Advanced MPI

Sebastian von Alfthan
CSC – the Finnish IT Center for Science
PRACE summer school

CSC

MPI-2

- Dynamic process management
  - Ability to start a MPI-processes during run time
  - Ability to connect to a separately started MPI-process
  - Collectives for inter-communicators
  - Not implemented on Cray
- One-sided communication
  - Read/write to memory of another process
  - Performance not optimized in many implementation
  - On Cray it has no performance benefits
- MPI-I/O: Parallel-I/O
- Other improvements
  - Better support for threads
  - Language interoperability
  - F9x/C++ support
  - Better support of user defined types

User defined datatypes

- **Standard MPI datatypes**
  - **Enable communication using contiguous memory sequence of identical elements (e.g. matrix)**
- **User defined datatypes can describe**
  - **Non-contiguous memory blocks (e.g. certain elements in a matrix)**
  - **Heterogenous data (structs in C, types in Fortran)**
- **User defined datatypes required for advanced use of MPI-I/O**
- **Higher level of programming is achieved**
  - **Code is more compact and maintainable**
  - **Performance is dependent on MPI-implementation**

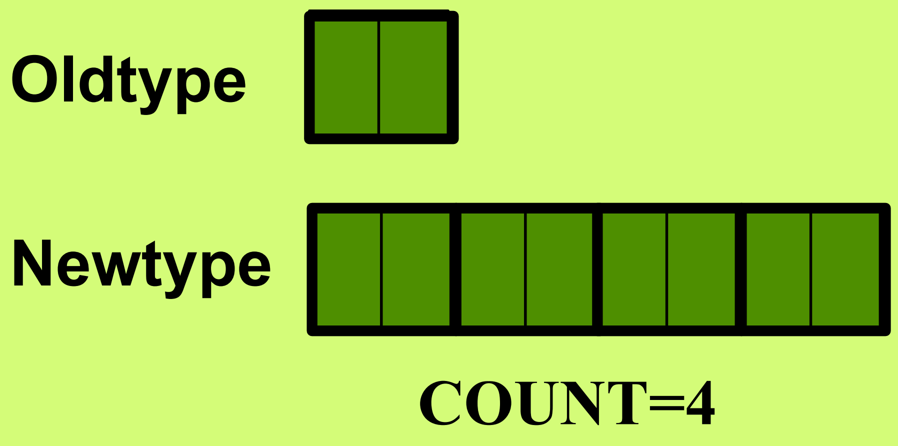
Creating a user defined datatype

- **A datatype is defined by a sequence of primitive datatypes and a sequence of displacements**
- **A new datatype is created from existing ones with a datatype constructor**
  - **Several different commands for different special cases**
- **A new datatype must be committed before using it.**
  - **F90: CALL MPI_TYPE_COMMIT(NEWTYPE,ERR)**
  - **C: err = MPI_Type_commit(&newtype)**
- **A type can be freed after it is no longer needed.**
  - **F90: CALL MPI_TYPE_FREE(NEWTYPE, ERR)**
  - **C: err = MPI_Type_free(&newtype)**
- **User defined datatypes can not be used for defining variables.**

Datatype constructors: MPI_TYPE_CONTIGUOUS

- **MPI_TYPE_CONTIGUOUS**
  - **Creates a new type from a contiguous list of identical elements, such as array column in Fortran or row in C.**
  - **F9x: MPI_TYPE_CONTIGUOUS(COUNT, OLDTYPE, NEWTYPE, ERR)**
    - **NEWTYPE is an INTEGER representing the new type**
    - **COUNT: Number of OLDTYPE elements.**
    - **OLDTYPE: Type of constructing elements (MPI datatype)**

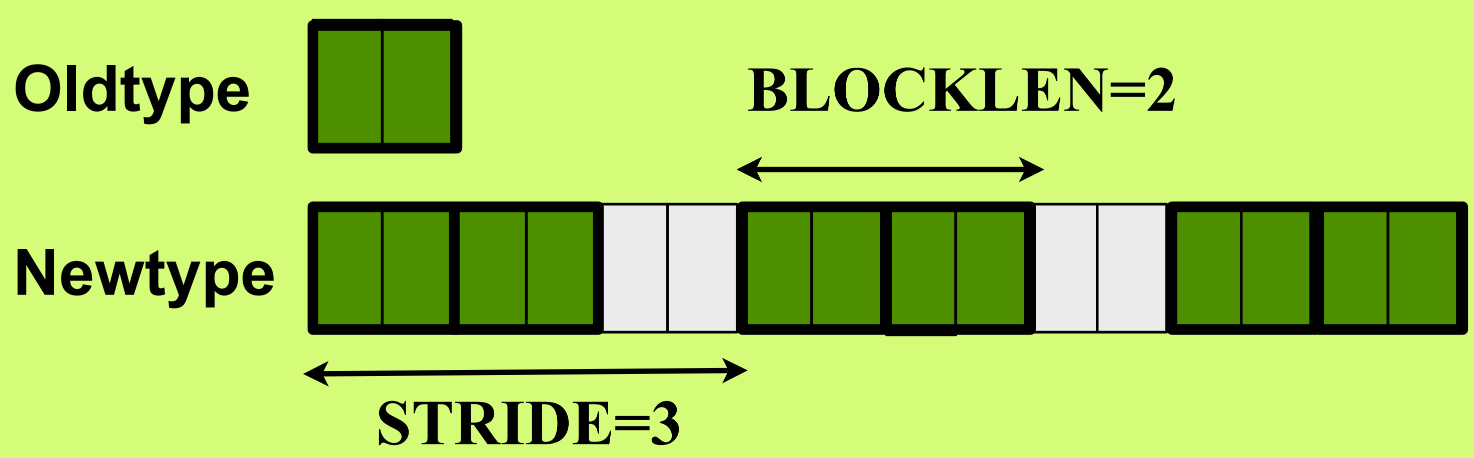
MPI_TYPE_CONTIGUOUS(4,OLDTYPE,NEWTYPE,ERR)



Datatype constructors: MPI_TYPE_VECTOR

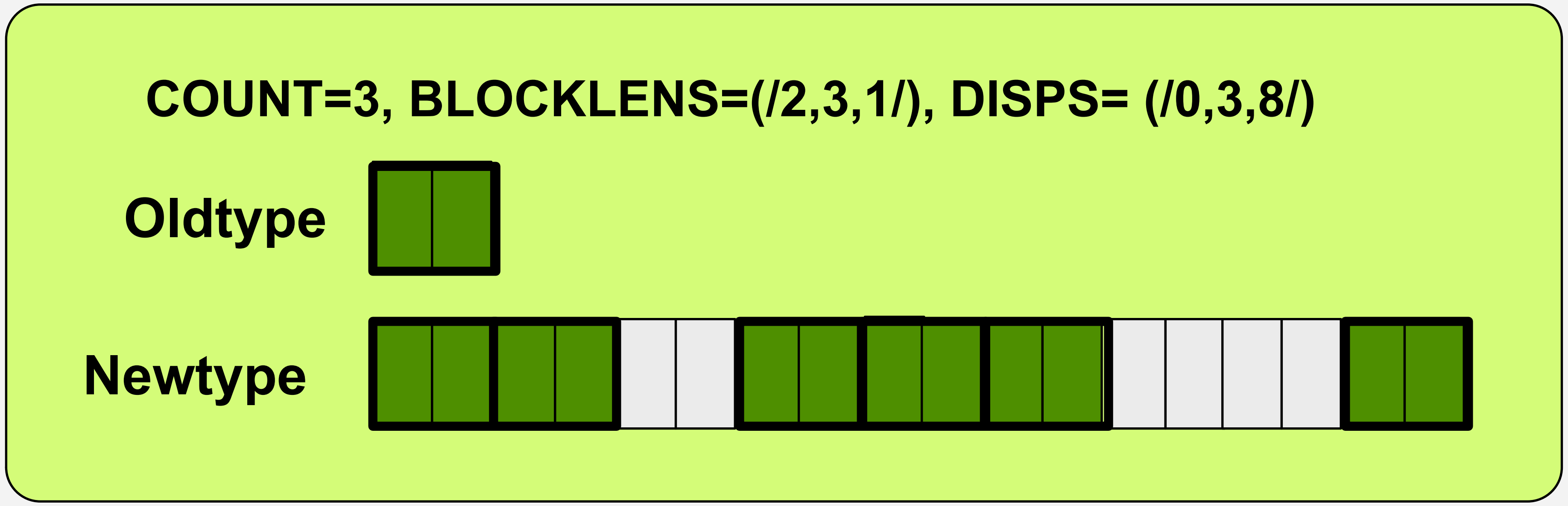
- **MPI_TYPE_VECTOR**
  - **Creates a new type from equally spaced identical blocks**
  - **F9x: MPI_TYPE_VECTOR(COUNT,BLOCKLEN, STRIDE,OLDTYPE,NEWTYPE,ERR)**
    - **COUNT=number of blocks**
    - **BLOCKLEN=number of elements in each block**
    - **STRIDE=displacement between the blocks in number of OLDTYPE elements**
- **MPI_TYPE_CREATE_HVECTOR**
  - **As MPI_TYPE_VECTOR, but STRIDE is in bytes**

MPI_TYPE_VECTOR(3,2,3,OLDTYPE,NEWTYPE,ERR)



Datatype constructors: MPI_TYPE_INDEXED

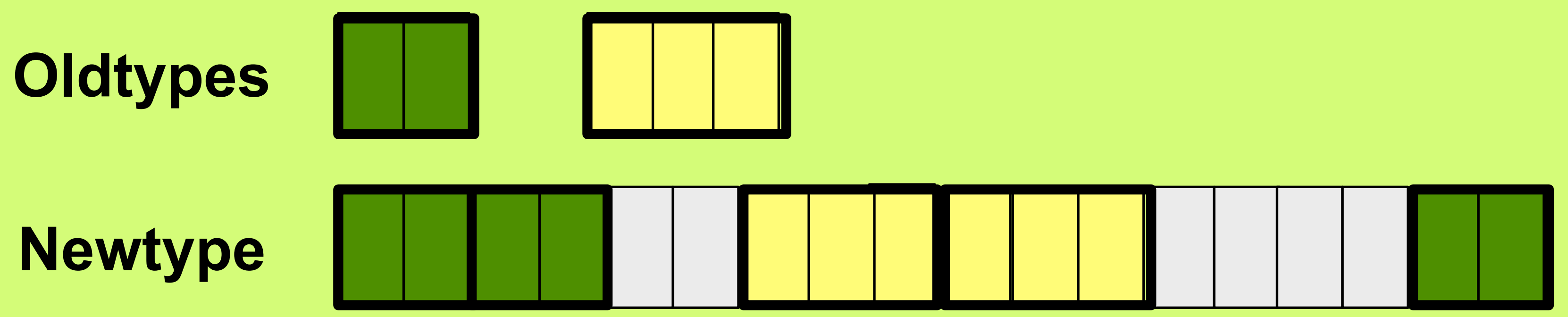
- **MPI_TYPE_INDEXED**
  - **Creates a new type from blocks comprising identical elements. The size and displacements of the blocks can vary (e.g. upper triangle of a matrix)**
  - **F9x: MPI_TYPE_INDEXED(COUNT, BLOCKLENS, DISPS,OLDTYPE,NEWTYPE, ERR)**
    - **BLOCKLENS=lengths of the blocks (array)**
    - **DISPLS=displacements (array) in OLDTYPES**
- **MPI_TYPE_CREATE_HINDEXED**
  - **As MPI_TYPE_INDEXED but displacements in bytes**



Datatype constructors: MPI_TYPE_CREATE_STRUCT

- **MPI_TYPE_CREATE_STRUCT**
  - **Most general type constructor.**
  - **Creates a new type from heterogeneous blocks**
  - **E.g. Fortran 77 common blocks, Fortran 9x and C structures.**
  - **F9x: MPI_TYPE_STRUCT(count, array_of_blocklengths, array_of_disp,array_of_types, newtype, error)**
    - **count,array_of_blocklengths:** as earlier (integer)
    - **array_of_disp:** Displacements in bytes (integer(KIND=MPI_ADDRESS_KIND)
    - **array_of_types:** Array of block types
- **MPI_GET_ADDRESS can be used to calculate displacement**

COUNT=3, BLOCKLENS=(/2,2,1/), DISPS= (/0,6,16/)



Example: **send an array leaving out every third number**

```
CALL MPI_TYPE_VECTOR(m,2,3,MPI_INTEGER,newtype,er)
CALL MPI_TYPE_COMMIT(newtype,er)
IF(myid==0) THEN
  a=(/ (i,i=1,n) /)
  CALL MPI_SEND(a,1,newtype,1,tag,MPI_COMM_WORLD,er)
  WRITE(*,'(A12,12I3)') "Sent:",a(1:12)
ELSE IF(myid==1) THEN
  a=0
  CALL MPI_RECV(a,1,newtype,0,tag,MPI_COMM_WORLD,status,er)
  WRITE(*,'(A12,12I3)') "Received:", a(1:12)
END IF
CALL MPI_TYPE_free(newtype,er)
```

```
Sent:        1  2  3  4  5  6  7  8  9 10 11 12
Received:    1  2  0  4  5  0  7  8  0 10 11  0
```

Example: **send an array leaving out every third number**

```
CALL MPI_TYPE_VECTOR(1,2,3,MPI_INTEGER,newtype,er)
CALL MPI_TYPE_COMMIT(newtype,er)
IF(myid==0) THEN
  a=(/ (i,i=1,n) /)
  CALL MPI_SEND(a,m,newtype,1,tag,MPI_COMM_WORLD,er)
  WRITE(*,'(A12,12I3)') "Sent:",a(1:12)
ELSE IF(myid==1) THEN
  a=0
  CALL MPI_RECV(a,m,newtype,0,tag,MPI_COMM_WORLD,status,er)
  WRITE(*,'(A12,12I3)') "Received:", a(1:12)
END IF
CALL MPI_TYPE_free(newtype,er)
```

```
Sent:        1  2  3  4  5  6  7  8  9 10 11 12
Received:    1  2  3  4  5  6  7  8  9 10 11 12
```

Extent of datatypes

- **The extent of a datatype defines how a sequence of elements are layed out in memory; it's the distance between subsequent elements**
- **Important to understand in order to send/recv more than one element of a user defined type**
- **In preceding example the extent of newtype was 2 x size of integer, not 3 x size of integer**
- **MPI_TYPE_CREATE_RESIZED is used to create a new type with user defined extent**
- **F9x: MPI_TYPE_CREATE_RESIZED(oldtype,lowerbound,extent,newtype,err)**
  - **oldtype: The old type (INTEGER)**
  - **lowerbound: Normally 0 (INTEGER(KIND=MPI_ADDRESS_KIND))**
  - **extent: Extent of newtype (INTEGER(KIND=MPI_ADDRESS_KIND))**
  - **newtype: copy of oldtype with new extent**
- **This is in MPI-2 style, MPI-1 uses another depreciated way**

Example: **send an array leaving out every third number**

```
   INTEGER(KIND=MPI_ADDRESS_KIND)::loc(2),displ
...
  CALL MPI_TYPE_CONTIGUOUS(2,MPI_INTEGER,temptype,ierror)
  CALL MPI_TYPE_COMMIT(temptype,ierror)
  CALL MPI_GET_ADDRESS(a(1),loc(1),ierror)
  CALL MPI_GET_ADDRESS(a(4),loc(2),ierror)
  displ=loc(2)-loc(1)
  CALL MPI_TYPE_CREATE_RESIZED(temptype,0,displ,newtype,ierror)
  CALL MPI_TYPE_COMMIT(newtype,ierror)
  CALL MPI_TYPE_free(temptype,ierror)
...
```

```
Sent:      1  2  3  4  5  6  7  8  9 10 11 12
Received:  1  2  0  4  5  0  7  8  0 10 11  0
```

Performance

- **Overhead is potentially reduced by:**
  - **Sending one long message instead of many small messages**
  - **Avoiding packing of data in buffers**
- **Some implementations are slow**
- **Performance should be tested on target platforms**
- **Example: Sending integers between two processes**
  - **Cray XT4 - mpi_type_vector with blocksize=2 and stride=20**
  - **Performance with user defined type 50% slower than sending same amount of data without any striding**
  - **Performance almost 10x better than naive manual packing**

MPI-I/O

- **Writing large output or scratch files is very slow**
  - **Flops are cheap, I/O is not!**
- **Alternatives:**
  - **One process takes care of all I/O. Increases communication and is slow.**
  - **Each process writes its local results to a separate file. Works for scratch but not for output files**
  - **MPI I/O: Scalable, standardized. Processes can access their own portions of a single file.**

MPI-I/O: Open/Close file

- **All processes in a communicator open a file using**
  - **MPI_FILE_OPEN(comm,filename,mode,info,fpointer,ierror)**
  - **comm: Communicator that performs parallel I/O**
  - **mode**
    - **MPI_MODE_RDONLY, MPI_MODE_WRONLY, MPI_MODE_CREATE, ...**
    - **Can be combined with + in Fortran, | in C/C++**
  - **Info:**
    - **Hints to implementation for optimal performance**
    - **No hints: MPI_INFO_NULL**
  - **fpointer: Parallel file pointer**
- **File closed using**
  - **MPI_FILE_CLOSE(fpointer,ierror)**

MPI-I/O: Read file

- File opened with MPI_MODE_RDONLY
- Change location of individual file pointer in file
  - MPI_FILE_SEEK(fpointer,disp, whence, err)
  - whence: MPI_SEEK_SET, MPI_SEEK_CUR, MPI_SEEK_END, ...
  - disp: Displacement in bytes (with default file view)
    - F9x type: INTEGER(KIND=MPI_OFFSET_KIND)
    - C type: MPI_Offset
- Read file at individual file pointer
  - MPI_FILE_READ(fpointer,buf,count,datatype, status, err)
  - Updates position of file pointer after reading
  - Not thread safe
- Determine location within the read statement (explicit offset)
  - CALL MPI_FILE_READ_AT(fpointer, disp, buf, count, datatype, status, err)
  - Thread-safe
- Amount of data read can be determined with MPI_GET_COUNT

MPI-I/O: Write file

- **Similar to reading**
- **File opened with MPI_MODE_WRONLY or MPI_MODE_CREATE**
- **Write file at individual file pointer**
  - **MPI_FILE_WRITE(fpointer,buf,count,datatype, status, err)**
  - **Updates position of file pointer after writing**
  - **Not thread safe**
- **Determine location within the write statement (explicit offset)**
  - **CALL MPI_FILE_WRITE_AT(fpointer, disp, buf, count, datatype, status, err)**
  - **Thread-safe**

Example

```
PROGRAM Output
 USE MPI
 IMPLICIT NONE
 INTEGER :: err, i, myid, file, intsize
 INTEGER :: status(MPI_STATUS_SIZE)
 INTEGER, PARAMETER :: count=100
 INTEGER, DIMENSION(count) :: buf
 INTEGER(KIND=MPI_OFFSET_KIND) :: disp
 CALL MPI_INIT(err)
 CALL MPI_COMM_RANK(MPI_COMM_WORLD, myid,&
   err)
 DO i = 1, count
  buf(i) = myid * count + i
 END DO
...
```

- **Multiple processes write to a binary file test.**
- **First process writes integers 1-100 to the beginning of the file, etc.**

```
...
 CALL MPI_FILE_OPEN(MPI_COMM_WORLD, 'test', &
      MPI_MODE_WRONLY + MPI_MODE_CREATE, &
      MPI_INFO_NULL, file, err)
 CALL MPI_TYPE_SIZE(MPI_INTEGER, intsize,err)
 disp = myid * count * intsize
 CALL CALL MPI_FILE_SEEK(file,disp,&
      MPI_SEEK_SET, err)
 CALL MPI_FILE_WRITE(file, buf, count, &
      MPI_INTEGER, status, err)
 CALL MPI_FILE_CLOSE(file, err)
 CALL MPI_FINALIZE(err)
END PROGRAM Output
```

MPI-I/O: Contiguous vs. non-contiguous

- **Contiguous access (previous example)**
  - **Each process accesses a contiguous slab of data**
  - **Very much like normal unix-like I/O read/write**
- **Non-Contiguous access**
  - **Each process has to access small pieces of data scattered throughout a file**
  - **Very expensive if implemented with separate reads/writes**
  - **Use file view's to implement the non contiguous access**

File view

- **Defines which part of a file is visible to a process**
  - **Non-contiguous file views defined with user defined datatype**
- **Defines type of data that is accessed**
  - **Useful for portability**
  - **Defines unit for offsets**
- **Default file view**
  - **Whole file is visible**
  - **All offsets are in bytes**

File view

- MPI_FILE_SET_VIEW(file, disp, etype, filetype,datarep,info, err)
  - disp: Offset from beginning of file. Always in bytes
  - etype:
    - MPI type or user defined type
    - Basic unit of data access
    - Offsets in I/O commands in units of etype
  - filetype:
    - Same type as etype or user defined type constructed of etypes
    - Specifies which part of the file is visible
  - datarep:
    - Data representation, sometimes useful for portability
    - “native”: store in same format as in memory
  - info:
    - Hints for implementation that can improve performance
    - MPI_INFO_NULL: No hints

Example: file view with contiguous data

```
PROGRAM Output
 USE MPI
 IMPLICIT NONE
 INTEGER :: err, i, myid, file, intsize
 INTEGER :: status(MPI_STATUS_SIZE)
 INTEGER, PARAMETER :: count=100
 INTEGER, DIMENSION(count) :: buf
 INTEGER(KIND=MPI_OFFSET_KIND) :: disp
 CALL MPI_INIT(err)
 CALL MPI_COMM_RANK(MPI_COMM_WORLD, myid,&
   err)
 DO i = 1, count
  buf(i) = myid * count + i
 END DO
...
```

- **Multiple processes write to a binary file test.**
- **First process writes integers 1-100 to the beginning of the file, etc.**

```
...
 CALL MPI_FILE_OPEN(MPI_COMM_WORLD, 'test', &
      MPI_MODE_WRONLY + MPI_MODE_CREATE, &
      MPI_INFO_NULL, file, err)
 CALL MPI_TYPE_SIZE(MPI_INTEGER, intsize,err)
 disp = myid * count * intsize
 CALL MPI_FILE_SET_VIEW(file, disp, &
      MPI_INTEGER, MPI_INTEGER, 'native', &
      MPI_INFO_NULL, err)
 CALL MPI_FILE_WRITE(file, buf, count, &
      MPI_INTEGER, status, err)
 CALL MPI_FILE_CLOSE(file, err)
 CALL MPI_FINALIZE(err)
END PROGRAM Output
```

Example: file view with non-contiguous data

```
...
 CALL MPI_FILE_OPEN(MPI_COMM_WORLD, 'test', &
      MPI_MODE_WRONLY + MPI_MODE_CREATE, &
      MPI_INFO_NULL, file, err)
 CALL MPI_TYPE_SIZE(MPI_INTEGER, intsize,err)
 CALL MPI_TYPE_VECTOR(count blocksize,blocksize,&
     nproc*blocksize,MPI_Integer,filetype,err)
 CALL MPI_COMMIT(filetype,err)
 etype=MPI_INTEGER
 disp = myid *  intsize * blocksize
 CALL MPI_FILE_SET_VIEW(file, disp, &
      MPI_INTEGER, MPI_INTEGER, 'native', &
      MPI_INFO_NULL, err)
 CALL MPI_FILE_WRITE(file, buf, count, &
      MPI_INTEGER, status, err)
 CALL MPI_FILE_CLOSE(file, err)
...
```

- **Multiple processes write to a binary file test.**
- **Each process writes cyclicly blocksize integers to file**

Collective operations

- **MPI_FILE_READ_ALL**
- **MPI_FILE_WRITE_ALL**
- **Same parameters as in independent I/O functions**
  - **MPI_FILE_READ**
  - **MPI_FILE_WRITE**
- **All processes in communicator that opened file must call function**
- **Performance potentially better than for individual functions**
  - **Even if each processor reads a non-contiguous segment, in total the read is contiguous**

Example: file view with non-contiguous data

```
...
 CALL MPI_FILE_OPEN(MPI_COMM_WORLD, 'test', &
      MPI_MODE_WRONLY + MPI_MODE_CREATE, &
      MPI_INFO_NULL, file, err)
 CALL MPI_TYPE_SIZE(MPI_INTEGER, intsize,err)
 CALL MPI_TYPE_VECTOR(count/blocksize,blocksize,&
     nproc*blocksize,MPI_Integer,filetype,err)
 CALL MPI_COMMIT(filetype,err)
 etype=MPI_INTEGER
 disp = myid *  intsize * blocksize
 CALL MPI_FILE_SET_VIEW(file, disp, &
      MPI_INTEGER, MPI_INTEGER, 'native', &
      MPI_INFO_NULL, err)
 CALL MPI_FILE_WRITE_ALL(file, buf, count, &
      MPI_INTEGER, status, err)
 CALL MPI_FILE_CLOSE(file, err)
...
```

- **Multiple processes write to a binary file test.**
- **Each process writes count integers cyclicly in blocks of blocksize integers**
- **With blocksize=1000 this is 22x faster on Cray XT4**

Performance

- **Use collective operations if possible**
- **Use derived datatypes if non-contiguous access is required**
- **Get to know hints that are useful on your platform**
- **Get to know tools and parameters that can be used on a filesystem level**

Shared pointers

- Value is shared between all processes in communicator. If one process writes/reads, the location is updated for all processes
- Blocking functions: MPI_File_seek/write/read_shared
- Non-blocking: MPI_File_iwrite/iread_shared
- Collective: MPI_File_read/write_ordered
- Still parallel-I/O
- Useful for logs, among other things

Questions!