Lecture 18 – MPI Part 2

Prof. Brendan Kochunas

NERS/ENGR 570 - Methods and Practice of Scientific Computing (F22)

COLLEGE OF ENGINEERING
NUCLEAR ENGINEERING & RADIOLOGICAL SCIENCES
UNIVERSITY OF MICHIGAN

Outline

- Revisit Lab 09
- Ping-Pong Example
- Collective communication
    - OMB Benchmarks
- Lab 10 analysis

Learning Objectives: By the end of Today's Lecture you should be able to

- (*Knowledge*) explain its good to use MPI Collective operations
- (*Skill*) Debug, Compile, and Run an MPI program

Ping Pong Example

P-I-N-G P-O-N-G

What does the ping pong program do?

- Original 6 Links
  - https://www.mpich.org/static/docs/v3.3/www3/MPI_Init.html
  - https://www.mpich.org/static/docs/v3.3/www3/MPI_Comm_size.html
  - https://www.mpich.org/static/docs/v3.3/www3/MPI_Comm_rank.html
  - https://www.mpich.org/static/docs/v3.3/www3/MPI_Send.html
  - https://www.mpich.org/static/docs/v3.3/www3/MPI_Recv.html
  - https://www.mpich.org/static/docs/v3.3/www3/MPI_Finalize.html

The most common bug for new MPI programmers

```
IF(MOD(myRank,2) == 0) THEN
  CALL MPI_Send(sbuffer, n, MPI_DOUBLE_PRECISION, &
    myRank+1, 0, MPI_COMM_WORLD, mpierr)
  CALL MPI_Recv(rbuffer, n, MPI_DOUBLE_PRECISION, &
    myRank+1, 0, MPI_COMM_WORLD, MPI_STATUS_NULL, mpierr)
ELSE
  CALL MPI_Send(sbuffer, n, MPI_DOUBLE_PRECISION, &
    myRank-1, 0, MPI_COMM_WORLD, mpierr)
  CALL MPI_Recv(rbuffer, n, MPI_DOUBLE_PRECISION, &
    myRank-1, 0, MPI_COMM_WORLD, MPI_STATUS_NULL, mpierr)
ENDIF
```

Collective Communication

MPI Collectives (1)

- These involve all MPI processes in a *communicator*
- Collectives can always be implemented with point-to-point routines
  - But it is often better to use the routines provided by MPI
- Common collective operations include:
  - Broadcast
  - Reduce
  - Scatter
  - Gather
  - Scan
  - Alltoall

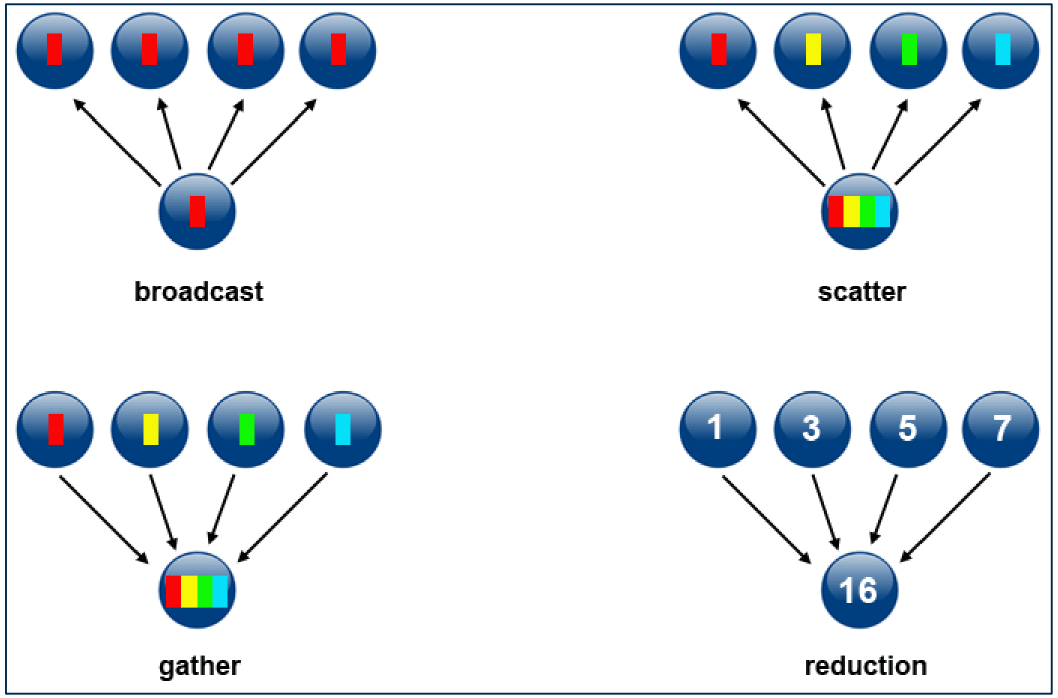


Figure from: https://computing.llnl.gov/tutorials/parallel_comp/

MPI Collectives (2)

Notable Variations

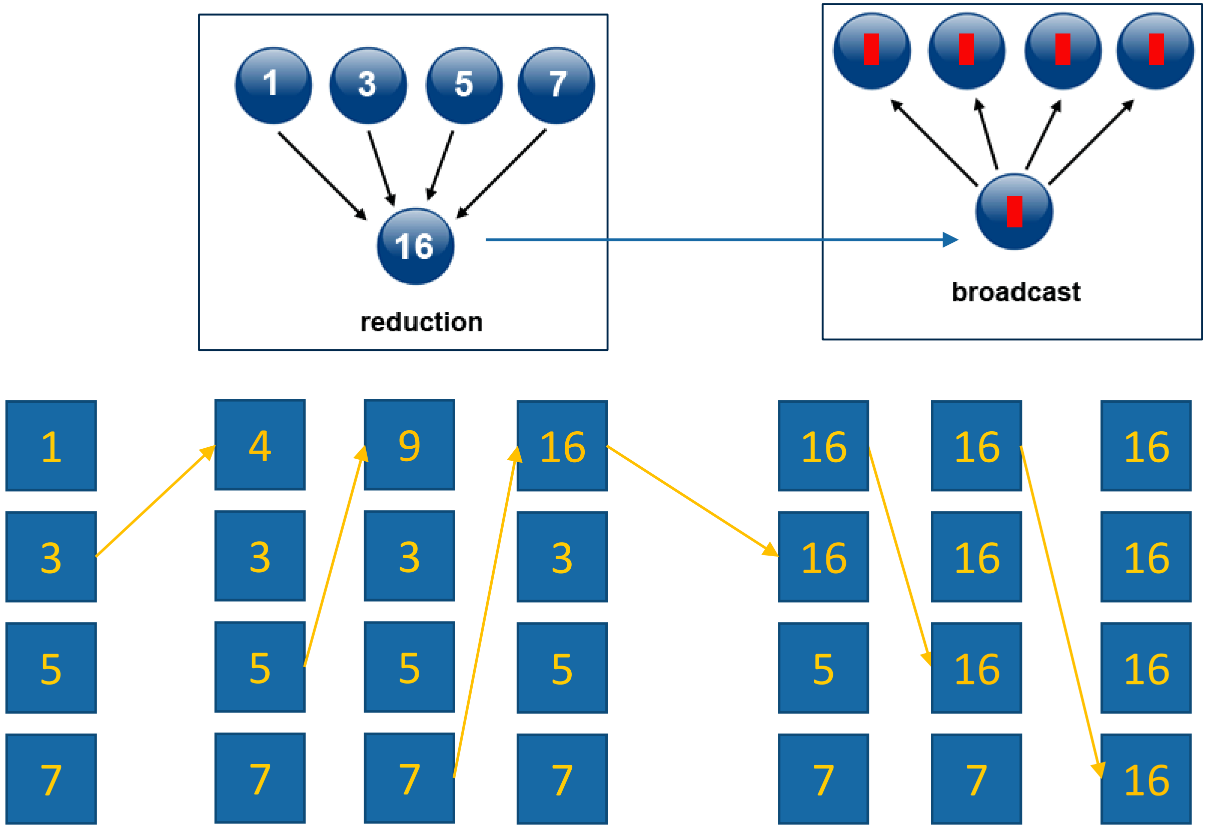
- The "`v`" suffix
  - Stands for vector
  - Means the *size of data may be different* for different processors
  - Gatherv & Scatterv, Alltoallv
- The "`All`" prefix
  - Means the *result of the operation is the same for all processors* in communicator
  - Allreduce & Allgather

Types of reduction operations

- Arithmetic
  - MPI_SUM
  - MPI_PROD
- Relation Operators (Mins & Maxes)
  - MPI_MAX
  - MPI_MIN
  - MPI_MAXLOC
  - MPI_MINLOC
- Logical Operators
  - MPI_LAND
  - MPI_LOR
  - MPI_LXOR
- Bit-wise operators also supported

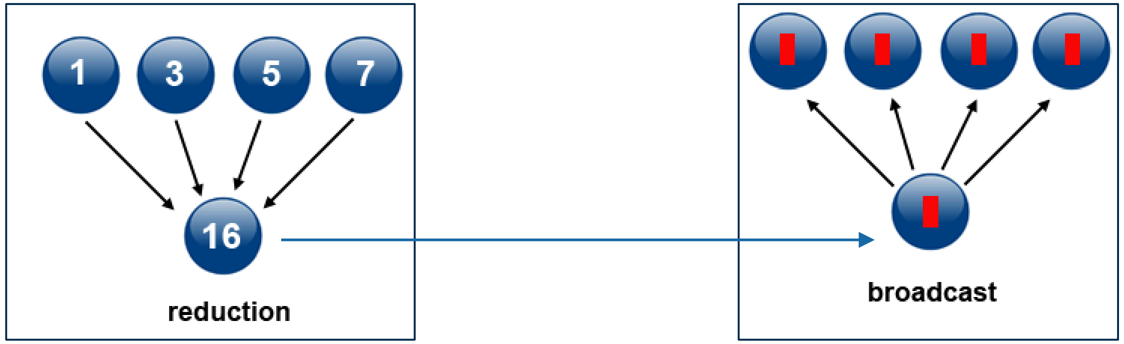
Example: MPI_Allreduce Algorithm

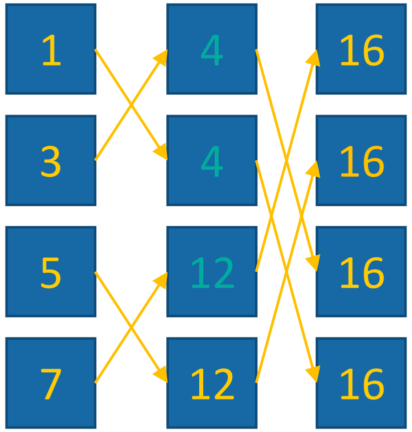
- Reduce + broadcast
- Reduce performed sequentially
  - P-1 steps
- Broadcast performed sequentially
  - Also P-1 steps
- Total of 6 steps



Example: Better Allreduce

- Use a binomial tree
  - Completed in $\lceil \log p \rceil$ steps
- Scales much better to higher number of processors





Even More Advanced Allreduce

- What about long messages?
  - Reduce_scatter + Allgather
- Different algorithms perform better under certain conditions

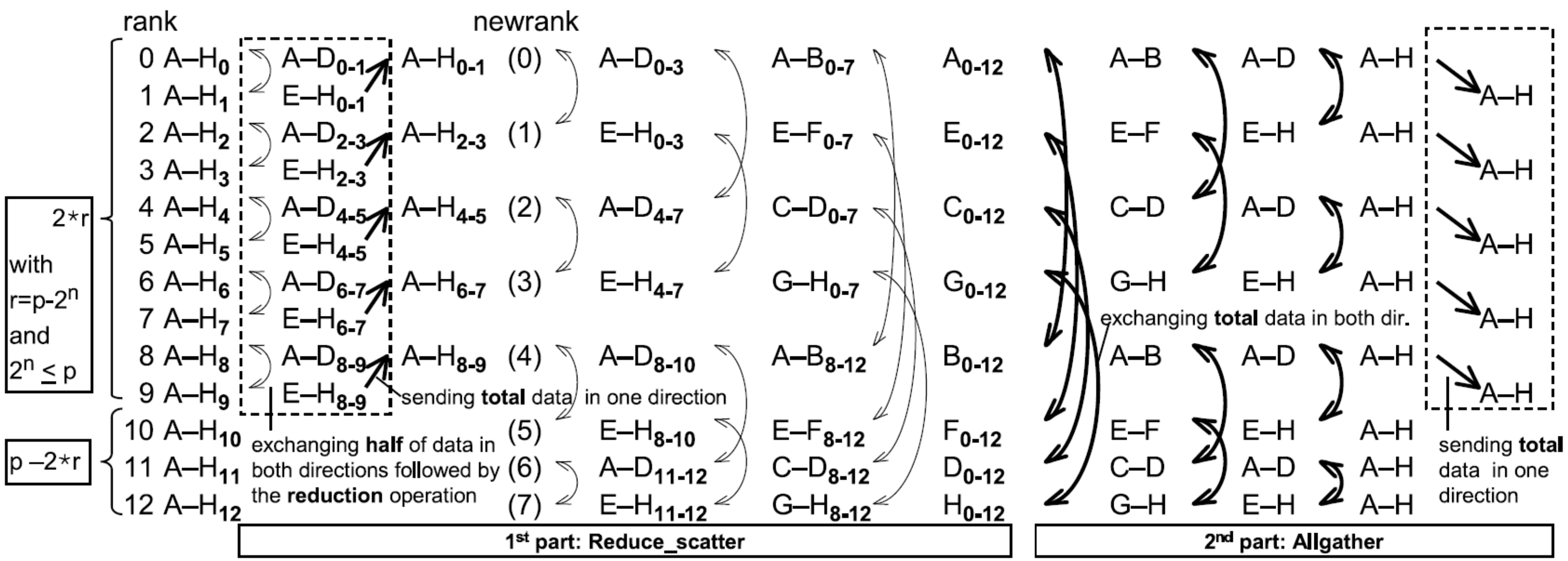


Figure 12: Allreduce using the recursive halving and doubling algorithm. The intermediate results after each communication step, including the reduction operation in the reduce-scatter phase, are shown. The dotted frames show the additional overhead caused by a non-power-of-two number of processes.

Source: http://www.mcs.anl.gov/~thakur/papers/ijhpca-coll.pdf

Summary of Collectives

- Provided as a convenience to the programmer
  - Collectives perform "common" operations that arise in programming
  - Often implemented with more complex and higher performing algorithms
    - Than what a beginner would implement.
- They represent a synchronization point in the program
- Always, always, always involves all processors *within communicator*
  - Otherwise, it causes a deadlock

The most common bug for new MPI programmers

```
IF(MOD(myRank,2) == 0) &
  CALL MPI_Reduce(sbuf,rbuf,n,MPI_DOUBLE_PRECISION, MPI_SUM, &
    0, MPI_COMM_WORLD, mpierr)
```

Lab 10

Extra Material

…for Extra Learning

OMB Benchmarks

Collective Communication Benchmarks

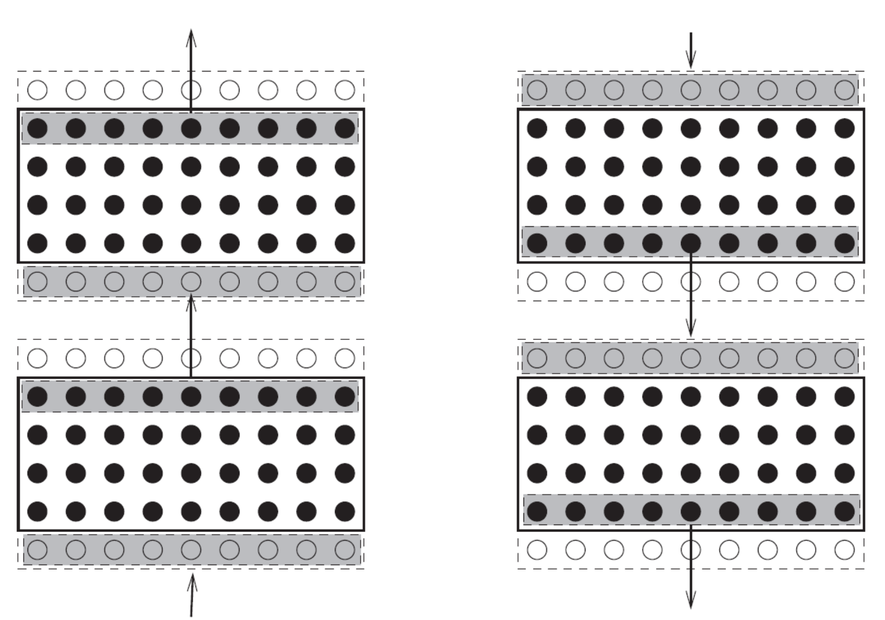
- Reduce
- Broadcast
- Allreduce

Non-blocking Collectives

Non-Blocking Communication

- Last lecture, we mentioned that non-blocking communication is more efficient.
- It also requires some extra steps (MPI calls)

```
  call MPI_IRECV(&
          a(1,s-1), nx, MPI_DOUBLE_PRECISION, nbrbottom, 0, &
          comm1d, req(1), ierr)
  call MPI_IRECV(&
          a(1,e+1), nx, MPI_DOUBLE_PRECISION, nbrtop, 1, &
          comm1d, req(2), ierr)
  call MPI_ISEND(&
          a(1,e), nx, MPI_DOUBLE_PRECISION, nbrtop, 0, &
          comm1d, req(3), ierr)
  call MPI_ISEND(&
          a(1,s), nx, MPI_DOUBLE_PRECISION, nbrbottom, 1, &
          comm1d, req(4), ierr)
!
  call MPI_WAITALL(4, req, MPI_STATUSES_IGNORE, ierr)
```



Non-blocking collectives

- Recently in the MPI-3 standard, non-blocking collective operations were defined.
- Interfaces follow same convention as point-to-point communication
  - Prefix operation with “I”
- Supported operations
  - Barrier
  - Broadcast
  - Gather
  - Scatter
  - Gather-to-all
  - All-to-all
  - Reduce
  - All-Reduce
  - Reduce-Scatter
  - Scan

```
MPI_Comm comm;
int array[100], array2[100];
Int root=0;
MPI_Request req;
…
MPI_Ibcast(array1, 100, MPI_INT, root, comm, &req);
Compute(array2, 100);
MPI_Wait(&req, MPI_STATUS_IGNORE);
```

Example: Start a broadcast of 100 ints from process 0 to every process in the group, perform some computation on independent data, and then complete the outstanding broadcast operation.

MPI I/O

MPI I/O

- Probably will not ever need to use this, just an FYI
  - HDF5 utilizes this
- Some simulations create really large data sets
  - Not an efficient use of resources to have one process write and 1000 just wait…
- Care must be taken for how this is done
  - Helps if natively supported by hardware (e.g. Lustre)

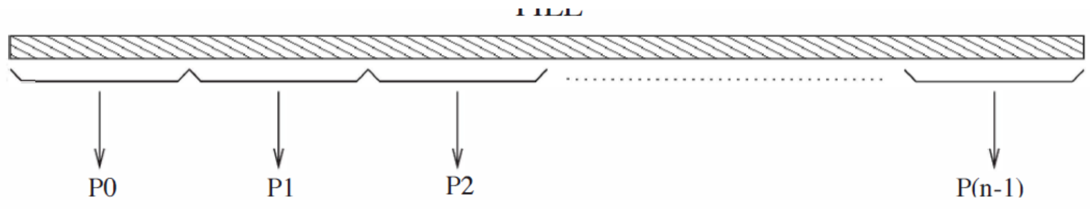


Figure 7.1: Example with n processes, each needing to read a chunk of data from a common file

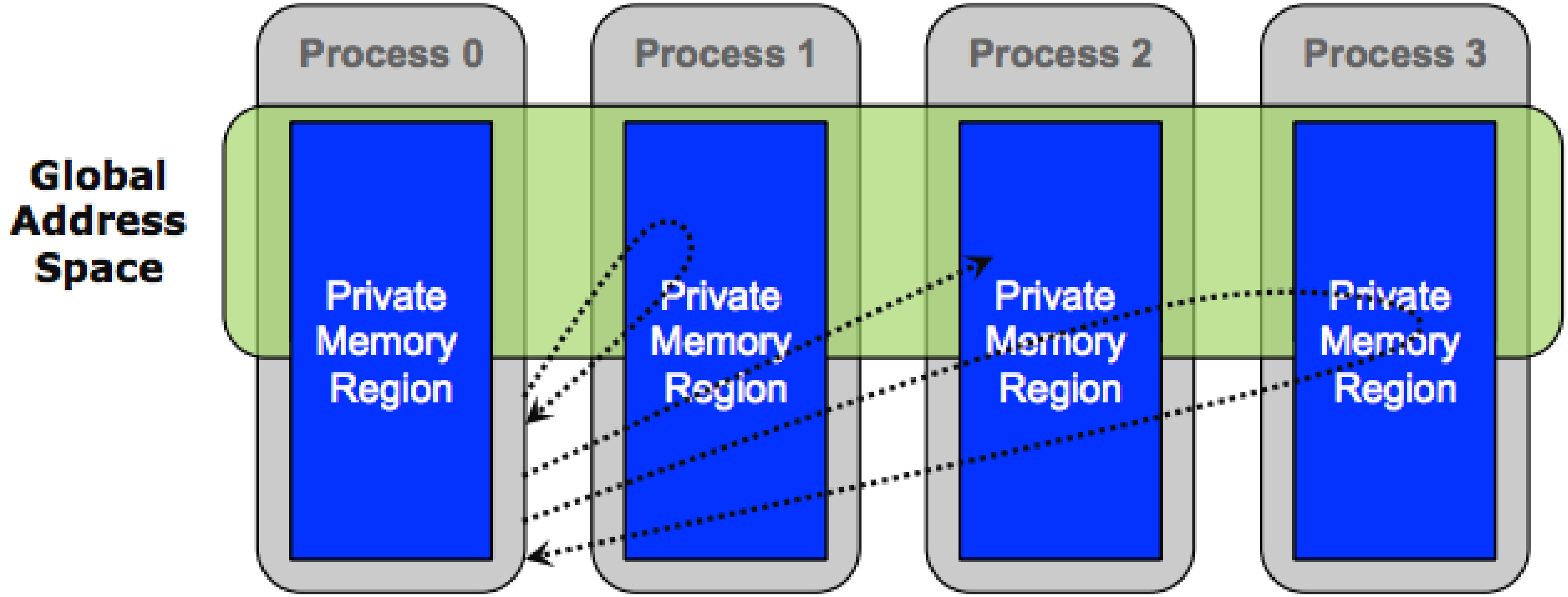
MPI_File_open	Opens a file
MPI_File_seek	goto a different pos
MPI_File_read	Read some data
MPI_File_write	Write some data
MPI_File_close	Closes a file

One-sided Communication

One-sided communication (Remote Memory Access)

- The basic idea of one-sided communication models is to decouple data movement with process synchronization
  - Should be able to move data without requiring that the remote process synchronize
  - Each process exposes a part of its memory to other processes
  - Other processes can directly read from or write to this memory
- Advantages
  - Multiple transfers with a single synchronization
  - Irregular communication patterns can be more economically expressed
  - Can be significantly faster than send/recv on systems with hardware support for RMA
- Example Monte Carlo “Tally Server”

Illustration of MPI One-sided communication



MPI Remote Memory Access (RMA)

- General steps to using:
  - Create a window
  - Put some data
  - Get some data
  - Accumulate some data
- Key concept is a "window object"
  - Exposes larger part of process's address space for access by other processes

```
MPI_Win_create
MPI_Put
MPI_Get
MPI_Accumulate
```

All are non-blocking; multiple operations can be active in same window object simultaneously

Hybrid Parallelism

MPI + X

- Distributed message passing parallel model with some other parallel programming model
  - X is usually "shared memory"
- Some examples
  - OpenMP
  - MPI Threads
  - CUDA
  - pthreads
  - Possibly some others

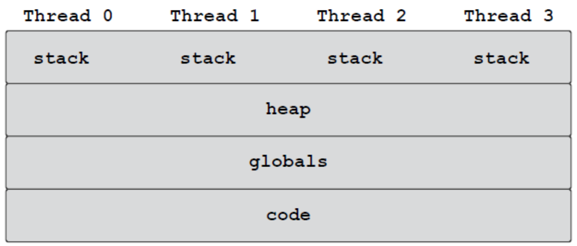


Figure 5.1: Full memory sharing in threaded environments

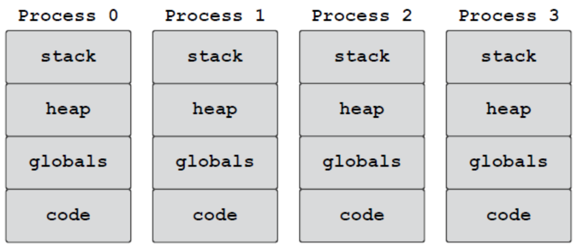


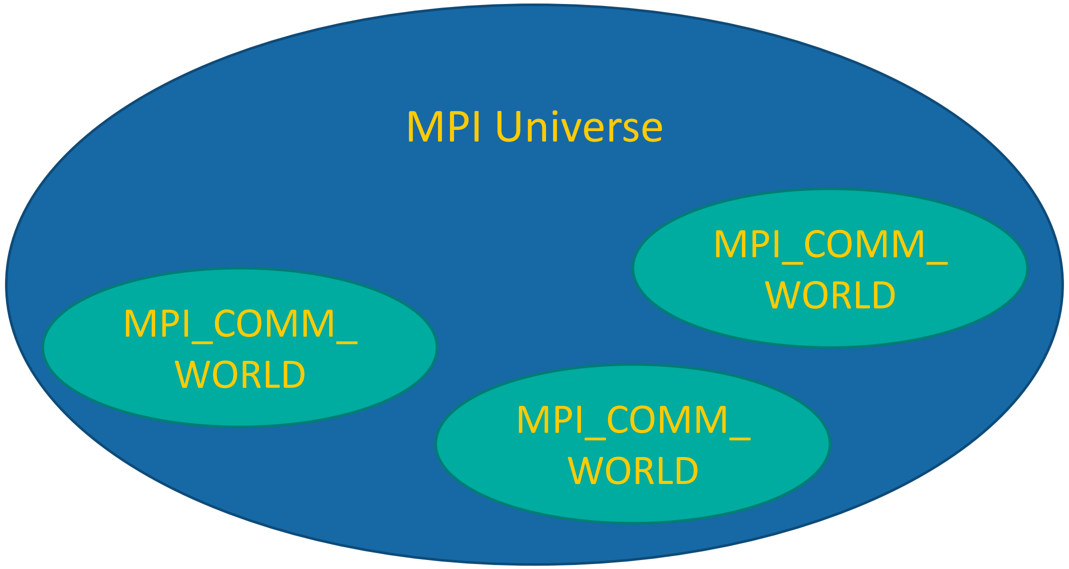
Figure 5.2: Standard MPI semantics—no sharing

MPI_Init_thread	Create a thread
MPI_Query_thread	Check threading support
MPI_Is_thread_main	Check for main thread
MPI_Finalize_thread	Destroy a thread

Dynamic Process Management

MPI Dynamic Process Management

- Create NEW MPI processes from our MPI processes



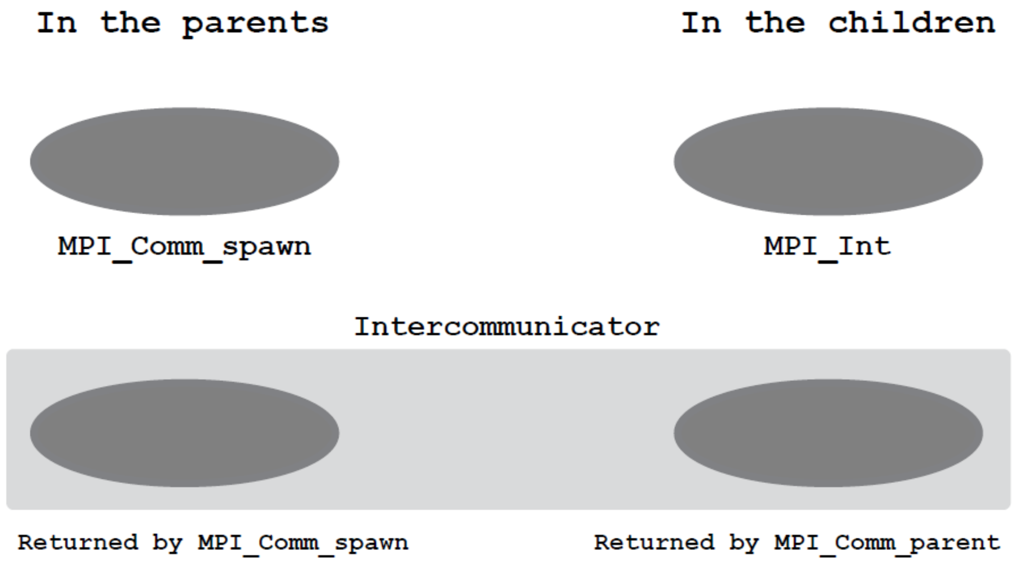


Figure 10.2: Spawning processes. Ovals are intracommunicators containing several processes.

Virtual Topologies

Virtual Topologies (1)

- A "virtual topology" is the topology that arises from the communication patterns of the application
  - e.g. the application topology
  - Not the physical or network topology of how the computers are connected.
- For more details see Bill Gropp's lecture http://wgropp.cs.illinois.edu/courses/cs598-s16/lectures/lecture28.pdf

Figure 4.2: Jacobi iteration

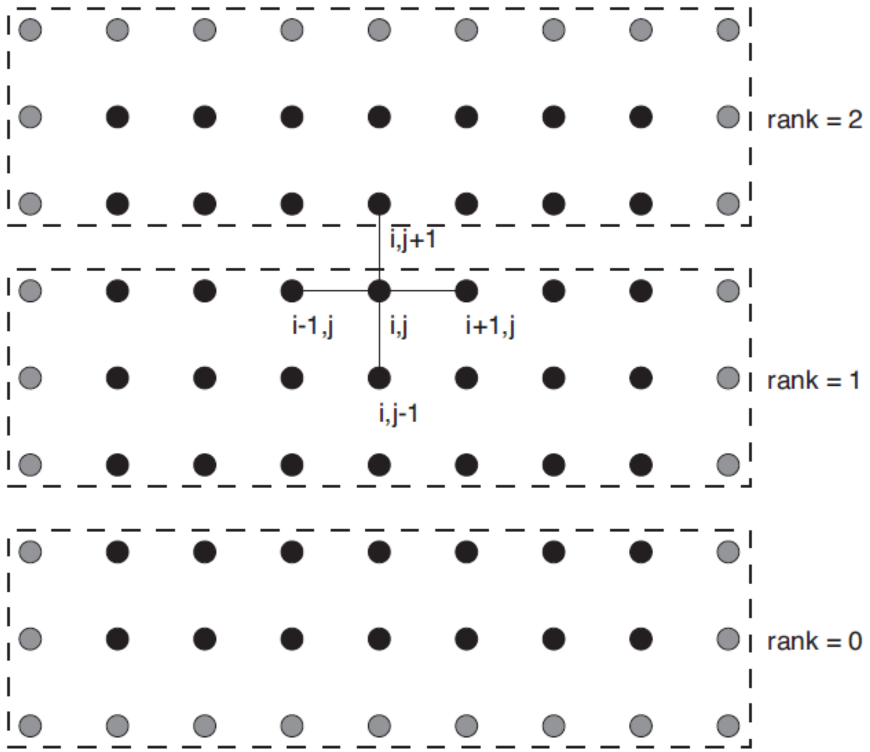


Figure 4.3: 1-D decomposition of the domain

Virtual Toplogies (2)

- Purpose of virtual topologies in MPI is to provide a better mapping of the MPI ranks to the physical hardware
  - e.g. process affinity
- Also simplifies identification of neighbors in nearest neighbor type communication

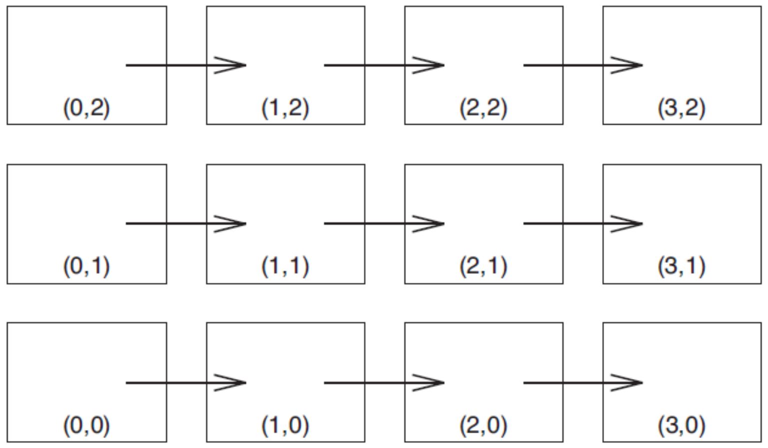


Figure 4.6: A two-dimensional Cartesian decomposition of a domain, also showing a shift by one in the first dimension. Tuples give the coordinates as would be returned by `MPI_Get_coords`.

MPI_Cart_create	Create Cartesian Virt. Topology
MPI_Cart_shift	Get ranks provided shift
MPI_Cart_get	Get your cords in topology
MPI_Cart_coords	Get topology coordinates given rank