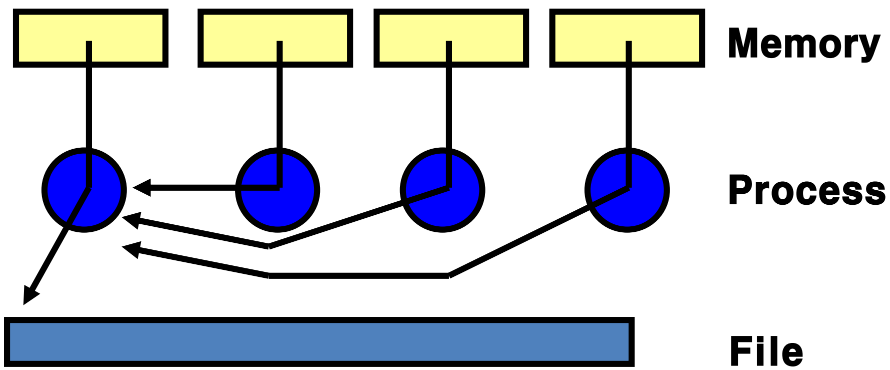
MPI-IO

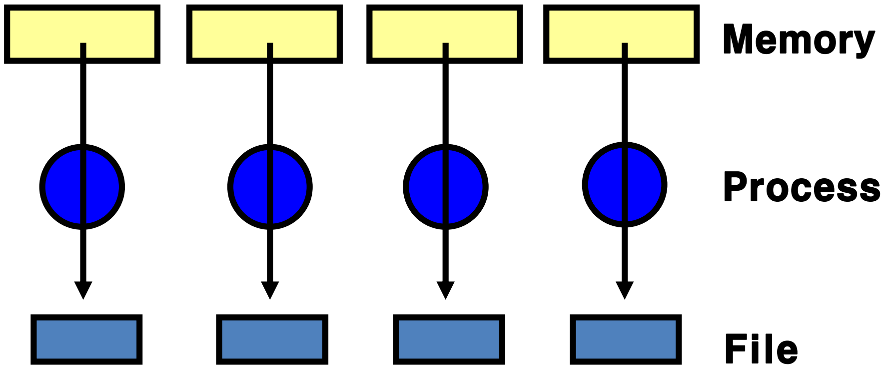
Ji-Hoon Kang
(jhkang@kisti.re.kr)

Parallel file I/O

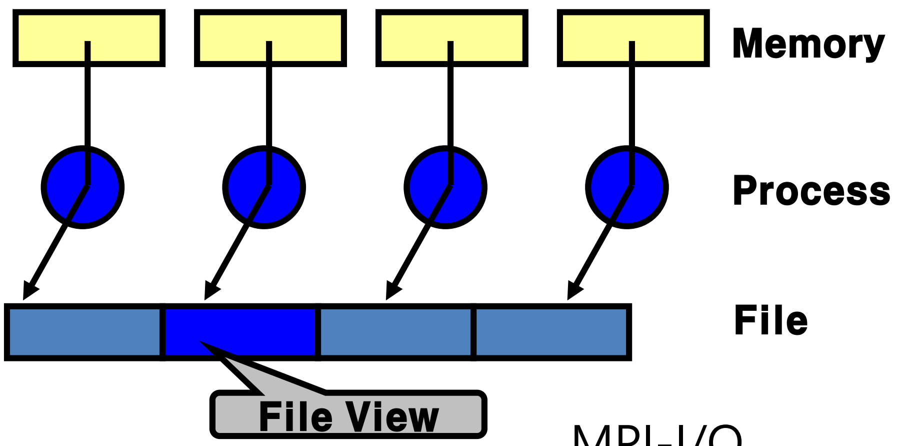
- Three I/O mode in parallel file write



Post Reassembly

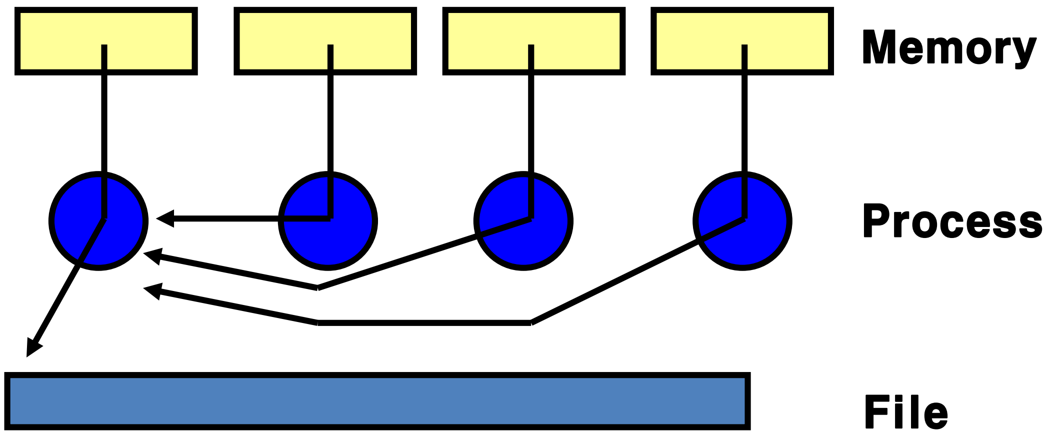


Single Task I/O



MPI-I/O

Post Reassembly



- **Example : write 100 integers which each process owns**

1. Initialization of temporal buffer, buf(), for each process
2. All process except process 0 send buf() to process 0
3. Process 0 write its own array to file at first, receive buf() from other processes and write it to file in turn.

Example of post reassembly (I)

- Fortran

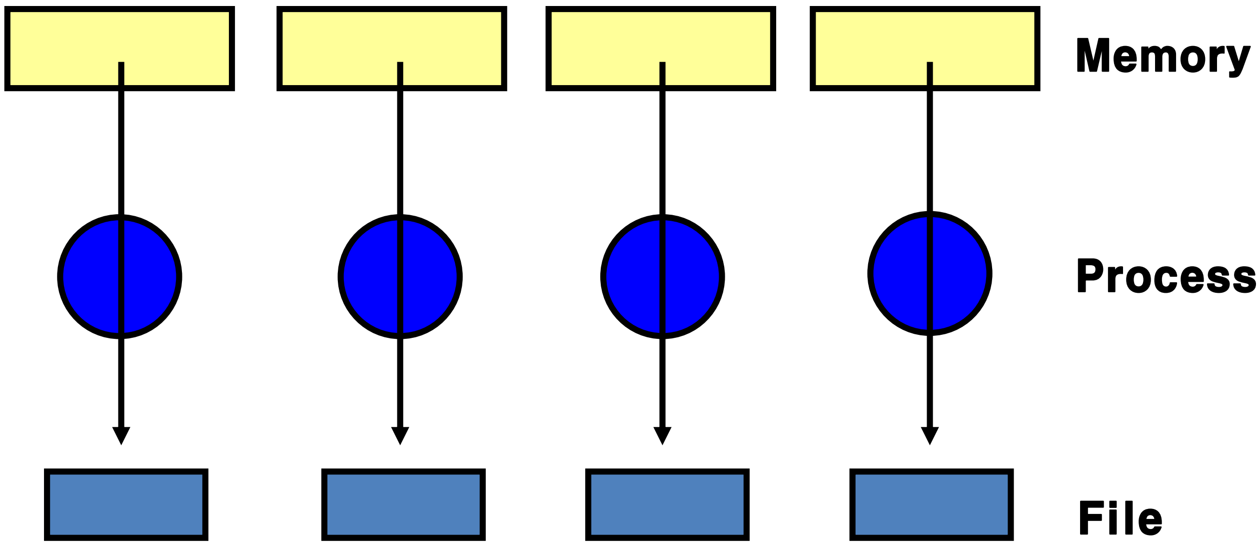
```
PROGRAM serial_IO1
INCLUDE 'mpif.h'
INTEGER BUFSIZE
PARAMETER (BUFSIZE = 100)
INTEGER nprocs, myrank, ierr, buf(BUFSIZE)
INTEGER status(MPI_STATUS_SIZE)

Call MPI_INIT(ierr)
Call MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
Call MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
DO i = 1, BUFSIZE
    buf(i) = myrank * BUFSIZE + i
ENDDO
IF (myrank /= 0) THEN
  CALL MPI_SEND(buf, BUFSIZE, MPI_INTEGER, 0, 99, MPI_COMM_WORLD, ierr)
ELSE
  OPEN (UNIT=10,FILE="testfile",STATUS="NEW",ACTION="WRITE")
  WRITE(10,*) buf
  DO i = 1, nprocs-1
    CALL MPI_RECV(buf, BUFSIZE, MPI_INTEGER, i, 99,MPI_COMM_WORLD, status, ierr)
    WRITE (10,*) buf
  ENDDO
ENDIF
CALL MPI_FINALIZE(ierr)
END
```

Example of post reassembly (II)

- **C**

```
/*example of serial I/O*/
#include <mpi.h>
#include <stdio.h>
#define BUFSIZE 100
void main (int argc, char *argv[]){
int i, nprocs, myrank, buf[BUFSIZE] ;
MPI_Status status;
FILE *myfile;

MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
for(i=0; i<BUFSIZE; i++)
    buf[i] = myrank * BUFSIZE + i;
if(myrank != 0)
    MPI_Send(buf, BUFSIZE, MPI_INT, 0, 99, MPI_COMM_WORLD);
else{
    myfile = fopen("testfile", "wb");
    fwrite(buf, sizeof(int), BUFSIZE, myfile);
    for(i=1; i<nprocs; i++){
        MPI_Recv(buf, BUFSIZE, MPI_INT, i, 99, MPI_COMM_WORLD, &status);
        fwrite(buf, sizeof(int), BUFSIZE, myfile);
    }
    fclose(myfile);
  }
MPI_Finalize();
}
```

Single Task I/O



- Each process opens its own file with different name (ex: file.(myrank))
- Additional pre/post processing is required for data processing
- Most efficient in terms of file I/O itself.

Example of single task I/O (I)

- Fortran

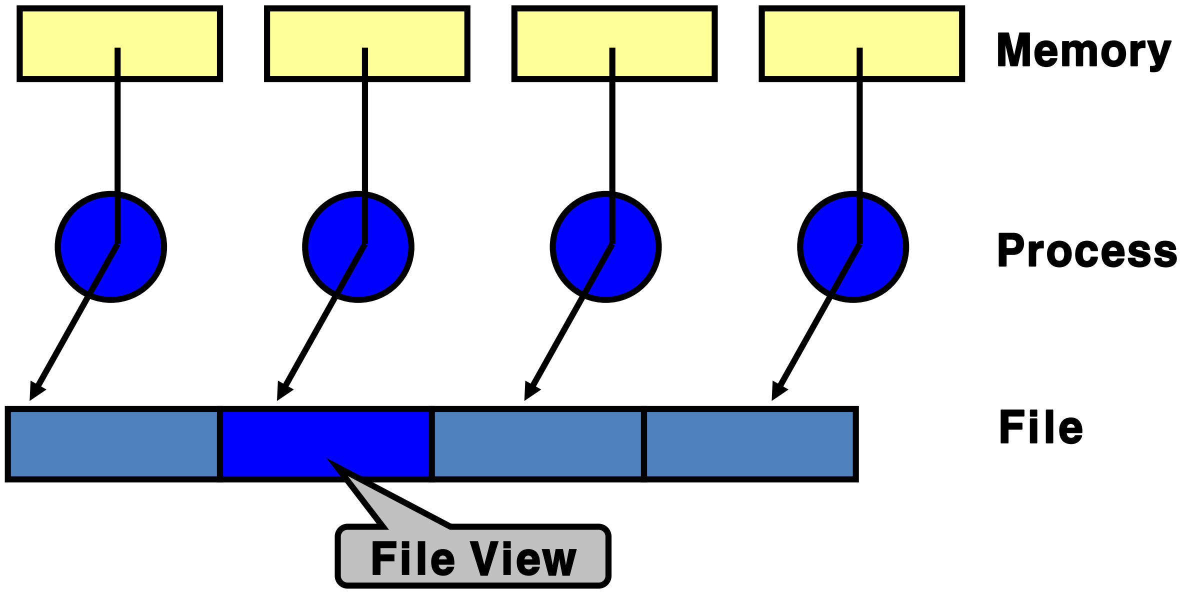
```
PROGRAM serial_IO2
INCLUDE 'mpif.h'
INTEGER BUFSIZE
PARAMETER (BUFSIZE = 100)
INTEGER nprocs, myrank, ierr, buf(BUFSIZE)
CHARACTER*2 number
CHARACTER*20 fname(0:128)
CALL MPI_INIT(ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
DO i = 1, BUFSIZE
    buf(i) = myrank * BUFSIZE + i
ENDDO
WRITE(number, 10) myrank
10 FORMAT(I0.2)
fname(myrank) = "testfile."//number
OPEN(UNIT=myrank+10,FILE=fname(myrank),STATUS="NEW",ACTION="WRITE")
WRITE(myrank+10,*) buf
CLOSE(myrank+10)
CALL MPI_FINALIZE(ierr)
END
```

Example of single task I/O (II)

- C

```
/*example of parallel UNIX write into separate files */
#include <mpi.h>
#include <stdio.h>
#define BUFSIZE 100
void main (int argc, char *argv[]){
  int i, nprocs, myrank, buf[BUFSIZE] ;
  char filename[128];
  FILE *myfile;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  for(i=0; i<BUFSIZE; i++)
    buf[i] = myrank * BUFSIZE + i;
  sprintf(filename, "testfile.%d", myrank);
  myfile = fopen(filename, "wb");
  fwrite(buf, sizeof(int), BUFSIZE, myfile);
  fclose(myfile);
  MPI_Finalize();
}
```

MPI-I/O



- **Basic functions**
  - MPI_FILE_OPEN
  - MPI_FILE_WRITE
  - MPI_FILE_CLOSE

MPI_FILE_OPEN (1/2)

C	`int MPI_File_open(MPI_Comm comm, char *filename, int amode, MPI_Info info, MPI_File *fh)`
Fortran	`MPI_FILE_OPEN(comm, filename, amode, info, fh, ierr)`

INTEGER comm : Communicator (IN)
CHARACTER filename : File name for open (IN)
INTEGER amode : File access mode (IN)
INTEGER info : info object (IN)
INTEGER fh : File handler (OUT)

- **Collective communication: sharing same file among processes in same communicator**
  - MPI_COMM_SELF : Communicator having its own single process

MPI_FILE_OPEN (1/2)

- **File access mode: OR(|:C), IOR(+:Fortran)**

MPI_MODE_APPEND	Append mode
MPI_MODE_CREATE	Create or overwrite if the file exists
MPI_MODE_DELETE_ON_CLOSE	Delete file on close
MPI_MODE_EXCL	Return error if the file exists
MPI_MODE_RDONLY	Read-only
MPI_MODE_RDWR	Read-write
MPI_MODE_SEQUENTIAL	Sequential access only
MPI_MODE_UNIQUE_OPEN	File will not be concurrently opened elsewhere
MPI_MODE_WDONLY	Write-only

- **info** argument
  - used to provide information regarding file access patterns and file system specifics. The constant MPI_INFO_NULL can be used when no info needs to be specified.

MPI_FILE_WRITE and MPI_FILE_CLOSE

C	`int MPI_File_write(MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)`
Fortran	`MPI_FILE_WRITE(fh, buf, count, datatype, status(MPI_STATUS_SIZE), ierr)`

INTEGER fh : File handler (INOUT)
CHOICE buf : Buffer address (IN)
INTEGER count : number of element in buffer(IN)
INTEGER datatype : Data type in buffer (IN)
INTEGER status(MPI_STATUS_SIZE) : status object (OUT)

C	`int MPI_File_close(MPI_File *fh)`
Fortran	`MPI_FILE_CLOSE(fh, ierr)`

Simple example for individual file write (I)

- Fortran

```
PROGRAM parallel_IO_1
INCLUDE 'mpif.h'
INTEGER BUFSIZE
PARAMETER (BUFSIZE = 100)
INTEGER nprocs, myrank, ierr, buf(BUFSIZE), myfile
CHARACTER*2 number
CHARACTER*20 filename(0:128)

CALL MPI_INIT(ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
DO i = 1, BUFSIZE
    buf(i) = myrank * BUFSIZE + i
ENDDO
WRITE(number, *) myrank
filename(myrank) = "testfile."//number

CALL MPI_FILE_OPEN(MPI_COMM_SELF, filename, &
    MPI_MODE_WRONLY+MPI_MODE_CREATE, MPI_INFO_NULL, myfile, ierr)
CALL MPI_FILE_WRITE(myfile, buf, BUFSIZE, MPI_INTEGER,MPI_STATUS_IGNORE, ierr)
CALL MPI_FILE_CLOSE(myfile, ierr)

CALL MPI_FINALIZE(ierr)
END
```

MPI_COMM_SELF : Communicator having its own single process

Simple example for individual file write (II)

- C

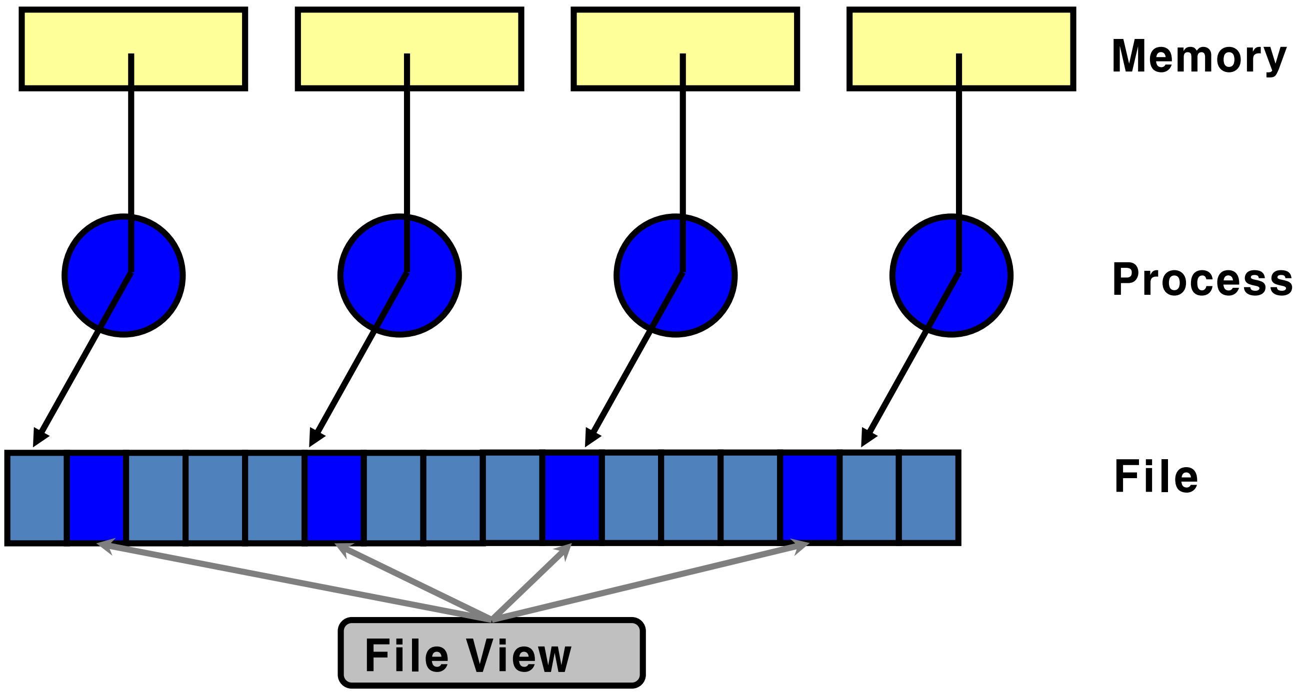
```
/*example of parallel MPI write into separate files */
#include <mpi.h>
#include <stdio.h>
#define BUFSIZE 100

void main (int argc, char *argv[]){
  int i, nprocs, myrank, buf[BUFSIZE] ;
  char filename[128];
  MPI_File myfile;

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  for(i=0; i<BUFSIZE; i++)
    buf[i] = myrank * BUFSIZE + i;
  sprintf(filename, "testfile.%d", myrank);

  MPI_File_open(MPI_COMM_SELF, filename,
           MPI_MODE_WRONLY | MPI_MODE_CREATE, MPI_INFO_NULL, &myfile);
  MPI_File_write(myfile, buf, BUFSIZE, MPI_INT, MPI_STATUS_IGNORE);
  MPI_File_close(&myfile);

  MPI_Finalize();
}
```

MPI_COMM_SELF : Communicator having its own single process

MPI-IO concept

- Processes share a single file
  - MPI_COMM_SELF → MPI_COMM_WORLD

- File view: part for the access of each process in the share file
  - MPI_FILE_SET_VIEW



MPI_FILE_SET_VIEW (1/2)

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

MPI_FILE_SET_VIEW(fh, disp, etype, filetype, datarep, info)

fh	file handle (handle)
disp	displacement (integer)
etype	elementary datatype (handle)
filetype	filetype (handle)
datarep	data representation (string)
info	info object (handle)

```
MPI_File_set_view(thefile, disp, MPI_INT, MPI_INT, "native",
MPI_INFO_NULL);
```

MPI_FILE_SET_VIEW (2/2)

- **Collective communication function called by every process in the communicator**
- **Data representation**
  - A string that specifies the representation of data in the file
  - Used for file portability
  - **native**
    - Data in this representation is stored in a file exactly as it is in memory.
    - The advantage of this data representation is that data precision and I/O performance are not lost in type conversions with a purely homogeneous environment
  - **internal**
    - This data representation can be used for I/O operations in a homogeneous or heterogeneous environment; the implementation will perform type conversions if necessary.
  - **external32**
    - This data representation states that read and write operations convert all data from and to the ``external32'' representation

For more information, refer to http://mpi-forum.org/docs/mpi-2.2/mpi22-report/node288.htm#Node288

Example of MPI-IO : File write (I)

- Fortran

```
PROGRAM parallel_IO_2
INCLUDE 'mpif.h'
INTEGER BUFSIZE
PARAMETER (BUFSIZE = 100)
INTEGER nprocs, myrank, ierr, buf(BUFSIZE), thefile
INTEGER(kind=MPI_OFFSET_KIND) disp
CALL MPI_INIT(ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
DO i = 1, BUFSIZE
    buf(i) = myrank * BUFSIZE + i
ENDDO
CALL MPI_FILE_OPEN(MPI_COMM_WORLD, 'testfile', &
  MPI_MODE_WRONLY + MPI_MODE_CREATE, MPI_INFO_NULL, &
  thefile, ierr)
disp = myrank * BUFSIZE * 4
CALL MPI_FILE_SET_VIEW(thefile, disp, MPI_INTEGER, &
        MPI_INTEGER, 'native', MPI_INFO_NULL, ierr)
CALL MPI_FILE_WRITE(thefile, buf, BUFSIZE, MPI_INTEGER, &
        MPI_STATUS_IGNORE, ierr)
CALL MPI_FILE_CLOSE(thefile, ierr)
CALL MPI_FINALIZE(ierr)
END
```

Example of MPI-IO : File write (II)

- C

```
/*example of parallel MPI write into single files */
#include <mpi.h>
#include <stdio.h>
#define BUFSIZE 100

void main (int argc, char *argv[]){
  int i, nprocs, myrank, buf[BUFSIZE] ;
  MPI_File thefile;
  MPI_Offset disp;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
 for(i=0; i<BUFSIZE; i++)
    buf[i] = myrank * BUFSIZE + i;
  MPI_File_open(MPI_COMM_WORLD, “testfile”,
          MPI_MODE_WRONLY | MPI_MODE_CREATE, MPI_INFO_NULL, &thefile);
  disp = myrank*BUFSIZE*sizeof(int);
  MPI_File_set_view(thefile, disp, MPI_INT, MPI_INT,“native”,MPI_INFO_NULL);
  MPI_File_write(thefile, buf, BUFSIZE, MPI_INT, MPI_STATUS_IGNORE);
  MPI_File_close(&thefile);
  MPI_Finalize();
}
```

MPI-IO : File read

- Processes can share file read data parallelely

- File size

  MPI_FILE_GET_SIZE

- Each process read data parallel using “File view”

  MPI_FILE_READ

MPI_FILE_GET_SIZE

C	`int MPI_File_get_size(MPI_File fh, MPI_Offset *size)`
Fortran	`MPI_FILE_GET_SIZE(fh, size, ierr)`

IN fh	file handle (handle)
OUT size	size of the file in bytes (integer)

- Return the file size in byte

MPI_FILE_READ

C	`int MPI_File_read(MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)`
Fortran	`MPI_FILE_READ(fh, buf, count, datatype, status(MPI_STATUS_SIZE), ierr)`

INOUT fh	file handle (handle)
OUT buf	initial address of buffer (choice)
IN count	number of elements in buffer (integer)
IN datatype	datatype of each buffer element (handle)
OUT status	status object (Status)

- Reads a file using the individual file pointer into buf according to count and datatype

Example of MPI-IO : File read (I)

- Fortran

```
PROGRAM parallel_IO_3
INCLUDE 'mpif.h'
INTEGER nprocs, myrank, ierr
INTEGER count, bufsize, thefile
INTEGER (kind=MPI_OFFSET_KIND) filesize, disp
INTEGER, ALLOCATABLE :: buf(:)
INTEGER status(MPI_STATUS_SIZE)

CALL MPI_INIT(ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
CALL MPI_FILE_OPEN(MPI_COMM_WORLD, 'testfile', MPI_MODE_RDONLY, MPI_INFO_NULL, thefile, ierr)
CALL MPI_FILE_GET_SIZE(thefile, filesize, ierr)
filesize = filesize/4
bufsize = filesize/nprocs + 1
ALLOCATE(buf(bufsize))
disp = myrank * bufsize * 4
CALL MPI_FILE_SET_VIEW(thefile, disp, MPI_INTEGER, MPI_INTEGER, 'native', MPI_INFO_NULL, ierr)
CALL MPI_FILE_READ(thefile, buf, bufsize, MPI_INTEGER, status, ierr)
CALL MPI_GET_COUNT(status, MPI_INTEGER, count, ierr)
print *, 'process ', myrank, 'read ', count, 'ints'
CALL MPI_FILE_CLOSE(thefile, ierr)
CALL MPI_FINALIZE(ierr)
END
```

Example of MPI-IO : File read (II)

- C

```
/* parallel MPI read with arbitrary number of processes */
#include <mpi.h>
#include <stdio.h>

void main (int argc, char *argv[]){
  int nprocs, myrank, bufsize, *buf, count;
  MPI_File thefile;
  MPI_Status status;
  MPI_Offset filesize;

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  MPI_File_open(MPI_COMM_WORLD, "testfile", MPI_MODE_RDONLY, MPI_INFO_NULL, &thefile);
  MPI_File_get_size(thefile, &filesize);
  filesize = filesize / sizeof(int);
  bufsize = filesize /nprocs + 1;
  buf = (int *) malloc(bufsize * sizeof(int));
  MPI_File_set_view(thefile,myrank*bufsize*sizeof(int),MPI_INT,MPI_INT, "native", MPI_INFO_NULL);
  MPI_File_read(thefile, buf, bufsize, MPI_INT, &status);
  MPI_Get_count(&status, MPI_INT, &count);
  printf("process %d read%c ints \n ", myrank, count);
  MPI_File_close(&thefile);
  MPI_Finalize();
}
```

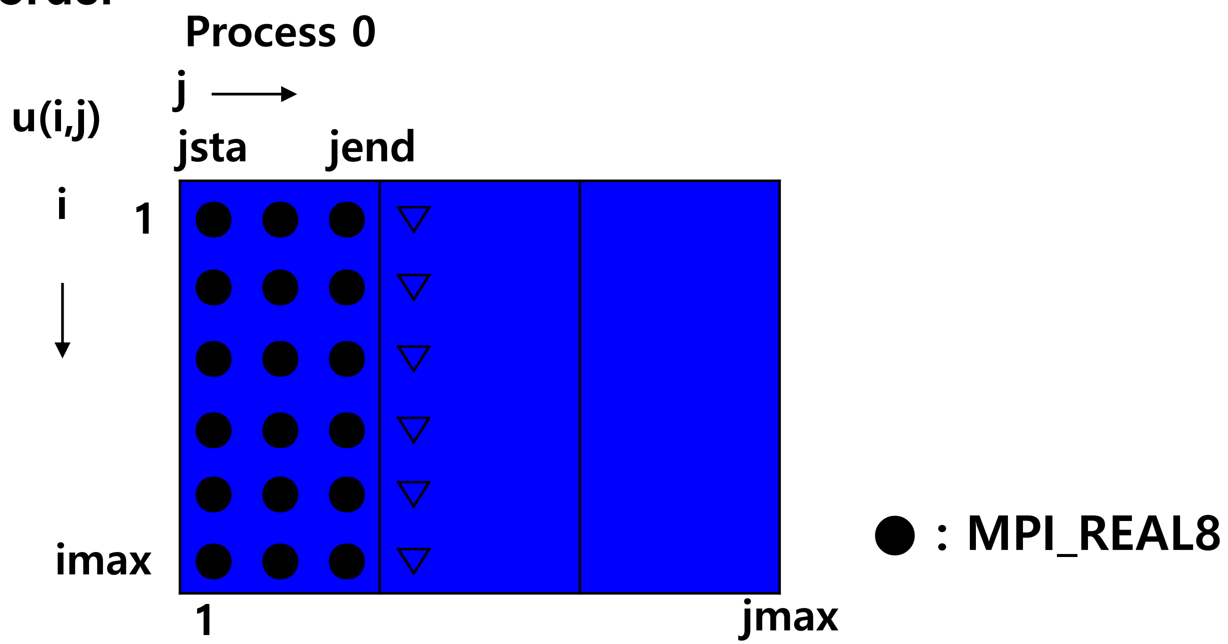
MPI_FILE_SEEK

C	`int MPI_File_seek(MPI_File fh, MPI_Offset offset, int whence)`
Fortran	`MPI_FILE_SEEK(FH, OFFSET, WHENCE, IERROR)`

- **Move file pointer to target position**
  - Input Parameters
    - fh : File handle (handle).
    - offset : File offset (integer).
    - whence : Update mode (integer).
  - Output Parameter
    - IERROR : Fortran only: Error status (integer).
  - whence
    - MPI_SEEK_SET - The pointer is set to offset.
    - MPI_SEEK_CUR - The pointer is set to the current pointer position plus offset.
    - MPI_SEEK_END - The pointer is set to the end of the file plus offset

MPI-I/O Example

- Fortran order




```
...
CALL MPI_FILE_OPEN(MPI_COMM_WORLD,"u.dat", MPI_MODE_CREATE    &
    + MPI_MODE_WRONLY, MPI_INFO_NULL, outfile, ierr)
CALL MPI_FILE_SEEK(outfile,(jsta-1)*imax*8,MPI_SEEK_SET,ierr)
DO j = jsta, jend
     CALL MPI_FILE_WRITE(outfile,u(1,j),imax,MPI_REAL8,istat,ierr)
ENDDO
CALL MPI_FILE_CLOSE(outfile,ierr)
...
```

Collective I/O Operations (I)

MPI_FILE_READ_ALL

C	`int MPI_File_read_all(MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)`
Fortran	`MPI_FILE_READ_ALL( FH, BUF, COUNT, DATATYPE, STATUS, IERROR)`

- Input Parameters
  - fh : File handle (handle).
  - count : Number of elements in buffer (integer).
  - datatype : Data type of each buffer element (handle).
- Output Parameters
  - buf : Initial address of buffer (choice).
  - status: Status object (status).

Reads a file starting at the locations specified by individual file pointers

Collective I/O Operations (II)

MPI_FILE_WRITE_ALL

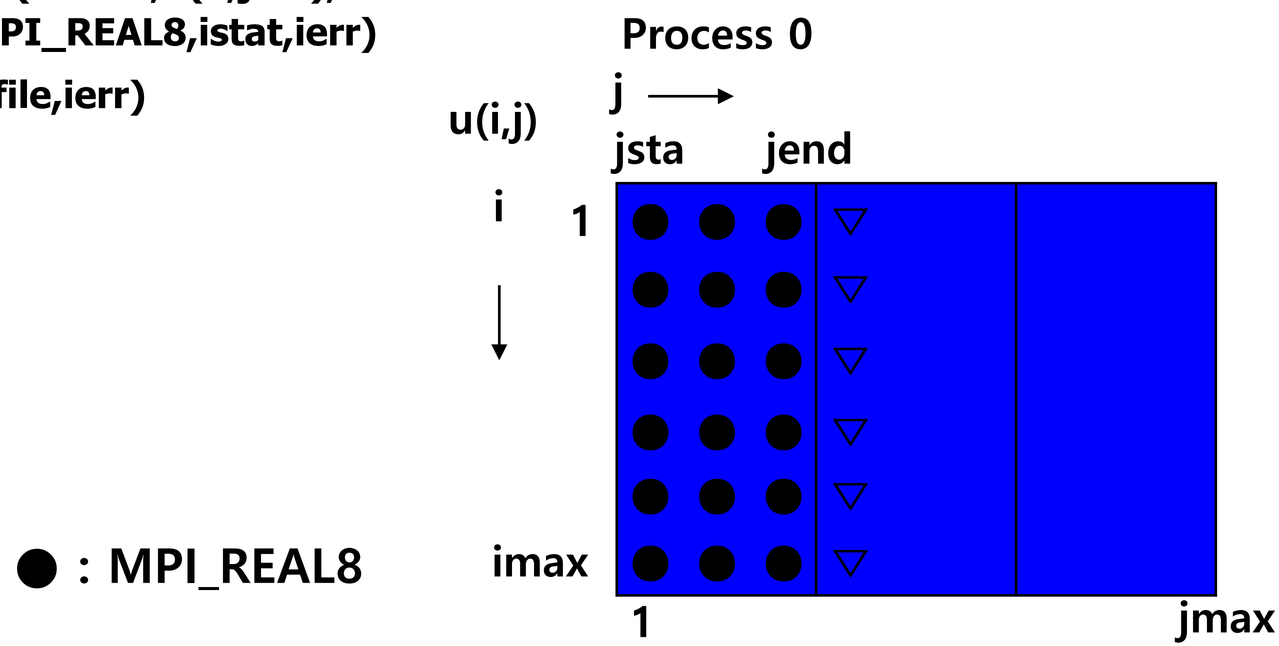
C	`int MPI_File_write_all(MPI_File fh, const void *buf, int count, MPI_Datatype datatype, MPI_Status *status)`
Fortran	`MPI_FILE_WRITE_ALL( FH, BUF, COUNT, DATATYPE, STATUS, IERROR)`

- Input Parameters
  - fh : File handle (handle).
  - buf : Initial address of buffer (choice).
  - count : Number of elements in buffer (integer).
  - datatype : Data type of each buffer element (handle).
- Output Parameters
  - status: Status object (status).

Writes a file starting at the locations specified by individual file pointers

Collective MPI-I/O

```
...
CALL MPI_TYPE_CONTIGUOUS(imax, MPI_REAL8,filetype,ierr)
CALL MPI_TYPE_COMMIT(filetype,ierr)
CALL MPI_FILE_OPEN(MPI_COMM_WORLD,'u.dat',    &
     MPI_MODE_CREATE+MPI_MODE_WRONLY,  &
     MPI_INFO_NULL,  outfile, ierr)
CALL MPI_FILE_SET_VIEW(outfile, (jsta-1)*imax*8,      &
     MPI_REAL8, filetype, 'native', MPI_INFO_NULL,ierr)
CALL MPI_FILE_WRITE_ALL(outfile,u(1,jsta), &
     (jend-jsta+1)*imax,    MPI_REAL8,istat,ierr)
CALL MPI_FILE_CLOSE(outfile,ierr)
...
```



File type

- Use of proper file type and data type → efficient MPI-I/O

Use derived data type!!

0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
2	2	2	3	3	3
2	2	2	3	3	3
2	2	2	3	3	3

Displacement: 0

etype: MPI_REAL

filetype (process 0):

0	0	0				0	0	0				0	0	0	…

MPI-I/O Example

- **Fortran order**

j →

u(i,j)	1								jmax
i ↓ 1	0	0	0	1	1	1	2	2	2
	0	0	0	1	1	1	2	2	2
	0	0	0	1	1	1	2	2	2
	0	0	0	1	1	1	2	2	2
	3	3	3	4	4	4	5	5	5
	3	3	3	4	4	4	5	5	5
	3	3	3	4	4	4	5	5	5
	3	3	3	4	4	4	5	5	5
	6	6	6	7	7	7	8	8	8
	6	6	6	7	7	7	8	8	8
	6	6	6	7	7	7	8	8	8
imax	6	6	6	7	7	7	8	8	8

MPI-I/O using a file view

```
...
array_of_sizes(1)=12; array_of_sizes(2)=9
array_of_subsizes(1)=4; array_of_subsizes(2)=3
array_of_starts(1)=istart-1; array_of_starts(2)=jstart-1

CALL MPI_TYPE_CREATE_SUBARRAY(2, array_of_sizes, array_of_subsizes,      &
          array_of_starts, MPI_ORDER_FORTRAN, MPI_REAL, filetype, ierr)
CALL MPI_TYPE_COMMIT(filetype, ierr)
CALL MPI_FILE_OPEN(MPI_COMM_WORLD, 'u.dat', MPI_MODE_CREATE +    &
   MPI_MODE_WRONLY, MPI_INFO_NULL, outfile, ierr)
CALL MPI_FILE_SET_VIEW(outfile, 0, MPI_REAL, filetype, 'native',
   MPI_INFO_NULL, ierr)
CALL MPI_FILE_WRITE(outfile, u, 12, MPI_REAL, istat, ierr)
CALL MPI_FILE_CLOSE(outfile, ierr)
...
```

Other functions

Asynchronous I/O Operations

- MPI_FILE_IREAD
- MPI_FILE_IWRITE

```
…
CALL MPI_TYPE_CONTIGUOUS(imax, MPI_REAL8,filetype,ierr)
CALL MPI_TYPE_COMMIT(filetype,ierr)
CALL MPI_FILE_OPEN(MPI_COMM_WORLD,'u.dat',    &
     MPI_MODE_CREATE+MPI_MODE_WRONLY,  &
     MPI_INFO_NULL,  outfile, ierr)
CALL MPI_FILE_SET_VIEW(outfile, (jstart-1)*imax*8,      &
     MPI_REAL8, filetype, 'native', MPI_INFO_NULL,ierr)
CALL MPI_FILE_IWRITE(outfile, u(1,jstart), (jend-jstart+1)*imax, &
     MPI_REAL8,req,ierr)
…
CALL MPI_WAIT(req,istat,ierr)
CALL MPI_FILE_CLOSE(outfile,ierr)
…
```

Split Collective I/O Operations

- **Collective + Non-Blocking**
  - MPI_FILE_READ_ALL_BEGIN
  - MPI_FILE_WRITE_ALL_BEGIN
  - MPI_FILE_READ_ALL_END
  - MPI_FILE_WRITE_ALL_END

```
…
CALL MPI_TYPE_CONTIGUOUS(imax, MPI_REAL8,filetype,ierr)
CALL MPI_TYPE_COMMIT(filetype,ierr)
CALL MPI_FILE_OPEN(MPI_COMM_WORLD,'u.dat',    &    MPI_MODE_CREATE+MPI_MODE_WRONLY,  &
     MPI_INFO_NULL,  outfile, ierr)
CALL MPI_FILE_SET_VIEW(outfile, (jstart-1)*imax*8,      &
     MPI_REAL8, filetype, 'native', MPI_INFO_NULL,ierr)
CALL MPI_FILE_WRITE_ALL_BEGIN(outfile,u(1,jstart), (jend-jstart+1)*imax,  & MPI_REAL8, ierr)
…
CALL MPI_FILE_WRITE_ALL_END(outfile, u(1,jstart), istat, ierr)
CALL MPI_FILE_CLOSE(outfile,ierr)
…
```