C-DAC Four Days Technology Workshop

ON

Hybrid Computing – Coprocessors/Accelerators
Power-Aware Computing – Performance of Applications Kernels

hyPACK-2013
(Mode-1: Multi-Core)

Lecture Topic:
Multi-Core Processors:MPI 1.0 Overview (Part-III)

Venue : CMSD, UoHYD ; Date : October 15-18, 2013

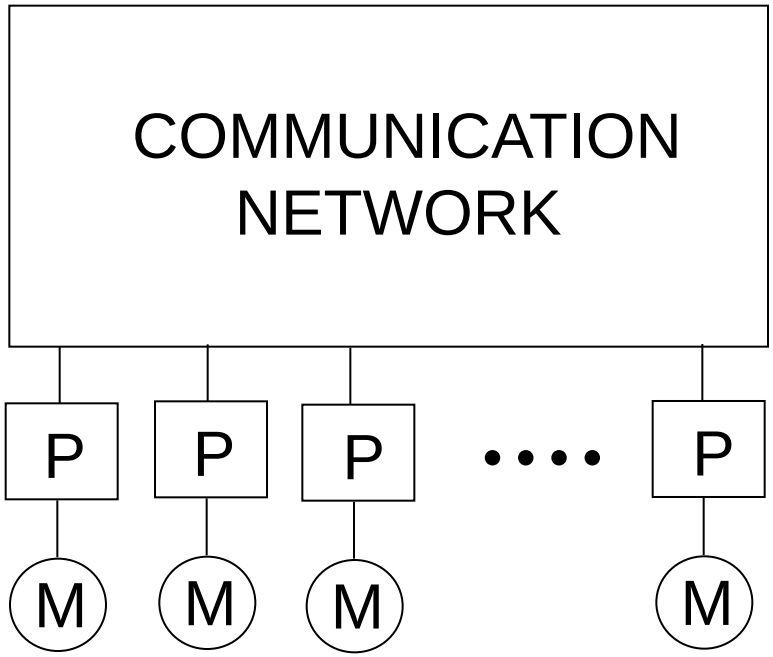
Lecture Outline

- MPI advanced point-to-point communication
- MPI Communication modes
- Cost of Message Passing
- Types of Synchronization
- MPI –2 Features
- Conclusions

Source : Reference : [11], [12], [25], [26]

Message Passing Architecture Model

Message-Passing Programming Paradigm : Processors are connected using a message passing interconnection network.



MPI Communication Modes

(Contd…)

Communication Modes

The mode of a point to point communication operation governs when a send operation is initiated, or when it completes

- **Standard mode**
  - A send may be initiated ***even if*** a matching receive ***has not been initiated***
- **Ready mode**
  - A send may be initiated ***only if*** a matching receive ***has been initiated***

MPI Communication Modes

(Contd…)

Communication Modes

- Two basic ways of checking on non-blocking send and receives
  - Call a wait routine that blocks until completion
  - Call a test routine that returns a flag to indicate if complete
- Use non-blocking and completion routines allow computation and communication to be overlapped
  - ***mpi_wait***(request_id,return_status,ierr)
  - ***mpi_test***(request_id,flag,return_status,ierr)

MPI Communication Modes

(Contd…)

Communication Modes

- **Non-blocking send**
  - Returns “immediately
  - Message buffer should not be written to after return
  - Must check for local completion
- **Blocking send**
  - Returns when send is locally complete
  - Message buffer should not be read from after return
  - Must check for local completion

MPI Communication Modes

(Contd…)

Communication Modes

- ***mpi_wait*** blocks until communication is complete
- ***mpi_status*** returns “immediately”, and sets ***flag*** to ***true*** if the communication is complete
- **Blocking receive**
  - Returns when receive is locally complete
  - Message buffer can be read from after return

Source : Reference : [11], [12], [25], [26]

MPI Communication Modes

(Contd…)

Sender mode	Notes
Synchronous send	Only completes when the receive has completed
Buffered send	Always completes (unless an error occurs), irrespective of receiver.
Standard send	Either synchronous or buffered.
Ready send	Always completes (unless an error occurs), irrespective of whether the receive has completed.
Receive	Completes when a message has arrived.

Source : Reference : [11], [12], [25], [26]

MPI Communication Modes

(Contd…)

MPI Sender Modes

OPERATION	MPI CALL
Standard send	MPI_SEND
Synchronous send	MPI_SSEND
Buffered send	MPI_BSEND
Ready send	MPI_RSEND
Receive	MPI_RECV

MPI Blocking Send and Receive

Blocking Send

A typical blocking send looks like

send (*dest, type, address, length*)

Where

- **`dest`** is an integer identifier representing the process to receive the message
- **`type`** is nonnegative integer that the destination can use to selectively screen messages
- **(`address`, `length`)** describes a contiguous area in memory containing the message to be sent

MPI Blocking Send and Receive

(Contd…)

Point-to-Point Communications

The sending and receiving of messages between pairs of processors.

- **BLOCKING SEND:** returns only after the corresponding RECEIVE operation has been issued and the message has been transferred.

  MPI_Send

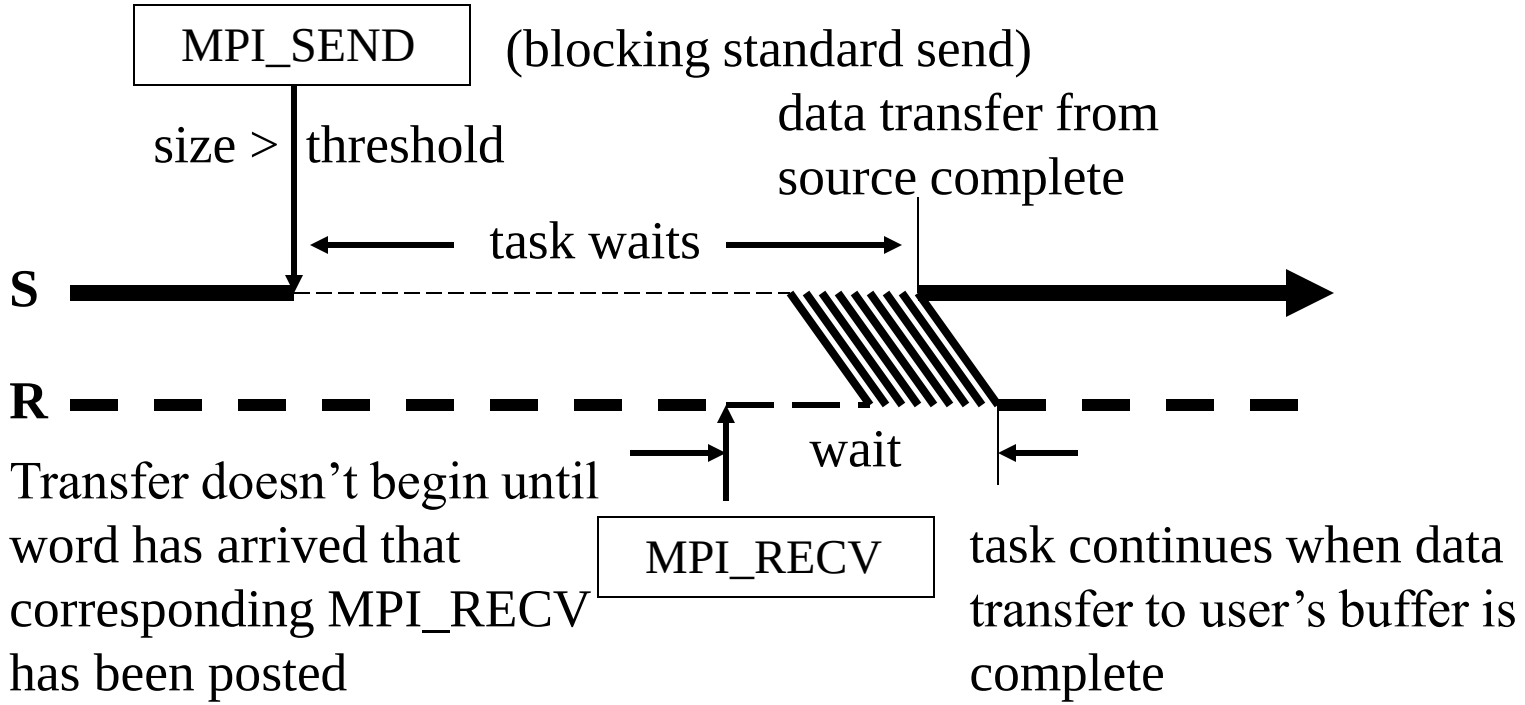
- **BLOCKING RECEIVE:** returns only after the corresponding SEND has been issued and the message has been received.

  MPI_Recv

Blocking Sends and Receives

If we are sending a large message, most implementations of blocking send and receive use the following procedure.

S = Sender R = Receiver



MPI Non-Blocking Send and Receive

Non-Blocking Send and Receive

- **Non-blocking Receive:** does not wait for the message transfer to complete, but immediate returns control back to the calling processor.

MPI_IRecv

C

MPI_Isend (buf, count, dtype, dest, tag, comm, request);

MPI_Irecv (buf, count, dtype, dest, tag, comm, request);

Fortran

MPI_Isend (buf, count, dtype, tag, comm, request, ierror)

MPI_Irecv (buf, count, dtype, source, tag, comm, request, ierror)

MPI Non-Blocking Send and Receive (Contd…)

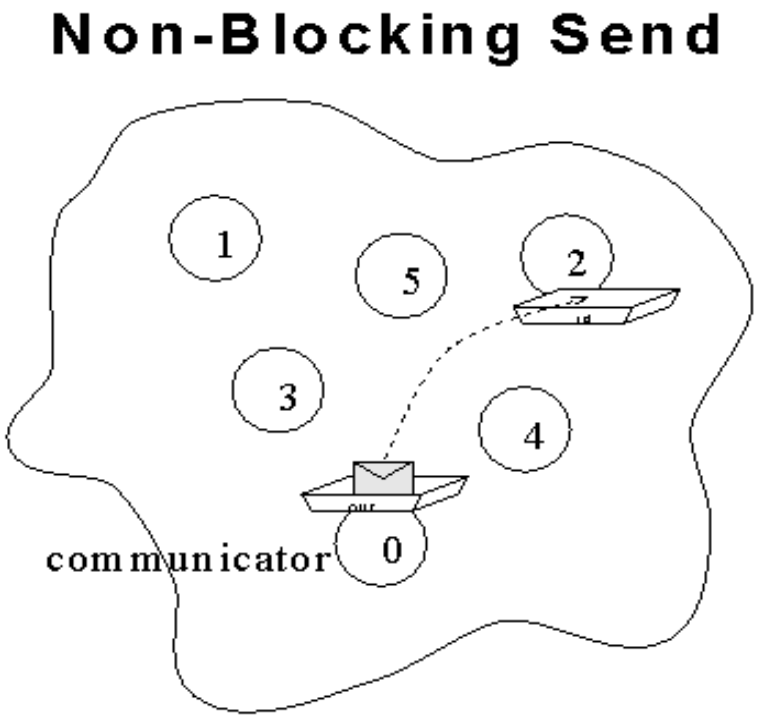
Non-Blocking Communications

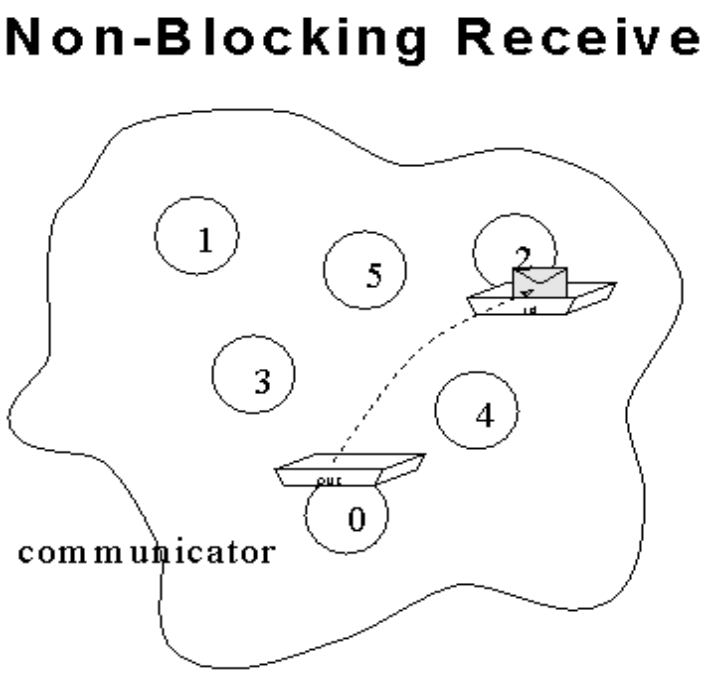
- Separate communication into three phases:
  - Initiate non-blocking communication.
  - Do some work (perhaps involving other communications ?)
  - Wait for non-blocking communication to complete.

Source : Reference : [11], [12], [25], [26]

MPI Non-Blocking Send and Receive

(Contd…)



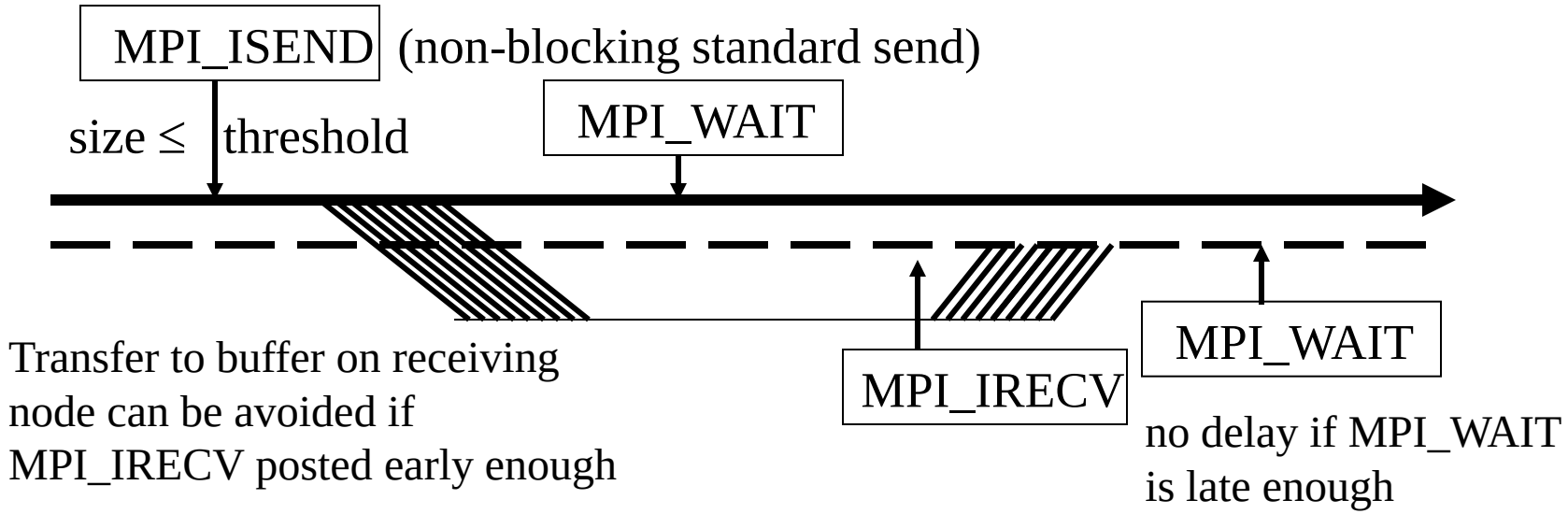


MPI Non-Blocking Send and Receive

If we are sending a `small` message, most implementations of non-blocking sends and receive use the following procedure. The message can be sent immediately and stored in a buffer on the receiving side.

S = Sender R = Receiver

An MPI-Wait checks to see it a non-blocking operation has completed. In this case, the MPI_Wait on the sending side believes the message has already been received.



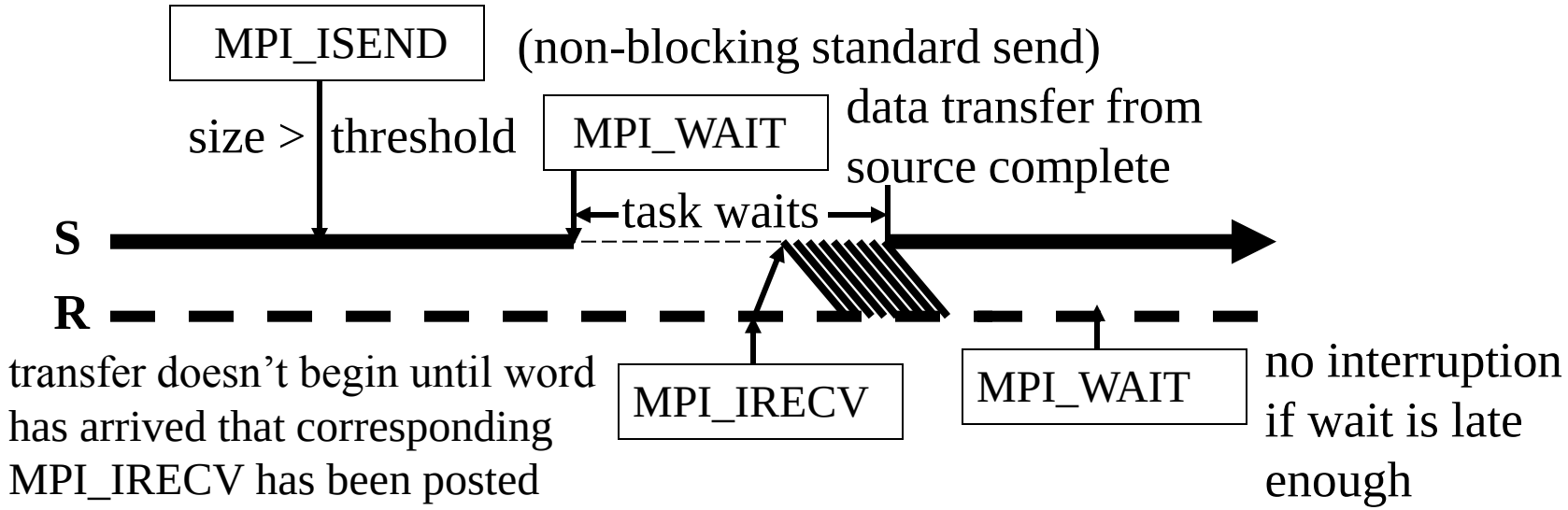
MPI Non-Blocking Send and Receive

(Contd…)

If we are sending a `large` message, most implementations of non-blocking sends and receive use the following procedure. The send is issued, but the data is not immediately sent. Computation is resumed after the send, but later halted by an MPI_Wait.

S = Sender R = Receiver

An MPI_Wait checks to see it a non-blocking operation has completed. In this case, the MPI_Wait on the sending side sees that the message has not been sent yet.



MPI Communication Modes

Communication Modes

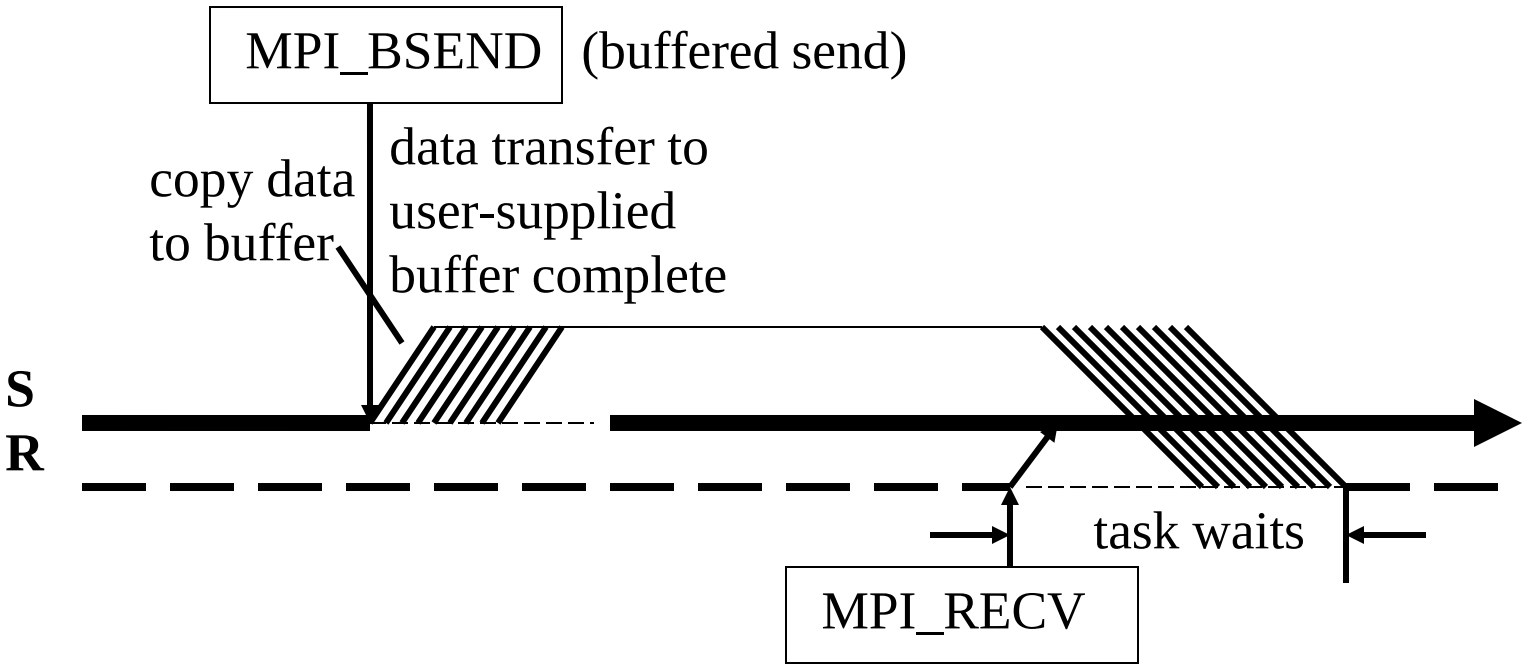
- **Synchronous mode**
  - The same as standard mode, except the send will not complete until message delivery is guaranteed
- **Buffered mode**
  - Similar to standard mode, but completion is always independent of matching receive, and message may be buffered to ensure this

Source : Reference : [11], [12], [25], [26]

Buffered Sends and Receives

If we the programmer allocate some memory (buffer space) for temporary storage on the sending processor, we can perform a type of non-blocking send.

S = Sender R = Receiver



MPI Persistent Communication

MPI : Nonblocking operations, overlap effective

- Isend, Irecv, Waitall

MPI : Persistent Operations

- Potential saving

  " Allocation of MPI_Request

- Variation of example

  " sendinit, recvinit, startall, waitall

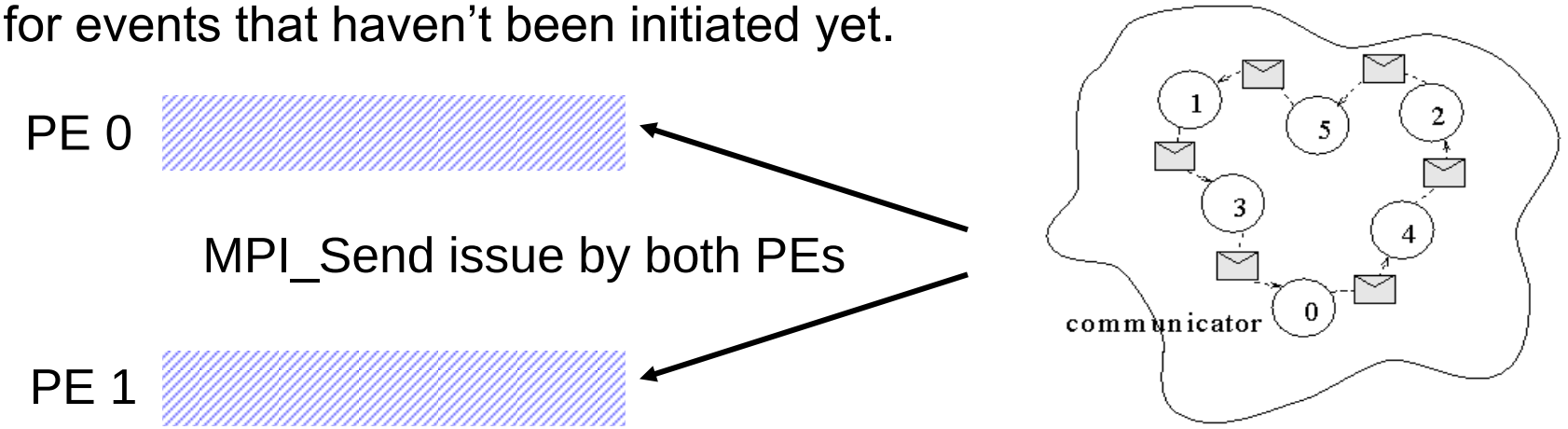
  " startall(recvs), sendrecv/barrier, startall(rsends), waitall

- Performance is based on Vendor implementations

MPI Cause of Deadlock

MPI Message Passing - Deadlock

Causes of Deadlock : Deadlock occurs when all tasks are waiting for events that haven't been initiated yet.



The diagram demonstrates that the two sends are each waiting on their corresponding receives in order to complete, but those receives are executed after the sends, so if the sends do not complete and return, the receives can never be executed, and both sets of communications will stall indefinitely.

Source : Reference : [11], [12], [25], [26]

MPI- Avoiding Deadlock

MPI Message Passing : Avoiding Deadlock

- **Change Ordering**
  - Different ordering of calls between tasks
  - Arrange for one task to post its receive first and for the other to post its send first.

- **Non-blocking calls**
  - Have each task post a non-blocking receive before it does any other communication.
  - This allows each message to be received, no matter what the task is working on when the message arrives or in what order the sends are posted.

MPI-Avoiding Deadlock

(Contd…)

MPI Message Passing Avoiding Deadlock

- **MPI_Sendrecv**
  - MPI_Sendrecv_replace use MPI_Sendrecv (S).
  - The send-receive combines in one call the sending of a message to a destination and the receiving from a source
- **Buffered mode**
  - Use buffered sends so that computation can proceed after copying the message to the user-supplied buffer.
  - This will allow the send to complete and the subsequent receive to be executed.

MPI Collective Communications

(Contd…)

Collective Communication

- Communications involving a group of processes.
- Called by all processes in a communicator.
- Examples:
  - Barrier synchronization.
  - Broadcast, scatter, gather.
  - Global sum, global maximum, etc.

MPI Collective Communications

(Contd…)

Characteristics of Collective Communication

- Collective action over a communicator
- All processes must communicate
- Synchronization may or may not occur
- All collective operations are blocking.
- No tags.
- Receive buffers must be exactly the right size

MPI Collective Communications

(Contd…)

Collective Communications

Communication is coordinated among a group of processes

- Group can be constructed "**by hand**" with MPI group-manipulation routines or by using MPI topology-definition routines
- Different communicators are used instead
- No non-blocking collective operations

Source : Reference : [11], [12], [25], [26]

MPI - Using topology

MPI : Support for Regular Decompositions

- Using topology routines

  "MPI_Cart_Create "

  User can define `virtual topology`

- Why you use the topology routines

  "Simple to use (why not?)

  "Allow MPI implementation to provide low expected contention layout of processes (contention can matter)

  "Remember,contention still matters; a good mapping can reduce contention effects

MPI Datatypes

Message type

- A message contains a number of elements of some particular datatype
- MPI datatypes:
  - Basic types
  - Derived Data types (Vectors; Structs; Others)
- Derived types can be built up from basic types
- C types are different from Fortran types

MPI Basic Datatypes

(Contd…)

MPI Basic Datatypes - C

MPI Datatype	C datatype
MPI_CHAR	Signed char
MPI_SHORT	Signed short int
MPI_INT	Signed int
MPI_LONG	Signed 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	

MPI Derived Data types

(Contd…)

Contiguous Data

- The simplest derived `datatype` consists of a number of contiguous items of the same `datatype`

- **C :**

  int MPI_Type_contiguous (int count, MPI_Datatype oldtype,MPI_Datatype *newtype);

- **Fortran :**

  MPI_Type_contiguous (count, oldtype, newtype)

  integer count, oldtype, newtype

MPI Derived Data types

(Contd…)

Constructing a Vector Datatype

- **C**

  int MPI_Type_vector (int count, int blocklength, int stride, MPI_Datatype oldtype, MPI_Datatype *newtype);

- **Fortran**

  MPI_Type_vector (count, blocklength, stride, oldtype, newtype, ierror)

Extent of a Datatype

- **C**

  int MPI_Type_extent (MPI_Datatype datatype, int *extent);

- **Fortran**

  MPI_Type_extent(datatype, extent, ierror)

  integer datatype, extent, ierror

MPI Derived Data types

(Contd…)

Constructing a Struct Datatype

- C :

```
int MPI_Type_struct (int count, int array_of_blocklengths,
    MPI_Aint *array_of_displacements,
    MPI_Datatype *array_of_types,
    MPI_Datatype *newtype);
```

- Fortran :

```
MPI_Type_Struct (count, array_of_blocklengths,
        array_of_displacements,  array_of_types, newtype, ierror)
```

MPI Derived Data types

(Contd…)

Committing a datatype

- Once a datatype has been constructed, it needs to be committed before it is used.
- This is done using `MPI_TYPE_COMMIT`
- **C**

  int MPI_Type_Commit (MPI_Datatype *datatype);

- **Fortran**

  MPI_Type_Commit (datatype, ierror)

  integer datatype, ierror

MPI Derived Data types

(Contd…)

MPI : Performance of MPI datatypes

- Handing non-contiguous data;
- Test of 1000 element vector of doubles with stride of 24 doubles.
  - "MPI_Type_vector" and "MPI_Type_struct(.*,*)";
  - "User packs and unpacks by hand
- Performance very dependent on implementation; should improve with time
- Collect many small messages into a single large message;
- Use of collective when many copies bcast /gather

Cost of Message Passing

- Message passing programs that exploit data parallelism often use messages to transfer **required data** among processors
- Startup time has both a *hardware* as well as *software* relate component
- Common message-passing operations
  - Point to point / Collective communication
  - Blocking and Non-Blocking type
  - Barrier
  - Broadcast, Reduction, Prefix, Gather, Scalter, All to All Gather and Scatter operations

Cost of Message Passing

(Contd…)

Startup-time and transfer-time

- The time required to send a message can be divided into two parts: startup-time and transfer-time
- Every time a message is passed between processors, the processors involved must spend some time sending and receiving the message
- The cost depends on the size of the message, how far it has to travel, and the status of the network at the time of transmission

Source : Reference : [11], [12], [25], [26]

Cost of Message Passing

(Contd…)

Hardware :

- Time required to prepare the message for under-lying network (such as adding header,trailer, and error correction information)
- The time to execute the writing algorithm
- The time to establish an interface between the local processor and router
- Startup time is fixed and does not depend on the size of message being sent

Cost of Message Passing

(Contd…)

Software :

- Depends on the protocol followed by the underlying message passing library
- In general, this may include the time spent by the message-passing library for creating various internal data structures
- Copying the message to internal buffers
- Negotiating the transmission of data between the sending and receiving processors

Cost of Message Passing

(Contd…)

The transfer time of a message

- Depends on the size of the message and bandwidth of
- The underlying interconnection network
- Per word transfer rate (t_w)=*1/r*
- Transfer time of a message of size *n* words (nt_w)
- In most cases, a message traveling from one processor to another will have to traverse many links

Cost of Message Passing

(Contd…)

Remarks :

- Different messages traverse the interconnection network concurrently, more than one message may want to traverse the same link of the interconnection network at the same time.
- In this case, only the one message at a time will traverse the link, forcing the remaining messages to wait.
- As the message sizes and distances being traveled increase, there is a greater likelihood that link contention will arise.

Cost of Message Passing

(Contd…)

Remarks :

- The software related startup time is usually much higher than of that of the hardware end
- Depending on the type of MPI communication operation performed
- May depend on the size of the message being sent
- The status of the receiving processor, and other traffic in the network
- The transfer time of a message depends on the size of the message and the bandwidth of the underlying interconnection network

Cost of Message Passing

(Contd…)

Cost of Collective Communication Operations

The time required by the MPI collective communication operation varies greatly and it depends on type of communication call used.

- Barrier operation
- Element wise Reduce Operation of *n* words
- BROADCAST of a message of size *n* words
- Algorithm is more important for efficient implementation of collective communication operations
- Time required by each operation depends on characteristic of the interconnection network

Cost of Message Passing

(Contd…)

(Cut-through- routing)

Operation	Complexity
BARRIER	*0(log p)*
BROADCAST	*0(n log p)*
REDUCE	*0(n log p)*
PREFIX	*0(np)*
SCATTER	*0(np)*

BROADCAST *n* is size of the message being broadcasted

REDUCE *n* is size of message stored in each processor

SCATTER *n* is size of the message received from each processor

Types of Synchronization

Synchronization Speed

- Refers to the time needed for all processors to agree that they have finished one step of a problem and are ready to go on together to the next step
- Cost of message passing also depends on Synchronization Speed of system

Synchronization methods

- Waiting all processes finish a loop
- Waiting until the first of any of the contributing processes finds a particular answer
- Assigning a unique task to each processor from a list of tasks

MPI- Synchronization

(Contd…)

MPI-Synchronization Delays

- Message passing is a cooperative method - if the partner doesn't react quickly, a delay results
- There is a performance tradeoff caused by reacting quickly - it requires devoting resources to checking.

Memory copies

- Memory copies are the primary source of performance problem and single processor `memcpy` is often much slower than the hardware
- Cost of non-contiguous data types
- Measured memcpy performance

Features of MPI

(Contd…)

- **Profiling**
  - Hooks allow users to intercept MPI calls
- **Environmental**
  - Inquiry and Error control
- **Collective**
  - Both built-in and user-defined collective operations
  - Large number of data movements routines
  - Subgroups defined directly or by topology
- **Application-oriented process topologies**
  - Built-in support for grids and graphs (uses groups)

Features of MPI

(Contd…)

- **General**
  - Communicators combine context & group for message security
  - Thread safety
- **Point-to-Point communication**
  - Structured buffers and derived datatypes, heterogeneity
  - Modes: normal (blocking and non-blocking), synchronous, ready (to allow access to fast protocols), buffered
- **Non-message-passing concepts included**
  - Active messages and Threads
- **Non-message-passing concepts not included:**
  - Process management; Remote memory transfers; Virtual shared memory

MPI - Performance

Tuning Performance (General techniques)

- Aggregation
- Decomposition
- Load Balancing
- Changing the Algorithm

Tuning Performance

- Performance Techniques
- MPI -Specific tuning
- Pitfalls

Summary of MPI

- Summary of MPI advanced point-to-point communication, Data Types, Topologies
- Summary of MPI Collective Communication and Computations
- Cost of Message Passing Operations

Source : Reference : [11], [12], [25], [26]

References

1. Andrews, Grogory R. **(2000),** Foundations of Multithreaded, Parallel, and Distributed Programming, Boston, MA : Addison-Wesley

2. Butenhof, David R **(1997),** Programming with POSIX Threads , Boston, MA : Addison Wesley Professional

3. Culler, David E., Jaswinder Pal Singh **(1999),** Parallel Computer Architecture - A Hardware/Software Approach , San Francsico, CA : Morgan Kaufmann

4. Grama Ananth, Anshul Gupts, George Karypis and Vipin Kumar **(2003),** Introduction to Parallel computing, Boston, MA : Addison-Wesley

5. Intel Corporation, **(2003),** Intel Hyper-Threading Technology, Technical User's Guide, Santa Clara CA : Intel Corporation Available at : http://www.intel.com

6. Shameem Akhter, Jason Roberts **(April 2006),** Multi-Core Programming - Increasing Performance through Software Multi-threading , Intel PRESS, Intel Corporation,

7. Bradford Nichols, Dick Buttlar and Jacqueline Proulx Farrell **(1996),** Pthread Programming O'Reilly and Associates, Newton, MA 02164,

8. James Reinders, Intel Threading Building Blocks – (**2007**) , O'REILLY series

9. Laurence T Yang & Minyi Guo (Editors), (**2006**) *High Performance Computing - Paradigm and Infrastructure* Wiley Series on Parallel and Distributed computing, Albert Y. Zomaya, Series Editor

10. Intel Threading Methodology ; Principles and Practices Version 2.0 copy right **(March 2003),** Intel Corporation

References

11. William Gropp, Ewing Lusk, Rajeev Thakur **(1999),** Using MPI-2, Advanced Features of the Message-Passing Interface, The MIT Press..

12. Pacheco S. Peter, **(1992),** Parallel Programming with MPI, , University of Sanfrancisco, Morgan Kaufman Publishers, Inc., Sanfrancisco, California

13. Kai Hwang, Zhiwei Xu, (**1998**), Scalable Parallel Computing (Technology Architecture Programming), McGraw Hill New York.

14. Michael J. Quinn (**2004**), Parallel Programming in C with MPI and OpenMP McGraw-Hill International Editions, Computer Science Series, McGraw-Hill, Inc. Newyork

15. Andrews, Grogory R. **(2000),** Foundations of Multithreaded, Parallel, and Distributed Progrmaming, Boston, MA : Addison-Wesley

16. SunSoft. Solaris multithreaded programming guide. SunSoft Press, Mountainview, CA, **(1996),** Zomaya, editor. Parallel and Distributed Computing Handbook. McGraw-Hill,

17. Chandra, Rohit, Leonardo Dagum, Dave Kohr, Dror Maydan, Jeff McDonald, and Ramesh Menon, **(2001)**,Parallel Programming in OpenMP San Fracncisco Moraan Kaufmann

18. S.Kieriman, D.Shah, and B.Smaalders **(1995),** Programming with Threads, SunSoft Press, Mountainview, CA. 1995

19. Mattson Tim, **(2002),** Nuts and Bolts of multi-threaded Programming Santa Clara, CA : Intel Corporation, Available at : http://www.intel.com

20. I. Foster **(1995,** Designing and Building Parallel Programs ; Concepts and tools for Parallel Software Engineering, Addison-Wesley (1995)

21. J.Dongarra, I.S. Duff, D. Sorensen, and H.V.Vorst **(1999),** Numerical Linear Algebra for High Performance Computers (Software, Environments, Tools) SIAM, 1999

References

22. OpenMP C and C++ Application Program Interface, Version 1.0". **(1998),** OpenMP Architecture Review Board. October 1998

23. D. A. Lewine. *Posix Programmer's Guide:* **(1991),** Writing Portable Unix Programs with the Posix. 1 Standard. O'Reilly & Associates, 1991

24. Emery D. Berger, Kathryn S McKinley, Robert D Blumofe, Paul R.Wilson, *Hoard : A Scalable Memory Allocator for Multi-threaded Applications* ; The Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX). Cambridge, MA, November (**2000)**. Web site URL : http://www.hoard.org/

25. Marc Snir, Steve Otto, Steyen Huss-Lederman, David Walker and Jack Dongarra, (**1998**) *MPI-The Complete Reference: Volume 1, The MPI Core, second edition* [MCMPI-07].

26. William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, William Saphir, and Marc Snir (**1998**) *MPI-The Complete Reference: Volume 2, The MPI-2 Extensions*

27. A. Zomaya, editor. Parallel and Distributed Computing Handbook. McGraw-Hill, **(1996)**

28. OpenMP C and C++ Application Program Interface, Version 2.5 (**May 2005**)", From the OpenMP web site, URL **: http://www.openmp.org/**

29. Stokes, Jon 2002 Introduction to Multithreading, Super-threading and Hyper threading *Ars Technica*, October **(2002)**

30. Andrews Gregory R. 2000, Foundations of Multi-threaded, Parallel and Distributed Programming, Boston MA : Addison – Wesley (**2000)**

31. Deborah T. Marr , Frank Binns, David L. Hill, Glenn Hinton, David A Koufaty, J . Alan Miller, Michael Upton, "Hyperthreading, Technology Architecture and Microarchitecture", Intel (**2000-01)**

Thank You

Any questions ?