COLLEGE OF ENGINEERING
NUCLEAR ENGINEERING & RADIOLOGICAL SCIENCES
UNIVERSITY OF MICHIGAN

Lecture 16
The Message Passing Interface (Part Deux)

Prof. Brendan Kochunas

10/30/2019

NERS 590-004

Outline

- Overview of Advanced MPI features
  - Non-blocking communication & collectives (new)
  - Virtual Topologies
  - I/O
  - Dynamic process management
  - One-sided Communication (new)

- Hands on examples

Learning Objectives

- Undestand what Virtual Topologies are
- Be aware of what's involved with one-sided communication
- Know how to use hybrid parallelism (MPI+X)
- Be able to write a simple MPI program

Source Material for this Lecture

(aka Further Reading)

- Gropp, W., Lusk, E. (2014). *Using MPI: portable parallel programming with the Message-Passing-Interface.* Third edition. Cambridge, Massachusetts: The MIT Press.
  - https://mirlyn.lib.umich.edu/Record/014888004
- Gropp, W., Hoefler, T. (2014). *Using advanced MPI: modern features of the Message-Passing-Interface.* Cambridge, Massachusetts: The MIT Press.
  - https://mirlyn.lib.umich.edu/Record/013606199
- The MPI Standard
  - https://www.mpi-forum.org/docs/

Overview of Other Advanced MPI Features

Advanced MPI Features

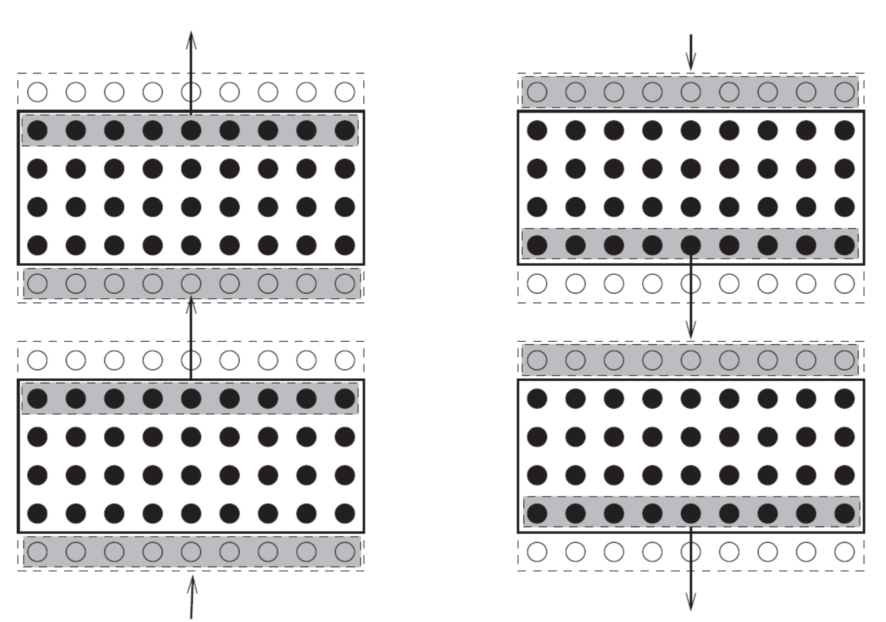
- Advanced because they are new (or because they require an advanced understanding)

- Non-blocking communication & collectives (new)
- Virtual Topologies
- I/O
- Dynamic process management
- One-sided Communication (new)

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.

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

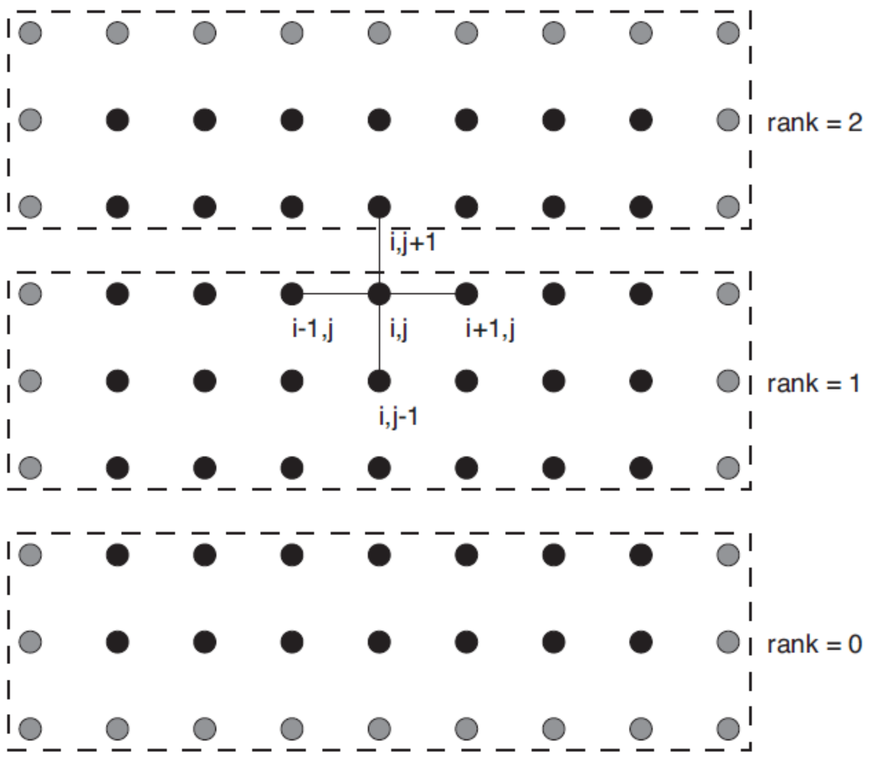


Figure 4.3: 1-D decomposition of the domain

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

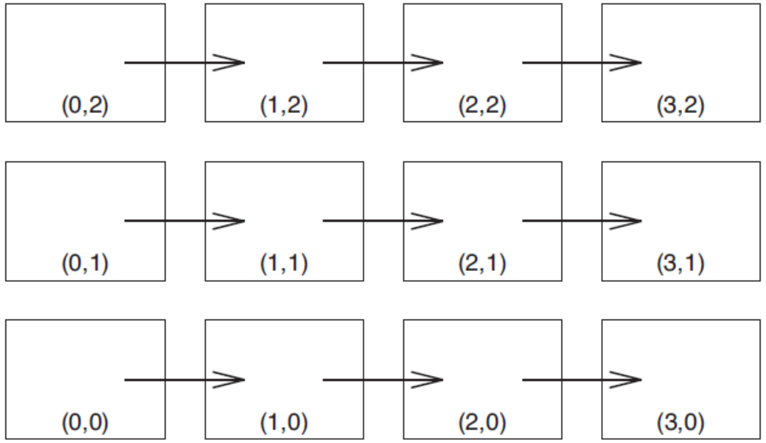


Figure 4.6: A two-dimensional Cartesian decomposition of a domain, also showing a shift by one in the first dimension. Tuples give the coordinates as would be returned by `MPI_Get_coords`.

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

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)

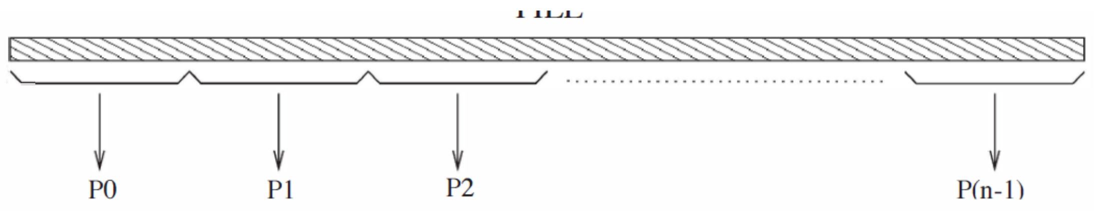
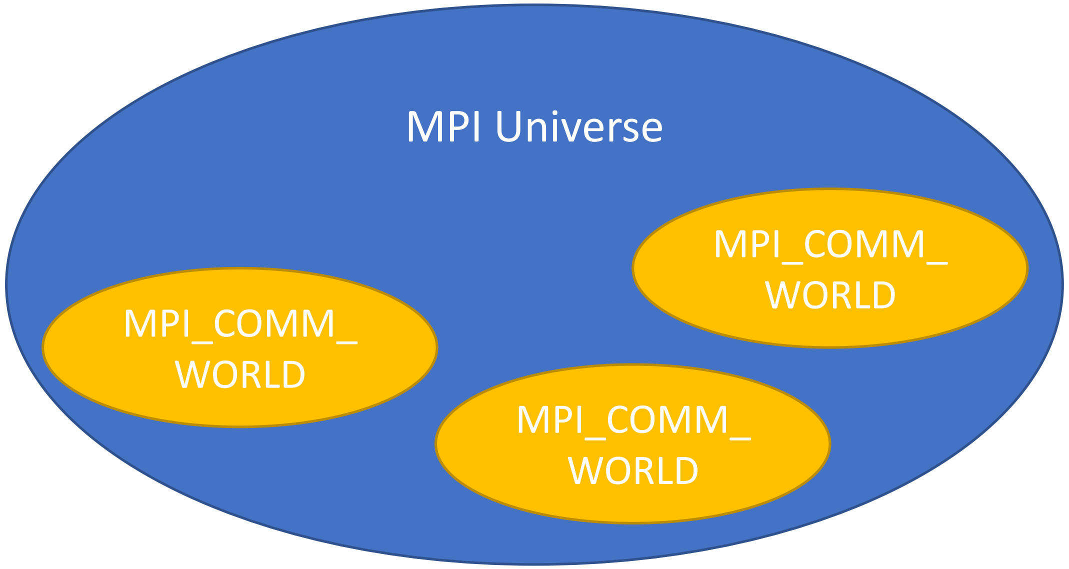


Figure 7.1: Example with n processes, each needing to read a chunk of data from a common file

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

MPI Dynamic Process Management

- Create NEW MPI processes from our MPI processes
  - aka FUBU



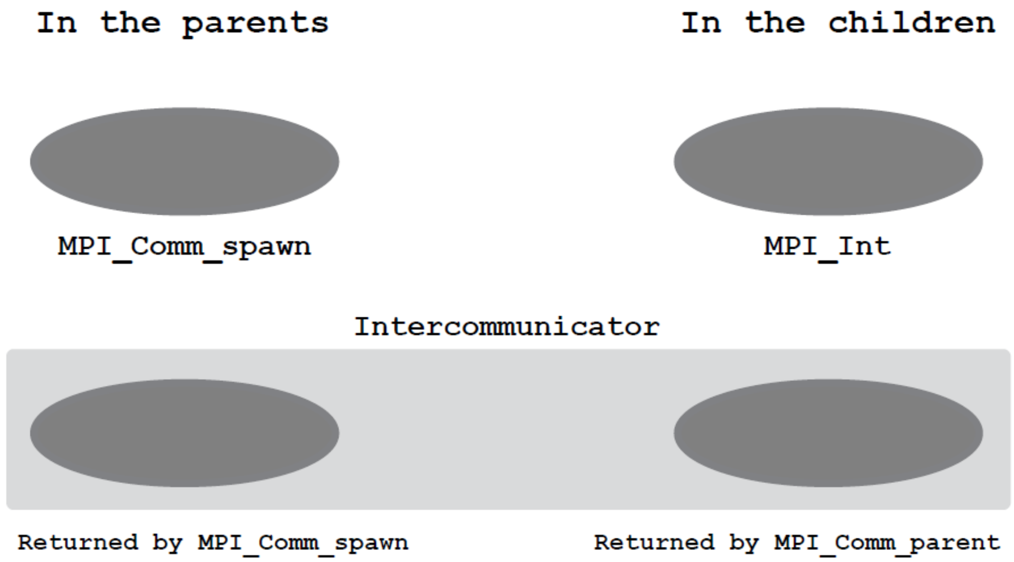


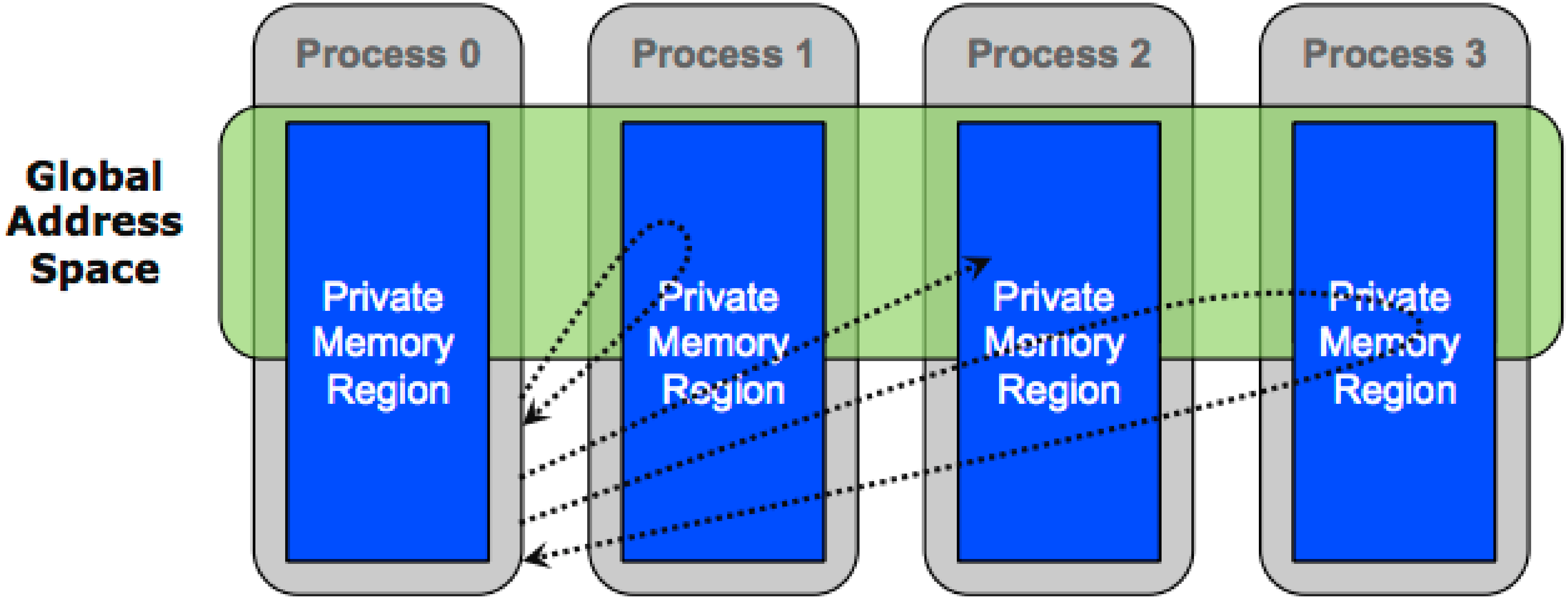
Figure 10.2: Spawning processes. Ovals are intracommunicators containing several processes.

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

```
MPI_Win_create
MPI_Put
MPI_Get
MPI_Accumulate
```

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

- Key concept is a “window object”
  - Exposes larger part of process’s address space for access by other processes

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

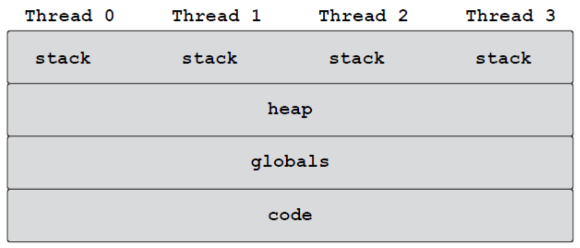


Figure 5.1: Full memory sharing in threaded environments

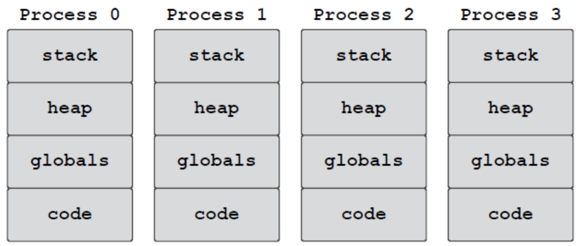


Figure 5.2: Standard MPI semantics—no sharing

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

Summary

- Message passing is one of the most common and heavily used parallel programming model in parallel computing
- MPI is a standard (not an implementation)
  - Many implementations
- Concepts in MPI are minimal
  - Capability of library is broad.
- MPI provides several communication models to support various algorithms
- MPI is still evolving

MPI+OpenMP Example

Ping Pong Example