KiSTi SUPERCOMPUTING EDUCATION CENTER
www.kisti.re.kr

MPI 고급

2021.06.

KISTI 강 지 훈

KISTI과학데이터스쿨
KISTI SCIENCE DATA SCHOOL
http://kacademy.kisti.re.kr

Syllabus (Advanced Course)

09:30	-	10:30	▪ Basic Environment / Virtual Topology
10:40	-	11:40	▪ One-sided Communication
11:40	-	13:00	▪ Lunch
13:00	-	14:20	▪ Parallel I/O
14:30	-	15:30	▪ Loop parallelization / Domain decomposition
15:40	-	16:40	▪ Advanced exercise / MPMD
16:40	-	17:00	▪ Summary

Resources

- http://k-academy.kisti.re.kr
- https://www.mpi-forum.org/
- https://computing.llnl.gov/tutorials/mpi/
- http://mpitutorial.com/
- http://incredible.egloos.com/3755171

Basic Environment

1. 시스템 접속
2. Linux 기초
   - 기본 명령어
   - VI Editor
3. Environment Module
   - Module 명령어
     - avail
     - add
     - rm
     - list
     - purge
4. Interactive 작업 제출

시스템 접속

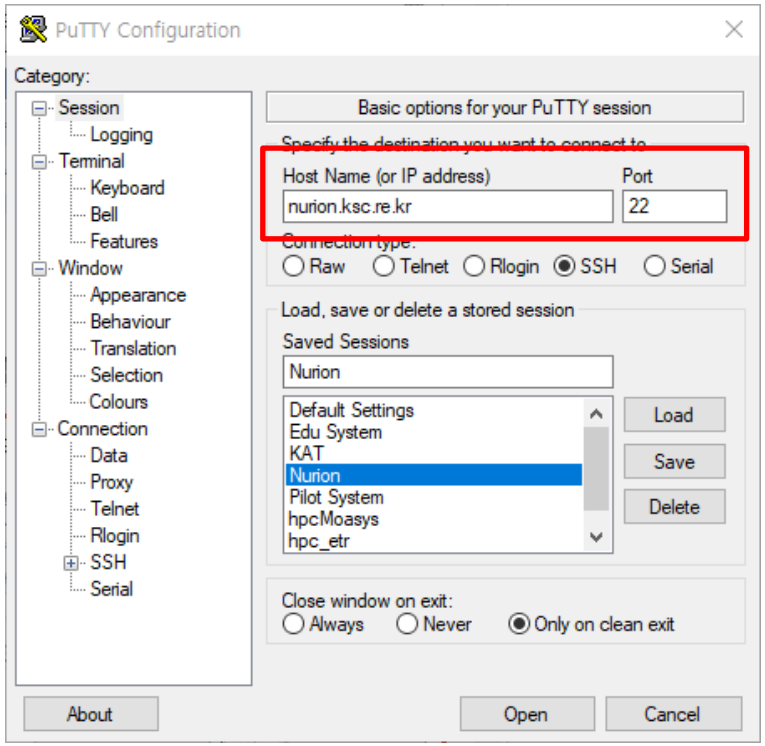
➢ 노드 구성

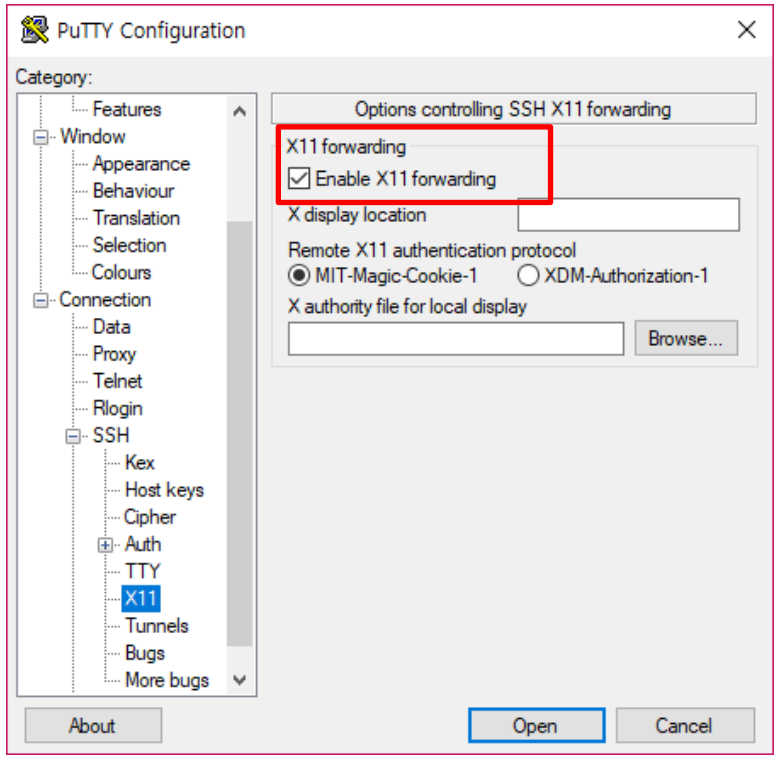
		호스트 명	CPU Limit	비고
로그인 노드		nurion.ksc.re.kr	20분	▪ ssh/scp/sftp 접속 가능 ▪ 컴파일 및 batch 작업제출용 ▪ ftp 접속 불가
Datamover 노드		nurion-dm .ksc.re.kr	-	▪ ssh/scp/sftp 접속 가능 ▪ ftp 접속 가능 ▪ 컴파일 및 작업 제출 불가
계산 노드	KNL	node[0001-8305]	-	▪ PBS 스케줄러를 통해 작업 실행 가능 ▪ 일반사용자 직접 접근 불가
	CPU-Only	cpu[0001-0132]	-	

시스템 접속

- Xming
  - X 환경 실행을 위해 필요
- Putty 사용
  - Host Name : nurion.ksc.re.kr(port : 22)

※ Xming 실행 필요





시스템 접속

- 접속 ID & otp
  - sedu##(31~49)
  - OTP : 강의시 안내
  - Passwd : 강의시 안내

```
Last login: Mon Jan  7 10:00:35 2019 from xxx.xxx.xxx.xx
================ KISTI 5th NURION System
====================
 * Compute Nodes(node[0001-8305],cpu[0001-0132)
 - KNL(XeonPhi 7250 1.40GHz 68C) / 16GB(MCDRAM),96GB(DDR4)
 - CPU-only(XeonSKL6148 2.40GHz 20C x2) / 192GB(DDR4)

 * Software
 - OS: CentOS 7.4(3.10.0-693.21.1.el7.x86_64)
 - System S/W: BCM v8.1, PBS v14.2, Lustre v2.10

 * Current Configurations
 - All KNL Cluster modes - Quadrant
 - Memory modes
  : Cache-node[0001-7980,8281-8300]/Flat-node[7981-8280]
  : PBS job sharing mode-Exclusive(running 1 job per node)
    (Except just the commercial queue)
  …
* Policy on User Job
    ….
  (Use the # showq & # pbs_status commands for more queue info.)
```

VI Editor

- vim(vi)
  - 가장 기본적인 텍스트 에디터, OS에 기본적으로 포함됨
  - VIsual display editor를 의미
- 파일 개방
  - $ vi file(편집 모드)
  - $ view file(읽기 모드)
- modes
  - 입력 모드
    - 입력모드로 전환 : i (,I, a, A, o, O, R)
    - 입력하는 모든 것이 편집 버퍼에 입력됨
    - 입력 모드에서 빠져 나올 때(명령 행 모드로 변경 시) : “ESC” key
  - 명령 행 모드
    - 입력하는 모든 것이 명령어 해석됨
- 파일 저장/종료 명령 : w, q

- **login 노드에서 gedit 사용가능**

Environment Module

- 사용자가 쉘 환경(shell environment)을 관리하도록 도와주는 도구
- ‘module’ 명령
  - 부명령 (subcommand)
    - avail(av)
      - 사용 가능한 모듈파일들(modulefiles)을 보여줌
    - add(load)
      - 쉘 환경으로 모듈파일들을 적재함(load)
    - rm(unload)
      - 쉘 환경에서 적재된 모듈파일들을 제거함
    - li(list)
      - 적재된 모듈파일들을 나열함
    - purge
      - 적재된 모든 모듈파일들을 제거함

Environment Module

Default modulefiles

- login을 하면, 기본 모듈파일이 적재됨

```
$ module list
Currently Loaded Modulefiles:
  1) craype-network-opa
```

module 명령

- 사용가능 모듈 확인 (avail)

```
$ module avail
-------- /opt/cray/craype/default/modulefiles ---------------------
craype-mic-knl     craype-network-opa craype-x86-skylake
---------------- /opt/cray/modulefiles ----------------------------
cdt/17.10                                cray-impi/1.1.4(default)
…
perftools-base/6.5.2(default)
--------- /apps/Modules/modulefiles/compilers --------------------
cce/8.6.3(default)    gcc/6.1.0             gcc/7.2.0
intel/17.0.5(default) intel/18.0.1          intel/18.0.3
…
```

Environment Module

➢ 모듈 명령

– 모듈 정보 출력

```
$ module help impi/17.0.5

----------- Module Specific Help for 'impi/17.0.5' ----------------

        This module is for use of impi/17.0.5
        use example:
            $ module load intel/17.0.5 impi/17.0.5
```

– 모듈 적재

```
$  module add craype-mic-knl gcc/7.2.0 openmpi/3.1.0
```

PBS Interactive 작업 제출

- 누리온 시스템은 debug 노드 대신 debug 큐를 제공
- /scratch 에서 작업 제출해야 함
- debug 큐를 이용하여 작업을 제출함으로써 디버깅
- qsub -I (대문자 i 임)
  - qsub를 이용한 Interactive 작업 사용 예 (MPI)

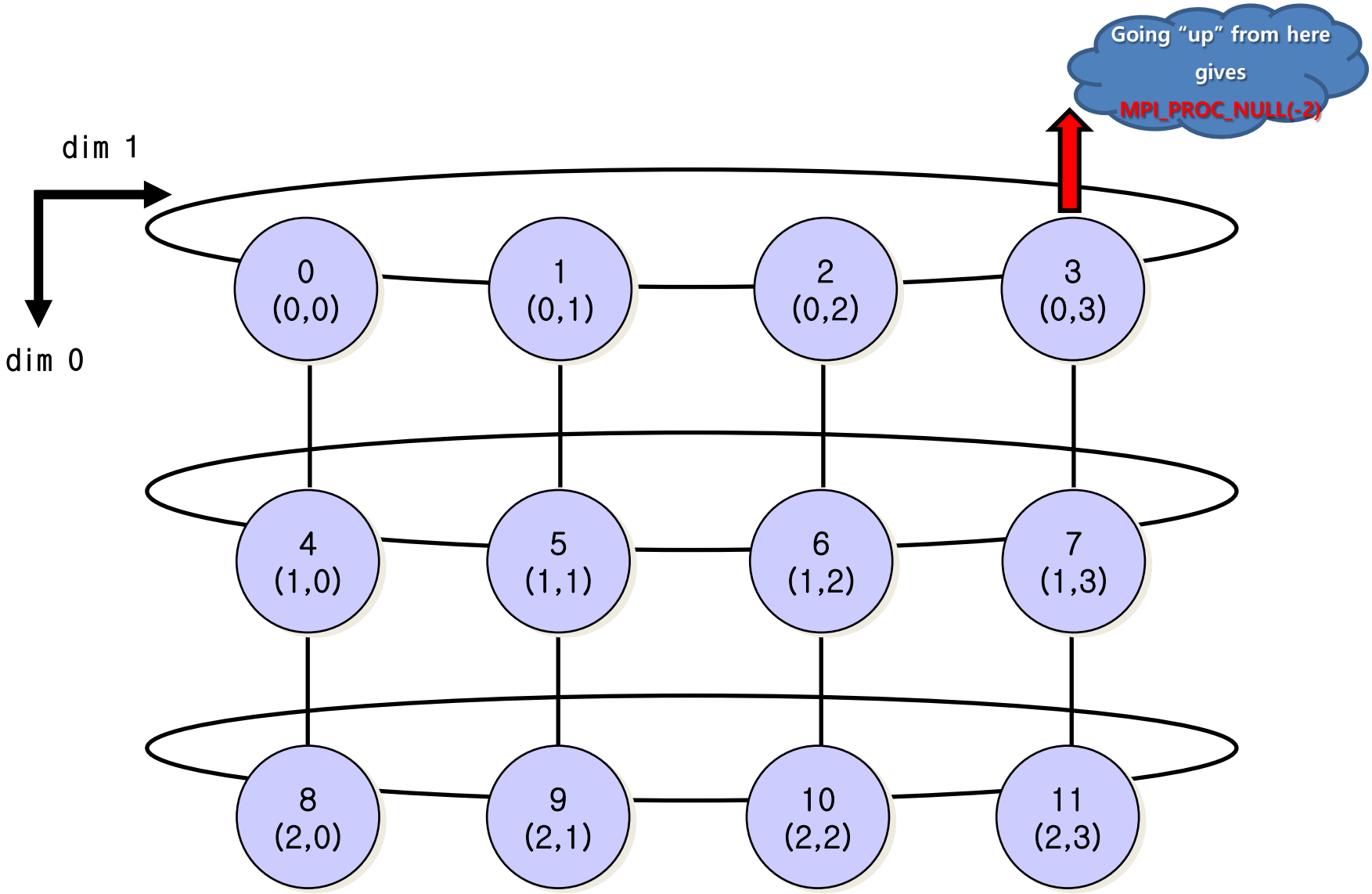
```
[sedu50@login03 C]$ qsub -I -V -l select=2:ncpus=68:mpiprocs=68:ompthreads=1
-l walltime=08:00:00 -q debug -A etc
qsub: waiting for job 1978322.pbs to start
qsub: job 1978322.pbs ready
[sedu50@node8281 MPI_Basic]$ mpirun -np 8 ./hostname.x
node8281
node8281
…
node8282
node8282
[sedu01@node8281 C]$ exit
```

Virtual Topology

Virtual Topology (1/2)

- Creating a Cartesian virtual topology
- Make MPI to optimize communications
- Easy to write codes
- Convenient process naming

Virtual Topology (2/2)



How to Use a Virtual Topology

- Creating a topology and make a new communicator
  - MPI_CART_CREATE
- MPI provides “mapping functions”
- Mapping functions compute processor ranks, based on the topology naming scheme
  - MPI_CART_RANK
  - MPI_CART_COORDS
  - MPI_CART_SHIFT

MPI_DIMS_CREATE

MPI_DIMS_CREATE(nnodes, ndims, dims)		
IN	nnodes	number of nodes in a grid (integer)
IN	ndims	number of Cartesian dimensions (integer)
INOUT	dims	integer array of size 'ndims' specifying the number of nodes in each dimension

- Cartesian topology에 대해서, 사용자가 좌표 방향당 프로세스의 균형 있는 분포를 선택하는데 도움을 줌
- C

int MPI_Dims_create(int nnodes, int ndims, int dims[])

- Fortran(mpi_f08 module)

MPI_Dims_create(nnodes, ndims, dims, ierror)	
	INTEGER, INTENT(IN) :: nnodes, ndims
	INTEGER, INTENT(INOUT) :: dims(ndims)
	INTEGER, OPTIONAL, INTENT(OUT) :: ierror

MPI_DIMS_CREATE : Fortran

```
PROGRAM Dimcreate
USE mpi_f08
IMPLICIT NONE
INTEGER::dims(3)=0, myrank
CALL MPI_Init
CALL MPI_Comm_rank(MPI_COMM_WORLD,myrank)
CALL MPI_Dims_create(6,2,dims(1:2))
if(myrank==0) PRINT*,'# of nodes (6) : dims=(',dims,')'

dims=0
CALL MPI_Dims_create(7,2,dims(1:2))
if(myrank==0) PRINT*,'# of nodes (7) : dims=(',dims,')'
dims=0

CALL MPI_Dims_create(27,3,dims(1:3))
if(myrank==0) PRINT*,'#of nodes (27) : dims=(',dims,')'

CALL MPI_Finalize
END PROGRAM Dimcreate
```

```
$ mpif90 dim_create.f90 -o dim_create.x
$ mpirun -np 1 ./dim_create.x
 # of nodes (6)  : dims=( 3   2   0 )
 # of nodes  (7) : dims=( 7   1   0 )
 # of nodes (27) : dims=( 3   3   3 )
```

MPI_CART_CREATE

MPI_CART_CREATE(comm_old, ndims, dims, periods, reorder, comm_cart)		
IN	comm_old	input communicator (handle)
IN	ndims	number of dimensions of Cartesian grid (integer)
IN	dims	integer array of size 'ndims' specifying the number of processes in each dimension
IN	periods	logical array of size 'ndims' specifying whether the grid is periodic(true) or not(false) in each dimension
IN	reorder	ranking may be reordered(true) or not(false) (logical)
OUT	comm_cart	communicator with new Cartesian topology(handle)

- Cartesian topology 정보가 부착된 새로운 communicator에 대한 handle 반환
- reorder
  - False : 새로운 그룹에서 각 프로세스의 rank는 old group의 rank와 동일
  - True : 프로세스 랭크를 재할당

MPI_CART_CREATE

- C

```
int MPI_Cart_create( MPI_Comm comm_old, int ndims, const int dims[],
                     const int periods[], int reorder, MPI_Comm *comm_cart )
```

- Fortran

```
MPI_Cart_create( comm_old, ndims, dims, periodes, reorder, comm_cart, ierror )
```

	TYPE(MPI_Comm), INTENT(IN) :: comm_old
	INTEGER, INTENT(IN) :: ndims, dims(ndims)
	LOGICAL, INTENT(IN) :: periods(ndims), reorder
	TYPE(MPI_Comm), INTENT(OUT) :: comm_cart
	INTEGER, OPTIONAL, INTENT(OUT) :: ierror

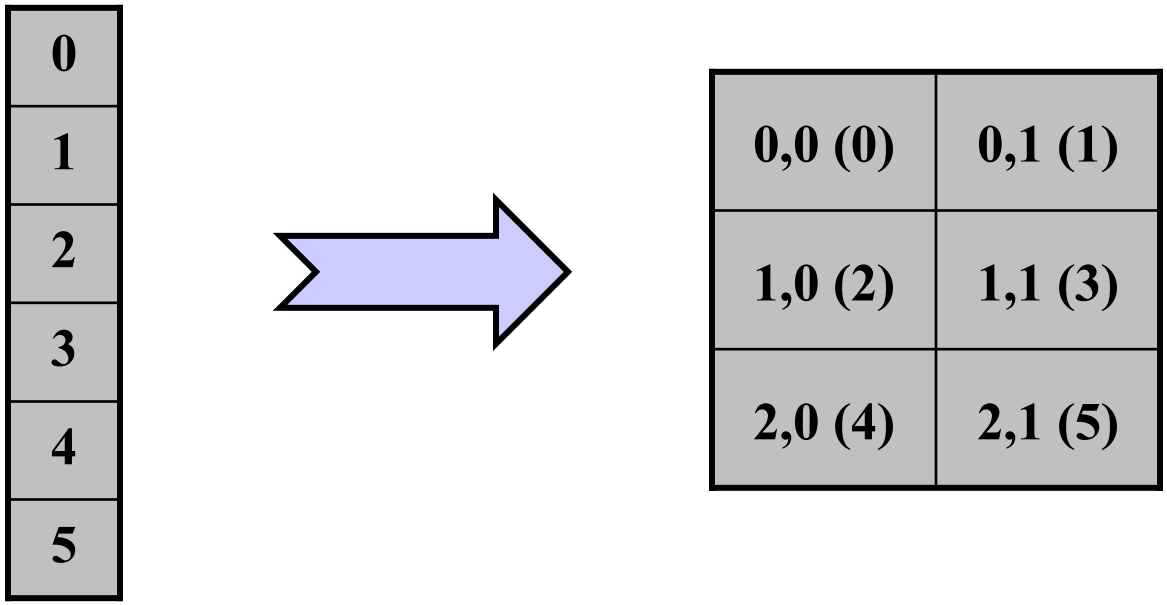
MPI_DIMS_CREATE : Fortran

```
PROGRAM cart_create
USE mpi_f08
IMPLICIT NONE
INTEGER,PARAMETER::NDIM=2
INTEGER::myrank, nprocs, dims(NDIM)=0, newprocs, newrank
LOGICAL::reorder=.TRUE., periods(NDIM)=.false.
TYPE(MPI_comm)::comm_cart
CALL MPI_Init
CALL MPI_Comm_rank(MPI_COMM_WORLD, myrank)
CALL MPI_Comm_size(MPI_COMM_WORLD, nprocs)
CALL MPI_Dims_create(nprocs,NDIM,dims)
if(myrank==0) PRINT*, 'DIMS=',dims
CALL MPI_Cart_create(MPI_COMM_WORLD,NDIM,dims,periods,reorder,comm_cart)
CALL MPI_Comm_size(comm_cart,newprocs)
CALL MPI_Comm_rank(comm_cart,newrank);
PRINT*,'myrank:',myrank, 'newrank:',newrank
CALL MPI_Finalize
END PROGRAM cart_create
```

MPI_DIMS_CREATE : Fortran

```
$ mpif90 cart_create.f90 -o cart_create.x
$ mpirun -np 12 ./cart_create.x
 DIMS=           4           3
 myrank:           0 newrank:           0
 myrank:           2 newrank:           2
 myrank:           3 newrank:           3
 myrank:           4 newrank:           4
 myrank:           5 newrank:           5
 myrank:           1 newrank:           1
 myrank:           6 newrank:           6
 myrank:           7 newrank:           7
 myrank:           8 newrank:           8
 myrank:           9 newrank:           9
 myrank:          10 newrank:          10
 myrank:          11 newrank:          11
```

MPI_CART_CREATE



MPI_CART_RANK

MPI_CART_RANK(comm, cords, rank)		
IN	comm	communicator with Cartesian structure (handle)
IN	coords	integer array(of size 'ndims') specifying the Cartesian coordinates of a process
OUT	rank	rank of specified process (integer)

- Cartesian 좌표에 대한 rank를 반환

- C

```
int MPI_Cart_rank( MPI_Comm comm, const int cords[], int *rank )
```

- Fortran

```
MPI_Cart_rank( comm, coords, rank, ierror )
```

```
TYPE(MPI_Comm), INTENT(IN) :: comm
INTEGER, INTENT(IN) :: coords(*)
INTEGER, INTENT(OUT) :: rank
INTEGER, OPTIONAL, INTENT(OUT) :: ierror
```

MPI_CART_COORDS

MPI_CART_COORDS(comm, rank, maxdims, coords)		
IN	comm	communicator with Cartesian structure (handle)
IN	rank	rank of process within group of 'comm' (integer)
IN	maxdims	length of vector 'cords' in the calling program (integer)
OUT	coords	integer array (of size 'ndims') containing the Cartesian coordinates of specified process (array of integers)

- Rank에 해당하는 좌표를 반환

- C

```
int MPI_Cart_coords( MPI_Comm comm, int rank, int maxdims, int coords[])
```

- Fortran(mpi_f08 module)

```
MPI_Cart_coords( comm, rank, maxdims, coords, ierror )
TYPE(MPI_Comm), INTENT(IN) :: comm
INTEGER, INTENT(IN) :: rank, maxdims
INTEGER, INTENT(OUT) :: coords(maxdims)
INTEGER, OPTIONAL, INTENT(OUT) :: ierror
```

Lab #1

MPI_CART_RANK/MPI_CART_COORDS : Fortran

```
PROGRAM cart_rank_coords
USE mpi_f08
IMPLICIT NONE
TYPE(MPI_Comm)::oldcomm, newcomm
INTEGER::ndims=2, dimsize(0:1)
LOGICAL::periods(0:1), reorder
INTEGER::myrank,nprocs,i,j,rank
INTEGER::coords(0:1)
CALL MPI_Init
CALL MPI_Comm_rank(MPI_COMM_WORLD,myrank)
CALL MPI_Comm_size(MPI_COMM_WORLD,nprocs)
oldcomm=MPI_COMM_WORLD
dimsize=(/3,2/)
periods=(/.TRUE., .FALSE./)
reorder=.TRUE.
CALL MPI_Cart_create(oldcomm, ndims,dimsize,periods, reorder,newcomm)
IF(myrank==0)THEN
  DO i=0,dimsize(0)-1
    DO j=0,dimsize(1)-1
      coords=(/i,j/)
      CALL MPI_Cart_rank(newcomm,coords,rank)
      PRINT*,'coords=',coords,'rank=',rank
    END DO
  END DO
ENDIF
```

MPI_CART_RANK/MPI_CART_COORDS : Fortran

```
IF(myrank==0)THEN
  DO rank=0,nprocs-1
    CALL MPI_Cart_coords(newcomm,rank,ndims,coords);
    PRINT*,'rank=',rank,'coords=',coords
  end do
END IF
CALL MPI_Finalize
END PROGRAM cart_rank_coords
```

```
$ mpif90 cart_rank_coords.f90 -o cart_rank_coords.x
$ mpirun -np 6 ./cart_rank_coords.x
 coords=           0           0 rank=           0
 coords=           0           1 rank=           1
 coords=           1           0 rank=           2
 coords=           1           1 rank=           3
 coords=           2           0 rank=           4
 coords=           2           1 rank=           5
 rank=           0 coords=           0           0
 rank=           1 coords=           0           1
 rank=           2 coords=           1           0
 rank=           3 coords=           1           1
 rank=           4 coords=           2           0
 rank=           5 coords=           2           1
```

MPI_CART_SHFIT

MPI_CART_SHIFT(comm, direction, disp, rank_source, rank_dest)		
IN	comm	communicator with Cartesian structure (handle)
IN	direction	coordinate direction of shift (integer)
IN	disp	displacement(>0: upwards shift, <0 : downwards shift) (integer)
OUT	rank_source	rank of source process (integer)
OUT	rank_dest	rank of destination process (integer)

- 직교 좌표 토폴로지에서 특정 방향을 따라 프로세스의 이웃 프로세스를 찾는데 사용
- 검색된 두 프로세스를 각각 source rank와 destination rank이라 함
- 루틴을 호출한 프로세스와의 인접 정도는 disp로 결정

MPI_CART_SHFIT

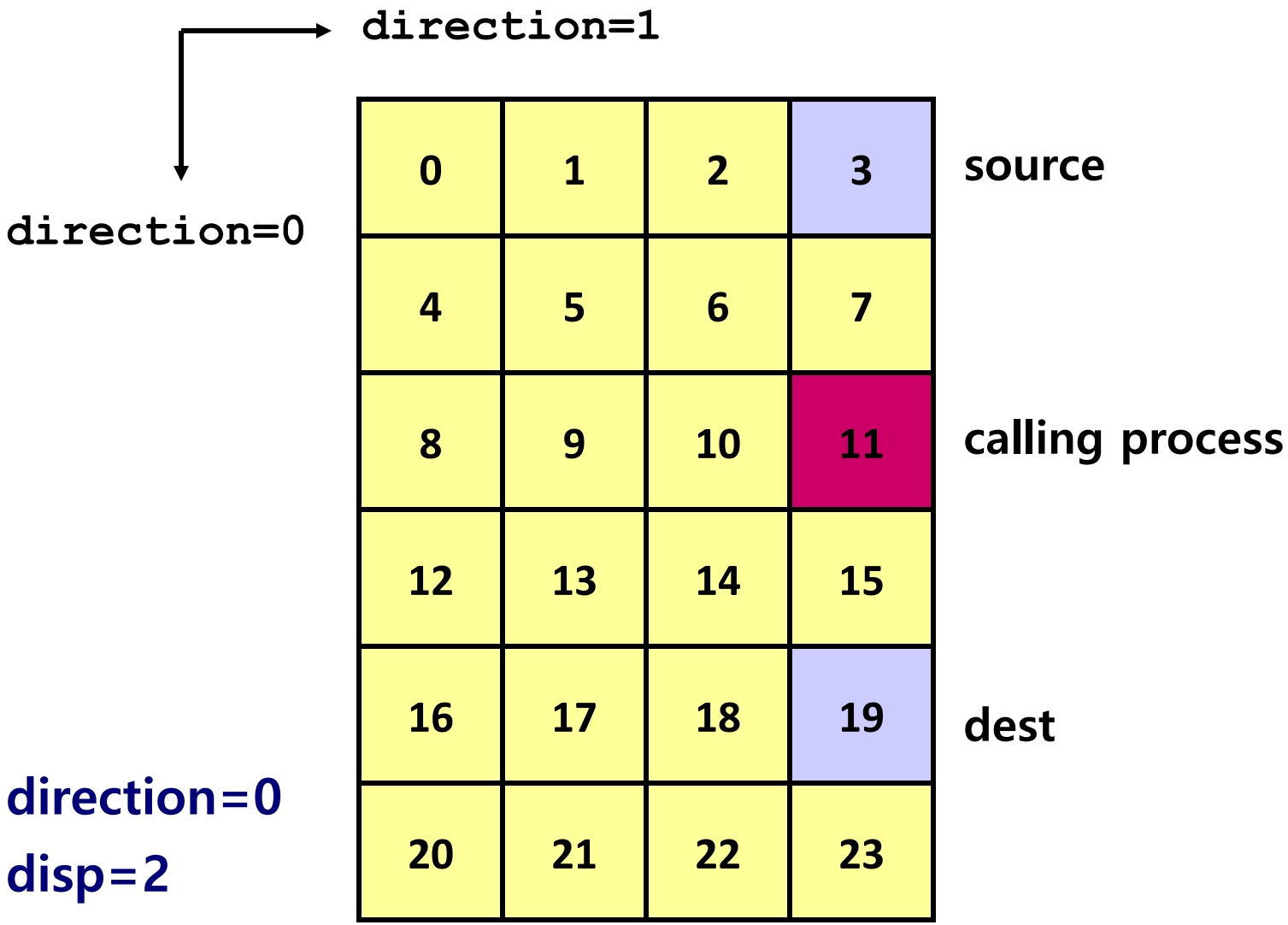
- C

```
int MPI_Cart_shift( MPI_Comm comm, int direction, int disp,
                    int* rank_source, int* rank_dest )
```

- Fortran(mpi_f08 module)

MPI_Cart_shift(comm, direction, disp, rank_source, rank_dest)	
	TYPE(MPI_Comm), INTENT(IN) :: comm
	INTEGER, INTENT(IN) :: direction, disp
	INTEGER, INTENT(OUT) :: rank_source, rank_dest
	INTEGER, OPTIONAL, INTENT(OUT) :: ierror

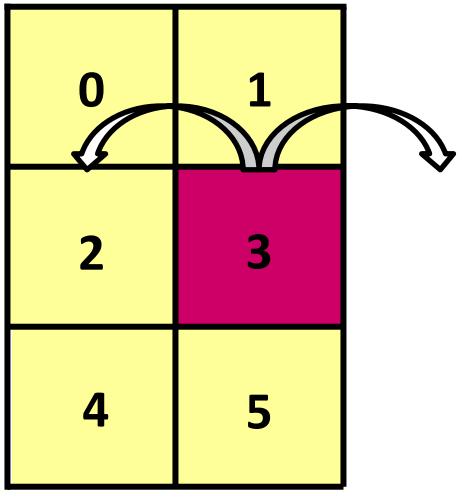
MPI_CART_SHIFT



- direction=0
- disp=2

MPI_CART_SHIFT Example

calling process : 3



- direction=1
- disp=1

Lab #2

MPI_CART_SHIFT : Fortran

```
PROGRAM cart_shift
USE mpi_f08
IMPLICIT NONE
TYPE(MPI_Comm)::oldcomm, newcomm
INTEGER::ndims=2, dimsize(0:1)
LOGICAL::periods(0:1), reorder
INTEGER::myrank,nprocs,i,j,rank
INTEGER::coords(0:1)
INTEGER::direction, disp,src,dest
CALL MPI_Init
CALL MPI_Comm_rank(MPI_COMM_WORLD,myrank)
CALL MPI_Comm_size(MPI_COMM_WORLD,nprocs)
oldcomm=MPI_COMM_WORLD
dimsize=(/3,2/)
periods=(/.TRUE., .FALSE./)
reorder=.TRUE.
CALL MPI_Cart_create(oldcomm, ndims,dimsize,periods, reorder,newcomm)
direction=0;  disp=1
CALL MPI_Cart_shift(newcomm, direction,disp,src,dest)
PRINT*,'rank:',myrank,'source=',src,'destination=',dest
CALL MPI_Finalize
END PROGRAM cart_shift
```

MPI_CART_SHIFT : Fortran

```
$ mpif90 cart_shfit.f90 -o cart_shfit.x
[sedu50@node8282 Fortran]$ mpirun -np 6 ./cart_shfit.x
 rank:           0 source=           4 destination=           2
 rank:           1 source=           5 destination=           3
 rank:           2 source=           0 destination=           4
 rank:           3 source=           1 destination=           5
 rank:           4 source=           2 destination=           0
 rank:           5 source=           3 destination=           1
```

Neighborhood Collective Communication

- 프로세스 토폴로지의 에지(edge)들을 따라서 통신을 수행하는 집합 통신
- Communicator의 모든 프로세스들에 의해 호출되어야 함
- 이웃이 존재하지 않으면(주기성이 false 일 때, 그 이웃은 MPI_PROCC_NULL로 정의됨
  - MPI_(I)NEIGHBOR_ALLGATHER, MPI_(I)NEIGHBOR_ALLGATHERV
  - MPI_(I)NEIGHBOR_ALLTOALL, MPI_(I)NEIGHBOR_ALLTOALLV

MPI_NEIGHBOR_ALLGATHER

MPI_NEIGHBOR_ALLGATHER(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm)		
IN	sendbuf	starting address of send buffer (choice)
IN	sendcount	number of elements sent to each neighbor(non-negative integer)
IN	sendtype	data type of send buffer elements (handle)
OUT	recvbuf	starting address of receive buffer (choice)
IN	recvcount	number of elements received from each neighbor(non-negative integer)
IN	recvtype	data type of receive buffer elements (handle)
IN	comm	communicator with topology structure (handle)

– ‘in place’ 옵션은 의미가 없음

- **통신 순서: dim 0(source, destination)**
  dim 1(source, destination)
- **MPI_PROC_NULL인 경우 통신이 발생하지 않음**

MPI_NEIGHBOR_ALLGATHER

- C

```
int MPI_Neighbor_allgather( const void* sendbuf, int sendcount,
                            MPIDatatype sendtype,
                            void* recvbuf, int recvcount,
                            MPI_Datatype recvtype, MPI_Comm comm)
```

- Fortran(mpi_f08 module)

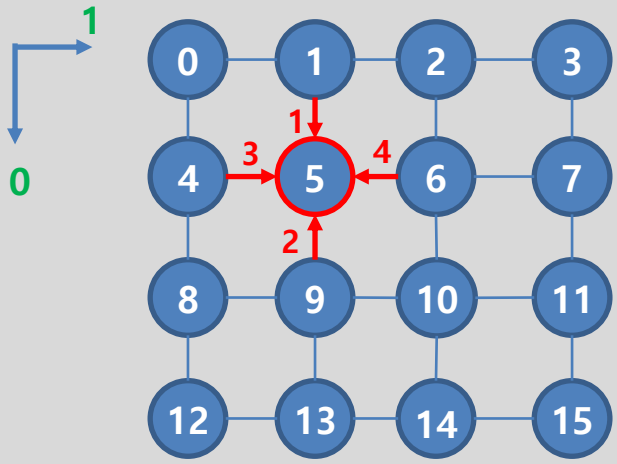
```
MPI_Neighbor_allgather( sendbuf, sendcount, sendtype,
                        recvbuf, recvcount, recvtype, comm, ierror )
```

	TYPE(*), DIMENSION(..), INTENT(IN) :: sendbuf
	TYPE(*), DIMENSION(..) :: recvbuf
	INTEGER, INTENT(IN) :: sendcount, recvcount
	TYPE(MPI_Datatype), INTENT(IN) :: sendtype, recvtype
	TYPE(MPI_Comm), INTENT(IN) :: comm
	INTEGER, OPTIONAL, INTENT(OUT) :: ierror

Lab #3

MPI_NEIGHBOR_ALLGATHER : Fortran

```
PROGRAM neighborhood_allgather
USE mpi_f08
IMPLICIT NONE
INTEGER::myrank,procs
TYPE(MPI_Comm)::comm_cart
INTEGER::NDIM=2,dims(2), coords(2),recvbuf(4)=-10
LOGICAL::periods(2)=.FALSE., reorder=.FALSE.
CALL MPI_Init
CALL MPI_Comm_rank(MPI_COMM_WORLD,myrank)
CALL MPI_Comm_size(MPI_COMM_WORLD,procs)
dims=(/4,4/)
CALL MPI_Cart_create(MPI_COMM_WORLD,NDIM,dims,periods,reorder,comm_cart)
CALL MPI_Cart_coords(comm_cart,myrank,NDIM,coords)
CALL MPI_Neighbor_allgather(myrank,1,MPI_INTEGER,recvbuf,1,MPI_INTEGER,comm_cart)
PRINT*,'myrank:',myrank,'recvbuf=',recvbuf
CALL MPI_Finalize
END PROGRAM neighborhood_allgather
```



- 통신 순서 : dim 0(source, destination)
  dim 1(source, destination)
- MPI_PROC_NULL인 경우 통신이 발생하지 않음

MPI_NEIGHBOR_ALLGATHER : Fortran

```
$ qsub -I -V -l select=1:ncpus=16:mpiprocs=16:ompthreads=1 -l walltime=08:00:00 -q
debug
$ mpif90 neighborhood_allgather.f90 -o neighborhood_allgather.x
$ mpirun -np 16 ./neighborhood_allgather.x
 myrank:            0 recvbuf=         -10           4         -10           1
 myrank:            1 recvbuf=         -10           5           0           2
 myrank:            2 recvbuf=         -10           6           1           3
 myrank:            3 recvbuf=         -10           7           2         -10
 myrank:            4 recvbuf=           0           8         -10           5
 myrank:            5 recvbuf=           1           9           4           6
 myrank:            6 recvbuf=           2          10           5           7
 myrank:            7 recvbuf=           3          11           6         -10
 myrank:            8 recvbuf=           4          12         -10           9
 myrank:            9 recvbuf=           5          13           8          10
 myrank:           10 recvbuf=           6          14           9          11
 myrank:           11 recvbuf=           7          15          10         -10
 myrank:           12 recvbuf=           8         -10         -10          13
 myrank:           13 recvbuf=           9         -10          12          14
 myrank:           14 recvbuf=          10         -10          13          15
 myrank:           15 recvbuf=          11         -10          14         -10
```

One-sided Communication

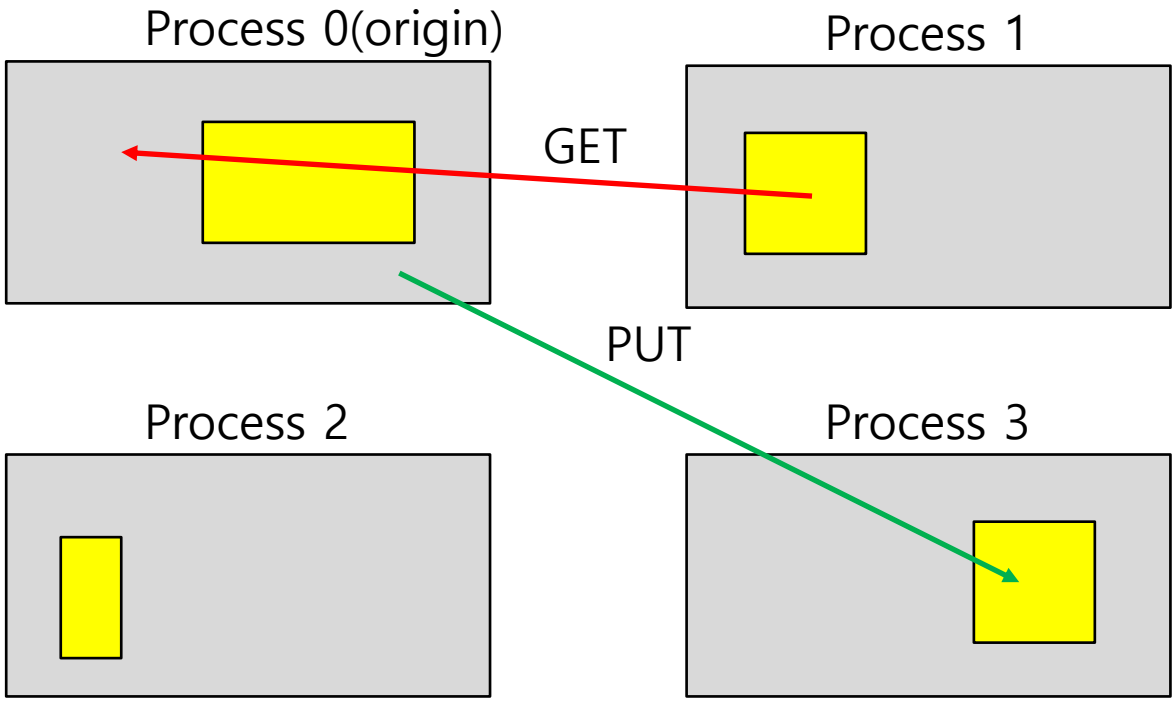
Introduction

- RMA(Remote Memory Access)
  - 한 프로세스가 송/수신측의 모든 통신 파라미터를 지정하는 것을 허용함으로써 MPI의 통신 메커니즘을 확장
  - Remote write: MPI_PUT, MPI_RPUT
  - Remote read : MPI_GET, MPI_RGET
  - Remote update : MPI_ACCUMULATE, MPI_RACCUMULATE
  - Remote read and update : MPI_GET_ACCUMULATE

- Origin : 호출을 수행하는 프로세스
- Target : 메모리 접근을 허용하는 프로세스

메모리 윈도우

- 단일 프로세스의 메모리 일부
  - 다른 프로세스들의 읽기(get), 쓰기(put), 갱신(accumulate)등의 메모리 연산을 허용하는 주소 공간
  - put과 get은 non-blocking 연산 ➔ 동기화 과정이 필요



메모리 윈도우 생성

- 윈도우 초기화 함수
  - 집합 함수
  - MPI_WIN_CREATE
  - MPI_WIN_ALLOCATE
  - MPI_WIN_ALLOCATE_SHARED
  - MPI_WIN_CREATE_DYNAMIC

MPI_WIN_CREATE

MPI_WIN_CREATE(base, size, disp_unit, info, comm, win)		
IN	base	initial address of window (choice)
IN	size	size of window in bytes (non-negative integer)
IN	disp_unit	local unit size for displacements, in bytes (positive integer)
IN	info	info argument (handle)
IN	comm	intra-communicator (handle)
OUT	win	window object returned by the call (handle)

- 집합 함수
- RMA 연산을 수행하기 위해 사용될 수 있는 window 객체 생성
- ‘base’ argument
  - In C : 메모리 영역의 시작 주소
  - In Fortran : 메모리 영역의 첫 번째 요소 혹은 단순 연속적인 전체 배열
- ‘size=0’로 지정함으로써 프로세스는 메모리 노출을 하지 않을 수 있음
  - base argument: MPI_BOTTOM
- ‘disp_unit’ : 보통 1을 사용

MPI_WIN_CREATE

- C

```
int MPI_Win_create( void *base, MPI_Aint size, int disp_unit, MPI_Info info,
                    MPI_Comm comm, MPI_Win *win)
```

- Fortran(mpi_f08 module)

MPI_Win_create(base, size, disp_unit, info, comm, win, ierror)	
	TYPE(*), DIMENSION(..), ASYNCHRONOUS :: base
	INTEGER(KIND=MPI_ADDRESS_KIND), INTENT(IN) :: size
	INTEGER, INTENT(IN) :: disp_unit
	TYPE(MPI_Info), INTENT(IN) :: info
	TYPE(MPI_Comm), INTENT(IN) :: comm
	TYPE(MPI_Win), INTENT(OUT) :: win
	INTEGER, OPTIONAL, INTENT(OUT) :: ierror

MPI_WIN_FREE

MPI_WIN_FREE(win)		
INOUT	win	window object (handle)

- 집합 호출(모든 프로세스가 호출해야 함)

- C

```
int MPI_Win_free(MPI_Win *win)
```

- Fortran(mpi_f08 module)

MPI_Win_free(win, ierror)	
	TYPE(MPI_Win), INTENT(INOUT) :: win
	INTEGER, OPTIONAL, INTENT(OUT) :: ierror

Lab #4

MPI_WIN_CREATE/MPI_WIN_FREE : C

```c
#include <stdio.h>
#include <mpi.h>
int main()
{
  int a,b,myrank;
  MPI_Win win;
  MPI_Init(NULL,NULL);
  MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
  if(myrank==0) a=1;
  if(myrank==1) a=10;
  if(myrank==2) a=100;
  printf("myrank:%d   a=%d\n",myrank,a);
  MPI_Win_create(&a,sizeof(int),1,MPI_INFO_NULL,MPI_COMM_WORLD,&win);
  MPI_Win_free(&win);
  MPI_Finalize();
  return 0;
}
```

```
$ mpicc win_create.c -o win_create.x
$ mpirun -np 3 ./win_create.x
myrank:0   a=1
myrank:1   a=10
myrank:2   a=100
```

MPI_WIN_CREATE/MPI_WIN_FREE : Fortran

```
PROGRAM win_create
USE mpi_f08
IMPLICIT NONE
INTEGER::a,b,myrank
INTEGER(KIND=MPI_ADDRESS_KIND)::size,lb
TYPE(MPI_Win)::win
CALL MPI_Init
CALL MPI_Comm_rank(MPI_COMM_WORLD,myrank)
IF(myrank==0) a=1
IF(myrank==1) a=10
IF(myrank==2) a=100
PRINT*,'myrnak:',myrank, 'a=',a
!size=STORAGE_SIZE(a)/8
CALL MPI_Type_get_extent(MPI_INTEGER,lb,size)
print*,'size',size
CALL MPI_Win_create(a,size,1,MPI_INFO_NULL,MPI_COMM_WORLD,win)
CALL MPI_Win_free(win)
CALL MPI_Finalize
END PROGRAM win_create
```

```
$ mpirun -np 3 ./win_create.x
 myrnak:           0 a=           1
 myrnak:           1 a=          10
 myrnak:           2 a=         100
```

MPI_WIN_FENCE

MPI_WIN_FENCE(assert, win)		
IN	assert	program assertion (integer)
IN	win	window object (handle)

- 메모리 윈도우에서 RMA 호출을 동기화 함
- ‘assert=0’는 언제나 유효함

- C

```
int MPI_Win_fence(int assert, MPI_Win win)
```

- Fortran(mpi_f08 module)

MPI_Win_fence(assert, win, ierror)	
	INTEGER, INTENT(IN) :: assert
	TYPE(MPI_Win), INTENT(IN) :: win
	INTEGER, OPTIONAL, INTENT(OUT) :: ierror

MPI_PUT

MPI_PUT(origin_addr, origin_count, origin_datatype, target_rank, target_disp, target_count, target_datatype, win)		
IN	origin_addr	initial address of origin buffer (choice)
IN	origin_count	number of entries in origin buffer (non-negative integer)
IN	origin_datatype	datatype of each entry in origin buffer (handle)
IN	target_rank	rank of target (non-negative integer)
IN	target_disp	displacement from start of window to target buffer (non-negative integer)
IN	target_count	number of entries in target buffer (non-negative integer)
IN	target_datatype	datatype of each entry in target buffer (handle)
IN	win	window object used for communication (handle)

- Origin 프로세스에 의한 send와 target 프로세스에 의한 receive 실행과 유사

MPI_PUT

- C

```
int MPI_Put(const void *origin_addr, int origin_count,
            MPI_Datatype origin_datatype, int target_rank, MPI_Aint target_disp,
            int target_count, MPI_Datatype target_datatype, MPI_Win win)
```

- Fortran(mpi_f08 module)

```
MPI_Put(origin_addr, origin_count, origin_datatype, target_rank, target_disp,
        target_count, target_datatype, win, ierror)
```

	TYPE(*), DIMENSION(..), INTENT(IN), ASYNCHRONOUS :: origin_addr
	INTEGER, INTENT(IN) :: origin_count, target_rank, target_count
	TYPE(MPI_Datatype), INTENT(IN) :: origin_datatype, target_datatype
	INTEGER(KIND=MPI_ADDRESS_KIND), INTENT(IN) :: target_disp
	TYPE(MPI_Win), INTENT(IN) :: win
	INTEGER, OPTIONAL, INTENT(OUT) :: ierror

MPI_GET

MPI_GET(origin_addr, origin_count, origin_datatype, target_rank, target_disp, target_count,target_datatype, win)		
OUT	origin_addr	initial address of origin buffer (choice)
IN	origin_count	number of entries in origin buffer (non-negative integer)
IN	origin_datatype	datatype of each entry in origin buffer (handle)
IN	target_rank	rank of target (non-negative integer)
IN	target_disp	displacement from window start to the beginning of the target buffer (non-negative integer)
IN	target_count	number of entries in target buffer (non-negative integer)
IN	target_datatype	datatype of each entry in target buffer (handle)
IN	win	window object used for communication (handle)

- 데이터 전송 방향이 반대인 것만 제외하면 MPI_PUT과 유사
- 데이터는 target memory에서 origin으로 복사됨

MPI_GET

- C

```
int MPI_Get( void *origin_addr, int origin_count, MPI_Datatype origin_datatype,
             int target_rank, MPI_Aint target_disp, int target_count,
             MPI_Datatype target_datatype, MPI_Win win)
```

- Fortran(mpi_f08 module)

```
MPI_Get( origin_addr, origin_count, origin_datatype, target_rank, target_disp,
         target_count, target_datatype, win, ierror)
```

	TYPE(*), DIMENSION(..), ASYNCHRONOUS :: origin_addr
	INTEGER, INTENT(IN) :: origin_count, target_rank, target_count
	TYPE(MPI_Datatype), INTENT(IN) :: origin_datatype, target_datatype
	INTEGER(KIND=MPI_ADDRESS_KIND), INTENT(IN) :: target_disp
	TYPE(MPI_Win), INTENT(IN) :: win
	INTEGER, OPTIONAL, INTENT(OUT) :: ierror

Lab #5

MPI_PUT/MPI_WIN_FENCE: Fortran

```
PROGRAM put
USE mpi_f08
IMPLICIT NONE
INTEGER::a,b,myrank
INTEGER(KIND=MPI_ADDRESS_KIND)::size, disp
TYPE(MPI_Win)::win
CALL MPI_Init
CALL MPI_Comm_rank(MPI_COMM_WORLD,myrank)
IF(myrank==0) a=1
IF(myrank==1) a=10
IF(myrank==2) a=100
PRINT*,'myrnak:',myrank, 'a=',a
size=STORAGE_SIZE(a)/8
CALL MPI_Win_create(a,size,1,MPI_INFO_NULL,MPI_COMM_WORLD,win)
CALL MPI_Win_fence(0,win)

IF(myrank==0) THEN
  disp=0
  CALL MPI_PUT(a,1,MPI_INTEGER,1,disp,1,MPI_INTEGER,win)
  CALL MPI_PUT(a,1,MPI_INTEGER,2,disp,1,MPI_INTEGER,win)
ENDIF
CALL MPI_Win_fence(0,win)
PRINT*,'MYRANK:',myrank, 'A=',a
CALL MPI_Win_free(win)
CALL MPI_Finalize
END PROGRAM put
```

MPI_PUT/MPI_WIN_FENCE: Fortran

```
$ mpif90 put.f90 -o put.x
$ mpirun -np 3 ./put.x
 myrnak:            0 a=           1
 myrnak:            1 a=          10
 myrnak:            2 a=         100

 MYRANK:            0 A=           1
 MYRANK:            1 A=           1
 MYRANK:            2 A=           1
```

MPI_PUT/MPI_WIN_FENCE: Fortran

```
PROGRAM put
USE mpi_f08
IMPLICIT NONE
INTEGER::a,b,myrank
INTEGER(KIND=MPI_ADDRESS_KIND)::size, disp
TYPE(MPI_Win)::win
CALL MPI_Init
CALL MPI_Comm_rank(MPI_COMM_WORLD,myrank)
IF(myrank==0) a=1
IF(myrank==1) a=10
IF(myrank==2) a=100
PRINT*,'myrnak:',myrank, 'a=',a
size=STORAGE_SIZE(a)/8
IF(MYRANK==0) THEN
  CALL MPI_Win_create(a,size,1,MPI_INFO_NULL,MPI_COMM_WORLD,win)
ELSE
  size=0
  ! 윈도우 생성 하지 않음
  CALL MPI_Win_create(MPI_BOTTOM,size,1,MPI_INFO_NULL,MPI_COMM_WORLD,win)
!  CALL MPI_Win_create(a,size,1,MPI_INFO_NULL,MPI_COMM_WORLD,win)
ENDIF
CALL MPI_Win_fence(0,win)
```

MPI_PUT/MPI_WIN_FENCE: Fortran

```
IF(myrank/=0) THEN
  disp=0
  CALL MPI_GET(a,1,MPI_INTEGER,0,disp,1,MPI_INTEGER,win)
ENDIF
CALL MPI_Win_fence(0,win)
IF(MYRANK==0)THEN
  disp=0
  a=200
  CALL MPI_PUT(a,1,MPI_INTEGER,1,disp,1,MPI_INTEGER,win)
  CALL MPI_PUT(a,1,MPI_INTEGER,2,disp,1,MPI_INTEGER,win)
ENDIF
CALL MPI_Win_fence(0,win)
PRINT*,'MYRANK:',myrank, 'A=',a
CALL MPI_Win_free(win)
CALL MPI_Finalize
END PROGRAM put
```

MPI_PUT/MPI_WIN_FENCE: Fortran

```
$ mpif90 win_create2.f90 -o win_create2.x
$ mpirun -np 3 ./win_create2.x ← 윈도우 생성하지 않았을 때
 myrnak:           0 a=           1
 myrnak:           1 a=          10
 myrnak:           2 a=         100

 MYRANK:           0 A=         200
 MYRANK:           1 A=           1
 MYRANK:           2 A=           1

$ mpirun -np 3 ./win_create2.x ← 윈도우 생성했을 때
 myrnak:           0 a=           1
 myrnak:           1 a=          10
 myrnak:           2 a=         100

 MYRANK:           0 A=         200
 MYRANK:           1 A=         200
 MYRANK:           2 A=         200
```

MPI_ACCUMULATE

MPI_ACCUMULATE(origin_addr, origin_count, origin_datatype, target_rank, target_disp, target_count, target_datatype, op, win)		
IN	origin_addr	initial address of buffer (choice)
IN	origin_count	number of entries in buffer (non-negative integer)
IN	origin_datatype	datatype of each entry (handle)
IN	target_rank	rank of target (non-negative integer)
IN	target_disp	displacement from start of window to beginning of target buffer (non-negative integer)
IN	target_count	number of entries in target buffer (non-negative integer)
IN	target_datatype	datatype of each entry in target buffer (handle)
IN	op	reduce operation (handle)
IN	win	window object (handle)

- 데이터를 기록하는 대신 연산 reduce 연산을 취하는 것을 제외하면 MPI_PUT과 동일

MPI_ACCUMULATE

- C

```
int MPI_Accumulate( const void *origin_addr, int origin_count,
                    MPI_Datatype origin_datatype, int target_rank,
                    MPI_Aint target_disp, int target_count,
                    MPI_Datatype target_datatype, MPI_Op op, MPI_Win win)
```

- Fortran(mpi_f08 module)

```
MPI_Accumulate(origin_addr, origin_count, origin_datatype, target_rank,
               target_disp, target_count, target_datatype, op, win, ierror)
```

	TYPE(*), DIMENSION(..), INTENT(IN), ASYNCHRONOUS :: origin_addr
	INTEGER, INTENT(IN) :: origin_count, target_rank, target_count
	TYPE(MPI_Datatype), INTENT(IN) :: origin_datatype, target_datatype
	INTEGER(KIND=MPI_ADDRESS_KIND), INTENT(IN) :: target_disp
	TYPE(MPI_Op), INTENT(IN) :: op
	TYPE(MPI_Win), INTENT(IN) :: win
	INTEGER, OPTIONAL, INTENT(OUT) :: ierror

Lab #6

MPI_ACCUMULATE: Fortran

```
PROGRAM put
USE mpi_f08
IMPLICIT NONE
INTEGER::a,b,myrank
INTEGER(KIND=MPI_ADDRESS_KIND)::size, disp
TYPE(MPI_Win)::win
CALL MPI_Init
CALL MPI_Comm_rank(MPI_COMM_WORLD,myrank)
IF(myrank==0) a=1
IF(myrank==1) a=10
IF(myrank==2) a=100
PRINT*,'myrnak:',myrank, 'a=',a
size=STORAGE_SIZE(a)/8
CALL MPI_Win_create(a,size,1,MPI_INFO_NULL,MPI_COMM_WORLD,win)
CALL MPI_Win_fence(0,win)

disp=0
IF(myrank==0) THEN
  CALL MPI_Accumulate(a,1,MPI_INTEGER,2,disp,1,MPI_INTEGER,MPI_SUM,win)
ENDIF
CALL MPI_Win_fence(0,win)
PRINT*,'MYRANK:',myrank, 'A=',a
CALL MPI_Win_free(win)
CALL MPI_Finalize
END PROGRAM put
```

MPI_ACCUMULATE: Fortran

```
$ mpif90 accumulate.f90 -o accumulate.x
$ mpirun -np 3 ./accumulate.x
 myrnak:            0 a=           1
 myrnak:            1 a=          10
 myrnak:            2 a=         100

 MYRANK:            0 A=           1
 MYRANK:            1 A=          10
 MYRANK:            2 A=         101
```

MPI_GET_ACCUMULATE

MPI_GET_ACCUMULATE(origin_addr, origin_count, origin_datatype, result_addr, result_count, result_datatype, target_rank, target_disp, target_count, target_datatype, op, win)		
IN	origin_addr	initial address of buffer (choice)
IN	origin_count	number of entries in origin buffer (non-negative integer)
IN	origin_datatype	datatype of each entry in origin buffer (handle)
OUT	result_addr	initial address of result buffer (choice)
IN	result_count	number of entries in result buffer (non-negative integer)
IN	result_datatype	datatype of each entry in result buffer (handle)
IN	target_rank	rank of target (non-negative integer)
IN	target_disp	displacement from start of window to beginning of target buffer (non-negative integer)
IN	target_count	number of entries in target buffer (non-negative integer)
IN	target_datatype	datatype of each entry in target buffer (handle)
IN	op	reduce operation (handle)
IN	win	window object (handle)

- 원격 데이터가 호출 측으로 반환된 후 accumulate 연산을 수행

MPI_GET_ACCUMULATE

- C

```
int MPI_Get_accumulate(const void *origin_addr, int origin_count,
                       MPI_Datatype origin_datatype, void *result_addr, int result_count,
                       MPI_Datatype result_datatype, int target_rank,
                       MPI_Aint target_disp, int target_count,
                       MPI_Datatype target_datatype, MPI_Op op, MPI_Win win)
```

- Fortran(mpi_f08 module)

```
MPI_Get_accumulate(origin_addr, origin_count, origin_datatype, result_addr,
                   result_count, result_datatype, target_rank, target_disp,
                   target_count, target_datatype, op, win, ierror)
```

	TYPE(*), DIMENSION(..), INTENT(IN), ASYNCHRONOUS :: origin_addr
	INTEGER, INTENT(IN) :: origin_count, result_count, target_rank, target_count
	TYPE(MPI_Datatype), INTENT(IN) :: origin_datatype, target_datatype, result_datatype
	INTEGER(KIND=MPI_ADDRESS_KIND), INTENT(IN) :: target_disp
	TYPE(MPI_Op), INTENT(IN) :: op
	TYPE(MPI_Win), INTENT(IN) :: win
	INTEGER, OPTIONAL, INTENT(OUT) :: ierror

Lab #7

MPI_GET_ACCUMULATE: Fortran

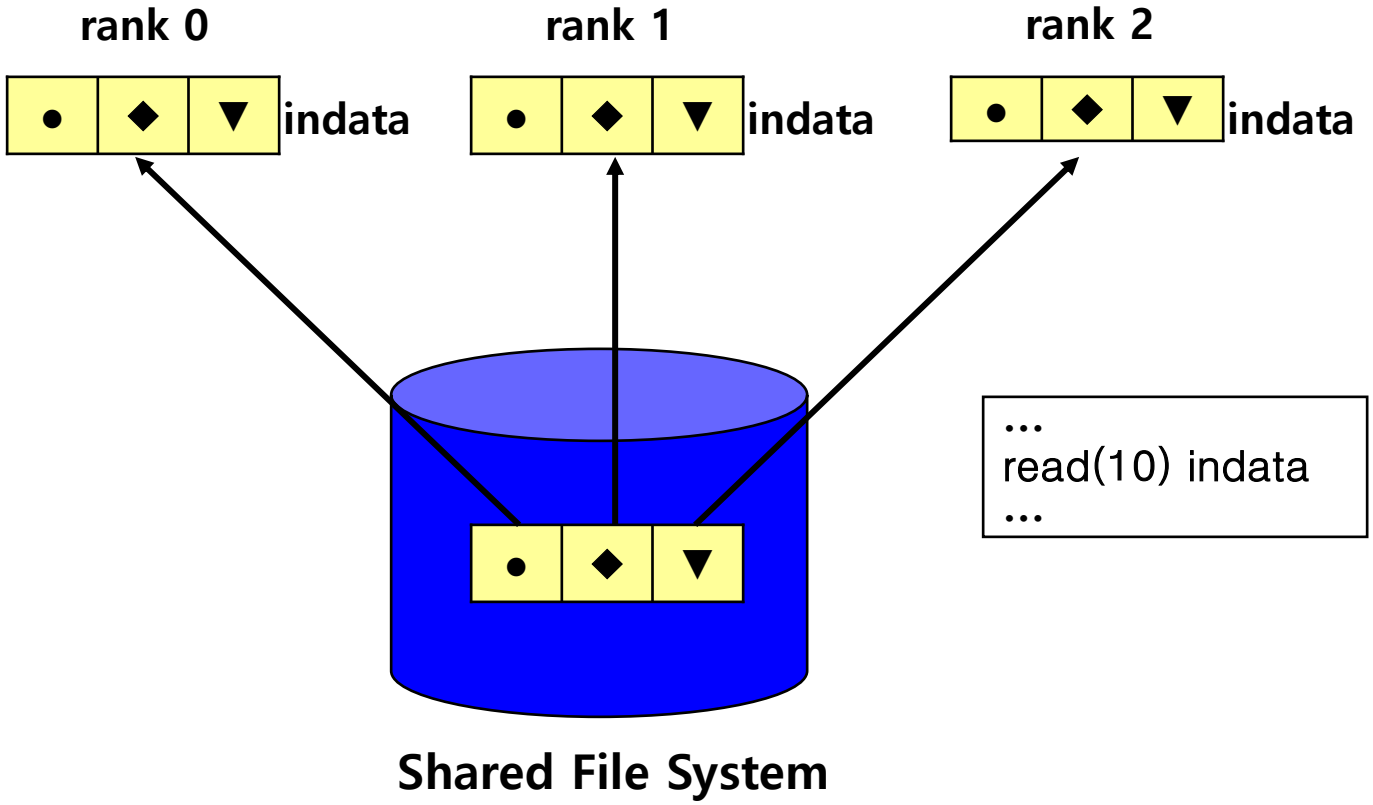
```
PROGRAM get_accumulate
USE mpi_f08
IMPLICIT NONE
INTEGER::a,b=0,myrank
INTEGER(KIND=MPI_ADDRESS_KIND)::size, disp
TYPE(MPI_Win)::win
CALL MPI_Init
CALL MPI_Comm_rank(MPI_COMM_WORLD,myrank)
IF(myrank==0) a=1
IF(myrank==1) a=10
IF(myrank==2) a=100
PRINT*,'myrnak:',myrank, 'a=',a
size=STORAGE_SIZE(a)/8
CALL MPI_Win_create(a,size,1,MPI_INFO_NULL,MPI_COMM_WORLD,win)
CALL MPI_Win_fence(0,win)

disp=0
IF(myrank==0) THEN
  CALL MPI_Get_accumulate(a,1,MPI_INTEGER,b,1,MPI_INTEGER,2,disp,1,&
                          MPI_INTEGER,MPI_SUM,win)
ENDIF
CALL MPI_Win_fence(0,win)
PRINT*,'MYRANK:',myrank, 'A=',a
PRINT*,'MYRANK:',myrank,'b=',b
CALL MPI_Win_free(win)
CALL MPI_Finalize
END PROGRAM get_accumulate
```

MPI_GET_ACCUMULATE: Fortran

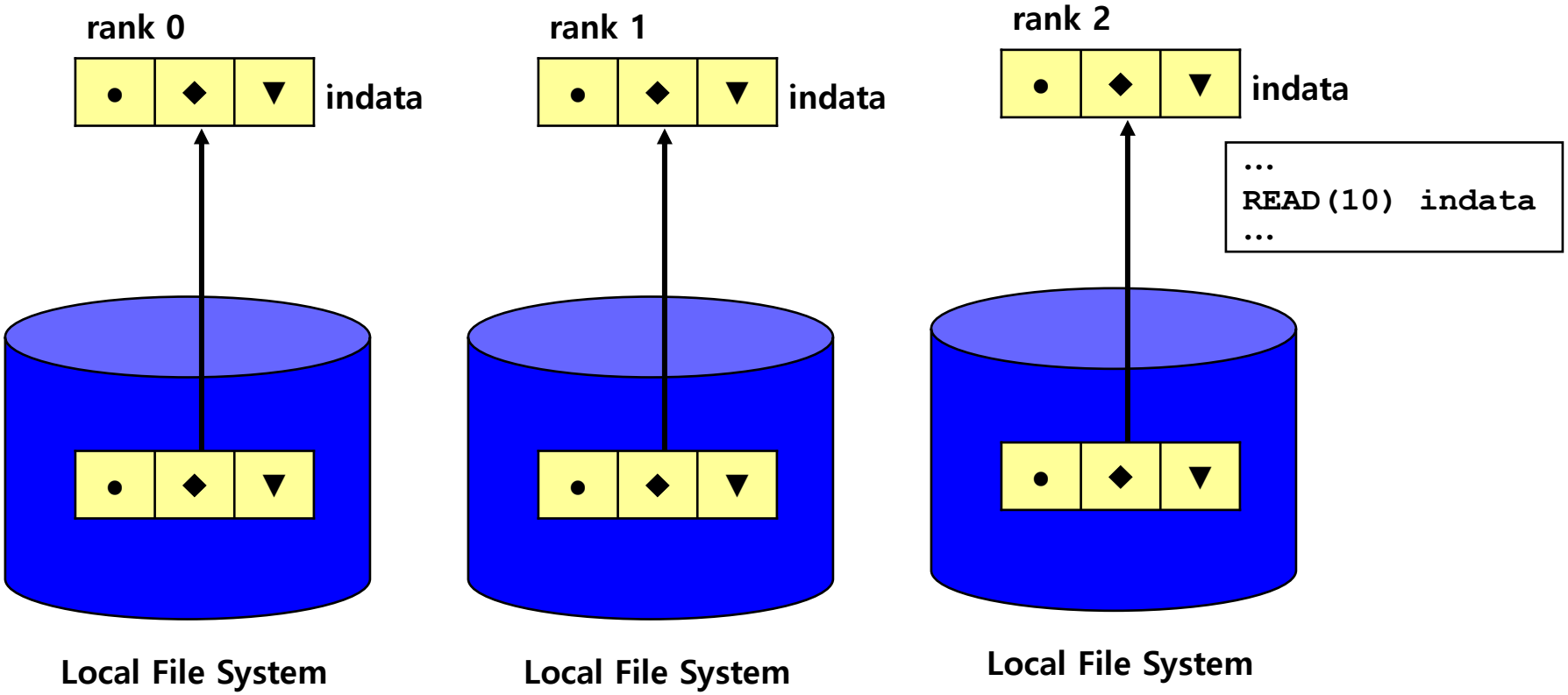
```
$ mpif90 get_accumulate.f90 -o get_accumulate.x
$ mpirun -np 3 ./get_accumulate.x
 myrnak:            0 a=           1
 myrnak:            1 a=          10
 myrnak:            2 a=         100

 MYRANK:            0 A=           1
 MYRANK:            1 A=          10
 MYRANK:            2 A=         101

 MYRANK:            0 b=         100
 MYRANK:            1 b=           0
 MYRANK:            2 b=           0
```

How to Parallelize Your Program: I/O

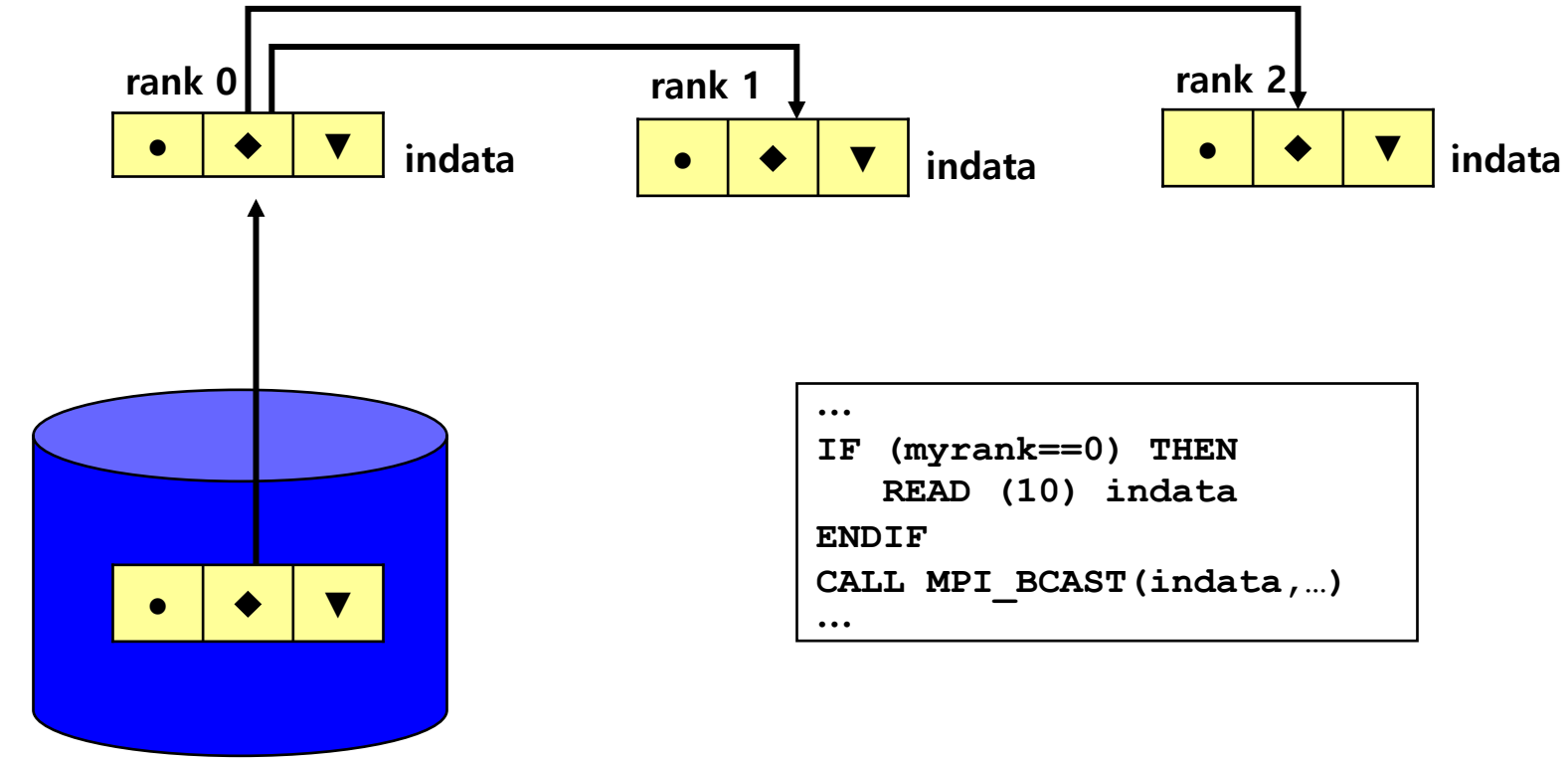
Parallelizing I/O: input 1.



Parallelizing I/O: input 2.



Parallelizing I/O: input 3.




```
…
IF (myrank==0) THEN
   READ (10) indata
ENDIF
CALL MPI_BCAST(indata,…)
…
```

Parallelizing I/O: standard out

➢ 표준 출력

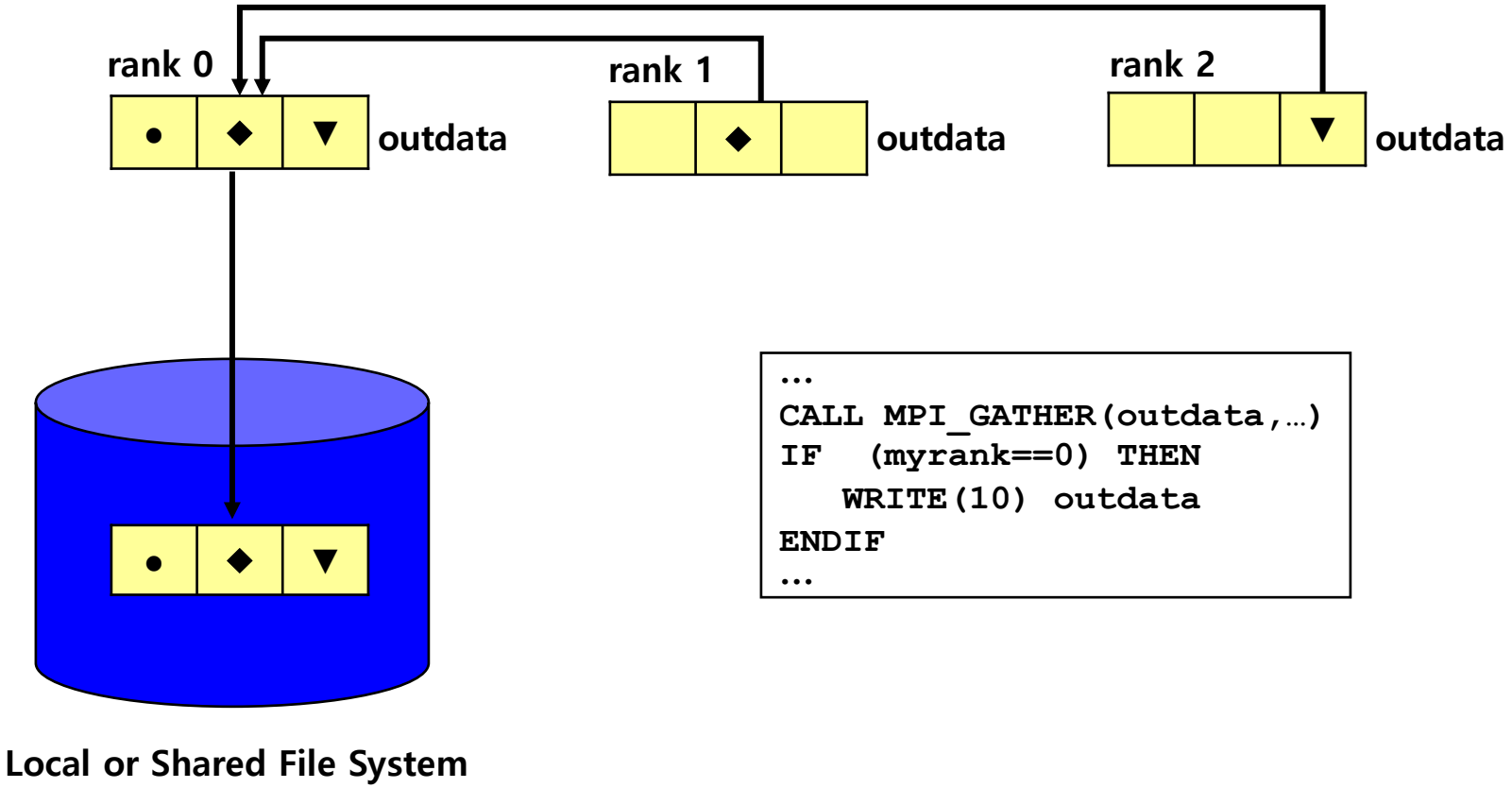
```
print *, 'I am :', myrank, 'Hello world!' ...
```

– 모든 프로세스 출력

```
if(myrank==0) then
  print *, 'I am :', myrank, 'Hello world!'
endif
```

– 원하는 프로세스만 출력

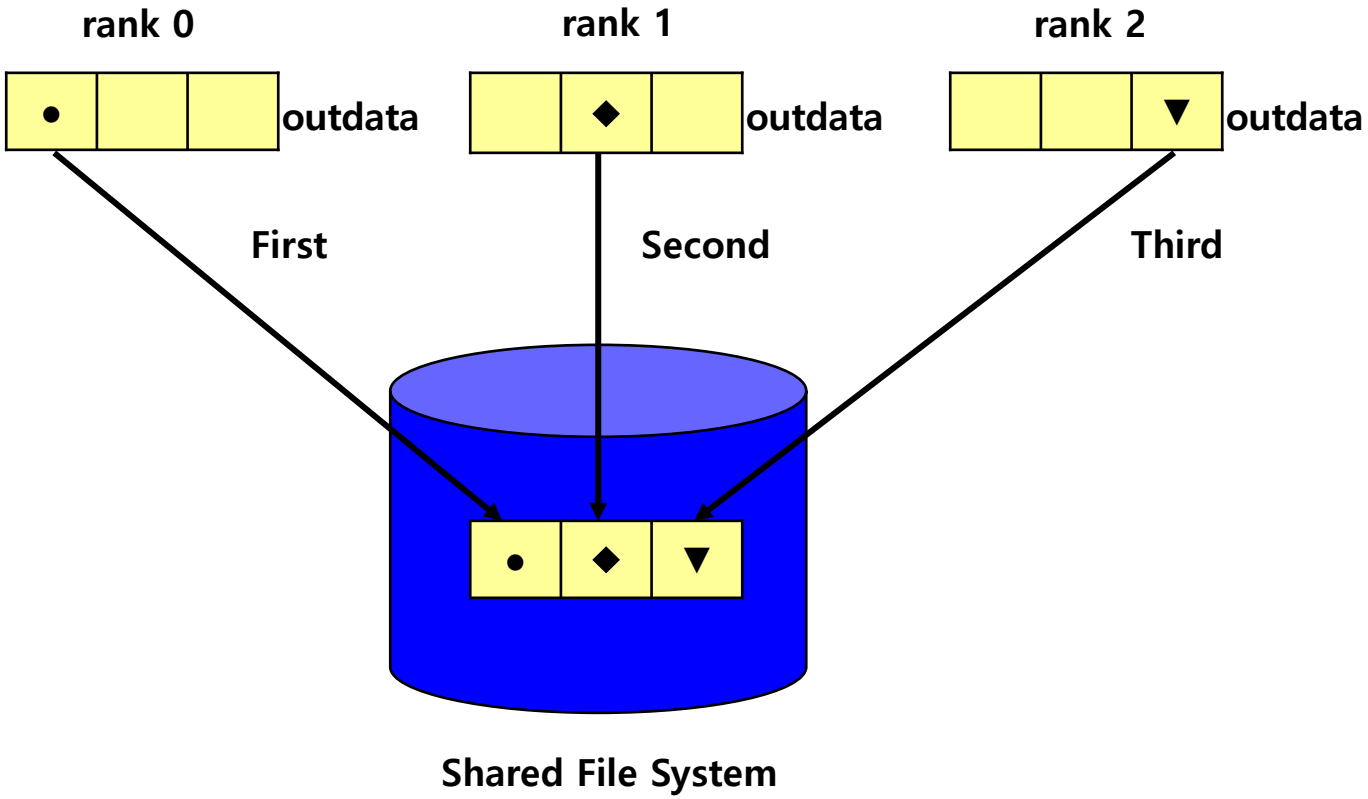
Parallelizing I/O: output 1.




```
...
CALL MPI_GATHER(outdata,...)
IF   (myrank==0) THEN
   WRITE(10) outdata
ENDIF
...
```

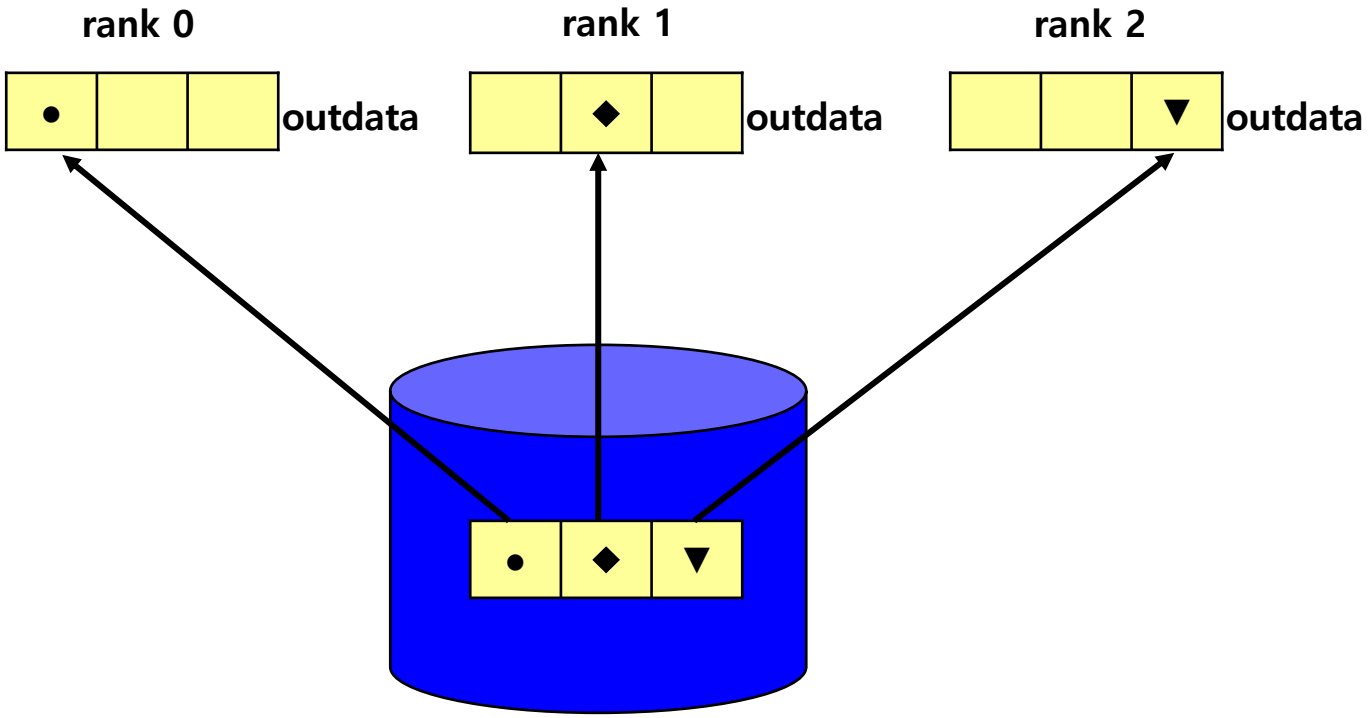
Parallelizing I/O: output 2.

- Sequential writing



MPI I/O: input

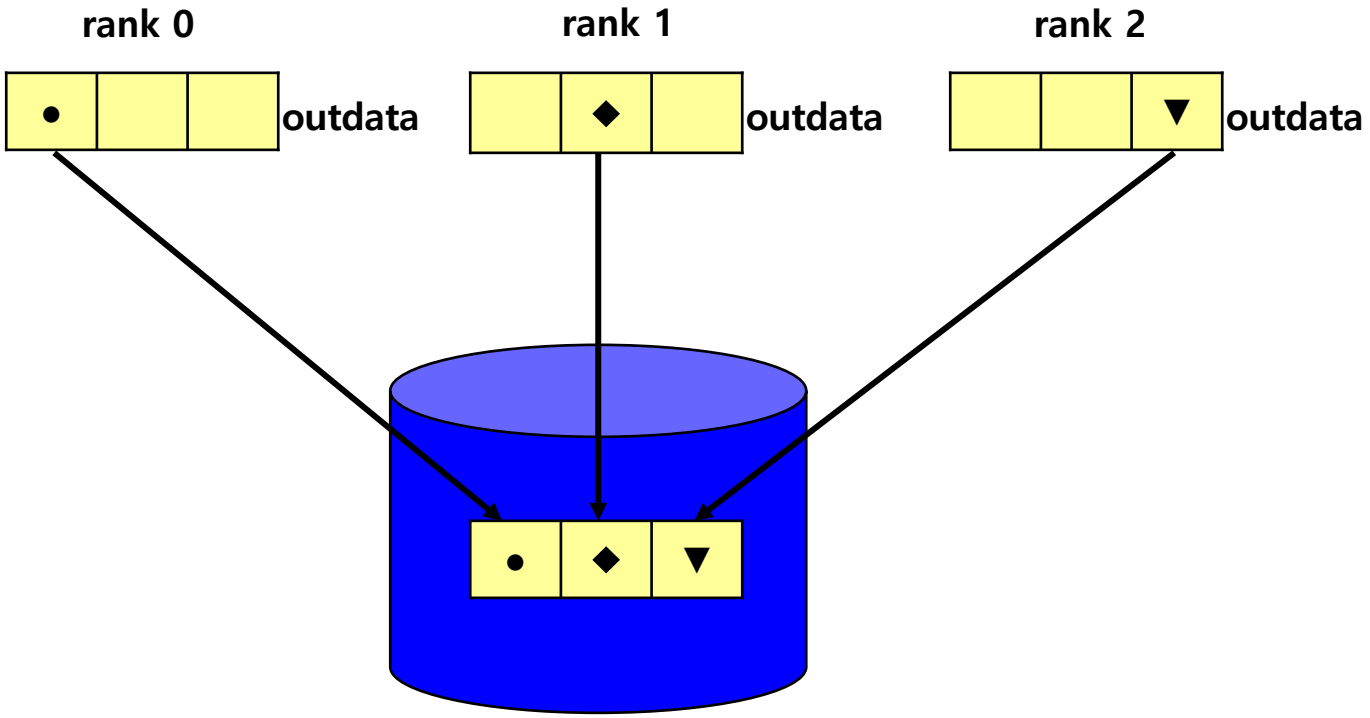
- Parallel reading




```
MPI_File_read_at(thefile,…);
```

MPI I/O: output

- Parallel writing




```
MPI_File_write_at(thefile,…);
```

MPI I/O

- 각 프로세스가 읽기/쓰기 하는 위치를 지정하는 방법에 세 가지 있음
  - Individual file pointers
    - MPI_File_seek / MPI_File_read
  - Calculate byte offsets
    - e.g., MPI_File_read_at
  - Access a shared file pointer
    - e.g., MPI_File_seek_shared, MPI_File_read_shared

MPI_FILE_OPEN

MPI_FILE_OPEN(comm, filename, amode, info, fh)		
IN	comm	communicator (handle)
IN	filename	name of file to open (string)
IN	amode	file access mode (integer)
IN	info	file access mode (integer)
OUT	fh	new file handle (handle)

- 집합 루틴
- 상수 MPI_INFO_NULL은 info가 지정될 필요 없을 때 사용
- Assess mode
  - MPI_MODE_RDONLY: read only
  - MPI_MODE_RDWR : reading and writing
  - MPI_MODE_WRONLY : write only
  - MPI_MODE_CREATE : crate the file if it does not exist
  - MPI_MODDE_EXCL : error if creating file that already exists
  - MPI_MODE_SEQUENTIAL : file will only be accessed sequentially
  - MPI_MODE_APPEND : set initial position of all file pointers to end of file.
- Flag를 결합하기 위해 ‘|’(C)나 ‘+’(Fortran) 사용

MPI_FILE_OPEN

- C

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

- Fortran(mpi_f08 module)

MPI_File_open(comm, filename, amode, info, fh, ierror)	
	TYPE(MPI_Comm), INTENT(IN) :: comm
	CHARACTER(LEN=*), INTENT(IN) :: filename
	INTEGER, INTENT(IN) :: amode
	YPE(MPI_Info), INTENT(IN) :: info
	TYPE(MPI_File), INTENT(OUT) :: fh
	INTEGER, OPTIONAL, INTENT(OUT) :: ierror

MPI_FILE_CLOSE

MPI_FILE_CLOSE(fh)		
INOUT	fh	new file handle (handle)

- C

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

- Fortran(mpi_f08 module)

MPI_File_close(fh, ierror)	
	TYPE(MPI_File), INTENT(INOUT) :: fh
	INTEGER, OPTIONAL, INTENT(OUT) :: ierror

MPI_FILE_SEEK

MPI_FILE_SEEK(fh, offset, whence)		
INOUT	fh	file handle (handle)
IN	offset	file offset (integer)
IN	whence	update mode (state)

- Whence에 따라 개별 파일 포인터를 업데이트
  - 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 file plus offset
- C

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

- Fortran(mpi_f08 module)

MPI_File_seek(fh, offset, whence, ierror)	
	TYPE(MPI_File), INTENT(IN) :: fh
	INTEGER(KIND=MPI_OFFSET_KIND), INTENT(IN) :: offset
	INTEGER, INTENT(IN) :: whence
	INTEGER, OPTIONAL, INTENT(OUT) :: ierror

MPI_FILE_READ

MPI_FILE_READ(fh, buf, count, datatype, status)		
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)

- 개별 파일 포인터를 사용하여 파일을 읽음
- C

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

- Fortran(mpi_f08 module)

MPI_File_read(fh, buf, count, datatype, status, ierror)	
	TYPE(MPI_File), INTENT(IN) :: fh
	TYPE(*), DIMENSION(..) :: buf
	INTEGER, INTENT(IN) :: count
	TYPE(MPI_Datatype), INTENT(IN) :: datatype
	TYPE(MPI_Status) :: status
	INTEGER, OPTIONAL, INTENT(OUT) :: ierror

MPI_FILE_WRITE

MPI_FILE_WRITE(fh, buf, count, datatype, status)		
INOUT	fh	file handle (handle)
IN	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)

– 개별 파일 포인터를 사용하여 파일을 읽음

MPI_FILE_WRITE

- C

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

- Fortran(mpi_f08 module)

MPI_File_write(fh, buf, count, datatype, status, ierror)	
	TYPE(MPI_File), INTENT(IN) :: fh
	TYPE(*), DIMENSION(..), INTENT(IN) :: buf
	INTEGER, INTENT(IN) :: count
	TYPE(MPI_Datatype), INTENT(IN) :: datatype
	TYPE(MPI_Status) :: status
	INTEGER, OPTIONAL, INTENT(OUT) :: ierror

MPI_FILE_READ_AT

MPI_FILE_READ_AT(fh, offset, buf, count, datatype, status)		
IN	fh	file handle (handle)
IN	offset	file offset (integer)
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)

- ‘offset’에 의해 지정된 위치에서 파일을 읽음

MPI_FILE_READ_AT

- C

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

- Fortran(mpi_f08 module)

MPI_File_read_at(fh, offset, buf, count, datatype, status, ierror)	
	TYPE(MPI_File), INTENT(IN) :: fh
	INTEGER(KIND=MPI_OFFSET_KIND), INTENT(IN) :: offset
	TYPE(*), DIMENSION(..) :: buf
	INTEGER, INTENT(IN) :: count
	TYPE(MPI_Datatype), INTENT(IN) :: datatype
	TYPE(MPI_Status) :: status
	INTEGER, OPTIONAL, INTENT(OUT) :: ierror

MPI_FILE_WRITE_AT

MPI_FILE_WRITE_AT(fh, offset, buf, count, datatype, status)		
INOUT	fh	file handle (handle)
IN	offset	file offset (integer)
IN	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)

- ‘offset’에 의해 지정된 위치에서 파일을 기록

MPI_FILE_WRITE_AT

- C

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

- Fortran(mpi_f08 module)

MPI_File_write_at(fh, offset, buf, count, datatype, status, ierror)	
	TYPE(MPI_File), INTENT(IN) :: fh
	INTEGER(KIND=MPI_OFFSET_KIND), INTENT(IN) :: offset
	TYPE(*), DIMENSION(..), INTENT(IN) :: buf
	INTEGER, INTENT(IN) :: count
	TYPE(MPI_Datatype), INTENT(IN) :: datatype
	TYPE(MPI_Status) :: status
	INTEGER, OPTIONAL, INTENT(OUT) :: ierror

Lab #8

FILE_SEEK/FILE_WRITE

```
#include <stdio.h>
#include "mpi.h"
int main()
{
  MPI_File fh;
  int buf[20]={0,}, rank,i,bufsize,nints,offset,data[20]={0,};
  MPI_Init(NULL,NULL);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  for(i=0;i<20;i++) buf[i]=i+1;
  MPI_File_open(MPI_COMM_WORLD,"test2.out",\
                MPI_MODE_CREATE|MPI_MODE_WRONLY,MPI_INFO_NULL, &fh);
  offset=rank*(20/4)*sizeof(int); // 20: # of data, 4: # of processes
  MPI_File_seek(fh,offset,MPI_SEEK_SET);
  MPI_File_write(fh,&buf[rank*5],5,MPI_INT,MPI_STATUS_IGNORE);

  MPI_File_close(&fh);
  MPI_Finalize();
  return 0;
}
```

```
$ mpicc file_write2.c -o file_write2.x
$ mpirun -np 4 ./file_write2.x

$ ls -l
…
-rw-r--r-- 1 sedu50 in0163   80 Mar 19 22:59 test2.out
…
```

FILE_SEEK/FILE_READ

```
#include <stdio.h>
#include <mpi.h>
int main()
{
  int rank,nprocs,bufsize,nints;
  int buf[20]={0,},i;
  MPI_File fh;
  MPI_Offset FILESIZE;
  MPI_Init(NULL,NULL);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  MPI_Comm_size(MPI_COMM_WORLD,&nprocs);
  MPI_File_open(MPI_COMM_WORLD,"./test2.out",\
                MPI_MODE_RDONLY,MPI_INFO_NULL,&fh);
  MPI_File_get_size(fh,&FILESIZE);
  bufsize=FILESIZE/nprocs;
  nints=bufsize/sizeof(int);
  MPI_File_seek(fh,rank*bufsize,MPI_SEEK_SET);
  MPI_File_read(fh,&buf[rank*nints],nints,MPI_INT,MPI_STATUS_IGNORE);
  printf("rank:%d  buf=",rank);
  for(i=0;i<20;i++) printf("%d  ",buf[i]);
  printf("\n");
  MPI_Finalize();
  return 0;
}
```

FILE_SEEK/FILE_READ

```
$ mpicc file_read.c -o file_read.x
$ mpirun -np 4 ./file_read.x
rank:0  buf=1 2 3 4 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
rank:1  buf=0 0 0 0 0 6 7 8 9 10 0 0 0 0 0 0 0 0 0 0
rank:2  buf=0 0 0 0 0 0 0 0 0 0 11 12 13 14 15 0 0 0 0 0
rank:3  buf=0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 16 17 18 19 20
```

Lab #9

FILE_WRITE_AT

```
#include <stdio.h>
#include "mpi.h"
int main()
{
  MPI_File fh;
  int buf[20]={0,}, rank,i,bufsize,nints,offset,data[20]={0,};
  MPI_Init(NULL,NULL);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  for(i=0;i<20;i++) buf[i]=i+1;
  MPI_File_open(MPI_COMM_WORLD,"test3.out", \
                MPI_MODE_CREATE|MPI_MODE_WRONLY,MPI_INFO_NULL, &fh);
  offset=rank*(20/4)*sizeof(int); // 20: # of data, 4: # of processes
  MPI_File_write_at(fh,offset,&buf[rank*5],5,MPI_INT,MPI_STATUS_IGNORE);

  MPI_File_close(&fh);
  MPI_Finalize();
  return 0;
}
```

```
$ mpicc file_write3.c -o file_write3.x
$ mpirun -np 4 ./file_write3.x
$ diff test3.out test2.out
```

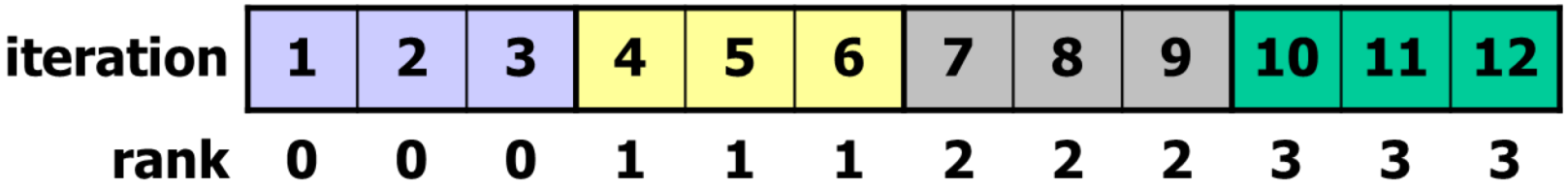
FILE_READ_AT

```
#include <stdio.h>
#include <mpi.h>
int main(){
  int rank,nprocs,bufsize,nints;
  int buf[20]={0,},i;
  MPI_File fh;
  MPI_Offset FILESIZE;
  MPI_Init(NULL,NULL);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  MPI_Comm_size(MPI_COMM_WORLD,&nprocs);
  MPI_File_open(MPI_COMM_WORLD,"./test2.out",MPI_MODE_RDONLY,\
                MPI_INFO_NULL,&fh);
  MPI_File_get_size(fh,&FILESIZE);
  bufsize=FILESIZE/nprocs;
  nints=bufsize/sizeof(int);
  MPI_File_read_at(fh,rank*5*sizeof(int),&buf[rank*nints],nints,\
                   MPI_INT,MPI_STATUS_IGNORE);
  printf("rank:%d  buf=",rank);
  for(i=0;i<20;i++) printf("%d  ",buf[i]);
  printf("\n");
  MPI_Finalize();
  return 0;
}
```

```
$ mpicc file_read2.c -o file_read2.x
$ mpirun -np 4 ./file_read2.x
rank:0  buf=1 2 3 4 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
rank:1  buf=0 0 0 0 0 6 7 8 9 10 0 0 0 0 0 0 0 0 0 0
rank:2  buf=0 0 0 0 0 0 0 0 0 0 11 12 13 14 15 0 0 0 0 0
rank:3  buf=0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 16 17 18 19 20
```

How to Parallelize Your Program: Loop

Block Distribution



Block Distribution

- Suppose when you divide n by p, the quotient is q and the remainder is r.
  - **n = p X q + r**

- Processes 0..r-1 are assigned *q* + 1 iterations each. The other processes are assigned *q* iterations.
  - **n = r(q+1) + (p-r)q**

Iteration	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Rank	0	0	0	0	1	1	1	1	2	2	2	3	3	3

Block Distribution: C

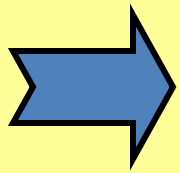
```
void  para_range(int  n1,int  n2,  int  nprocs,  int
  myrank, int *ista, int *iend){
   int iwork1, iwork2;
   iwork1 = (n2-n1+1)/nprocs;
   iwork2 = (n2-n1+1) % nprocs;
   *ista= myrank*iwork1 + n1 + min(myrank, iwork2);
   *iend = *ista + iwork1 - 1;
   if(iwork2 > myrank) *iend = *iend + 1;
}

int min(int x, int y){
 int v;
 if (x>=y) v = y;
 else v = x;
 return v;
}
```

Block Distribution: Fortran

```
SUBROUTINE para_range(n1, n2, nprocs, irank,
  ista, iend)
  iwork1 = (n2 - n1 + 1) / nprocs
  iwork2 = MOD(n2 - n1 + 1, nprocs)
  ista = irank * iwork1 + n1 + MIN(irank,
  iwork2)
  iend = ista + iwork1 - 1
  IF (iwork2 > irank) iend = iend + 1
END
```

Cyclic Distribution

```
DO i = n1, n2                    DO i = n1+myrank, n2, nprocs
   computation                      computation
ENDDO                            ENDDO
```

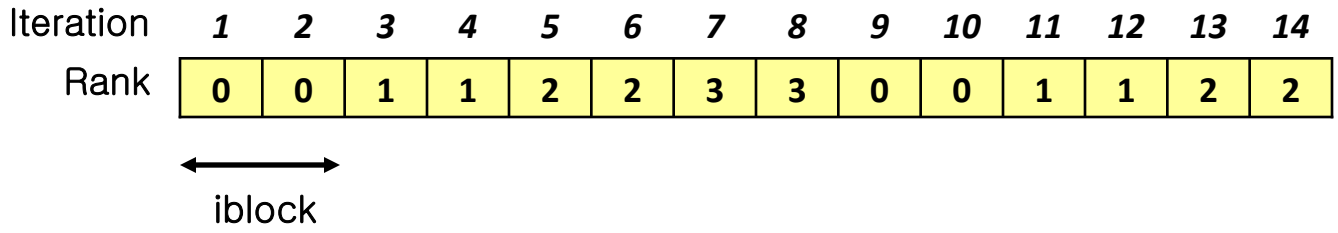


Iteration	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Rank	0	1	2	3	0	1	2	3	0	1	2	3	0	1

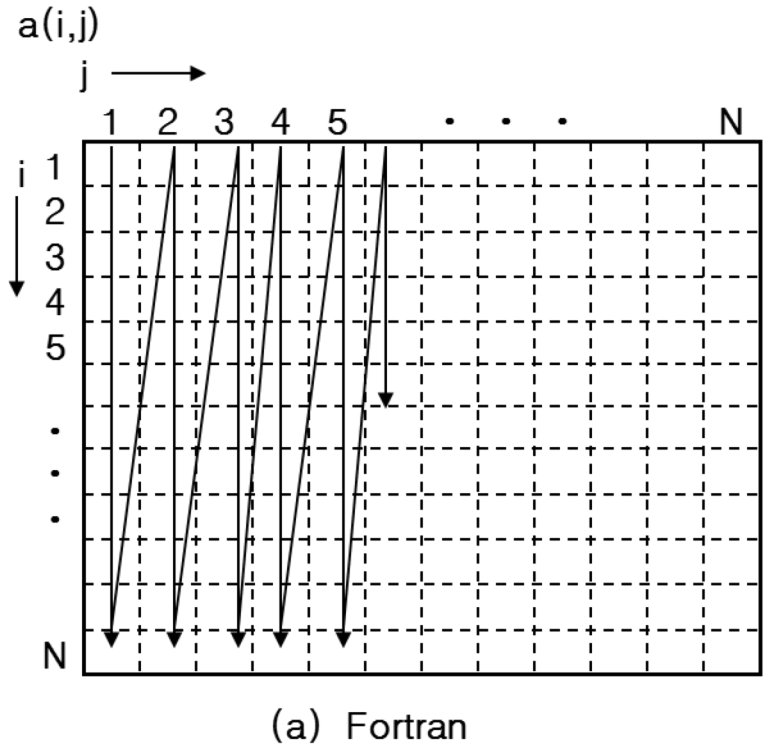
- More balanced workload for processes than the block distribution
- More cache misses than the block distribution

Block-Cyclic Distribution

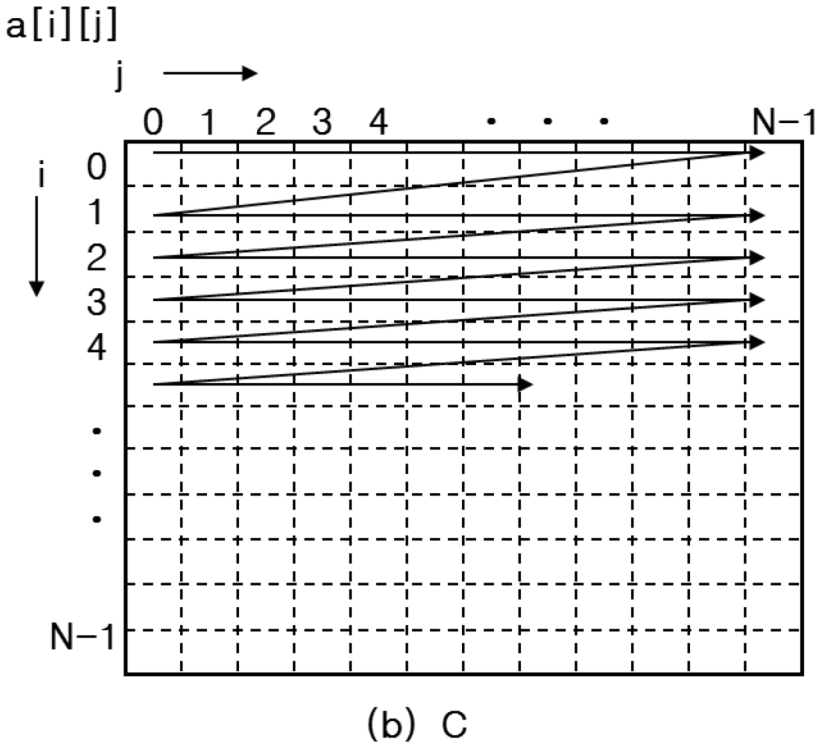
```
DO ii = n1+myrank*iblock, n2, nprocs*iblock
   DO i = ii, MIN(ii+iblock-1, n2)
      computation
   ENDDO
ENDDO
```



Nested Loop



(a) Fortran



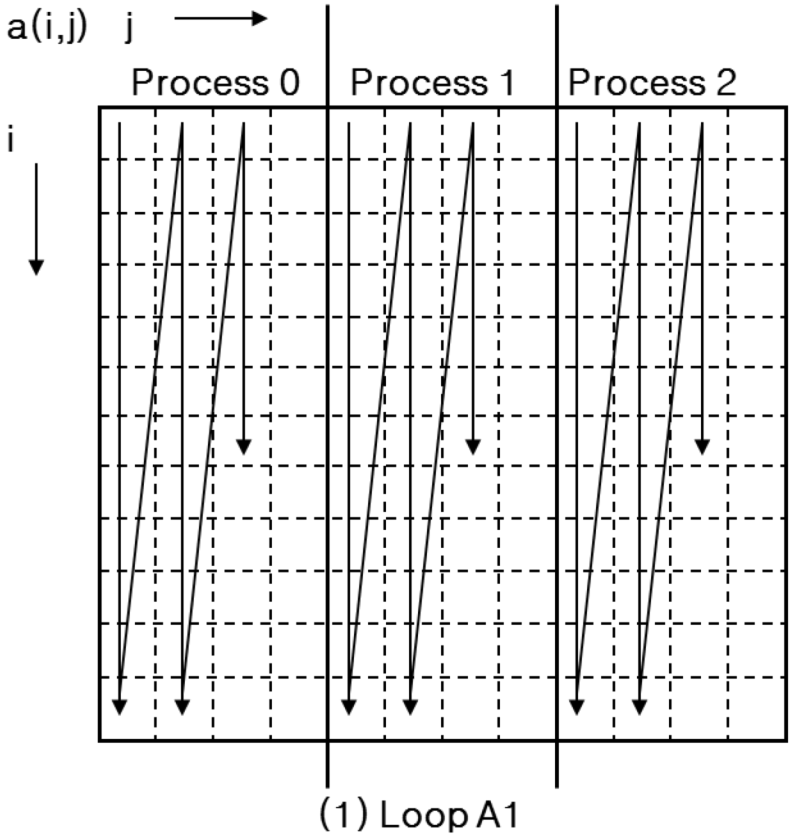
(b) C

Nested Loop Parallelization

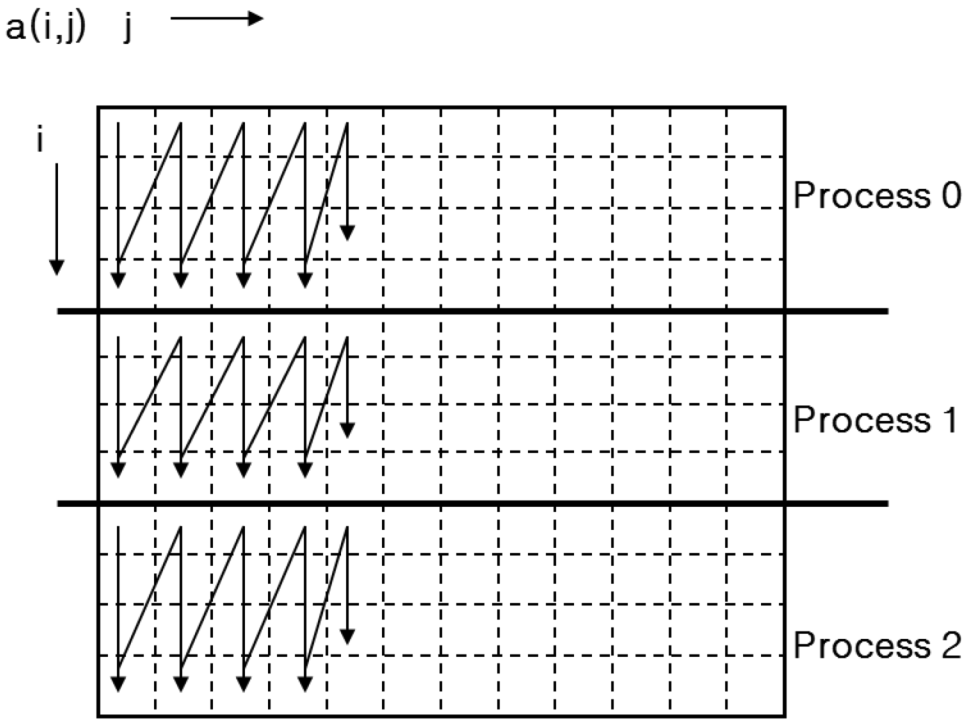
Loop A1	Loop A2

DO j = jsta, jend
  DO i = 1, n
  a(i,j) = b(i,j) + c(i,j)
  ENDDO
ENDDO
	DO j = 1, n
	  DO i = ista, iend
	  a(i,j) = b(i,j) + c(i,j)
	  ENDDO
	ENDDO

Nested Loop Parallelization



(1) Loop A1



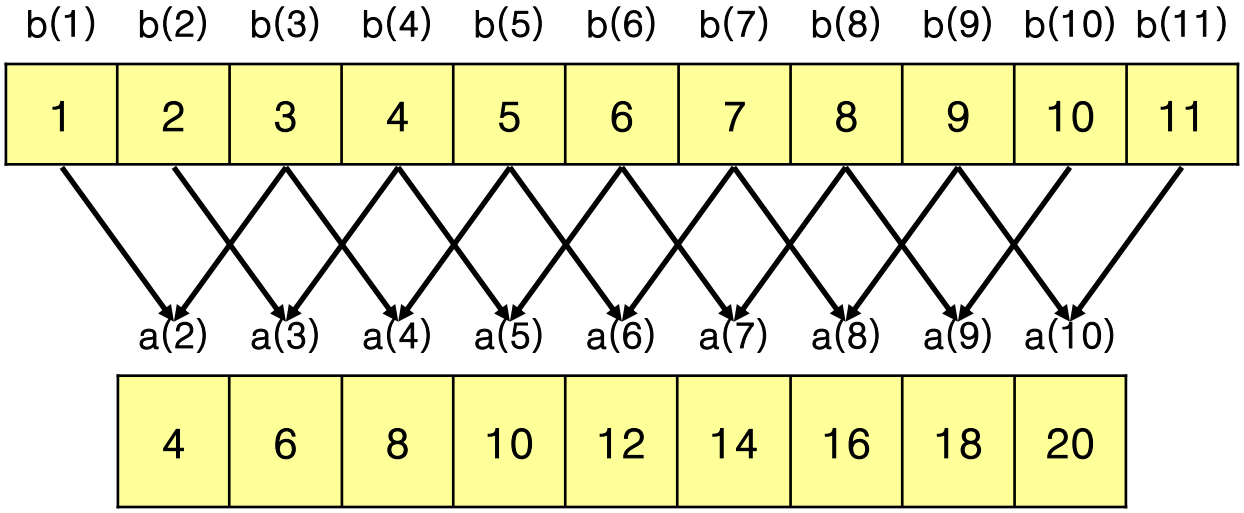
(2) Loop A2

1D FDM

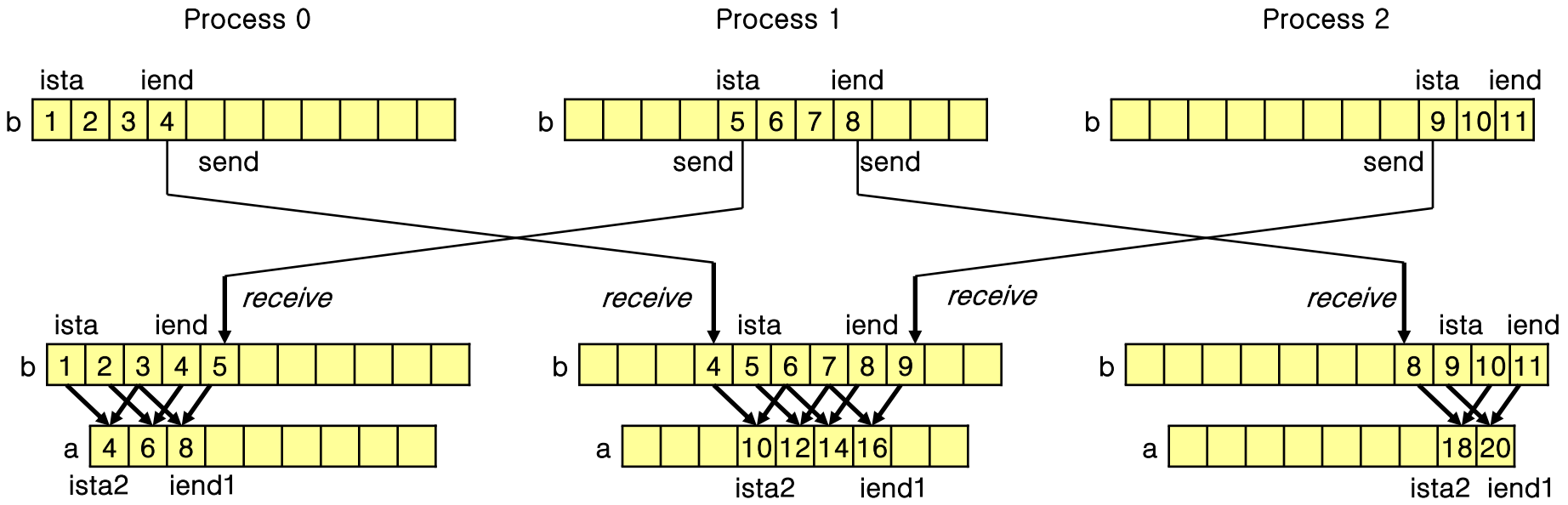
Fortran	C

PROGRAM 1D_fdm_serial
IMPLICIT REAL*8 (a-h, o-z)
PARAMETER (n=11)
DIMENSION a(n), b(n)
DO i = 1, n
  b(i) = i
ENDDO
DO i = 2, n-1
  a(i) = b(i-1) + b(i+1)
ENDDO
END
	/*1D_fdm_serial*/
	#define n 11
	main(){
	  double a[n], b[n];
	  int i;
	  for(i=0; i<n; i++)
	  b[i] = i+1;
	  for(i=1; i<n-1; i++)
	  a[i] = b[i-1] + b[i+1];
	}

1D FDM: data dependence



1D FDM: Data Dependence and Movements



Lab #10

1D FDM: C

```
/*parallel_1D_fdm*/
#include <mpi.h>
#define n 11
void para_range(int, int, int, int, int*, int*);
int min(int, int);
main(int argc, char *argv[]){
  int i, nprocs, myrank ;
  double a[n], b[n];
  int ista, iend, ista2, iend1, inext, iprev;
  MPI_Request isend1, isend2, irecv1, irecv2;
  MPI_Status istatus;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  para_range(0, n-1, nprocs, myrank, &ista, &iend);
  ista2 = ista; iend1 = iend;
  if(myrank==0) ista2=1;
  if(myrank==nprocs-1)iend1=n-2;
```

1D FDM: C

```
inext=myrank+1; iprev=myrank-1;
  if(myrank==nprocs-1) inext=MPI_PROC_NULL;
  if(myrank==0) iprev=MPI_PROC_NULL;
  MPI_Isend(&b[iend], 1, MPI_DOUBLE, inext ,1, MPI_COMM_WORLD,
   &isend1);
  MPI_Isend(&b[ista], 1, MPI_DOUBLE, iprev, 1, MPI_COMM_WORLD,
   &isend2);
  MPI_Irecv(&b[ista-1], 1, MPI_DOUBLE, iprev, 1, MPI_COMM_WORLD,
   &irecv1);
  MPI_Irecv(&b[iend+1], 1, MPI_DOUBLE, inext, 1, MPI_COMM_WORLD,
   &irecv2);
  MPI_Wait(&isend1, &istatus);
  MPI_Wait(&isend2, &istatus);
  MPI_Wait(&irecv1, &istatus);
  MPI_Wait(&irecv2, &istatus);
  for(i=ista2; i<=iend1; i++) a[i] = b[i-1] + b[i+1];
  MPI_Finalize();
}
```

1D FDM: Fortran

```
PROGRAM parallel_1D_fdm
INCLUDE 'mpif.h'
PARAMETER (n=11)
DIMENSION a(n), b(n)
INTEGER istatus(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 para_range(1, n, nprocs, myrank, ista, iend)
ista2 = ista; iend1 = iend
IF (myrank == 0) ista2 = 2
IF (myrank == nprocs-1) iend1 = n-1
inext = myrank + 1; iprev = myrank - 1
IF (myrank == nprocs-1) inext = MPI_PROC_NULL
IF (myrank == 0)        iprev = MPI_PROC_NULL

DO i = ista, iend
   b(i) = i
ENDDO
```

1D FDM: Fortran

```
CALL MPI_ISEND(b(iend), 1, MPI_REAL, inext, 1, &
     MPI_COMM_WORLD,  isend1, ierr)
CALL MPI_ISEND(b(ista), 1, MPI_REAL, iprev, 1, &
     MPI_COMM_WORLD, isend2, ierr)
CALL MPI_IRECV(b(ista-1), 1, MPI_REAL, iprev, 1, &
     MPI_COMM_WORLD, irecv1, ierr)
CALL MPI_IRECV(b(iend+1), 1, MPI_REAL, inext, 1, &
     MPI_COMM_WORLD, irecv2, ierr)
CALL MPI_WAIT(isend1, istatus, ierr)
CALL MPI_WAIT(isend2, istatus, ierr)
CALL MPI_WAIT(irecv1, istatus, ierr)
CALL MPI_WAIT(irecv2, istatus, ierr)

DO i = ista2, iend1
   a(i) = b(i-1) + b(i+1)
ENDDO
CALL MPI_FINALIZE(ierr)
END
```

Break!!

Lab #11

2D FDM

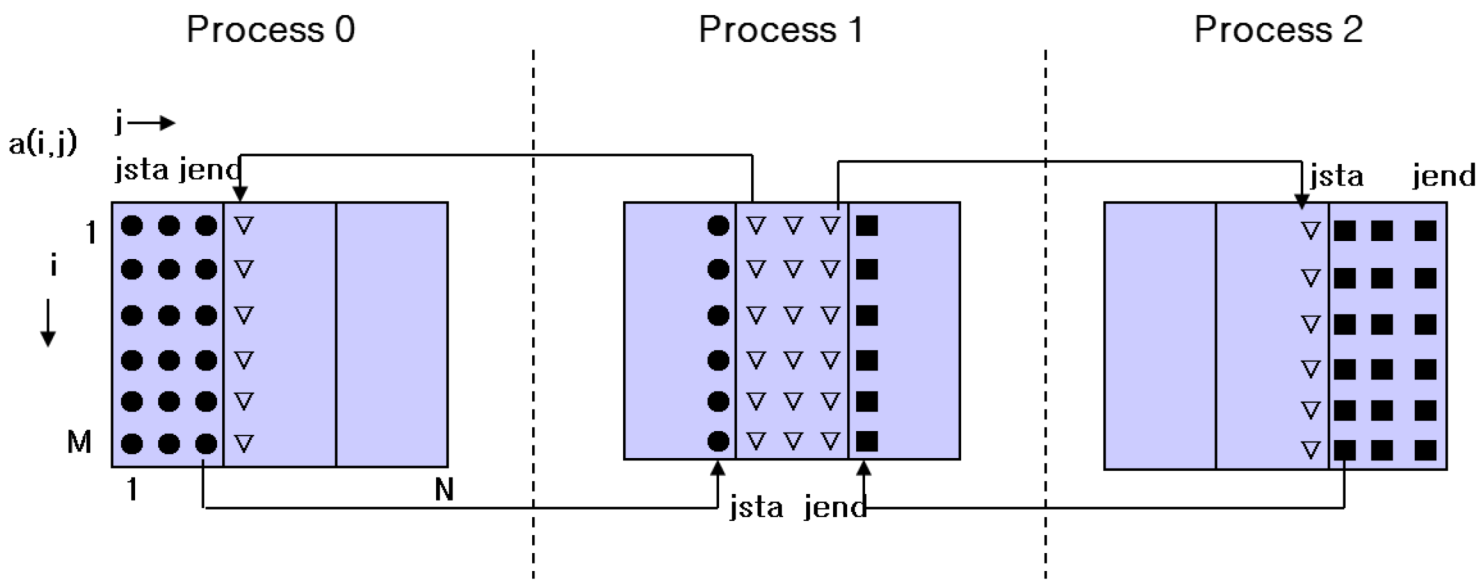
Fortran

```
...
PARAMETER (m=6,n=9)
DIMENSION a(m,n), b(m,n)
DO j = 1, n
   DO i = 1, m
      a(i,j) = i+10.0*j
   ENDDO
ENDDO
DO j = 2, n-1
   DO i = 2, m-1
      b(i,j) = a(i-1,j)+a(i,j-1) &
           + a(i,j+1) + a(i+1,j)
   ENDDO
ENDDO
...
```

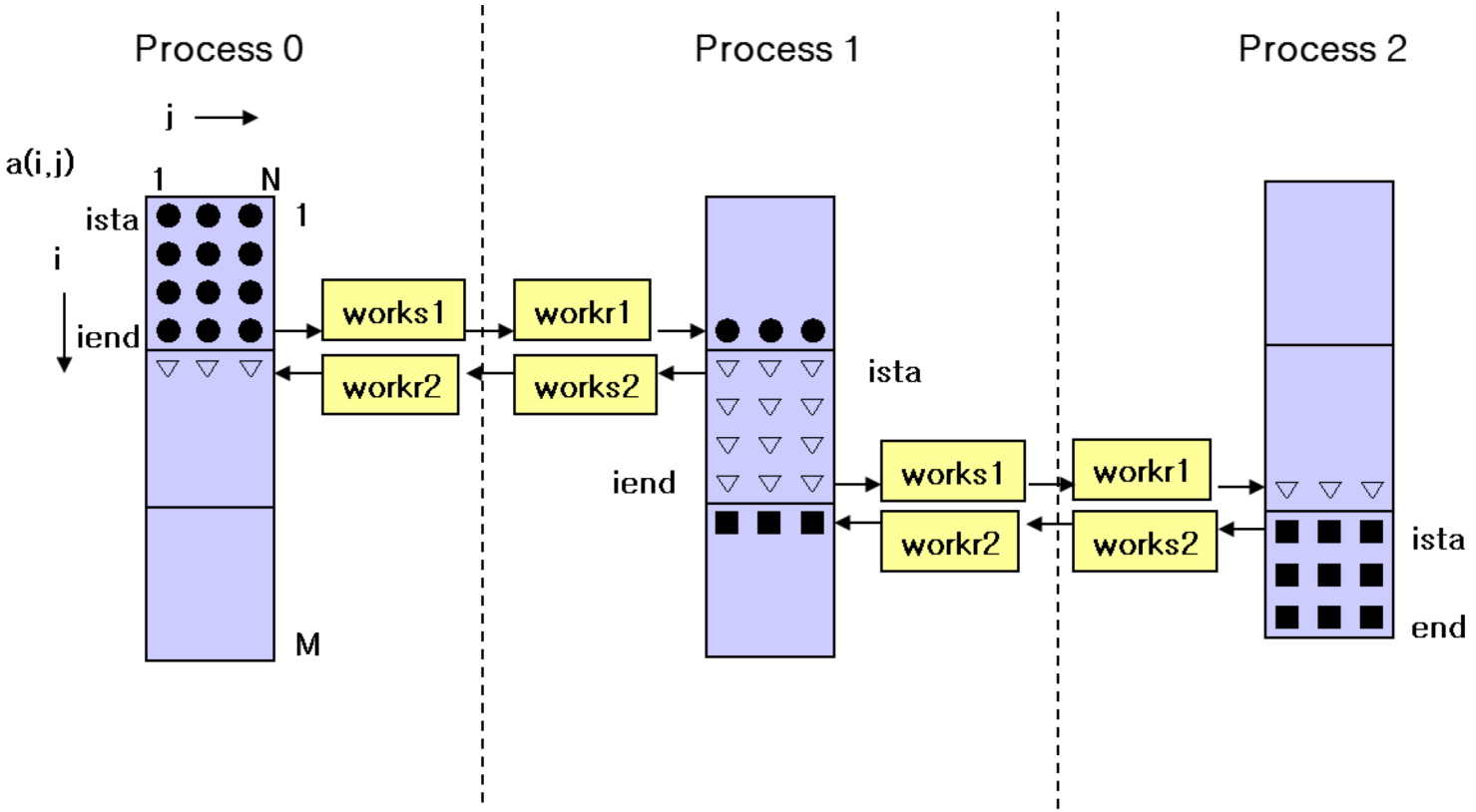
C

```
...
#define m 6
#define n 9
main(){
  double a[m][n], b[m][n];
  for(i=0; i<m; i++)
     for(j=0; j<n; j++)
        a[i][j] = (i+1)+10.*(j+1);
  for(i=1; i<m-1; i++)
     for(j=1; j<n-1; j++)
        b[i][j] =
           a[i-1][j] + a[i][j-1]
         + a[i][j+1] + a[i+1][j]
  ...
}
```

2D FDM: Column-Wise



2D FDM: Row-Wise



2D FDM(Column-Wise): C

```
/*parallel_2D_FDM_column*/
#include <mpi.h>
#define m 6
#define n 9
void para_range(int, int, int, int, int*, int*);
int min(int, int);
main(int argc, char *argv[]){
  int i, j, nprocs, myrank ;
  double a[m][n],b[m][n];
  double works1[m],workr1[m],works2[m],workr2[m];
  int jsta, jend, jsta2, jend1, inext, iprev;
  MPI_Request isend1, isend2, irecv1, irecv2;
  MPI_Status istatus;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
```

2D FDM(Column-Wise): C

```
para_range(0, n-1, nprocs, myrank, &jsta, &jend);
jsta2 = jsta; jend1 = jend;
if(myrank==0) jsta2=1;
if(myrank==nprocs-1) jend1=n-2;
inext = myrank + 1;
iprev = myrank - 1;
if (myrank == nprocs-1) inext = MPI_PROC_NULL
if (myrank == 0) iprev = MPI_PROC_NULL
for(i=0; i<m; i++)
   for(j=jsta; j<=jend; j++)  a[i][j] = i + 10.0 * j
if(myrank != nprocs-1)
   for(i=0; i<m; i++) works1[i]=a[i][jend];
if(myrank != 0)
   for(i=0; i<m; i++) works2[i]=a[i][jsta];
```

2D FDM(Column-Wise): C

```
MPI_Isend(works1, m, MPI_DOUBLE, inext, 1,
          MPI_COMM_WORLD, &isend1);
MPI_Isend(works2, m, MPI_DOUBLE, iprev, 1,
          MPI_COMM_WORLD, &isend2);
MPI_Irecv(workr1, m, MPI_DOUBLE, iprev, 1,
          MPI_COMM_WORLD, &irecv1);
MPI_Irecv(workr2, m, MPI_DOUBLE, inext, 1,
          MPI_COMM_WORLD, &irecv2);
MPI_Wait(&isend1, &istatus);
MPI_Wait(&isend2, &istatus);
MPI_Wait(&irecv1, &istatus);
MPI_Wait(&irecv2, &istatus);
if (myrank != 0)
   for(i=0; i<m; i++) a[i][jsta-1] = workr1[i];
if (myrank != nprocs-1)
   for(i=0; i<m; i++) a[i][jend+1] = workr2[i];
```

2D FDM(Column-Wise): C

```
  for (i=1; i<=m-2; i++)
     for(j=jsta2; j<=jend1; j++)
         b[i][j] = a[i-1][j] + a[i][j-1]
                         + a[i][j+1] + a[i+1][j];
  MPI_Finalize();
}
```

2D FDM(Column-Wise): Fortran

```
PROGRAM parallel_2D_FDM_column
INCLUDE 'mpif.h'
PARAMETER (m = 6, n = 9)
DIMENSION a(m,n), b(m,n)
INTEGER istatus(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 para_range(1, n, nprocs, myrank, jsta, jend)
jsta2 = jsta; jend1 = jend
IF (myrank == 0) jsta2 = 2
IF (myrank == nprocs - 1) jend1 = n - 1
inext = myrank + 1
iprev = myrank - 1
```

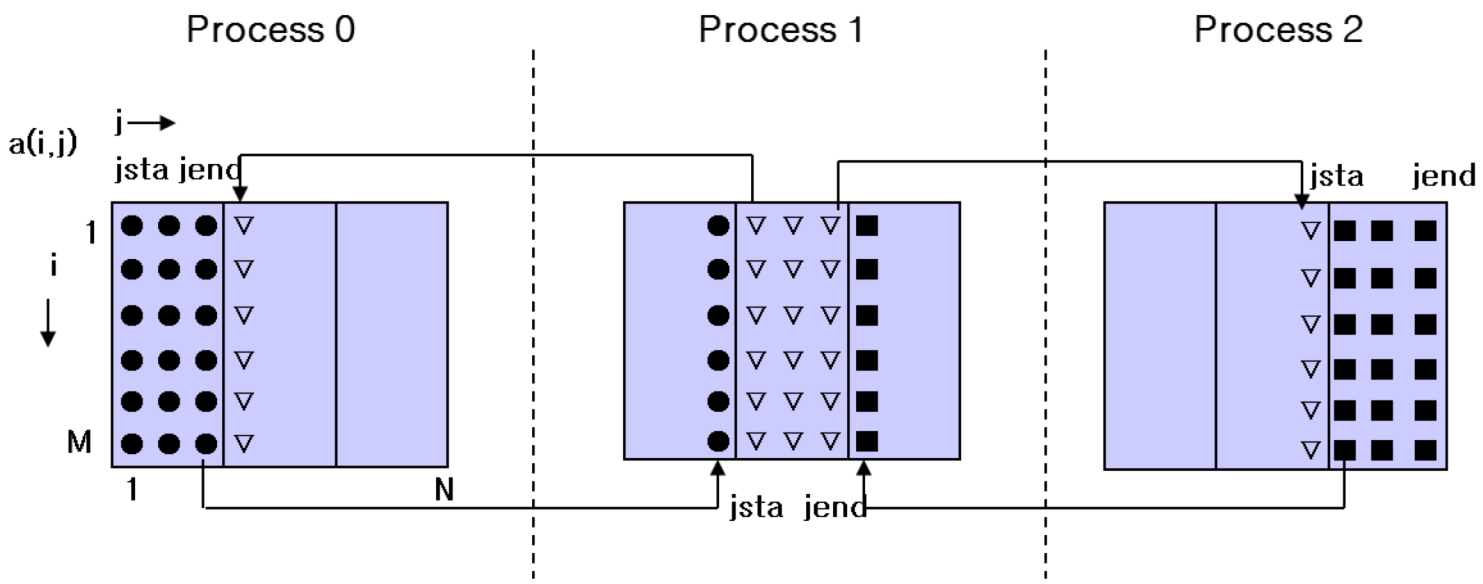
2D FDM(Column-Wise): Fortran

```
IF (myrank == nprocs - 1) inext = MPI_PROC_NULL
IF (myrank == 0) iprev = MPI_PROC_NULL
DO j = jsta, jend
   DO i = 1, m
      a(i,j) = i + 10.0 * j
   ENDDO
ENDDO
CALL MPI_ISEND(a(1,jend), m, MPI_REAL, inext, 1, &
        MPI_COMM_WORLD, isend1, ierr)
CALL MPI_ISEND(a(1,jsta), m, MPI_REAL, iprev, 1, &
        MPI_COMM_WORLD, isend2, ierr)
CALL MPI_IRECV(a(1,jsta-1), m, MPI_REAL, iprev, 1, &
        MPI_COMM_WORLD, irecv1, ierr)
CALL MPI_IRECV(a(1,jend+1), m, MPI_REAL, inext, 1, &
        MPI_COMM_WORLD, irecv2, ierr)
```

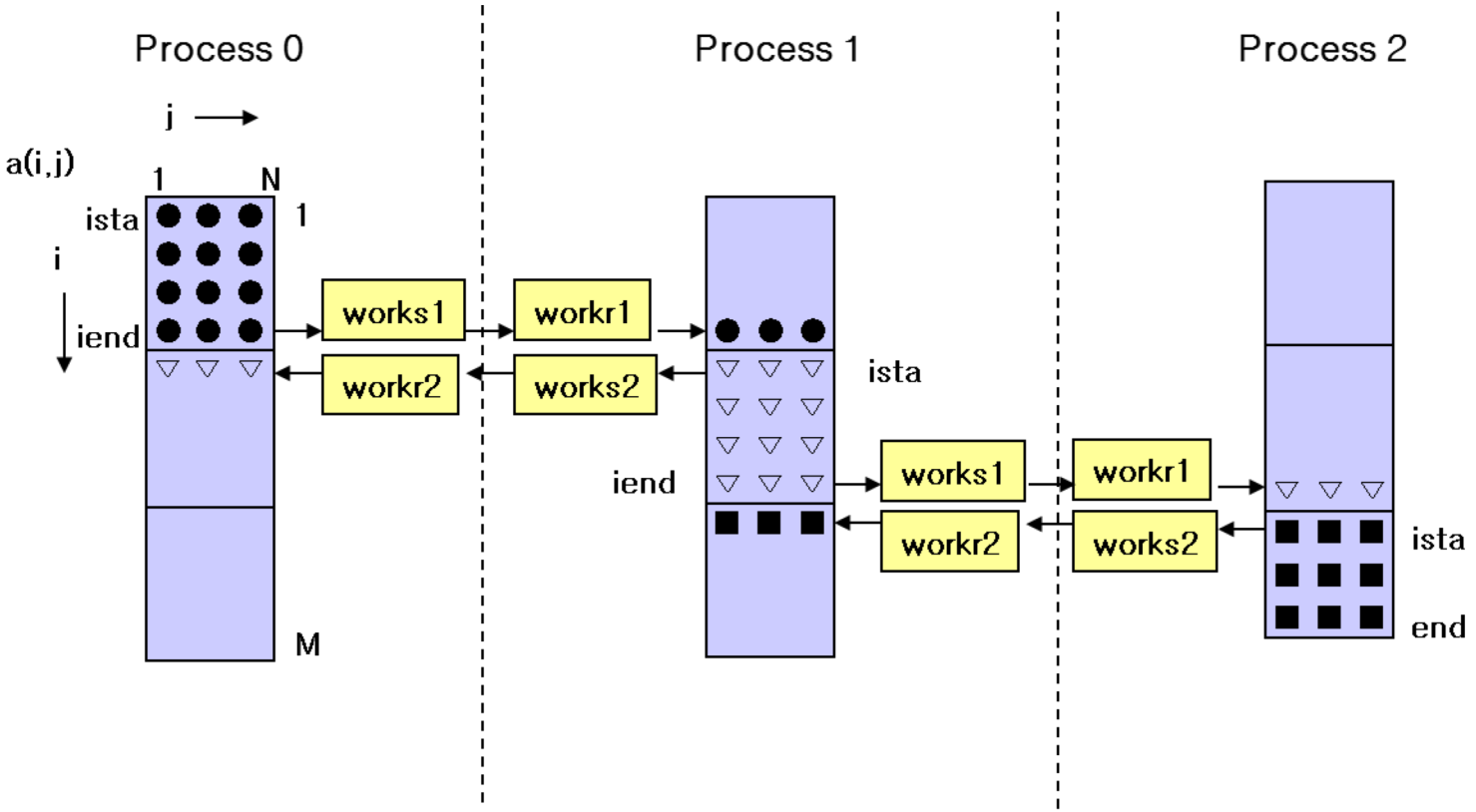
2D FDM(Column-Wise): Fortran

```
CALL MPI_WAIT(isend1, istatus, ierr)
CALL MPI_WAIT(isend2, istatus, ierr)
CALL MPI_WAIT(irecv1, istatus, ierr)
CALL MPI_WAIT(irecv2, istatus, ierr)
DO j = jsta2, jend1
   DO i = 2, m - 1
      b(i,j) = a(i-1,j) + a(i,j-1) + a(i,j+1) + a(i+1,j)
   ENDDO
ENDDO
CALL MPI_FINALIZE(ierr)
END
```

2D FDM: Column-Wise



2D FDM: Row-Wise



2D FDM(Row-Wise): C

```
/*parallel_2D_FDM_row*/
#include <mpi.h>
#define m 12
#define n 3
void para_range(int, int, int, int, int*, int*);
int min(int, int);
main(int argc, char *argv[]){
  int i, j, nprocs, myrank ;
  double a[m][n],b[m][n];
  int ista, iend, ista2, iend1, inext, iprev;
  MPI_Request isend1, isend2, irecv1, irecv2;
  MPI_Status istatus;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
```

2D FDM(Row-Wise): C

```
para_range(0, m-1, nprocs, myrank, &ista, &iend);
ista2 = ista; iend1 = iend;
if(myrank==0) ista2=1;
if(myrank==nprocs-1) iend1=m-2;
inext = myrank + 1;
iprev = myrank - 1;
if (myrank == nprocs-1) inext = MPI_PROC_NULL
if (myrank == 0) iprev = MPI_PROC_NULL
for(i=ista; i<=iend; i++)
   for(j=0; j<n; j++)  a[i][j] = i + 10.0 * j
MPI_Isend(&a[iend][0], n, MPI_DOUBLE, inext, 1,
          MPI_COMM_WORLD, &isend1);
MPI_Isend(&a[ista][0], n, MPI_DOUBLE, iprev, 1,
          MPI_COMM_WORLD, &isend2);
```

2D FDM(Row-Wise): C

```
    MPI_Irecv(&a[ista-1][0], n, MPI_DOUBLE, iprev, 1,
                  MPI_COMM_WORLD, &irecv1);
    MPI_Irecv(&a[iend+1][0], n, MPI_DOUBLE, inext, 1,
                  MPI_COMM_WORLD, &irecv2);
    MPI_Wait(&isend1, &istatus);
    MPI_Wait(&isend2, &istatus);
    MPI_Wait(&irecv1, &istatus);
    MPI_Wait(&irecv2, &istatus);
    for (i=ista2; i<=iend1; i++)
       for(j=1; j<=n-2; j++)
          b[i][j] = a[i-1][j] + a[i][j-1] +
                          a[i][j+1] + a[i+1][j];
    MPI_Finalize();
}
```

2D FDM(Row-Wise): Fortran

```
PROGRAM parallel_2D_FDM_row
INCLUDE 'mpif.h'
PARAMETER (m = 12, n = 3)
DIMENSION a(m,n), b(m,n)
DIMENSION   works1(n),   workr1(n),   works2(n),
   workr2(n)
INTEGER istatus(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  para_range(1,  m,  nprocs,  myrank,  ista,
   iend)
ista2 = ista; iend1 = iend
IF (myrank == 0) ista2 = 2
IF (myrank == nprocs - 1) iend1 = m-1
inext = myrank + 1; iprev = myrank - 1
```

2D FDM(Row-Wise): Fortran

```
IF (myrank == nprocs - 1) inext = MPI_PROC_NULL
IF (myrank == 0) iprev = MPI_PROC_NULL

DO j = 1, n
   DO i = ista, iend
      a(i,j) = i + 10.0 * j
   ENDDO
ENDDO

IF (myrank /= nprocs - 1) THEN
   DO j = 1, n
      works1(j) = a(iend,j)
   ENDDO
ENDIF
IF (myrank /= 0) THEN
   DO j = 1, n
      works2(j) = a(ista,j)
   ENDDO
ENDIF
```

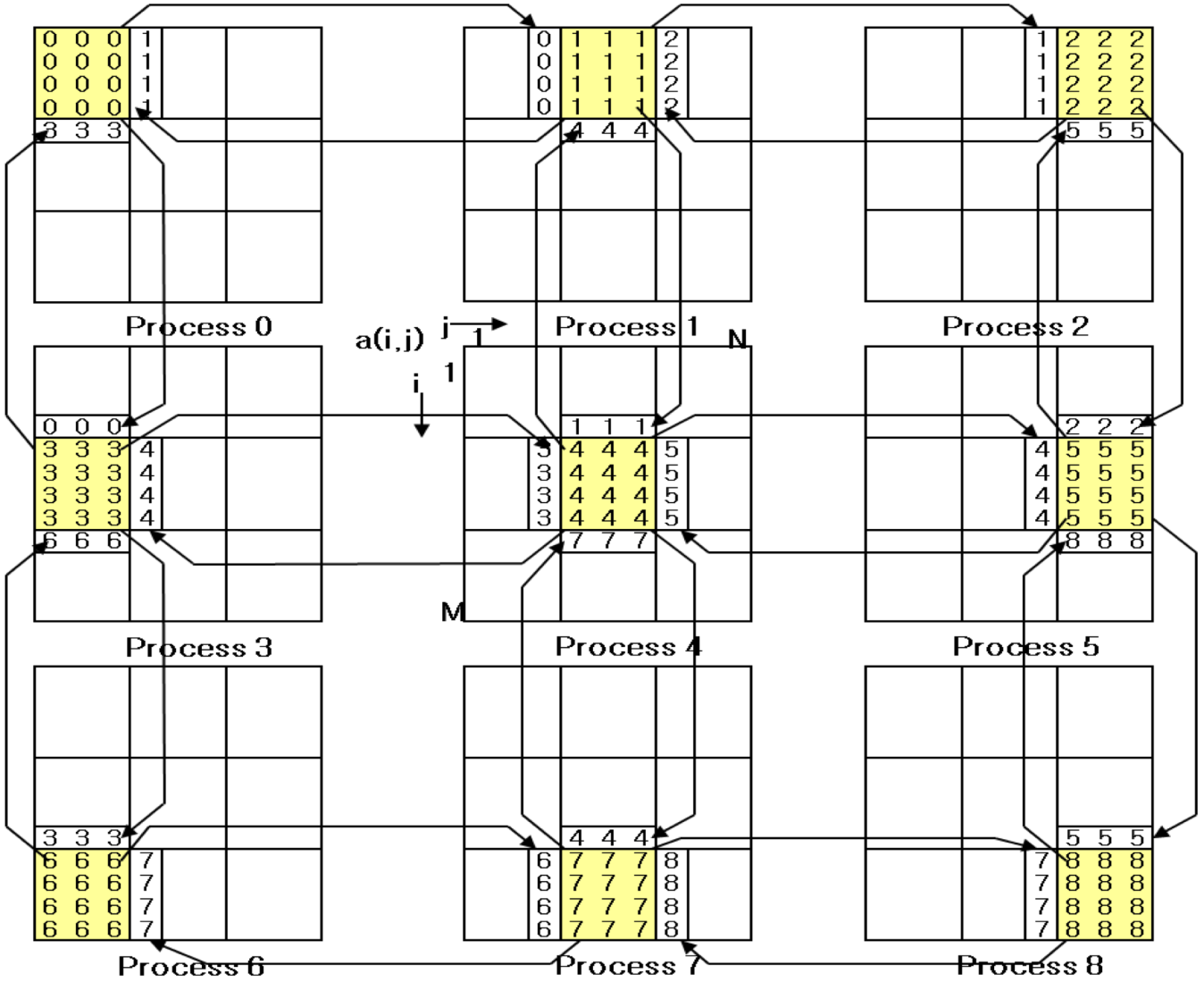
2D FDM(Row-Wise): Fortran

```
CALL MPI_ISEND(works1,n,MPI_REAL,inext,1, &
      MPI_COMM_WORLD, isend1,ierr)
CALL MPI_ISEND(works2,n,MPI_REAL,iprev,1, &
      MPI_COMM_WORLD, isend2,ierr)
CALL MPI_IRECV(workr1,n,MPI_REAL,iprev,1, &
      MPI_COMM_WORLD, irecv1,ierr)
CALL MPI_IRECV(workr2,n,MPI_REAL,inext,1, &
      MPI_COMM_WORLD, irecv2,ierr)
CALL MPI_WAIT(isend1, istatus, ierr)
CALL MPI_WAIT(isend2, istatus, ierr)
CALL MPI_WAIT(irecv1, istatus, ierr)
CALL MPI_WAIT(irecv2, istatus, ierr)
```

2D FDM(Row-Wise): Fortran

```
IF (myrank /= 0) THEN
   DO j = 1, n
      a(ista-1,j) = workr1(j)
   ENDDO
ENDIF

IF (myrank /= nprocs - 1) THEN
   DO j = 1, n
      a(iend+1,j) = workr2(j)
   ENDDO
ENDIF

DO j = 2, n - 1
   DO i = ista2, iend1
      b(i,j) = a(i-1,j) + a(i,j-1) + a(i,j+1) + a(i+1,j)
   ENDDO
ENDDO
CALL MPI_FINALIZE(ierr)
END
```

2D FDM: Block distribution in both dimensions



Advanced Exercise

Advanced Lab #1

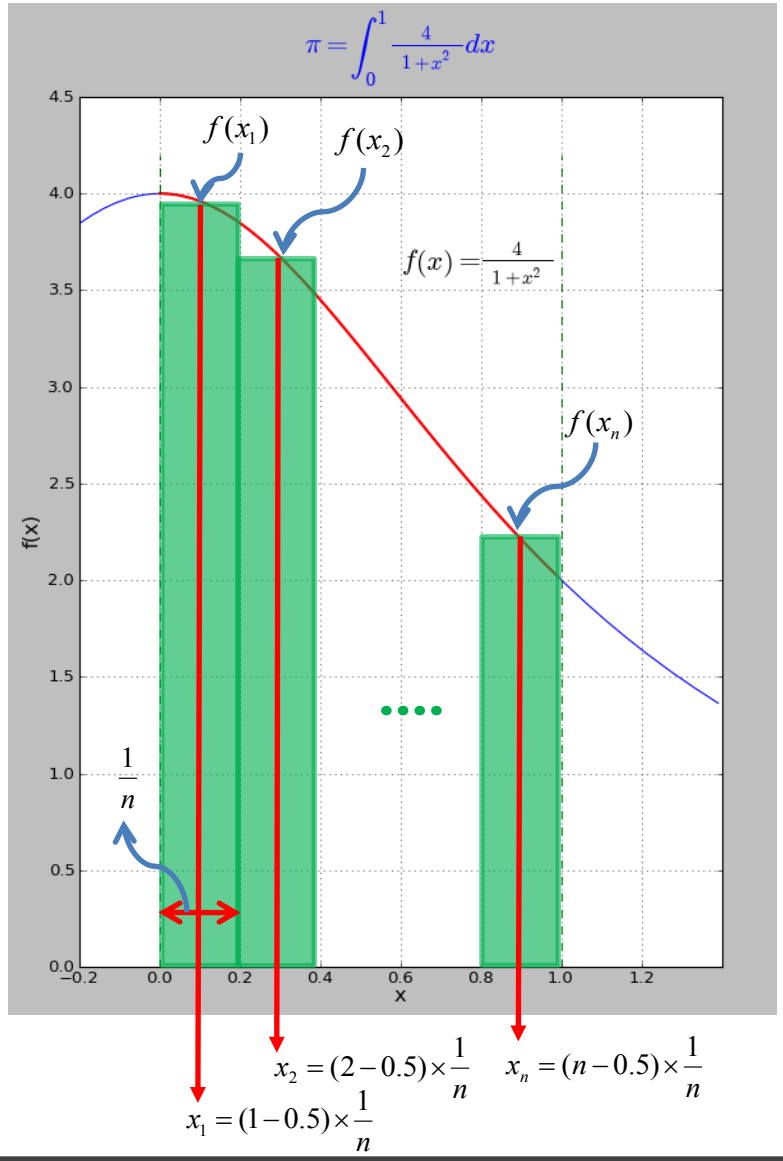
Advanced Lab #2 PI (Numerical Integration)

- 〈Problem〉
  - Get PI using Numerical integration

$$\int_0^1 \frac{4.0}{(1+x^2)} \, dx = \pi$$

- 〈Requirement〉
  - Point to point communication

$$\pi \approx \sum_{i=1}^{n} \frac{4}{1+((i-0.5)\times\frac{1}{n})^2} \times \frac{1}{n}$$



Advanced Lab #2 PI Numerical Integration :serial - C

```c
#include <stdio.h>
#include <math.h>
#define num_steps 1000000000

void main(int argc, char *argv[]) {
  double sum, step, x, pi;
  double t1, t2;
  int i;

  sum=0.0;
  step=1./(double)num_steps;

  for(i=1; i<num_steps; i++){
     x = (i-0.5)*step;
     sum = sum + 4.0/(1.0+x*x);
  }

  pi = step*sum;
  printf(" numerical  pi = %.15f \n", pi);
  printf(" analytical pi = %.15f \n", acos(-1.0));
  printf(" Error = %E \n", fabs(acos(-1.0)-pi));
}
```

Advanced Lab #2 PI Numerical Integration - C(1/2)

```
#include <stdio.h>
#include <math.h>
#include <mpi.h>
#define num_steps 1000000000
void main(int argc, char *argv[]) {
  double sum, step, x, pi;
  double tsum;
  int i, nprocs, myrank;
  int ista, iend;
  MPI_Status status;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

  sum=0.0;
  step=1./(double)num_steps;

  para_range(1, num_steps, nprocs, myrank, &ista, &iend);
  printf(" ista=%d, iend = %d \n", ista,iend);
```

Advanced Lab #2 PI Numerical Integration - C(2/2)

```
  for(i=ista; i<=iend; i++){
     x = (i-0.5)*step;
     sum = sum + 4.0/(1.0+x*x);
  }

  MPI_Reduce(&sum, &tsum, 1, MPI_DOUBLE,
   MPI_SUM,0,MPI_COMM_WORLD);

  if(myrank ==0){
    pi = step*tsum;
    printf(" numerical  pi = %.15lf \n", pi);
    printf(" analytical pi = %.15f \n", acos(-1.0));
    printf(" Error = %E \n", fabs(acos(-1.0)-pi));
  }
  MPI_Finalize();
}
```

Advanced Lab #2 PI Compile & Run

```
$ mpicc -g -o pi_integral pi_integral.c

$ time mpirun -np 4 -hostfile hosts pi_integral
```

```
integer, parameter:: num_steps=1000000000
real(8) sum, step, x, pi;

sum=0.0
step=1./dble(num_steps)

do i=1, num_steps
   x = (i-0.5)*step
   sum = sum + 4.0/(1.0+x*x)
enddo

pi = step*sum
print*, "numerical  pi = ", pi
print*, "analytical pi = ", dacos(-1.d0)
print*, " Error = ", dabs(dacos(-1.d0)-pi)

end
```

Advanced Lab #2 PI Numerical Integration - Fortran(1/2)

```
PROGRAM pi_integral
include "mpif.h"

  integer, parameter:: num_steps=1000000000
  real(8) sum, step, x, pi, tsum;
  integer :: i, nprocs, myrank, ierr

  call MPI_INIT(ierr);
  call MPI_Comm_size(MPI_COMM_WORLD,nprocs,ierr)
  call MPI_Comm_rank(MPI_COMM_WORLD,myrank,ierr)

  sum=0.0
  step=1./dble(num_steps)

  call para_range(1, num_steps, nprocs, myrank, ista, iend)
  print*, "myrank =", myrank, ":", ista," ~ ", iend
```

```
do i=ista, iend
   x = (i-0.5)*step
   sum = sum + 4.0/(1.0+x*x)
enddo

call MPI_REDUCE(sum, tsum, 1, MPI_REAL8, MPI_SUM, 0, &
                MPI_COMM_WORLD, ierr)

if(myrank ==0) then
  pi = step*tsum
  print*, "numerical  pi = ", pi
  print*, "analytical pi = ", dacos(-1.d0)
  print*, " Error = ", dabs(dacos(-1.d0)-pi)
endif

call MPI_FINALIZE(ierr)

end
```

Advanced Lab #2 PI Compile & Run

```
$ mpif90 -g -o pi_integral.x pi_integral.f90

$ time mpirun -np 4 -hostfile hosts pi_integral.x
```

Advanced Lab #2

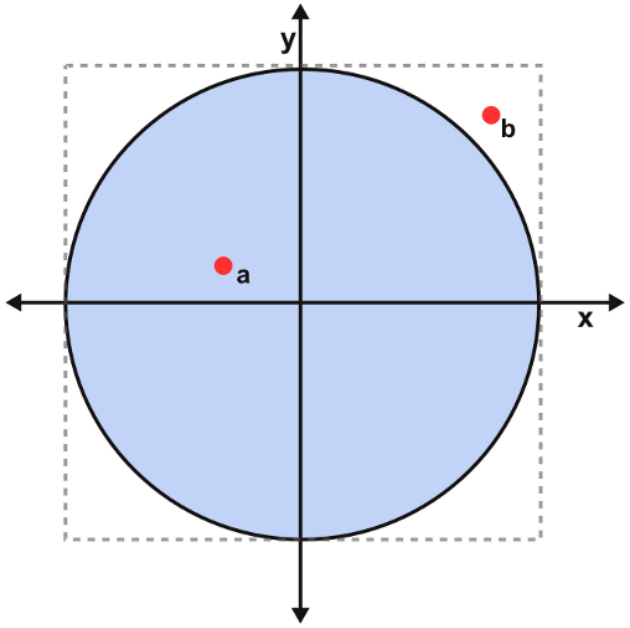
Advanced Lab #1 PI (Monte Carlo Simulation)

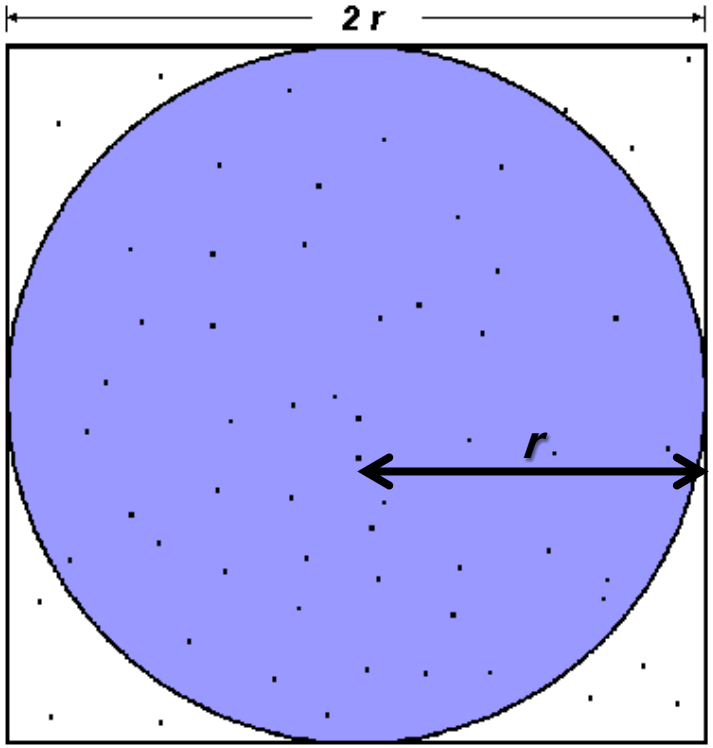
➢ 〈Problem〉

- Monte carlo simulation
- Random number use
- ***PI = 4 x Ac/As***

➢ 〈Requirement〉

- N's processor(rank) use
- P2p communication





$$A_S = (2r)^2 = 4r^2$$

$$A_C = \pi r^2$$

$$\pi = 4 \times \frac{A_C}{A_S}$$

Advanced Lab #1 PI MC Simulation - C (1/3)

```
/*
     Example Name    : pi_monte.c
     Compile         : $ mpicc -g -o pi_monte -Wall pi_monte.c
     Run             : $ mpirun -np 4 -hostfile hosts pi_monte
*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <mpi.h>

#define   SCOPE     100000000

/*
** PI (3.14) Monte Carlo Method
*/
int main(int argc, char *argv[])
{
     int nProcs, nRank, proc, ROOT = 0;
     MPI_Status status;
     int nTag = 55;

     int i, nCount = 0, nMyCount = 0;
     double x, y, z, pi, z1;
```

Advanced Lab #1 PI MC Simulation - C (2/3)

```
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &nProcs);
MPI_Comm_rank(MPI_COMM_WORLD, &nRank);

srand(time(NULL)*(nRank+1));

for (i = 0; i < SCOPE; i++) {
      x = (double)rand() / (double)(RAND_MAX);
      y = (double)rand() / (double)(RAND_MAX);

      z = x*x + y*y;
      z1 = sqrt(z);

      if (z1 <= 1)         nMyCount++;
}

if (nRank == ROOT) {
      nCount = nMyCount;

      for (proc=1; proc<nProcs; proc++) {
                MPI_Recv(&nMyCount, 1, MPI_REAL, proc, nTag, MPI_COMM_WORLD,
                         &status);
                nCount += nMyCount;
      }

      pi = (double)4*nCount/(SCOPE * nProcs);
```

Advanced Lab #1 PI MC Simulation - C (3/3)

```
            printf("Processor %d sending results = %d to ROOT processor\n", nRank,
                  nMyCount);
            printf("\n # of trials (cpu#: %d, time#: %d) = %d, estimate of pi is %f\n",
                              nProcs, SCOPE, SCOPE*nProcs, pi);

      }
      else {
            printf("Processor %d sending results = %d to ROOT processor\n", nRank,
                  nMyCount);
            MPI_Send(&nMyCount, 1, MPI_REAL, ROOT, nTag, MPI_COMM_WORLD);
      }

      printf("\n");

      MPI_Finalize();

      return 0;
}
```

Advanced Lab #1 PI MC Simulation - Compile & Run

```
$ mpicc -g -o pi_monte -Wall pi_monte.c
$ mpirun -np 4 -hostfile hosts pi_monte
Processor 2 sending results = 78541693 to ROOT processor
Processor 1 sending results = 78532183 to ROOT processor
Processor 3 sending results = 78540877 to ROOT processor
Processor 0 sending results = 78540877 to ROOT processor

 # of trials (cpu#: 4, time#: 100000000) = 400000000, estimate of pi is 3.141516
```

Advanced Lab #1 PI MC Simulation - Fortran (1/3)

```
! Example Name  : pi_monte.f90
! Compile      : $ mpif90 -g -o pi_monte.x -Wall pi_monte.f90
! Run          : $ mpirun -np 4 -hostfile hosts pi_monte.x

PROGRAM pi_monte
IMPLICIT NONE
INCLUDE 'mpif.h'
    INTEGER nRank, nProcs, iproc, iErr
    INTEGER :: ROOT = 0, scope = 100000000
    INTEGER :: i, nMyCount = 0, nCount = 0
    REAL :: x, y, z, pi, z1

    INTEGER status(MPI_STATUS_SIZE)

    CALL MPI_INIT(iErr)
    CALL MPI_COMM_RANK(MPI_COMM_WORLD, nRank, iErr)
    CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nProcs, iErr)

    ! Get Random #
    CALL INIT_RANDOM_SEED(nRank+1)

    DO i=0, scope-1
        CALL RANDOM_NUMBER(x)
        CALL RANDOM_NUMBER(y)
```

Advanced Lab #1 PI MC Simulation - Fortran (2/3)

```
        z = x*x + y*y
        z1 = SQRT(z)

        IF (z1 <= 1) nMyCount = nMyCount + 1
    END DO

    IF (nRank == ROOT) THEN
        nCount = nMyCount

        DO iproc=1, nProcs-1
            CALL MPI_RECV(nMyCount, 1, MPI_INTEGER, iproc, 55, MPI_COMM_WORLD,
                          status, iErr)
            nCount = nCount + nMyCount
        END DO

        pi = 4 * REAL(nCount) / (scope * nProcs)

        WRITE (*, '(A, I2, A, I10, A)') 'Processor=', nRank, ' sending results = ',
               nMyCount, ' to ROOT process'
        WRITE (*, '(A, I2, A, F15.10)') '# of trial is ', nProcs, ' estimate of PI is ',
              pi

    ELSE
        CALL MPI_SEND(nMyCount, 1, MPI_INTEGER, ROOT, 55, MPI_COMM_WORLD, iErr)
        WRITE (*, '(A, I2, A, I10)') 'Processor=', nRank, ' sending results = ',
               nMyCount
    END IF
```

Advanced Lab #1 PI MC Simulation - Fortran (3/3)

```
    CALL MPI_FINALIZE(iErr)

CONTAINS
SUBROUTINE INIT_RANDOM_SEED(rank)
IMPLICIT NONE
INTEGER rank
    INTEGER :: i, n, clock
    INTEGER, DIMENSION(:), ALLOCATABLE :: seed

    CALL RANDOM_SEED(size = n)
    ALLOCATE(seed(n))

    CALL SYSTEM_CLOCK(COUNT=clock)

    seed = clock + 37 * (/ (i - 1, i = 1, n) /)
    seed = seed * rank * rank

    CALL RANDOM_SEED(PUT = seed)

    DEALLOCATE(seed)
END SUBROUTINE

END
```

Advanced Lab #1 PI MC Simulation - Compile & Run

```
$ mpif90 -g -o pi_monte.x -Wall pi_monte.f90
$ mpirun -np 4 -hostfile hosts pi_monte.x
Processor= 3    sending results =   78540734
Processor= 2    sending results =   78537818
Processor= 0 sending results =   78540734       to ROOT process
Processor= 1    sending results =   78542358

# of trial is  4 estimate of PI is    3.1415631771
```

Q&A


Thank
you!