第七讲：MPI并行I/O

陈俊清

jqchen@math.tsinghua.edu.cn

清华大学数学科学系

April 18, 2013

Outline

Outline

基本术语

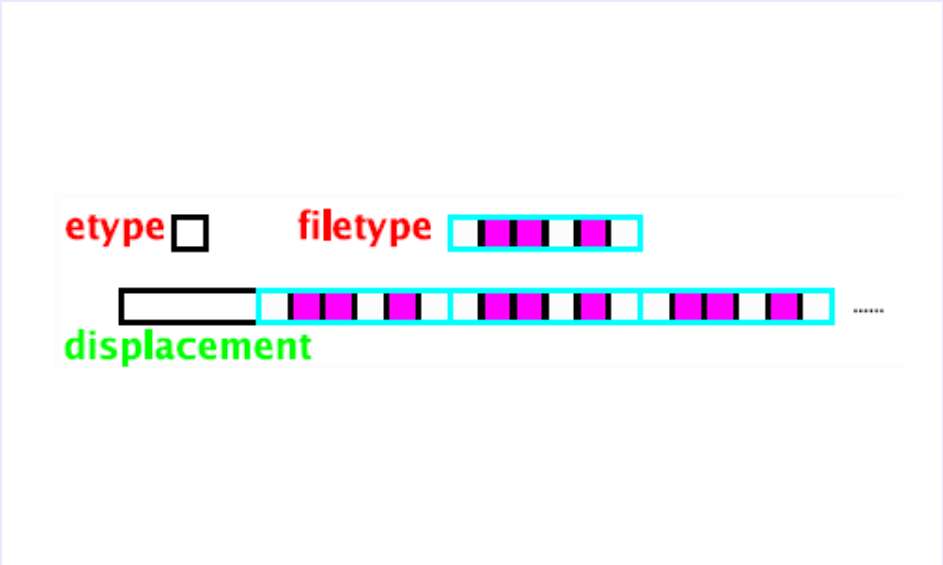
- **文件(file)**：MPI的文件可以看成具有相同或者不同类型的数据项构成的序列。 MPI支持对文件的顺序和随机访问。 MPI的文件是和进程组相关联的：MPI打开文件的函数（MPI_File_open）中要求指定一个通信器， 并且该通信器中所有进程必须同时对文件进行打开或关闭操作(聚合)。
- **起始位置(displacement)**：一个文件的起始位置指相对于文件开头以字节为单位的一个绝对地址， 它用来定义一个“文件视窗”的起始位置。（绝对偏移）

基本术语

- **基本单元类型(etype)**：基本单元类型(elementary type)是定义一个文件最小访问单元的MPI数据类型。一个文件的基本单元类型可以是任何预定义或者用户构造的并已递交的MPI数据类型，但其类型图中的位移必须非负并且位移序列是（非严格）单调上升的。MPI的文件操作完全以基本单元类型为单位：文件中的位移(offset)以基本单元类型的个数而非字节数为单位，文件指针总是指向一个基本单元的起始地址。
- **文件单元类型(filetype)**：文件单元类型也是一个MPI数据类型，它定义了对一个文件的存取图案，文件单元类型可以等于基本单元类型，也可是在基本单元类型的基础上构造并已递交的任意MPI数据类型。文件单元类型的域必须是基本单元类型的域的倍数，并且文件单元类型中间的“洞”的大小也必须是基本单元类型的域的倍数。

基本术语

- **视窗(view)**：文件视窗是指一个文件中目前可以访问的数据集。 文件视窗由三个参数定义：起始位置，基本单元类型，文件单元类型。 文件视窗指从起始位置开始将文件单元类型连续重复排列构成的图案，MPI对文件进行存取操作时将跳过图案中的空洞，见下图：



- **位移(offset)**：MPI的I/O函数中位移总是相对于文件起始位置(当前视窗)计算，并且以基本单元类型的域为单位。

基本术语

- **文件大小(file size)**：文件大小指从文件开头到文件结尾的总字节数。
- **文件指针(file pointer)**：文件指针是MPI管理的两个内部位移（隐式位移）。MPI在每个进程中为每个打开的文件定了两个文件指针，一个供本进程独立使用，称为独立文件指针（individual file pointer），另一个供打开文件的进程组中所有进程共同使用，称为共享文件指针（shared file pointer）。
- **文件句柄(file handle)**：MPI打开一个文件后，返回给调用程序一个文件句柄，供以后访问及关闭该文件时用。MPI的文件句柄在文件关闭时被释放。

基本术语

例7.1 假设ext(MPI_INT)=4, etype=MPI_INT，打开文件fh的进程组包括4个进程$p_i, i = 0, 1, 2, 3$，四个进程中文件单元类型分别定义如下：

$$p_0 : filetype = \{(int, 0), (LB, 0), (UB, 16)\}$$
$$p_1 : filetype = \{(int, 4), (LB, 0), (UB, 16)\}$$
$$p_2 : filetype = \{(int, 8), (LB, 0), (UB, 16)\}$$
$$p_3 : filetype = \{(int, 12), (LB, 0), (UB, 16)\}$$

如果四个进程中独立文件指针均为0，则调用：

```
MPI_File_read(fh,A,1,MPI_INT,status)
```

将文件开头的四个数依次赋给四个进程中变量A。

Outline

基本文件操作

打开文件:

```
int MPI_File_open(MPI_Comm comm, char *filename,
        int amode,MPI_info info, MPI_File *fh)
```

文件成功打开后，在参数fh中返回该文件的句柄，供以后对该文件进行操作。comm指定打开文件的通信器， 所有属于comm的进程必须同时调用该函数。filename是打开的文件名，comm所有进程提供的文件名必须代表同一个文件。 amode给出文件的打开模式，comm中所有进程必须提供同样的amode参数。
输入参数info提供给MPI系统一些附加提示信息，它由MPI的实现具体定义。 用户可以用常数MPI_INFO_NULL代替它，表示没有提示信息。

基本文件操作

文件打开模式:
amode参数与普通操作系统中文件访问模式类似。MPI为其定义的值有:

- MPI_MODE_RDONLY:只进行读操作;
- MPI_MODE_RDWR:同时进行读操作和写操作;
- MPI_MODE_WRONLY:只进行写操作;
- MPI_MODE_CREATE:如果文件不存在则创建一个新文件;
- MPI_MODE_EXCL:创建文件时若文件存在则打开失败;
- MPI_MODE_DELETE_ON_CLOSE:关闭文件后将其删除;
- MPI_MODE_UNIQUE_OPEN:用户可以确保只有当前程序访问该文件;
- MPI_MODE_SEQUENTIAL:只能对文件进行顺序读写;
- MPI_MODE_APPEND:打开后将文件指针置于文件结尾处。

基本文件操作

前述各模式可以用二进制“或”运算进行叠加（C中为“|”，同一模式不出现两次以上）。但是:

- MPI_MODE_CREATE、MPI_MODE_EXCL不可与MPI_MODE_RDONLY合用;
- MPI_MODE_RDWR不可与MPI_MODE_SEQUENTIAL合用。

基本文件操作

关闭MPI文件:

```
int MPI_File_close(MPI_File *fh)
```

文件关闭完成后，文件句柄被释放，fh被置成MPI_FILE_NULL。如果打开文件时使用的amode为MPI_MODE_DELETE_ON_CLOSE，则关闭后还会自动调用MPI_File_delete。用户应该确保调用该函数前所有与该文件有关的操作请求均已完成。

MPI_File_open、MPI_File_close均为聚合型函数，进程组中所有进程必须同时调用并且提供同样的参数。

基本文件操作

删除文件:

```
int MPI_File_delete(char *filename, MPI_Info info)
```

如果指定的文件不存在，则返回MPI_ERR_NO_SUCH_FILE错误。要删除的文件通常应该是没打开的或已关闭的。

基本文件操作

设定文件长度:

```
int MPI_File_set_size(MPI_File fh, MPI_Offset size)
```

将指定文件的长度（指从文件开头到文件结尾的字节数）设成size。如果当前文件长度大于size，则文件将被截断成size字节。 如果当前文件长度小于size，则文件大小被设定为指定长度，此时操作系统不一定为该文件实际分配存储空间。
事实上，该函数为一个写操作。

基本文件操作

- MPI_File_set_size是聚合型函数，进程组中所有进程必须同时调用并且提供同样的参数。
- 如果MPI_File_set_size缩小了文件长度，则允许原来的文件指针指向超过文件长度之外的范围（即不影响文件指针的使用）。
- 如果使用amode=MPI_MODE_SEQUENTIAL，则不能使用这个函数。

基本文件操作

为文件预留空间:

```
int MPI_File_preallocate(MPI_File fh, MPI_Offset size)
```

如果当前文件长度大于或等于size，则该函数不起任何作用。 否则它将文件长度调整到size指定的大小，并且强制操作系统为文件分配好存储空间。
MPI_File_preallocate是聚合型函数，进程组中所有进程必须同时调用并且提供相同的参数。

基本文件操作

查询文件长度:

```
int MPI_File_get_size(MPI_File fh, MPI_Offset *size)
```

在参数size中返回指定文件的当前长度（以字节为单位）。该操作为数据访问操作。

Outline

查询文件参数

查询打开文件的进程组:

```
int MPI_File_get_group(MPI_File fh, MPI_Group *group)
```

该函数在参数group中返回与文件句柄fh相关联（即打开该文件）的进程组的副本。 用户应该负责在不再需要该句柄时将其释放（调用MPI_Group_free）。

查询文件参数

查询文件访问模式:

```
int MPI_File_get_amode(MPI_File fh, int *amode)
```

该函数在参数amode中返回文件句柄fh所对应的文件的访问模式。

Outline

设定文件视窗

```
int MPI_File_set_view(MPI_File fh, MPI_Offset disp,
        MPI_Datatype etype, MPI_Datatype filetype,
        char *datarep, MPI_Info info)
```

将文件视窗的起始位置设为disp（从文件开头以字节为单位计算），基本单元类型设为etype，文件单元类型设为filetype。 参数datarep给出文件中的数据表示格式。参数info用来重新指定附加提示信息。

MPI_File_set_view是聚合型函数，进程组中所有进程必须同时调用。 不同进程可以提供不同的disp,filetype和info参数，但必须提供相同的datarep参数和具有相同域的etype参数。

如果文件打开是使用了MPI_MODE_SEQUENTIAL模式，则disp参数必须写成MPI_DISPLACEMENT_CURRENT(代表文件的当前位置： 共享文件指针指向的位置）。

文件中数据表示格式

参数datarep是一个字符串，给出文件中使用的数据表示格式。它有下面一些可能值:

- "native"： 文件中数据完全按其在内存中的表示形式存放。 使用该数据表示的文件不能在数据格式不兼容的计算机间交换使用。
- "internal"： 指MPI内部格式，具体由MPI的实现定义。 使用该数据表示的文件可以确保能在使用同一MPI系统的计算机件进行交换使用，即使这些计算的数据格式不兼容。
- "external32"： 使用IEEE定义的一种通用数据表示格式， external data representation(简称XDR),使用该数据表示的文件可以在所有支持MPI的计算机件交换使用。 该格式可用于在数据表示不兼容的计算机间交换数据。

目前，许多MPI系统尚未全部实现上述三种格式（它们通常只支持"native"格式）。

文件中数据表示格式

除前述数据表示外，用户还可以通过函数MPI_Register_datarep定义自己的数据表示形式。
MPI不将有关数据表示格式的信息写在文件中，因此用户须保证在设定文件窗口时指定的数据表示格式与文件中的实际表示格式相符。
特别地，当datarep不等于"native"时，基本单元类型（etype）和文件单元类型（filetype） 在文件中的形式有可能与它们在内存中的形式不一样。 此时，如果用作单元类型的数据类型是“可移植的”（portable datatype）， 则MPI在函数MPI_File_set_view中会自动对其进行调整（缩放）以便与文件中的数据表示格式相匹配。 如果用作单元类型的数据类型不是可移植的，则用户必须保证他们与文件中的数据表示格式相符， 必要时使用MPI_Type_lb和MPI_Type_ub来进行调整。

可移植数据类型

MPI中一个数据类型称为是可移植的，如果它是一个预定义数据类型，或者是一个可移植的数据类型的基础上用下述函数之一创建的:

```
MPI_Type_contiguous, MPI_Type_vector, MPI_Type_indexed,
MPI_Type_dup, MPI_Type_create_subarray,
MPI_Type_indexed_block, MPI_Type_create_darray
```

(后四个是MPI2中新增加的数据类型创建函数)。因此，可移植数据类型的位移和上下界都是以某一预定义数据类型为单位的，换言之，可移植的数据类型在其构造过程中不能使用下述函数:

```
MPI_Type_hvector, MPI_Type_hindexed, MPI_Type_struct
```

即不能直接以字节为单位来设定数据类型的位移和上下界。

查询数据类型相应于文件表示格式的域

MPI提供了一个函数来查询一个（内存中的）数据类型在文件中的域（当文件的数据表示格式不等于"native"时，数据类型在文件中的域可能与它在内存中的域不同）。

```
int MPI_File_get_type_extent(MPI_File fh,
        MPI_Datatype datatype,MPI_Aint *extent)
```

Outline

文件的读写操作

文件的读写操作函数由下表给出，其中xxxx可代表read和write，分别对应于读操作和写操作的函数。

定位方式	同步方式	进程组进程间的协同方式	
		非聚合式	聚合式
显式位移	阻塞型	MPI_xxxx_AT	MPI_xxxx_AT_ALL
	非阻塞或分裂型	MPI_Ixxxx_AT	MPI_xxxx_AT_ALL_BEGIN MPI_xxxx_AT_ALL_END
独立文件指针	阻塞型	MPI_xxxx	MPI_xxxx_ALL
	非阻塞或分裂型	MPI_Ixxxx	MPI_xxxx_ALL_BEGIN MPI_xxxx_ALL_END
共享文件指针	阻塞型	MPI_xxxx_SHARED	MPI_xxxx_ORDERED
	非阻塞或分裂型	MPI_Ixxxx_SHARED	MPI_xxxx_ORDERED_BEGIN MPI_xxxx_ORDERED_END

文件读写操作的分类

按指定数据在文件中的位置:

- 显式位移：直接在函数中指定位移量，以基本单元类型的域为单位;
- 使用独立文件指针
- 使用共享指针

每种类型的操作不会对其它类型的操作的位置产生影响，如使用显式位移的操作不会改变独立文件指针或共享文件指针， 使用独立文件指针的操作不会改变共享文件指针，而使用共享文件指针的操作也不会改变独立文件指针。

文件读写操作的分类

按进程间的协同方式：

- 非聚合式：函数的完成只依赖本进程，它们不要求进程组中的所有进程同时调用，而由各进程分别独立地调用， 当多个进程同时调用非聚合式函数时，不同进程间对数据读写的先后顺序是不确定的。
- 聚合式：完全依赖于同组所有进程间的协调，它们要求进程组中全部进程同时调用， 各进程对数据读写的先后顺序由进程的序号确定。

文件读写操作的分类

按函数是否阻塞：

- 阻塞型：阻塞型函数返回后即表明读写操作已经完成，进程马上可以对读写缓冲区进行后续操作或关闭文件。
- 非阻塞型：与非阻塞型消息传递函数类似，只向系统发出一个读或写请求， 随后需要调用MPI_Wait或 MPI_Test等函数来等待操作的完成。
- 分裂型：分裂型函数将文件的读写操作分解成开始（begin）和结束（end）两步， 以便允许进程在读写开始和结束之间进行一些其它的计算或通信。

显式位移的阻塞型文件读写

所有使用显式位移的阻塞型文件读写函数（*_at, *_at_all）的接口参数完全一样， 这里仅列出函数MPI_File_read_at的接口参数供参考.

```
int MPI_File_read_at(MPI_File fh, MPI_Offset offset,
        void *buf, int count, MPI_Datatype datatype,
        MPI_Status)
```

fh为文件句柄，offset为位移，buf，count和datatype分别为数据的缓冲区地址、个数和类型。 status返回操作结果状态（与通信函数类似）。

使用独立文件指针的阻塞型文件读写

使用独立文件指针的阻塞型文件读写函数与使用显式位移的阻塞型文件读写函数的功能完全一样，只是文件位移由独立文件指针隐式设定。这些函数的接口参数中比使用显式位移的函数少了一个offset参数，其它参数完全一样，这里给出MPI_File_read和MPI_File_write的接口参数：

```
int MPI_File_read(MPI_File fh, void *buf, int count,
        MPI_Datatype datatype, MPI_Status *status)
int MPI_File_write(MPI_File fh, void *buf, int count,
        MPI_Datatype datatype, MPI_Status *status)
```

使用独立文件指针阻塞聚合型读写

```
int MPI_File_read_all(MPI_File fh, void *buf, int count,
        MPI_Datatype datatype, MPI_Status *status)
int MPI_File_write_all(MPI_File fh, void *buf, int count,
        MPI_Datatype datatype, MPI_Status *status)
```

fh为文件句柄，buf，count和datatype分别为数据缓冲区地址、个数和类型，status返回操作结果状态。

使用共享文件指针的阻塞型文件读写

使用共享文件指针的阻塞型文件读写函数的接口参数与使用独立文件指针的阻塞型文件读写函数的接口参数完全一样

```
int MPI_File_read_ordered(MPI_File fh, void *buf,
        int count, MPI_Datatype datatype,
        MPI_Status *status)
```

由于使用共享文件指针的文件操作函数中进程组的全部进程共同使用和修改同一个文件指针， 因此这类操作非常类似于以文件为“根进程”的数据收集和散发，即它们相当于将进程组中各进程的数据块合并写入文件 （收集）或读取文件中的数据并分发给各进程（散发）。 当使用非聚合类函数MPI_File_read_shared和 MPI_File_write_shared时， 各进程从文件中读取或写入文件的数据块在文件中的相对位置是不确定的， 而聚合式函数MPI_File_read_ordered和 MPI_File_write_ordered则可以确保这些数据块在文件中严格按进程号排列。 (参看例子ex7-1.c)

非阻塞型文件读写函数

每个阻塞型非聚合式文件读写函数都有一个对应的非阻塞型函数，由阻塞型函数的函数名中在read或write前加I构成。 非阻塞型函数的接口参数中只需将对应的阻塞型函数的参数表中的status参数换成request，其它参数完全一样。

```
int MPI_File_iread_at(MPI_File fh, MPI_Offset offset,
        void *buf, int count, MPI_Datatype datatype,
        MPI_Request *request)
```

分裂型文件读写函数

MPI为每个阻塞型聚合式文件读写函数定义了一对分裂型函数，分别在阻塞型函数的函数名后面加_begin或_end构成， 分裂型函数将文件读写操作分解成开始和结束两步，用户可以在开始和结束之间插入其它通信或计算， 从而实现计算或通信与文件输入输出重叠进行。

```
int MPI_File_read_at_all_begin(fh,offset,buf,
        count, datatype)
int MPI_File_read_at_all_end(fh,buf, status)
... ...
```

Outline

独立文件指针操作

移动独立文件指针操作:

```
int MPI_File_seek(MPI_File fh, MPI_Offset offset,
        int whence)
```

改变独立文件指针的位置，参数whence可取下列值:

- MPI_SEEK_SET: 将指针的位置设为offset
- MPI_SEEK_CUR: 将指针的位置设为当前位移加上offset
- MPI_SEEK_END: 将指针的位移设为文件结尾加上offset

参看程序ex7-2.c

独立文件指针操作

查询独立文件指针的当前位移:

```
int MPI_File_get_position(MPI_File fh,
        MPI_Offset *offset)
```

在参数offset中返回独立文件指针的位移。

共享文件指针操作

移动文件指针：

```
int MPI_File_seek_shared(MPI_File fh, MPI_Offset offset,
        int whence)
```

改变共享文件指针的位置，参数whence可取下列值：

- MPI_SEEK_SET：将指针的位置设为offset
- MPI_SEEK_CUR：将指针的位置设为当前位移加上offset
- MPI_SEEK_END：将指针的位移设为文件结尾加上offset

MPI_File_seek_shared是聚合型函数，进程组中所有进程必须同时调用并且提供相同的参数。

共享文件指针操作

查询文件指针的当前位移:

```
int MPI_File_get_position_shared(MPI_File fh,
        MPI_Offset *offset)
```

在参数offset中返回共享文件指针的位移。

文件位移在文件中的绝对地址

```
int MPI_File_get_byte_offset(MPI_File fh,
        MPI_Offset offset, MPI_Offset *disp)
```

该函数将以etype为单位相对于当前文件视窗的位移（offset）换算成以字节为单位从文件开头计算的绝对地址（disp）。

Outline

读写操作的相容性

当单个或多个进程同时对同一个文件进行访问时，MPI称这些访问是相容的， 如果这些访问可以等效地看成是以某种顺序依次进行的，即便他们的先后顺序是不确定的。 换言之，对同一个文件的多个访问是相容的， 如果他们中的任一访问都不会在操作过程中间由于被另一个访问打断或干扰而影响到访问的结果。
MPI系统允许用户将对一个文件的访问设置成具有“原子性”（atomicity，意即不可分的） 来保证属于与该文件关联的进程组中的进程对该文件的访问的相容性。

设定文件访问的原子性

```
int MPI_File_set_atomicity(MPI_File fh, int flag)
```

该函数设定是否需要保证打开文件的进程组中进程对该文件的访问的原子性。 当flag为true时，MPI系统将保证文件访问的原子性从而保证属于（与该文件相关联的）同一进程组的进程对该文件的访问的相容性。而当flag为false时，MPI不保证对文件访问的原子性，而需要用户通过其它途径来保证对文件的不同访问间的相容性。
MPI_File_set_atomicity是聚合型函数，进程组中所有进程必须同时调用并且提供相同的参数。

设定文件访问的原子性

例：在文件的同一个位置上一个进程写、另一个进程读。

```
MPI_Status status;
MPI_File fh;
int A[10];
... ...
MPI_File_open(MPI_COMM_WORLD, 'myfile',
        MPI_MODE_RDWR|MPI_MODE_CREATE,MPI_INFO_NULL,&fh);
MPI_File_set_view(fh,0,MPI_INT,'native',MPI_INFO_NULL);
MPI_File_set_atomicity(fh,1);
if(myrank == 0){
    for(i=0;i<10;i++)
        A[i]=5;
    MPI_File_write_at(fh, 0, A, 10, MPI_INT,&status);
}
else if(myrank == 1)
    MPI_File_read_at(fh, 0 , A, 10, MPI_INT,&status);
```

设定文件访问的原子性

在该例中，因为atomocity被设为true，因此1进程将总是读到0个数或10个5。如果改变上面的程序将atomocity设为false， 则进程1读到的结果是不确定的，它与具体的MPI实现和程序运行过程有关。 (参看程序ex7-3.c)

查询atomicity的当前值

```
int MPI_File_get_atomicity(MPI_File fh, int *flag)
```

该函数在参数flag中返回atomicity的当前值。

文件读写与存储设备间的同步

```
int MPI_File_sync(MPI_File fh)
```

该函数确保将调用它的进程新写入文件的数据写入存储设备。如果文件在存储设备中的内容已经被其它进程改变， 则调用该函数可以确保调用它的进程随后读该文件时得到的是改变后的数据。 调用该函数时不能有尚未完成的对该文件的非阻塞型或分裂型读写操作。
注意，如果打开文件的进程组的两个进程中一个进程往文件中写入一组数据， 另一个进程希望从文件的同一位置读到这组数据，则各进程可能需要调用两次MPI_File_sync。 并在两次调用中间进行一次同步MPI_Barrier。第一次MPI_File_sync的调用确保第一个进程写的数据被写入存储设备， 而第二次调用则可确保写入存储设备的数据被另一个进程读到。
MPI_File_sync是聚合型函数，进程组中所有进程必须同时调用并且提供相同的参数。

子数组数据类型创建函数

```
int MPI_Type_create_subarray(int ndims,
        int array_of_sizes[], int array_of_subsizes[],
        int array_of_starts[], int order,
        MPI_Datatype oldtype, MPI_Datatype *newtype)
```

这是一个辅助数据类型创建函数，可以用于对分布式数组的读写操作。该函数创建一个“子数组”数据类型，即描述一个n维（全局）数组中的一个n维子数组。 创建的新数据类型的域对应于全局数组，类型中数据的位移由子数组的元素在全局数组中的位移确定。

子数组数据类型创建函数

参数ndims给出数组的维数，array_of_sizes[i]给出全局数组第i维的大小。array_of_subsizes[i]给出子数组第i维的大小，array_of_starts[i]给出子数组第i维在全局数组中的起始位置。 参数order给出数组元素的排列顺序，order=MPI_ORDER_C表示数组元素按C的数组顺序排列，order=MPI_ORDER_FORTRAN表示数组元素按FORTRAN的数组顺序排。oldtype给出数组元素的数据类型。 newtype返回所创建的子数组数据类型的句柄。 子数组各维的大小必须大于0并且小于或等于全局数组相应维的大小。 子数组的起始位置可以是全局数组中的任何位置，但必须确保子数组被包含在全局数组中，否则出错。
如果数据类型oldtype是可移植的，则新数据类型也是可移植的。

子数组数据类型创建函数

例：在之前的jacobi迭代中我们忽略了近似解的输出， 这里我们用本次课中介绍的MPI并行I/O函数实现近似解的并行输出，特别地，我们要求输出的近似解按自然序排列， 且包含边界节点。
参看例子程序jacobi3.c