UiT THE ARCTIC UNIVERSITY OF NORWAY

Lecture 3a: Message-Passing Computing

Parallell Programming (INF-3201)

John Markus Bjørndalen

Message-Passing Programming using User-level Message-Passing Libraries

Two primary mechanisms needed:

1. A method of creating separate processes
   - typically executing on different computers

2. A method of sending and receiving messages

Process creation

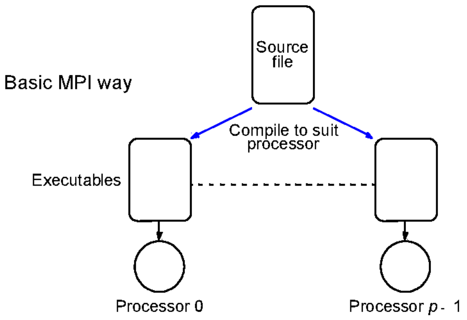
- Static
  - All processes are specified before execution
  - The number of processes is fixed during execution.

- Dynamic
  - Processes can be created and their execution initiated during the execution of other processes.
  - The number of processes may vary during execution

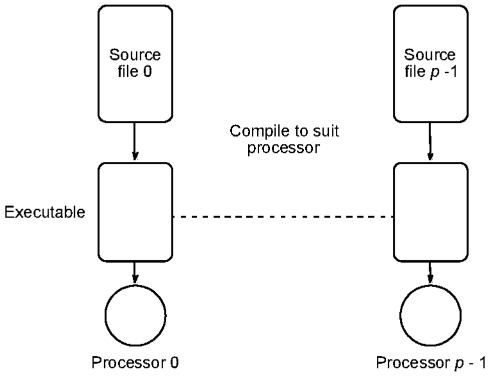
- The code for processes is normally written and compiled before the execution of any process

Single Program Multiple Data (SPMD) model

Different processes merged into one program. Control statements select different parts for each processor to execute. All executables started together - static process creation

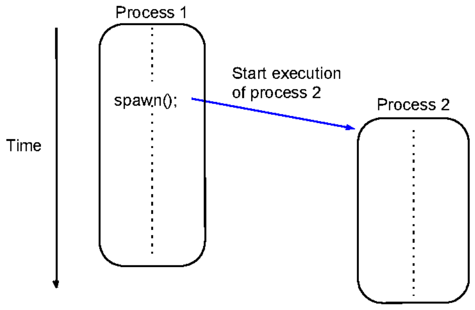


Multiple program, multiple data (MPMD) model



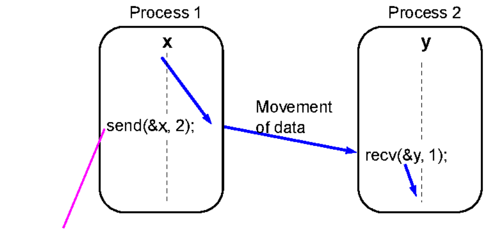
Multiple Program Multiple Data (MPMD) Model

Separate programs for each processor. One processor executes master process. Other processes started from within master process - dynamic process creation.



Basic "point-to-point" Send and Receive Routines

Passing a message between processes using send() and recv() library calls:



Generic syntax (actual formats later)

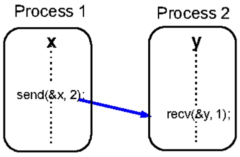
Synchronous Message Passing

Performs two actions:

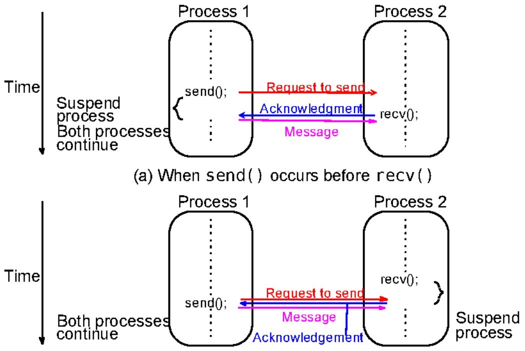
- Transfer data
- Synchronize processes.

Send - returns when message can be accepted by receiver.

Receive – returns when message received.



Synchronous send() and recv() using 3-way protocol



(a) When send() occurs before recv()

(b) When recv() occurs before send()

Asynchronous Message Passing

- Routines that do not wait for actions to complete before returning. Usually require local storage for messages.
- More than one version depending upon the actual semantics for returning.
- In general, they do not synchronize processes but allow processes to move forward sooner. Must be used with care.
  - Process 1:
    ```
    x = a
    Isend(&x, 2)
    x = b
    ```
  - Which value will be sent to process 2, a or b?

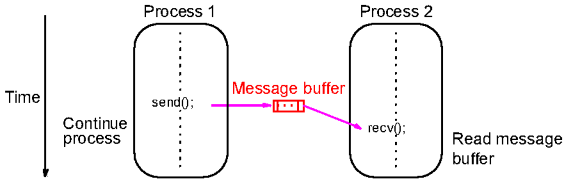
MPI Definitions of Blocking and Non-Blocking

- Blocking - return after their local actions complete, though the message transfer may not have been completed.
- Non-blocking - return immediately.
  - Assumes that data storage used for transfer not modified by subsequent statements prior to being used for transfer, and it is left to the programmer to ensure this.

These terms may have different interpretations in other systems.

How message-passing routines return before message transfer completed

Message buffer needed between source and destination to hold message:



Asynchronous (blocking) routines changing to synchronous routines

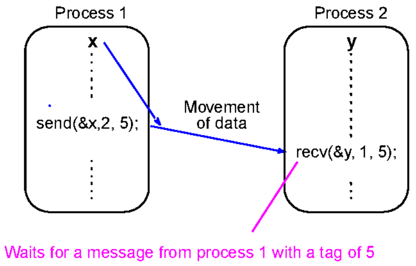
- Once local actions completed and message is safely on its way, sending process can continue with subsequent work.
- Buffers only of finite length and a point could be reached when send routine held up because all available buffer space exhausted.
- Then, send routine will wait until storage becomes re-available - i.e then routine behaves as a synchronous routine.

Message selection - Message Tag

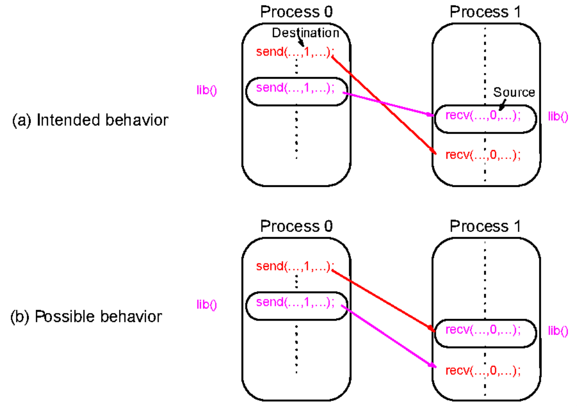
- Used to differentiate between different types of messages being sent.
  - P_1: send(&x, 2, 5)
  - P_2: recv(&y, 1, 5)
- Message tag is carried within message.
- If special type matching is not required, a wild card message tag is used, so that the recv() will match with any send().

Message Tag Example

To send a message, x, with message tag 5 from a source process, 1, to a destination process, 2, and assign to y:



Unsafe message passing - Example



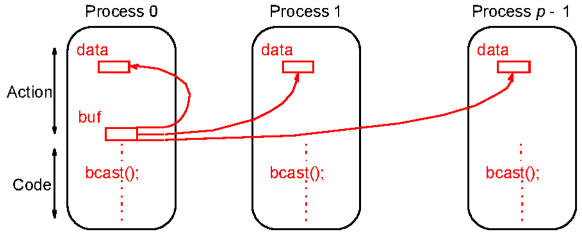
“Group” message passing routines

- Have routines that send message(s) to a group of processes or receive message(s) from a group of processes
- Higher efficiency than separate point-to-point routines although not absolutely necessary.
  - Example: IP-multicast

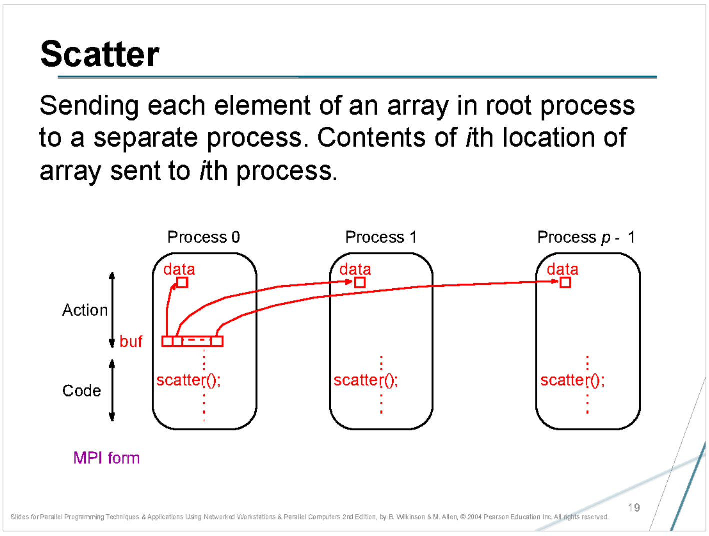
Broadcast

Sending same message to all processes concerned with problem.

Multicast - sending same message to defined group of processes.

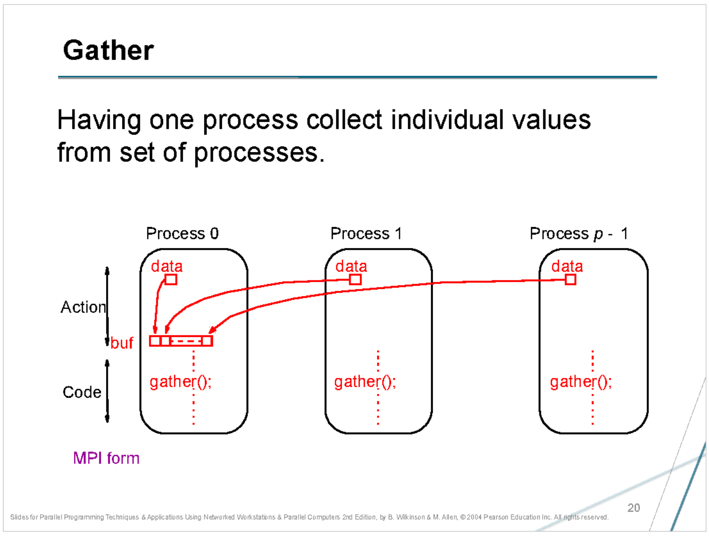


MPI form



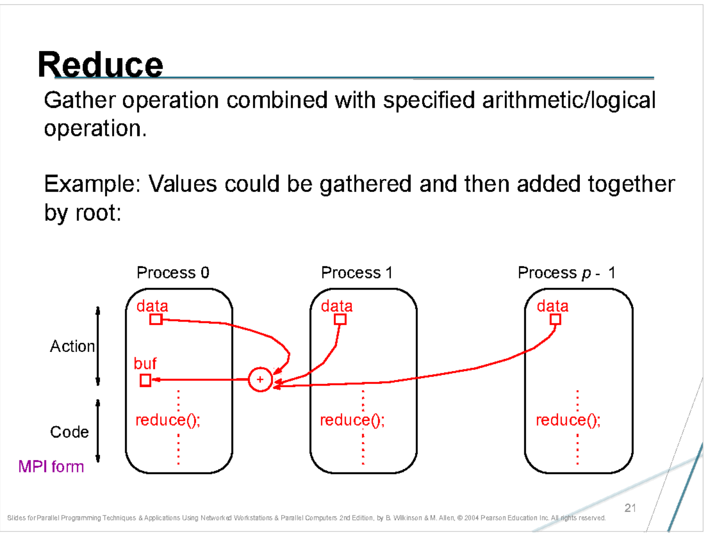
Scatter

Sending each element of an array in root process to a separate process. Contents of *i*th location of array sent to *i*th process.



Gather

Having one process collect individual values from set of processes.



Reduce

Gather operation combined with specified arithmetic/logical operation.

Example: Values could be gathered and then added together by root: