C-DAC Four Days Technology Workshop

ON

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

hyPACK-2013
(Mode-12 : Multi-Core)

Lecture Topic:

Multi-Core Processors: MPI -2 Part-III (MPI 2.0 -Multi-threaded Prog. -Thread Safety & MPI 3.0 Efforts)

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

Part-III (MPI 2.0 -Multi-threaded Programming - Thread Safety & MPI 3.0 Efforts

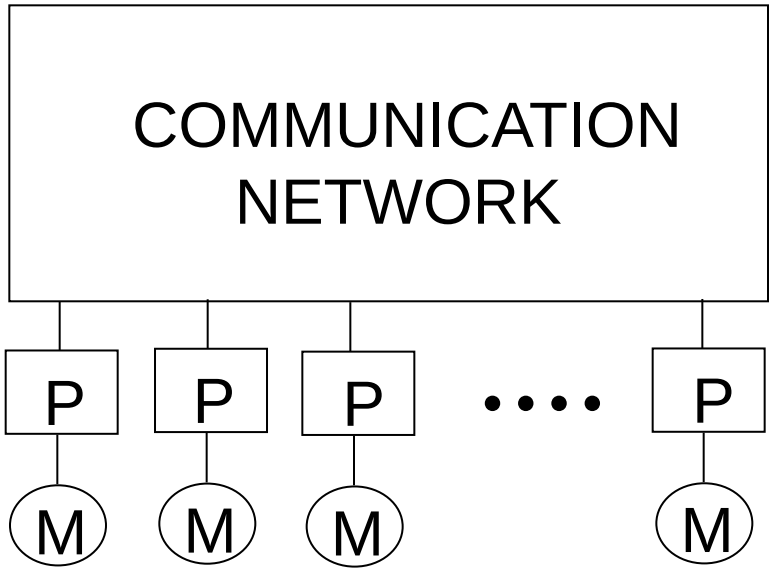
Quick overview of what this Lecture is all about

- Extensions to MPI
- MPI & Thread Safety
- MPI & Multi-threaded Programming
- MPI Library Calls
- MPI 3.0 Efforts

Source : Reference : [4], [6], [11],[12],[24],[25], [26], [36], [37], [38], [39], [40], [41]

Message Passing Architecture Model

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



- On most Parallel Systems, the processes involved in the execution of a parallel program are identified by a sequence of non-negative integers. If there are *p* processes executing a program, they will have ranks 0, 1,2, ……., *p-1.*

Information about MPI

Where to use MPI –2.0 ?

- You need a portable parallel program on Multi-Core Processors
- You are writing a parallel Library & You have irregular data relationships that do not fit a data parallel model

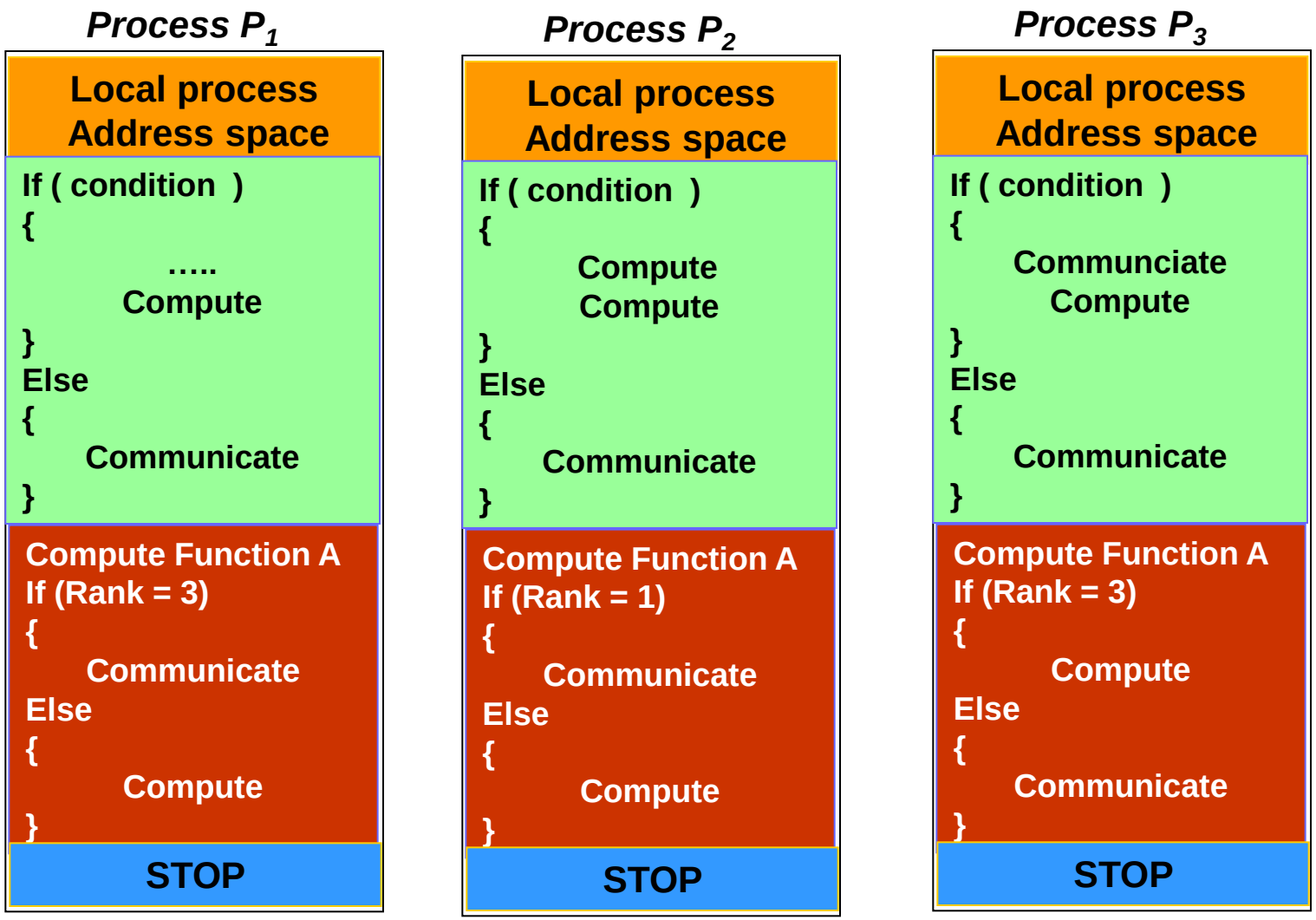
Why learn MPI-2 ?

- Portable & Expressive, Universal acceptance
- Good way to learn about subtle issues in parallel computing

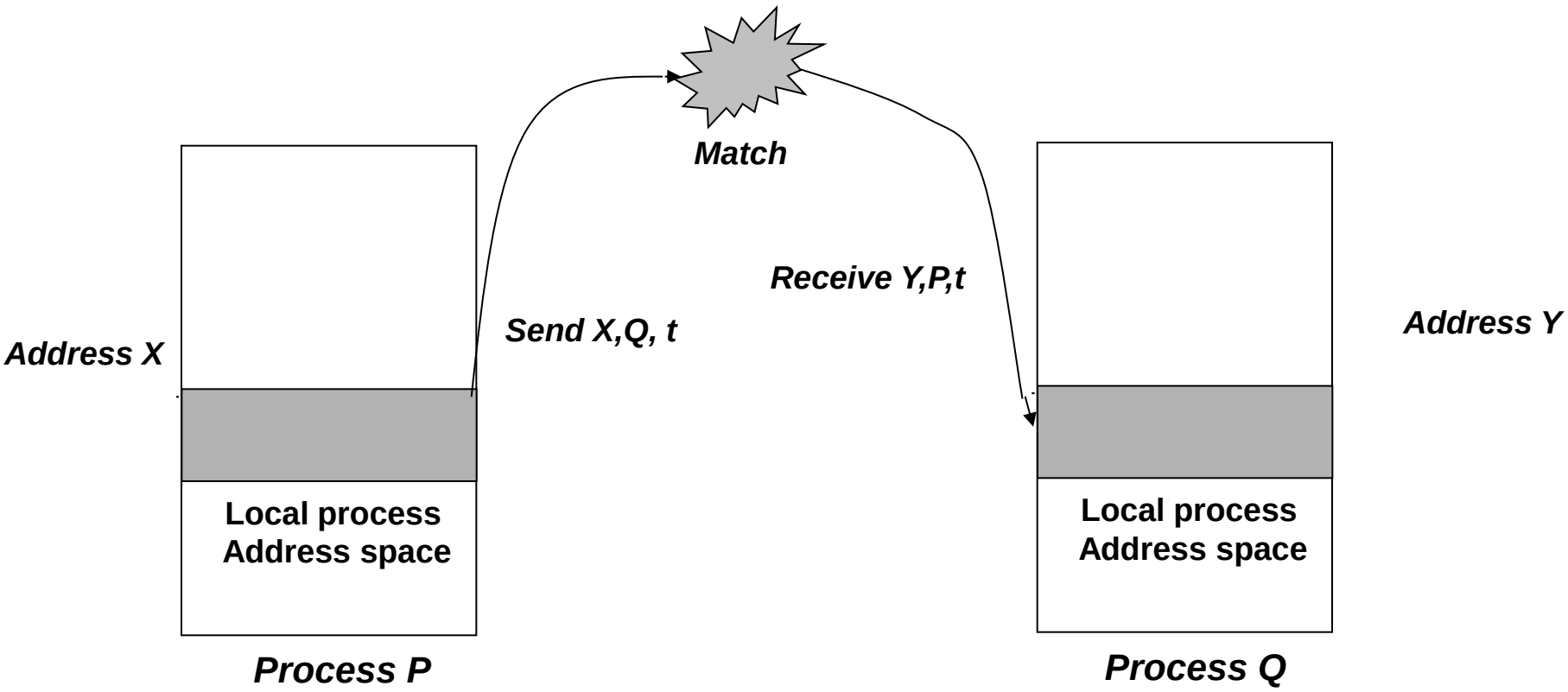
What is MPI-3 ?

- Multi-threaded Programming :Multi-core Processors
- Programming - Treating threads as MPI Processes

The Message Passing Abstraction



The Message Passing Abstraction



User-Level Send/receive message-passing abstraction : A data transfer from one local address space to another occurs when a *send* to particular processes matches with a *receive* posted by that process

MPI-2 Contents & MPI 3.0 Efforts

- Extensions to the message-passing model
  - Parallel I/O
  - One-sided operations
  - Dynamic process management
  - Thread Safety
- Making MPI more robust and convenient
  - C++ and Fortran 90 bindings
  - External interfaces, handlers
  - Extended collective operations
  - Language interoperability
  - MPI interaction with threads
- Making MPI - Multi-threaded Programming for Multi-core Processors
  - C++ and Fortran 90 bindings

Source : Reference : [4], [6], [11],[12],[24],[25], [26], [36], [37], [38], [39], [40], [41]

MPI-2 : Introduction to Thread Safety

- Thread Safety & Message Passing Library
  - Thread safety means that multiple threads can be executing Message Passing library calls without interfacing with one another
    - Thread unsafety occurs when when the message passing system is expected to hold certain parts of the process state.
    - It is impossible to hold certain parts of the process state and it is impossible to hold that process state for more than one thread at time.
  - POSIX Standard also known as Pthreads – widely used definition of threads

Source : Reference : [4], [6], [11],[12],[24],[25], [26], [36], [37], [38], [39], [40], [41]

MPI-2 : Introduction to Thread Safety

- Threads are becoming the parallel programming model for Multi-core processors and Shared Memory Machines
- Threads can be used in conjunction with Message Passing Library
- Threads can improve performance and reduce overheads in communication
- Threads provide a natural implementation of non- blocking communication operations
- Threads can increase the efficiency of the implementation of collective operations

Source : Reference : [4], [6], [11],[12],[24],[25], [26]

MPI-2 : Introduction to Thread Safety

- **Thread Safety & Message Passing Library**
  - Example : The concept of “the most recently received message” to avoid passing a status stored on the process’s stack
    - That is user code will look something like

```
recv(msg, type);
src = get_src();
len = get_len();
```

    - Single threaded case - Woks well
    - Multi-threaded case - several *receives* may be in progress simultaneously
    - When **`get_src`** is called, it ma not be clear for which message the source is supposed to be returned.
  - MPI provides thread safe implementations so that MPI can work hand to hand with thread libraries

Threads and Processes in MPI-2

- Thread systems where the operating system (the kernel) is not involved in managing the individual threads are called *user threads*.
  - *User threads* tend to be *faster* than kernel threads (User threads takes smaller time to switch between the threads within the same process)
  - Restriction : System calls will block all threads in the process containing the thread, made the system call.) Not just the calling thread)
- Difficult to write truly portable multithreaded programs (Application can not assume the the entire process will not be blocked when a thread calls a library routine)
- The POSIX thread (Pthreads) specification does not specify whether the threads are user or kernel; it it is upto threads implementation

Source : Reference : [4], [6], [11],[12],[24],[25], [26]

MPI-2.0 : Thread Safe Library

- Mixed-Model Programming - MPI for SMP Clusters
  - MPI for Multi core Processors

  - Thread safe - libmpi_mt.so
  - Non-thread-safe (Default) - libmpi.so

For programs that are not multi-threaded, the user should use libmpi.so whenever possible for maximum performance.

Reference: Sun MPI 3.0

Source : Reference : [4], [6], [11],[12],[24],[25], [26], [36], [37], [38], [39], [40], [41]

Threads and MPI in MPI-2

Thread Safety & MPI – Issues to be addressed

- Performance tradefoffs between multi-threaded and single-threaded code.
  - I/O operations
  - Against Inconsistent updates to the same memory location from different threads
  - Software locks and System locks are quite expensive
- Vendors sometimes provide single threaded /Multi-threaded libraries
  - Have I been linked with the right library ?
  - DO I suffer with occasional and mysterious errors

Source : Reference : [4], [6], [11],[12],[24],[25], [26]

MPI - Shared Memory Allocation

- Supports communication via shared memory between MPI Processes
  - Based on Message size (Short or Long Messages)
  - Compromise between Performance and Memory Use
  - Implementation varies as per Vendor Specification
  - Works for SMPs and NUMA based shared Memory Computer.

Source : Reference : [4], [6], [11],[12],[24],[25], [26], MPI-2 or SunMPI 3.0

MPI – Multithreaded Programming

- MPI-2 Specification - MPI & threads
  - Use thread-safe library (For ex : libmpi_mt.so in SUN MPI 3.0)
  - When two concurrently running threads make MPI calls, the outcome will be as if the calls executed in some order.
  - Blocking MPI calls will block the calling thread only. A blocked calling thread will not prevent progress of other runnable threads on the same process, nor will it prevent them from executing MPI calls.
  - Multiple sends and receives are concurrent

Reference: Sun MPI 3.0

Source : Reference : [36], [37], [38], [39], [40], [41]

MPI – Multithreaded Programming (Thread Safe)

- Each thread within an MPI process may issue MPI calls; however, threads are not separately addressable.
  - That is, the rank of a send or receive call identifies a process, not a thread, meaning that no order is defined for the case where two threads call
  - **`MPI_Recv`** with the same tag and communicator. Such threads are said to be in conflict.

- **Note :** Overheads in MPI implementation –
  - How to handle conflicts and Data Races ?
  - How to write thread Safe programs ?

MPI – Multithreaded Programming (Thread Safe)

- Each thread within an MPI process may issue MPI calls; however, threads are not separately addressable.
  - If threads within the same application post conflicting communication calls, data races will result.
    - You can prevent such data races by using distinct communicators or tags for each thread.
  - Prevention of data races and conflict - User can write thread safe programs

Source : Reference : [36], [37], [38], [39], [40], [41]

MPI – Multithreaded Programming (Thread Safe)

- Adhere to these guidelines:
  - You must not have an operation posted in one thread and then completed in another.
  - you must not have a request serviced by more than one thread.
  - A data type or communicator must not be freed by one thread while it is in use by another thread.
  - Once MPI_Finalize has been called, subsequent calls in any thread will fail.

MPI – Multithreaded Programming (Thread Safe)

- **Adhere to these guidelines:**

  - You must ensure that a sufficient number of lightweight processes (LWPs) are available for your multithreaded program.Failure to do so may degrade performance or even result in deadlock.

  - You cannot stub the thread calls in your multithreaded program by omitting the threads libraries in the link line.

  - The libmpi.so library automatically calls in the threads libraries, which effectively overrides any stubs.

Reference: Sun MPI 3.0

MPI – Multithreaded Programming (Thread Safe)

MPI Library Calls - Guidelines

- Provides specific guidelines that apply for specific some - routines - Collective calls and Communicator operations

- **MPI_Wait, MPI_Waitall, MPI_Waitany, MPI_Waitsome**

In a program where two or more threads call one of these routines, you must ensure that they are not waiting for the same request. Similarly, the same request cannot appear in the array of requests of multiple concurrent wait calls.

Source : Reference : 25], [26], [36], [37], [38], [39], [40], [41]

MPI – Multithreaded Programming (Thread Safe)

MPI Library Calls - Guidelines

- **MPI_Cancel**

  One thread must not cancel a request while that request is being serviced by another thread.

- **MPI_Probe, MPI_Iprobe**

  A call to MPI_Probe or MPI_Iprobe from one thread on a given communicator should not have a source rank and tags that match those of any other probes or receives on the same communicator. Otherwise, correct matching of message to probe call may not occur.

Source : Reference : 25], [26], [36], [37], [38], [39], [40], [41]

MPI – Multithreaded Programming (Thread Safe)

MPI Library Calls - Guidelines

❖ Collective Calls

- Collective calls are matched on a communicator according to the order in which the calls are issued at each processor.
- All the processes on a given communicator must make the same collective call.
- You can avoid the effects of this restriction on the threads on a given processor by using a different communicator for each thread.

Source : Reference : [4], [6], [11],[12],[24],[25], [26], MPI-2 or SunMPI 3.0

MPI – Multithreaded Programming (Thread Safe)

MPI Library Calls - Guidelines

❖ **Communicator Operations**

- Use the same or different communicators.
- threads in different processes participating in the same communicator operation require grouping
- Do not free a communicator in one thread if it is still being used by another thread.

Source : Reference : [4], [6], [11],[12],[24],[25], [26], [36], [37], [38], [39], [40], [41]

MPI – Multithreaded Programming (Thread Safe)

- Communicator Operations

  Each of the communicator functions operates simultaneously with each of the noncommunicator functions, regardless of what the parameters are and of whether the functions are on the same or different communicators. However, if you are using multiple instances of the same communicator function on the same communicator, where all parameters are the same, it cannot be determined which threads belong to which resultant communicator. Therefore, when concurrent threads issue such calls, you must assure that the calls are synchronized in such a way that threads in different processes participating in the same communicator operation are grouped. Do this either by using a different base communicator for each call or by making the calls in single-thread mode before actually using them within the separate threads. Do not free a communicator in one thread if it is still being used by another thread.

Reference: Sun MPI 3.0

Source : Reference : [4], [6], [11],[12],[24],[25], [26], [36], [37], [38], [39], [40], [41]

Threads and MPI in MPI-2

MPI Library Calls - Guidelines

- MPI-2 function to initialize

  - **int MPI_Init_thread**
    int *argc,char ***argv,int required, int *provided)
    (C-Binding)

  - **MPI_INIT_THREAD(required, provided, ierror)**
    Fortran binding

Regardless of of whether **MPI_Init** or **MPI_Init_thread** is called, the MPI program must end with a call to **MPI_finalize**

Source : Reference : [4], [6], [11],[12],[24],[25], [26], [36], [37], [38], [39], [40], [41]

MPI Library Calls - Guidelines

- MPI-2 specifies four levels of thread safety
  - MPI_THREAD_SINGLE: only one thread
  - MPI_THREAD_FUNNELED: only one thread that makes MPI calls
  - MPI_THREAD_SERIALIZED: only one thread at a time makes MPI calls
  - MPI_THREAD_MULTIPLE: any thread can make MPI calls at any time
- MPI_Init_thread(…, required, &provided) can be used instead of MPI_Init

Source : Reference : [4], [6], [11],[12],[24],[25], [26], [36], [37], [38], [39], [40], [41]

MPI 3.0 Efforts

- Increasing prevalence of multi- and many-core processors calls for extended MPI facilities for dealing with threads as first class MPI entities.
- This leads to issues like the Probe/Recv consistency issue in MPI
- Efforts seeks to introduce a powerful and convenient way of direct addressing of the threads as MPI processes.

MPI 3.0 Efforts

- Treating Threads as MPI Processes
- Dynamic Thread levels
- I/O threads
- Address Thread Locks & MPI

Source : Reference : MPI-3 or SunMPI 3.0

MPI 3.0 Thread Init/Finalize Routines

- **Problem :**
  - MPI currently does not explicitly know threads
    - Process can be mapped to different cores/SMTs
    - Thread scheduling is left to the OS

- **Relevant Issues:**
  - Explicit Thread Init/Finalize Routines
- Allow the process manager to perform intelligent mapping
- Optional calls - application does not necessarily have to call

MPI 3.0 - Dynamic Threads Levels

- Problem: MPI specifies thread-level support at Init time
  - Even if a small fraction of the code uses THREAD-MULTIPLE, the entire code is forced to go through locks
- Performance Impact (messaging rate)

- Efforts
  - Add calls for **MPI_Set_thread_level()** to dynamically change thread-level within the application

Source : Reference : Intel MPI, MPI-3, SunMPI 3.0, 36,37,38,39,40,41

MPI 3.0 - Dynamic Threads Levels

- MPI_Set_thread_level(int required, int* provided)
  - Hinting mechanism only

- Relevant Issues:
  - If an implementation allows the thread-level reduction, but not increase, the application might not be able to deal with it
  - Asynchronous Progress Threads

- Requires synchronization with the progress thread to change level
  - Collective Operations: Some MPI implementations use different collective operations based on the thread-level

Source : Reference : Intel MPI, MPI-3, SunMPI 3.0, 36,37,38,39,40,41

MPI treating Threads as MPI process

- A new collective routine MPI_Comm_thread_register() is introduced to create a communicator in which existing threads become MPI processes with unique ranks.
- The existing routine MPI_Comm_free() is extended to operate on the resulting communicators.
- Prerequisite: MPI_THREAD_MULTIPLE thread support level

Reference: Intel MPI

Source : Reference : Intel MPI, MPI-3, SunMPI 3.0, 36,37,38,39,40,41

MPI treating Threads as MPI process

MPI_comm_thread_register (basic language binding)

MPI_Comm_thread_register(comm, local_thread_index, local_num_threads, newcomm)

IN comm	original communicator
IN local_thread_index	index of the calling thread (0 to local_num_threads – 1) on the current MPI process in comm
IN local_num_threads	total number of threads issuing this call on the current MPI process in comm
OUT newcomm	new communicator based on threads

Reference: Intel MPI

Source : Reference : Intel MPI, MPI-3, SunMPI 3.0, 36,37,38,39,40,41

MPI treating Threads as MPI process

MPI_comm_thread_register (basic language binding)

C:

int MPI_Comm_thread_register(MPI_Comm comm, int local_thread_

Index, int_local_num_threads, MPI_Comm *newcomm)

Fortran:

MPI_COMM_THREAD_REGISTER(INTEFER COMM, INTEGER LOCAL_THREAD_INDEX, INTEGER LOCAL_NUM_THREADS, INTEGER NEWSOMM, INTEGER IERROR)

Reference: Intel MPI

Source : Reference : Intel MPI, MPI-3, 36,37,38, 40,41

MPI treating Threads as MPI process

Example: OpenMP parallel section

```
!$OMP parallel num_treads(4)
call MPI_COMM_THREAD_REGISTER(       &
MPI_COMM_WORLD, OMP_GET_THREAD_NUM(),
        & OMP_GET_NUM_THREADS(), NEWCOMM)
!
!       Whatever MPI operations on and in NEWCOMM
!
        call    MPI_COMM_FREE(NEWCOMM)
!$OMP parallel end
```

Source : Reference : Intel MPI, MPI-3, 36,37,38, 40,41

Summary

- MPI-2 provides how MPI interfaces with threads
- MPI-2 can deliver to libraries and applications portability across a diverse set of environments.
- MPI-3 provides major extensions to the original message-passing model targeted by MPI-1 &MPI-2
- Many Vendor efforts on Treating threads as MPI processes

Source : [4], [6], [11],[12],[24],[25], [26]
Intel MPI, MPI-3, SunMPI 3.0, 36,37,38,39,40,41

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)**

References

32. http://www.erc.msstate.edu/mpi/

33. http://www.arc.unm.edu/workshop/mpi/mpi.html

34. http://www.mcs.anl.gov/mpi/mpich

35. The MPI home page, with links to specifications for MPI-1 and MPI-2 standards : http://www.mpi–forum.org

36. Hybrid Programming Working Group Proposals, Argonne National Laboratory, Uchiacago (2007-2008)

37. TRAC Link : https://svn.mpi-forum.org/trac/mpi-form-web/wiki/MPI3Hybrid

38. Threads and MPI Software, Intel Software Products and Services 2008 - 2009

39. Sun MPI 3.0 Guide November 2007

40. Treating threads as MPI processes thru Registration/deregistration –Intel Software Products and Services 2008 - 2009

41. Intel MPI library 3.2 - http://www.hearne.com.au/products/Intelcluster/edition/mpi/663/

Thank You

Any questions ?