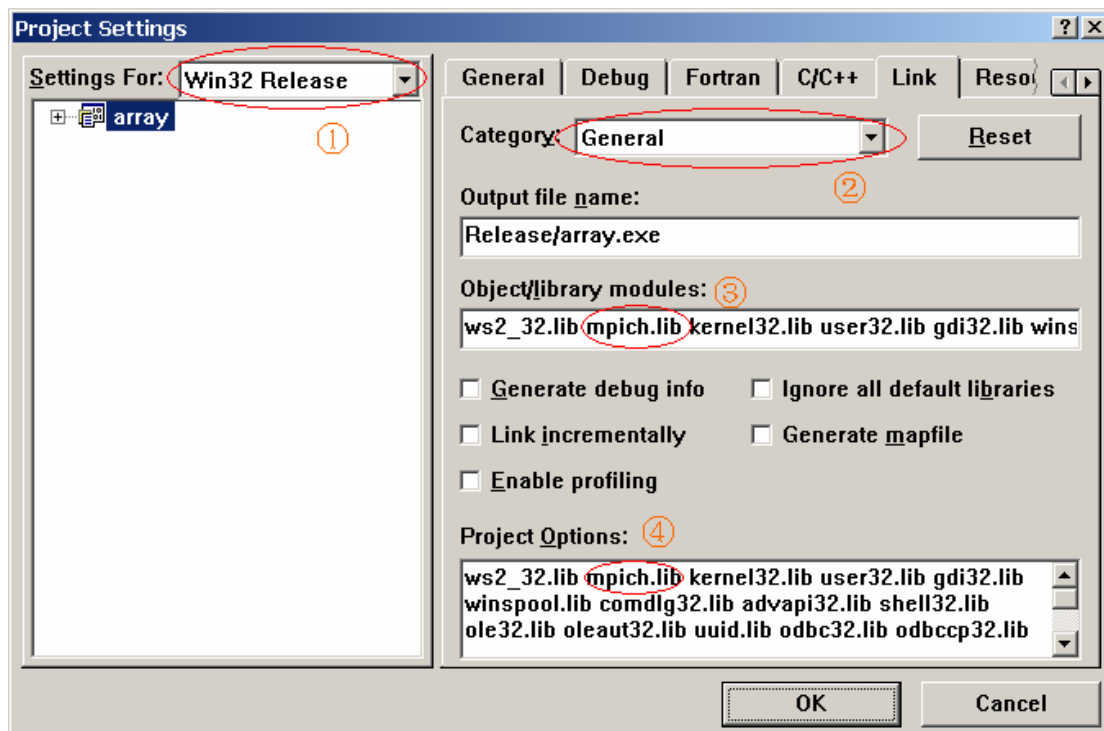
图七 Visual C++设置 (5)

8. 在Link选项卡中,选择“ Win32 Debug ”(的下拉框)然后再选择“ General ”(的下拉框),然后在“ Object/library modules ”()的文本框中添加“ mpichd.lib ”。点击“ OK ”。这时在“ Project Options ”()中会出现“ mpichd.lib ”。(如图八)



图八 Visual C++设置 (6)

9. 在Link选项卡中,选择“Win32 Release”。(的下拉框)然后再选择“General”(的下拉框),然后在“Object/library modules”()的文本框中添加“mpich.lib”。点击“OK”。这时在“Project Options”()中会出现“mpich.lib”。(如图九)



图九 Visual C++设置 (7)

10. 关闭工程设置对话框。

1.5.2 Compaq Visual Fortran 6.5

Compaq Visual Fortran 6.5的Develop Studio跟Visual C++ 6.0基本一致,所以设置方法也大致相同,在这里不再重复了。

1.5.3 Visual Fortran 5.0

MPICH手册中并没有提到支持Visual Fortran 5.0。这时因为Visual Fortran 5.0所附带的Link.exe是Microsoft Develop Studio 97里的版本(5.00.7022),如果在Visual Fortran 5.0中直接编译MPI程序会提示如下错误:

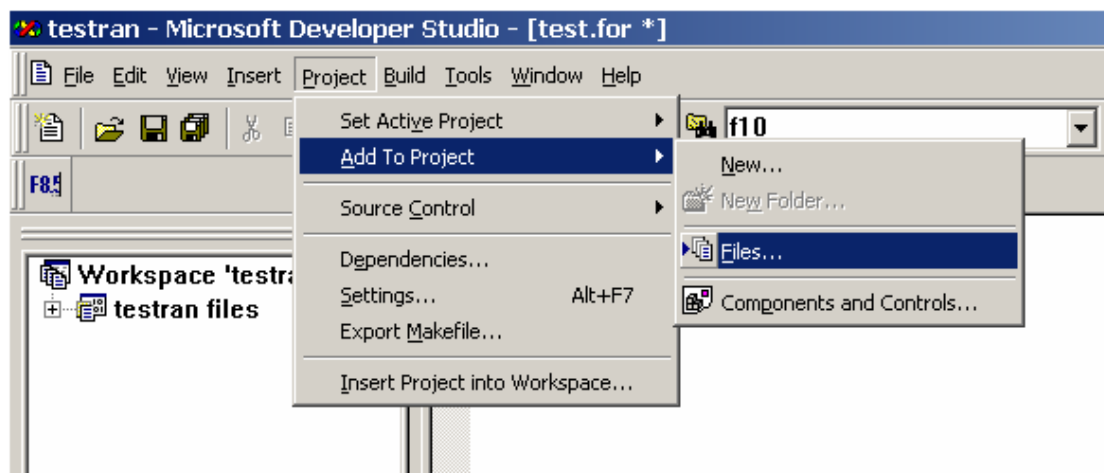
```
-----Configuration: testran - Win32 Debug-----
Compiling Fortran...
C:\Program Files\DevStudio\MyProjects\testran\test.for
Linking...
..\..\..\MPICH\SDK\Lib\mpich.lib : fatal error LNK1106: invalid file or disk full: cannot seek to 0x3ea20b5c
Error executing link.exe.

testran.exe - 1 error(s), 0 warning(s)
```

解决方法是:用新版的Link.exe来代替。如果有Visual C++ 6.0以上版本的话可以把新版的Link.exe复制到\DevStudio\VC\BIN中覆盖原来的Link.exe。如果没有Visual C++ 6.0,则可以到Microsoft的网站下载Windows DDK,其中所包含的Microsoft的免费汇编器MASM,其中的Link.exe也可以用来作代替。

修正了Link.exe的版本错误之后,就可以用Visual Fortran 5.0来编译MPI+Fortran程序了。由于Visual Fortran 5.0的Develop Studio跟Visual C++ 6.0的有所不同,所以不能使用上述的方法来

设置。由于对于Fortran程序来说,MPI可以看作一个库,所以只要建立一个工程,并把MPICH所附带的头文件和库文件添加到工程就可以编译了。具体方法是:在菜单中选择“Project->Add to project->Files...”然后在跳出的对话框选择MPI的头文件和库文件。(图十)



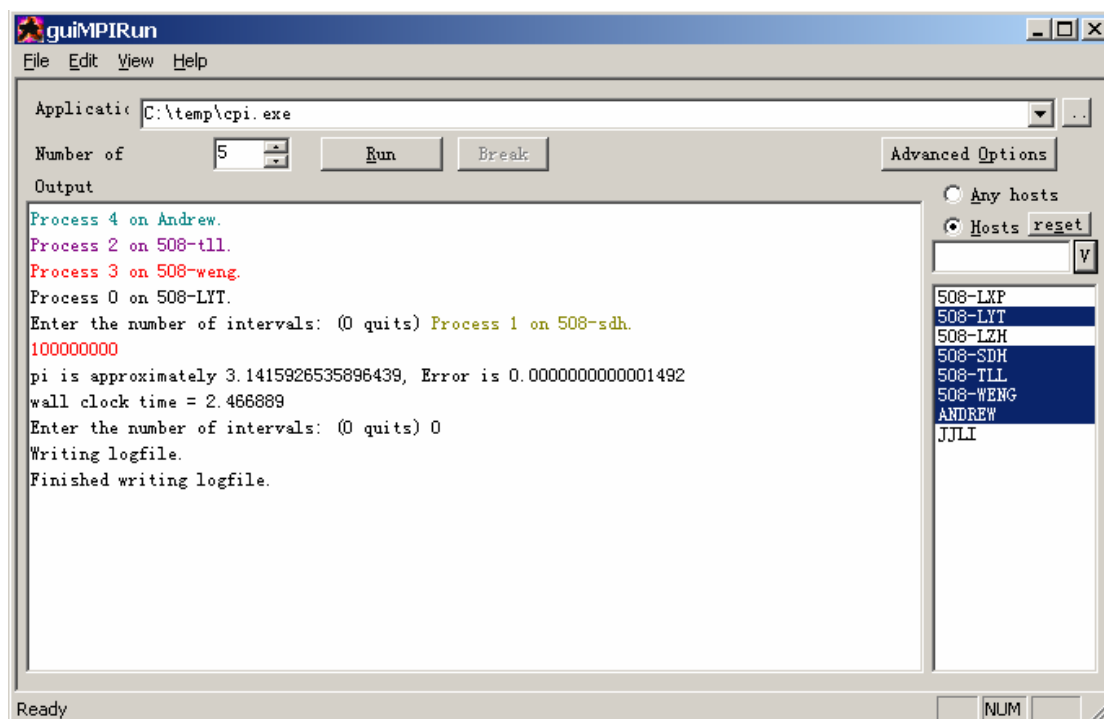
图十 在Visual Fortran 5.0中添加头文件和库文件

1.6 样例程序的运行

在“MPICH\SDK\Examples\nt\Basic”有一个简单的计算 π 的程序,在VC下编译成可执行程序就可应用来测试并行环境。

在MPICH中运行并行程序有两种方式:命令行方式和图形方式。命令行方式可以使用配置文件,功能比较强,但操作复杂易出错。图形方式简单明了,下面以图形方式说明如何运行一个测试程序来测试并行环境。

在运行一个程序之前,首先要做的事是将待运行的目标程序复制到所有的主机的相同目



图十一 MPIRun

录下,运行程序的步骤如下:从“开始->程序->MPICH->mpd->MPIRun”启动图形方式的

MPI环境。（如上图所示）首先，选择待运行的程序，点击右上角的“..”按钮，选择相应的程序；然后选择需要用于计算的主机个数，指定参加运算的主机名；最后点击Run，运行程序。运行的结果会显示在文本框，不同主机的输出信息会以不同的颜色标出。

2 MPI 程序设计简介

使用MPI进行程序设计的关键是灵活的使用MPI所提供的各种函数。下面是常见的MPI函数的说明。（这里提供的是C版本，Fortran版本与之类似）

2.1 MPI 初始化

功能：初始化MPI环境

```
int MPI_Init(int *argc, char ***argv)
```

2.2 MPI 结束

功能：终止MPI环境

```
int MPI_Finalize(void)
```

2.3 当前进程标识

功能：返回当前进程的标识。如果有 n 个进程，则进程的标识号从0到 $n-1$ 。

```
int MPI_Comm_rank(MPI_Comm comm, int *rank)
```

[IN Comm]	该进程所在的通信域（句柄）
[OUT rank]	调用进程在COMM中的标识号（整型）

2.4 通信域包含的进程数

功能：返回指定通信域中的进程个数。

```
int MPI_Comm_size(MPI_Comm comm, int *SIZE)
```

[IN comm]	该进程所在的通信域（句柄）
[OUT size]	通信域COMM内包含的进程数（整型）

2.5 消息发送

功能：将首地址为buf的count个数据类型为datatype的数据由当前进程发送给dest进程。

```
int MPI_Send(void* buf, int count, MPI_Datatype Datatype,
             int dest, int tag, MPI_Comm comm)
```

[IN buf]	发送缓冲区的起始地址（可选类型）
[IN count]	将发送的数据的个数（非负整型）
[IN datatype]	发送数据的数据类型（句柄）
[IN dest]	目的进程标识号（整型）
[IN tag]	消息标志（整形）
[IN comm]	通信域（句柄）

2.6 消息接收

功能：从source进程接受count个数据类型为datatype的数据并保存到首地址为buf的内存空间中。

```
int MPI_Recv(void* buf, int count, MPI_Datatype datatype,
             int source, int tag, MPI_Comm comm,
             MPI_Status *status)
```

[OUT buf]	发送缓冲区的起始地址（可选类型）
[IN count]	将发送的数据的个数（非负整型）
[IN datatype]	发送数据的数据类型（句柄）
[IN source]	源进程标识号（整型）
[IN tag]	息标识，与发送操作的标识匹配（整型）
[IN comm]	通信域（句柄）
[OUT status]	返回状态（状态类型）

2.7 消息广播

功能：将首地址为buf的count个数据类型为datatype的数据由root进程广播给指定通信域中所有进程。

```
int MPI_Bcast(void* buf, int count, MPI_Datatype datatype,
              int root, MPI_Comm comm)
```

[IN/OUT buf]	通信消息缓冲区的起始地址（可选类型）
[IN count]	将广播出去或接收的数据个数（非负整型）

[IN datatype]	广播 / 接收数据的数据类型 (句柄)
[IN root]	广播数据的根进程的标识号 (整型)
[IN comm]	通信域 (句柄)

2.8 收集

功能：将各个进程中的数据收集到根进程中。

```
int MPI_Gather(void* sendbuf, int sendcount,
              MPI_Datatype sendtype, void* recvbuf, int recvcount,
              MPI_Datatype Recvtype, int root, int comm)
```

[IN sendbuf]	发送消息缓冲区的起始地址 (可选类型)
[IN sendcount]	发送消息缓冲区中的数据个数 (非负整型)
[IN sendtype]	发送消息缓冲区中的数据类型 (句柄)
[OUT recvbuf]	接收消息缓冲区的起始地址 (可选类型)
[IN recvcount]	接收消息缓冲区中的数据个数 (非负整型)
[IN recvtype]	接收消息缓冲区中的数据类型 (句柄)
[IN root]	接收数据的根进程的标识号 (整型)
[IN comm]	通信域 (句柄)

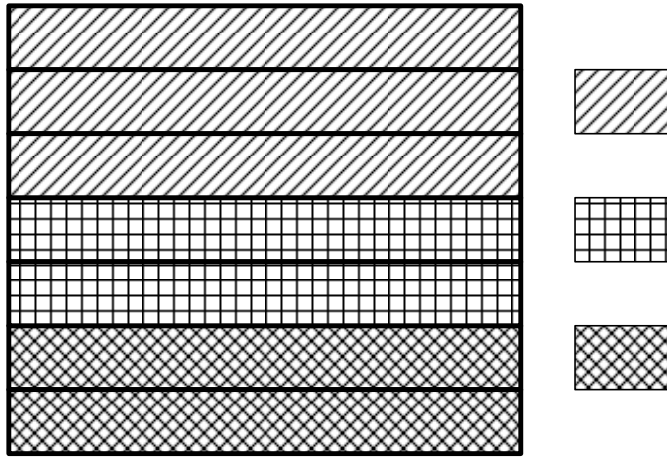
3 基于 MPI 的并行程序设计

并行的程序设计的核心问题是任务分配和通信, MPI 对通信提供了良好的支持, 所以如何分配任务成了我们主要关心的问题。这里通过对一个实例的分析和实现来简要的介绍一下如何使用 MPI 来构造一个对等模式的并行程序。

实例：求一个二维数组 (Data) 中所有元素之和。

3.1 分析

这个问题中的处理对象是一个二维数组 (下标从 1 开始), 即我们要把这个二维数组分配到各个处理器上去。在每个处理器上分别求和, 然后再将所有处理器计算出来的“部分和”累加起来, 就可以得出最后结果。如何分配呢? 按行来分配是首先想到的方法, 我们可以按处理器的个数把二维数组按行分块, 每个处理器一块。对于 7 行 ($ArraySize[0]=7$), 3 个处理器 ($size=3$) 的情况下, 分配方案如下图所示:



图十二 数组的分配

如何实现呢？假设首先，我们可以计算出每个处理器至少分配到的行数：

$$AverageLineNumber = \lfloor ArraySize[0] / size \rfloor$$

如果处理器个数不能整除行数，这样将有

$$HeavyProcessorNumber = ArraySize[0] \bmod size$$

个处理器分得 $AverageLineNumber + 1$ 行，我们规定编号小的处理器分得更多的行，这样我们就可以得到编号为 $rank$ 的处理器分得的行数：

$$MyLineNumber = \begin{cases} AverageLineNumber + 1 & rank < HeavyProcessorNumber \\ AverageLineNumber & rank \geq HeavyProcessorNumber \end{cases}$$

有了每个处理器分配的行数，就可以分配任务了。

在实际操作中会存在这样一个问题，MPI 的通信函数对传递的信息是这样组织的：给出缓冲区起始地址和缓冲区的长度。而实际的程序中，一方面，二维数组是按行或按列存放的，另一方面，数组中存放的元素一般不会达到数组的上界，就是说数组一般不会存满元素。这样就造成了数组中的元素在内存中实际上是离散存放的。如果直接给出二维数组的首地址和长度，并不能把我们预想的元素在处理器之间进行传递。解决办法是，用一维数组来存放二维数组，由程序员完成地址的映射，这样就可以顺利的解决上面的问题了。在本问题的解决中，采用了 *SendArray* 来存储 *Data* 中的元素来完成数据的传递和计算。

另外一个在实际操作中容易出错的地方是，发送缓冲区和接收缓冲区的长度的一致性问题。由于基于 MPI 的并行程序一般是由一个主控程序读取数据然后将任务分配，但是分配任务的时候，被控程序并不知道主控程序发送的数据缓冲区的长度。对这个问题的一种解决方法是，主控程序在发送数据前先发送数据缓冲区的长度；另一种方法是，由被控程序自己计算缓冲区的长度。本着减少通信的原则，通常采取第二种方法解决上述问题。这个问题尽管看起来很简单，但在实际操作中极易被忽视，应该引起重视。

上面我们提到，在处理二维数组的时候，通常先把二维数组转换为一维数组进行传递。实际中的许多问题，在被控程序计算完毕之后，可能要把结果返还给主控程序，而结果可能是根据这个一维数组计算出来的一个数组，所以在被控程序的处理过程中，往往不把一维数组还原为二维数组，而对其直接运算，以省去返还数据时的一次转换。同时，主控程序在分

配任务的时候往往采取这样的方法,将一个处理其应分配的数据由二维数组转化为一维数组后马上把这个一维数组传递出去,然后在用这个一维数组存储应分配给下一个处理器的数据。这样,如果我们选择了0号处理器作为主控程序,就必须从后向前分配任务。因为如果我们以相反的顺序分配任务的话,0号处理器首先将自己应分配的数据转化为一维数组,但这个数组它应该自己保存,所以它必须另找地方保存这个一维数组,在分配好任务之后,再将这个数组转移回来,这就造成了程序控制和存储上的冗余。而如果采取从后向前的方法分配,主控程序在分配完受控程序的数据后,刚好得到了自己应得的数据。

3.2 流程图

整个程序的大体思路是:由主控程序从文件中读出数组的信息,然后把数组的尺寸广播给所有处理器,以便每个处理器计算出自己应得的数据长度,做好接收准备。然后,主控程序分配任务,从处理器接收任务。接下来,各个处理器完成自己的计算任务。最后,主处理器将所有结果收集起来,完成最后的处理,并输出结果。流程图如图十三所示。

3.3 源代码

```
#include "mpi.h"
#include <iostream.h>
#include <fstream.h>

void main(int argc, char *argv[])
{
    //初始化 MPI 环境
    int rank, size;
    MPI_Status Status;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    //定义与数组有关的数据结构
    const int MAXX=8, MAXY=3;
    const int MAXPROCESSOR=64;
    float Data[MAXX][MAXX];
    int ArraySize[2];
    int i, j, k;

    //从文件中读入矩阵信息
    if (rank == 0)
    {
        ifstream in("input.txt");

        in >> ArraySize[0] >> ArraySize[1];

        for (i=1; i<=ArraySize[0]; i++)
            for (j=1; j<=ArraySize[1]; j++)
                in >> Data[i][j];
    }

    //广播矩阵的尺寸
```

```
MPI_Bcast(ArraySize,2,MPI_INT,0,MPI_COMM_WORLD);

//定义与任务分配有关的数据结构
int AverageLineNumber,HeavyProcessorNumber,MyLineNumber;
int CurrentLine,StartLine,SendSize;
float SendArray[MAXX*MAXY];

//计算每个处理其应分得的矩阵行数
AverageLineNumber=ArraySize[0] / size;
HeavyProcessorNumber=ArraySize[0] % size;
if (rank < HeavyProcessorNumber)
    MyLineNumber=AverageLineNumber+1;
else
    MyLineNumber=AverageLineNumber;

//如果是 0 号处理器,进行任务分配
if (rank == 0)
{
    CurrentLine=ArraySize[0];
    for (i=size-1; i >= 0; i--)
    {
        SendSize=0;
        if (i < HeavyProcessorNumber)
            StartLine=CurrentLine-AverageLineNumber;
        else
            StartLine=CurrentLine-AverageLineNumber+1;

        for (j=StartLine; j <= CurrentLine; j++)
            for (k=1; k <= ArraySize[1]; k++)
                SendArray[SendSize++]=Data[j][k];

        if (i != 0)
            MPI_Send(SendArray,SendSize,
                    MPI_FLOAT,i,10,MPI_COMM_WORLD);

        CurrentLine=StartLine-1;
    }
}
//非 0 号处理其接收分配的矩阵
else
    MPI_Recv(SendArray,MyLineNumber*ArraySize[1],
            MPI_FLOAT,0,10,MPI_COMM_WORLD,&Status);

//定义每个处理器的“部分和”变量
float *Sum=new(float);

//每个处理器,完成自己的计算任务
*Sum=0;
for (i=0; i < MyLineNumber*ArraySize[1]; i++)
    *Sum+=SendArray[i];
float AllSum[MAXPROCESSOR];

//将所有计算结果收集到 0 号处理器
MPI_Gather(Sum,1,MPI_FLOAT,AllSum,1,MPI_FLOAT,0,MPI_COMM_WORLD);

//如果是 0 号处理器,则进行最后的计算,并输出结果
if (rank == 0)
{
```

```
*Sum=0;
for (i=0; i < size; i++)
    *Sum+=AllSum[i];
cout<<"The Sum of the Array is:"<<*Sum<<endl;
}

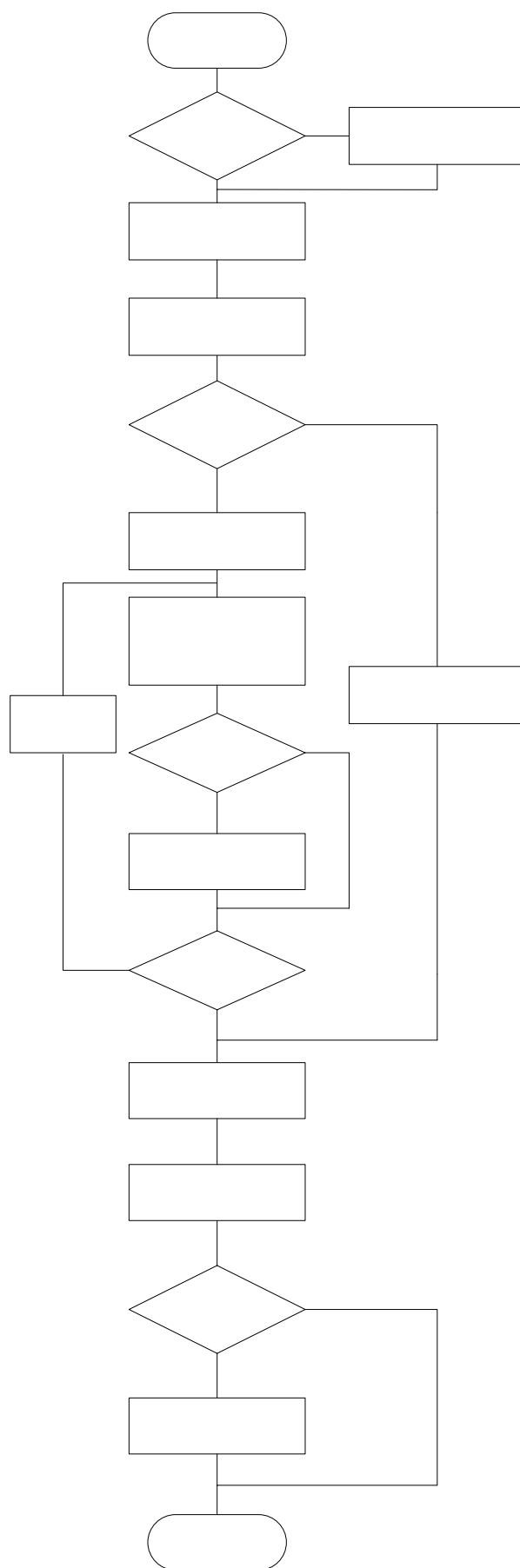
//MPI 终止
MPI_Finalize();
}
```

3.4 如何使用 MPI 并行化一个已存在的串行程序

串行程序并行化的关键是：发掘串行程序中的并行成分，合理的分配任务。其程序结构跟上述的直接构造并行程序类似。构造方法大致为：在原来的程序中添加 MPI 初始化以及终止的调用；明晰串行部分和并行部分的接口关系，尽量简化接口的数据量；在两个接口部分分别编写任务分配和数据收集的代码；最后，寻找程序结构中可以进行优化的部分，提高程序运行的效率。

4 FAQ

- Q :MPICH 有 Windows 和 Linux 的版本 ,那么可以同时两个环境下运行一个程序吗？
A :不行。同一个 MPI 程序可以分别在 Windows 和 Linux 环境下编译和运行，但却不能同时运行。这是因为 MPICH 确立和管理主机间的 Socket 连接的代码是不兼容的。
- Q :为什么得到错误信息 “ Unhandled exception caught in RedirectIOThread ” ？
A :这是由于向标准输出（屏幕）打印了过多的信息而造成的。由于在运行 MPIRun 的时候，所有主机的信息全部输出在 0 号进程所在的主机的屏幕上，如果每个程序输出了过多的信息，这就会造成上述错误。解决方法是，减少不必要的调试信息的输出。
- Q :为什么得到错误信息 “ MPI_RECV : Message truncated ” ？
A :这是由于 MPI_RECV 的接收缓冲区比对应的 MPI_SEND 的发送缓冲区小造成的，解决办法是，调整 MPI_RECV 的接收缓冲区尺寸，使其与发送缓冲区的尺寸一致。
- Q :为什么得到错误信息 “ Error 64 – GetQueuedCompletenessStatus failed ” ？
A :通常这是由于 MPI_RECV 与 MPI_SEND 不匹配造成的。解决方法是，检查每个 MPI_RECV 是否都有相应的 MPI_SEND 与之匹配。如果都匹配，再查看是否有进程非正常终止。
- Q :为什么得到错误信息 “ Failed to launch the root process: c:\cpi.exe LaunchProcess Failed, 登陆失败：未知的用户名或错误的密码 ”
A :这是由于某台运行程序的主机，没有运行 MPIRegister 进行注册，或者注册的用户名或密码不一致。



MPI初始化

当前处理器是
为0号处理器

否

广播矩阵的尺寸

各处理器计算自己
分配到的行数

当前处理器是
为0号处理器

是

i=最后一个
处理器序号

计算分配给处理
器的部分并转化成
数组

图十三 实例的流程图

$i := i - 1$

$i == 0?$