Lecture 12.1
MPI Messaging and Deadlock

EN 600.320/420

Instructor: Randal Burns

7 March 2018

Department of Computer Science, *Johns Hopkins University*

Point-to-Point Messaging

- This is the fundamental operation in MPI
  - Send a message from one process to another
- Blocking I/O
  - Blocking provides built in synchronization
  - Blocking can lead to deadlock
- Send and receive, let's do an example

See nodeadlock.c

What's in a message?

- First three arguments specify content

```
int MPI_Send (
    void* sendbuf,
    int count,
    MPI_Datatype datatype,
    int dest,
    int tag,
    MPI_Comm comm )
```

What's in a message?

- First three arguments specify content

```
int MPI_Recv (
    void* recvbuf,
    int count,
    MPI_Datatype datatype,
    int source,
    . . . )
```

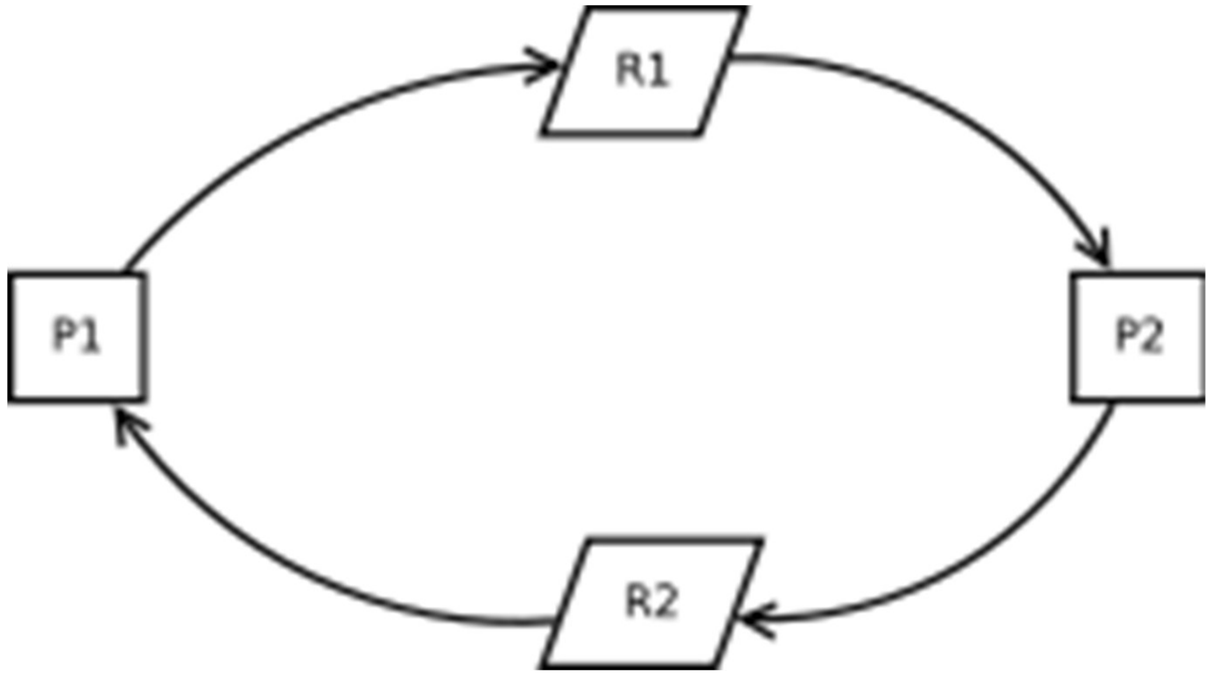
- All MPI data are arrays
  - Where is it?
  - How many?
  - What type?

MPI Datatypes

Table 3.1 Some Predefined MPI Datatypes

MPI datatype	C datatype
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_LONG_LONG	signed long long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

Deadlock



- Conditions for deadlock
  - Mutual exclusion
  - Hold and wait
  - No preemption
  - Circular wait
- Deadlocks are cycles in a resource dependency graph

http://en.wikipedia.org/wiki/Deadlock

Deadlock in MPI Messaging

- Synchronous: the caller waits on the message to be delivered prior to returning
  - *So why didn't our program deadlock?*

See deadlock.c

Deadlock in MPI Messaging

- Synchronous: the caller waits on the message to be delivered prior to returning
  - *So why didn't our program deadlock?*
- Blocking ***standard*** send may be implemented by the MPI runtime in a variety of ways
  - MPI_Send(…, MPI_COMM_WORLD)
  - Buffered at sender or receiver
  - Depending upon message size, number of processes
- Converting to a mandatory synchronous send reveals the deadlock
  - MPI_Ssend(…, MPI_COMM_WORLD)
  - But so could increasing the # of processors

Standard Mode

- MPI runtime chooses best behavior for messaging based on system/message parameters:
  - Amount of buffer space
  - Message size
  - Number of processors

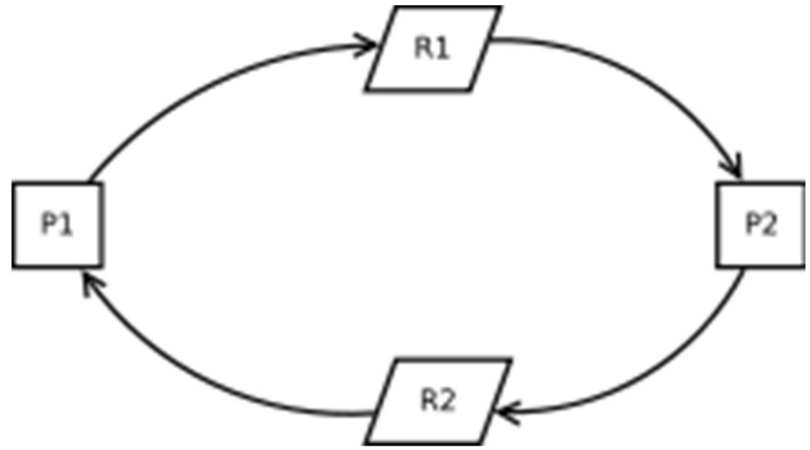
- Preferred way to program??
  - Commonly used and realizes good performance
  - System take available optimizations

- Can lead to horrible errors
  - Because semantics/correctness changes based on job configuration. **Dangerous!**

Standard Mode Danger

- You develop program on small cluster
  - Has plenty of memory for small instances
  - Messages get buffered which hides unsafe (deadlock) messaging protocol
- You launch code on big cluster with big instance
  - Bigger messages and more memory consumption means that MPI can't buffer messages
  - Standard mode falls back to synchronous sends
  - Your code breaks
- Best practice: test messaging protocols with synchronous sends, deploy code in standard mode

Avoiding Deadlock



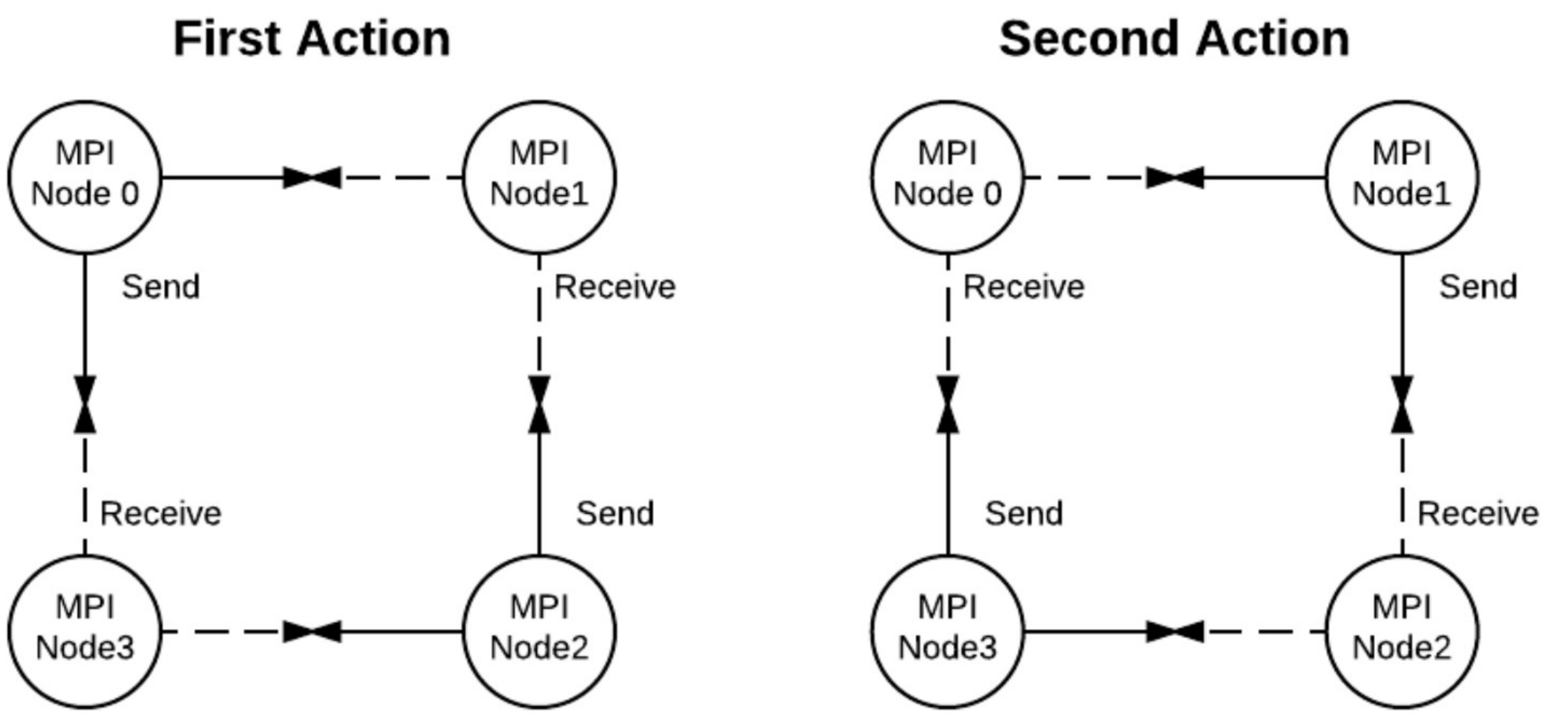
http://en.wikipedia.org/wiki/Deadlock

- Conditions for deadlock
  - Mutual exclusion
  - Hold and wait
  - No preemption
  - Circular wait
- Deadlocks are cycles in a resource dependency graph
- Avoiding deadlock in MPI
  - Create cycle-free messaging disciplines
  - Synchronize actions

See passitforward.c

Messaging Topology

- Pair sends and receives
  - No circular dependencies
  - Relies on/assumes even number of nodes!



Messaging Topologies

- Order/pair sends and receives to avoid deadlocks
- For linear orderings and rings
  - Simplest and sufficient: (n-1) send/receive, 1 receive/send
  - More parallel, alternate send/receive and receive/send
- For more complex communication topologies?
- Messaging topology dictates parallelism
  - Important part of parallel design