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 2.0 Overview Part-II

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

C-DAC Five Days Technology Workshop

ON

Heterogeneous Computing –
Many Core / Multi GPU
Performance Algorithms, Application Kernels

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

Lecture Topic:
Multi-Core Processors: MPI 2.0 Overview (Part-II)

Venue : CMSD, UoHYD ; Dates : Oct 17-21, 2012

Contents of MPI-2

- Extensions to the message-passing model
  - Parallel I/O
  - One-sided operations
  - Dynamic process management

- Making MPI more robust and convenient
  - C++ and Fortran 90 bindings
  - External interfaces, handlers
  - Extended collective operations
  - Language interoperability
  - MPI interaction with threads

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

MPI-2 An Overview

Remote Memory Access/Dynamic Process Management

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

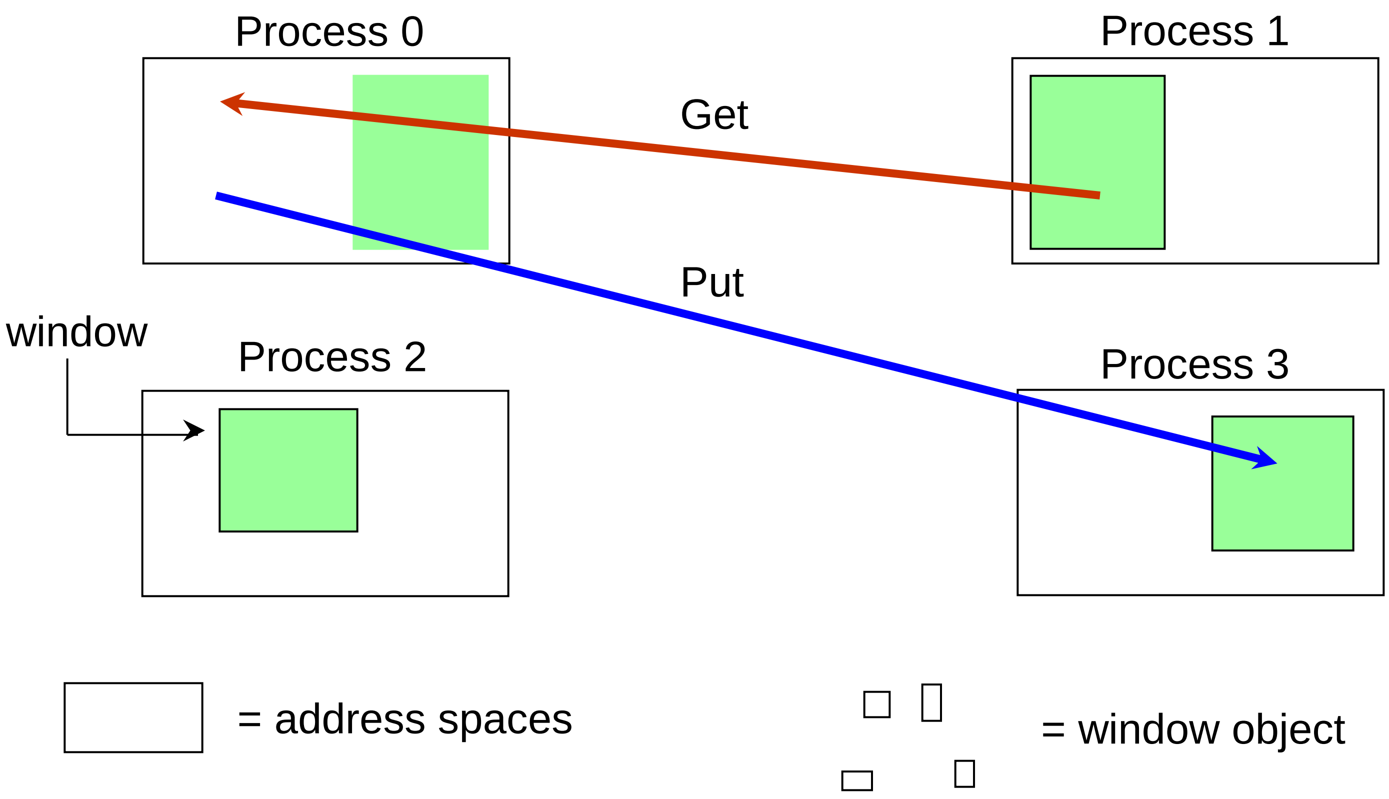
MPI-2: Introduction to Remote Memory Access

- Issues
- Windows
- One-sided operations
- Synchronization
- A simple example

MPI-2 : Introduction to Remote Memory Access

- Balancing efficiency and portability across a wide class of architectures
  - shared-memory multiprocessors
  - NUMA architectures
  - distributed-memory MPP’s, clusters
  - Workstation networks
- Retaining “look and feel” of MPI-1
- Dealing with subtle memory behavior issues: cache coherence, sequential consistency
- Synchronization is separate from data movement.

Remote Memory Access Windows and Window Objects



MPI-2 :Remote Memory Access Window Objects

`MPI_Win_create(base, size, disp_unit, info, comm, win)`

- Exposes memory given by `(base, size)` to RMA operations by other processes in `comm`.
- `win` is window object used in RMS operations.
- `Disp_unit` scales displacements:
  - 1 (no scaling) or `sizeof(type)`, where window is an array of elements of type `type`.
  - Allows use of array indices.
  - Allows heterogeneity

MPI-2 :One-Sided Communications Calls

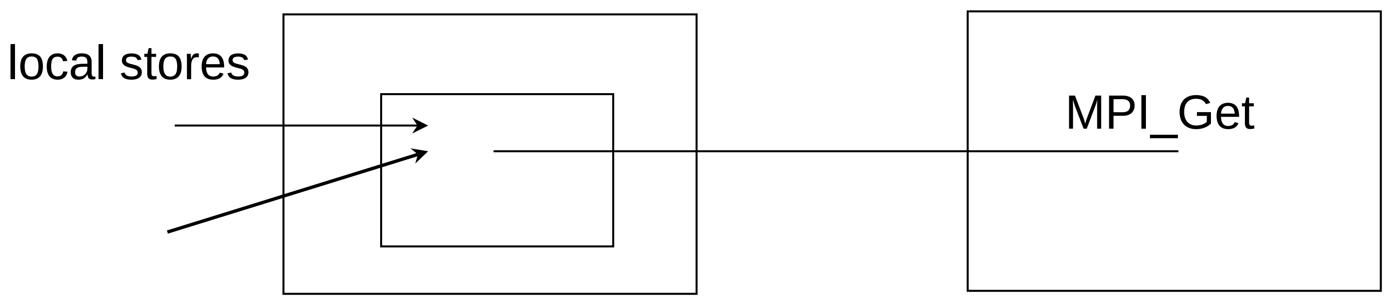
- `MPI_Put` – stores into remote memory
- `MPI_Get` – reads from remote memory
- `MPI_Accumulate` – updates remote memory
- All are non-blocking: data transfer is described, may be even initiated, but may continue after call returns.
- Subsequent synchronization on window object is needed to ensure operations are complete.

MPI-2 : Put, Get, and Accumulate

- `MPI_Put( origin_addr, origin_count, origin_datatype, target_addr, target_count, target_datatype, window)`
- `MPI_Get(...)`
- `MPI_Accumulate(..., op, ...)`
- `op` is an in `MPI_Reduce`, but no user-defined operations are allowed.

MPI-2 : The Synchronization Issue

- Which value is retrieved?



- Which value is retrieved?
- MPI provides multiple forms

MPI-2 : Synchronization

Multiple methods for synchronizing on window objects:

- `MPI_Win_fence` – like barrier, supports BSP model
- `MPI_Win_{start, complete, post, wait}` – for more scalable control, involves groups of processes
- `MPI_Win_{lock, unlock}` – provides part of remote-memory model.

MPI –2 : Remote Memory Access

- ❖ Separates data transfer from synchronization
- ❖ In message-passing, they are combined

Proc 0	Proc 1		Proc 0		Proc 1
store			fence		fence
send	receive		put		
	load		fence		fence
				or	Load
			store		
			fence		fence
					load

MPI-2 Advantages of Remote Memory Access Operations

- Can do multiple data transfer with a single synchronization operation
  - like BSP model
- Bypass tag matching
  - effectively precomputed as part of remote offset
- Significantly faster than send/receive on SGI Origin, for example, provided special memory is used.

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

MPI- 2: Cpi with Bcast/Reduce

```
n = 0;
while (!done) {
   if (myid == 0) {
        printf("Enter the number of intervals
   (0quits):");
        scanf("%d". &n);
   }
   MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
   if (n == 0)
        done = 1;
   else {
        h = 1.0 / (double) n;
        sum = 0.0;
        for (i = myid + 1; i <= n; i += numprocs) {
             x = h * ((double)i - 0.5);
             sum += f(x)
   }
   mypi = h * sum;
```

MPI- 2: Cpi with Bcast/Reduce

```
 MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE,
                        MPI_SUM, 0,
 MPI_COMM_WORLD);
    if (myid == 0) {
    printf(*pi is %.16f, Error is %.16f\n",
                      pi, fabs(pi - PI25DT));
       }
 }
 MPI_Finalize();
 return 0;
}
```

MPI-2 Cpi With One-sided Operations

```
int n, numprocs, muid, i;
double mypi, pi, h, sum, x;
MPI_WIN nwin, piwin;

MPI_Init(&argc,&argv);
MPI_Comm_site(MPI_COMM_WORLD,&numprocs);
MPI_Comm_rank(MPI_COMM_WORLD,&myid);
if (myid == o) {
  MPI_Win_create(&n, sizeof(int), 1, MPI_INFO_NULL,
                 MPI_COMM_WORLD, &nwin);
  MPI_Win_create(&pi,sizeof(double),1,MPI_INFO_NULL,
                 MPI_COMM_WORLD, &piwin);
  }
else{
  MPI_Win_create(MPI_BOTTOM, 0, 1, MPI_INFO_NULL,
                 MPI_COMM_WORLD, &win);
  MPI_Win_create(MPI_BOTTOM, 0, 1, MPI_INFO_NULL,
                 MPI_COMM_WORLD, &piwin);
}
```

MPI-2 One-sided cpi - 2

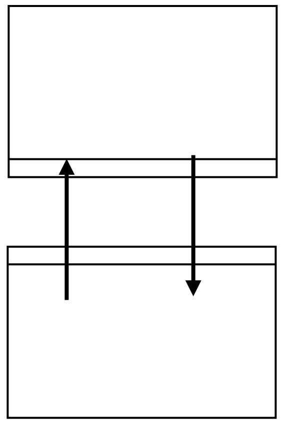
```
while (1) {
   if (myid ==) {
       printf("Enter the number of intervals: (0 quits)
   ");
       scanf("%d",&n);
       pi = 0.0;
   }
   MPI_Win_fence(0, nwin);
   if (myid ! = 0)
       MPI_Get(&n, 1, MPI_INT, 0, 0, 1, MPI_INT, nwin);
   MPI_Win_fence(0, nwin);
   if (n == 0)
       break;
   else
       h = 1.0 / (double) n;
       sum = 0.0;
       for (i = myid + 1; i <=n; i += numprocs) {
       x = h * ((double)i - 0.5);
```

MPI-2 One-sided cpi - 3

```
        sum += (4.0 / (1.0 + x*x));
    }
    mypi = h * sum;
    MPI_Win_fence( 0, piwin);
    MPI_Accumulate(&mympi, 1, MPI_DOUBLE, 0, 0, 1,
                   MPI_DOUBLE, MPI_SUM, piwin);
    MPI_Win_fence(0, piwin);
    if (myid == 0)
        printf("pi is %.16f, Error is %.16f\n",
               pi, fabs(pi - PI25DT));
    }
  }
  MPI_Win_free(&nwin);
  MPI_Win_free(&piwin);
  MPI_Finalize();
  return 0;
```

MPI-2 RMA Advanced Topics

- Exchanging boundary data
- Allocating special memory for one-sided operations
- Alternate synchronization methods
- Fetch and increment operation
- datatypes



Introduction to Dynamic Process Management in MPI

- Standard way of starting processes in PVM
- Not so necessary in MPI
- Useful in assembling complex distributed applications

Dynamic Process Management

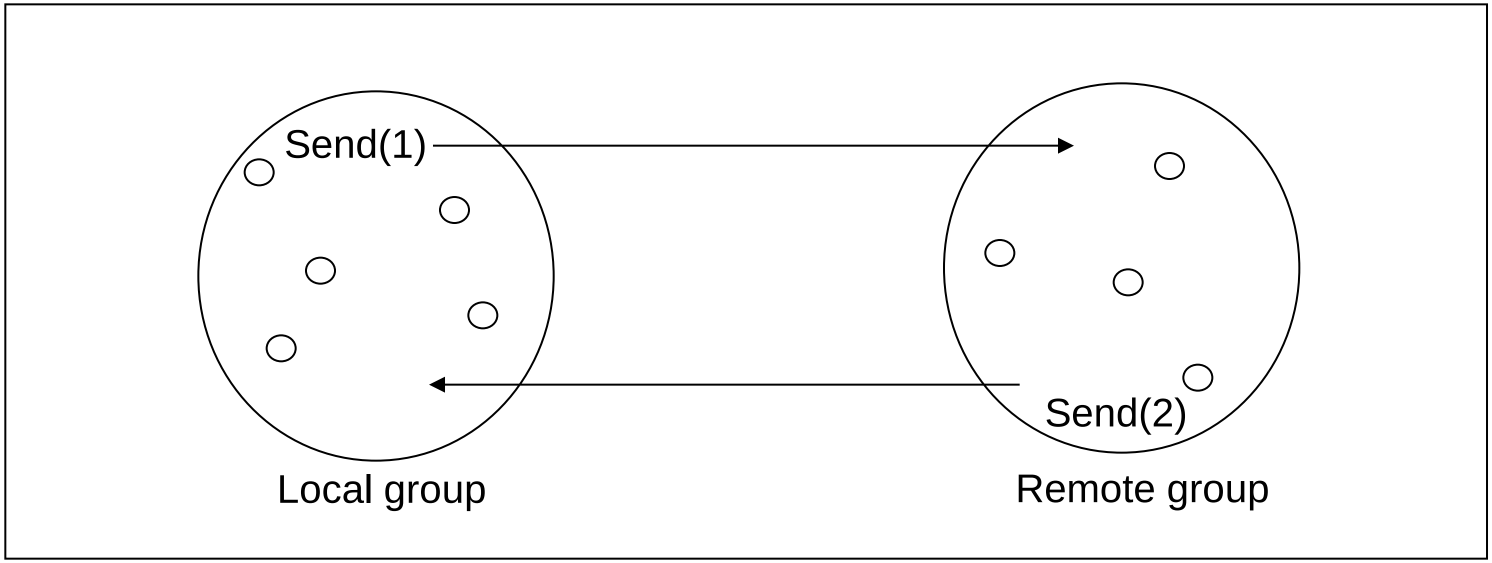
- Issues
  - maintaining simplicity, flexibility, and correctness
  - Interaction with operating system, resource manager, and process manager
  - connecting independently started processes
- Spawning new processes is collective, returning an intercommunicator.
  - Local group is group of spawning processes.
  - Remote group is group of new processes.
  - New processes have own `MPI_COMM_WORLD`
  - `MPI_Comm_get_parent` lets new processes find parent communicator

Intercommunicators

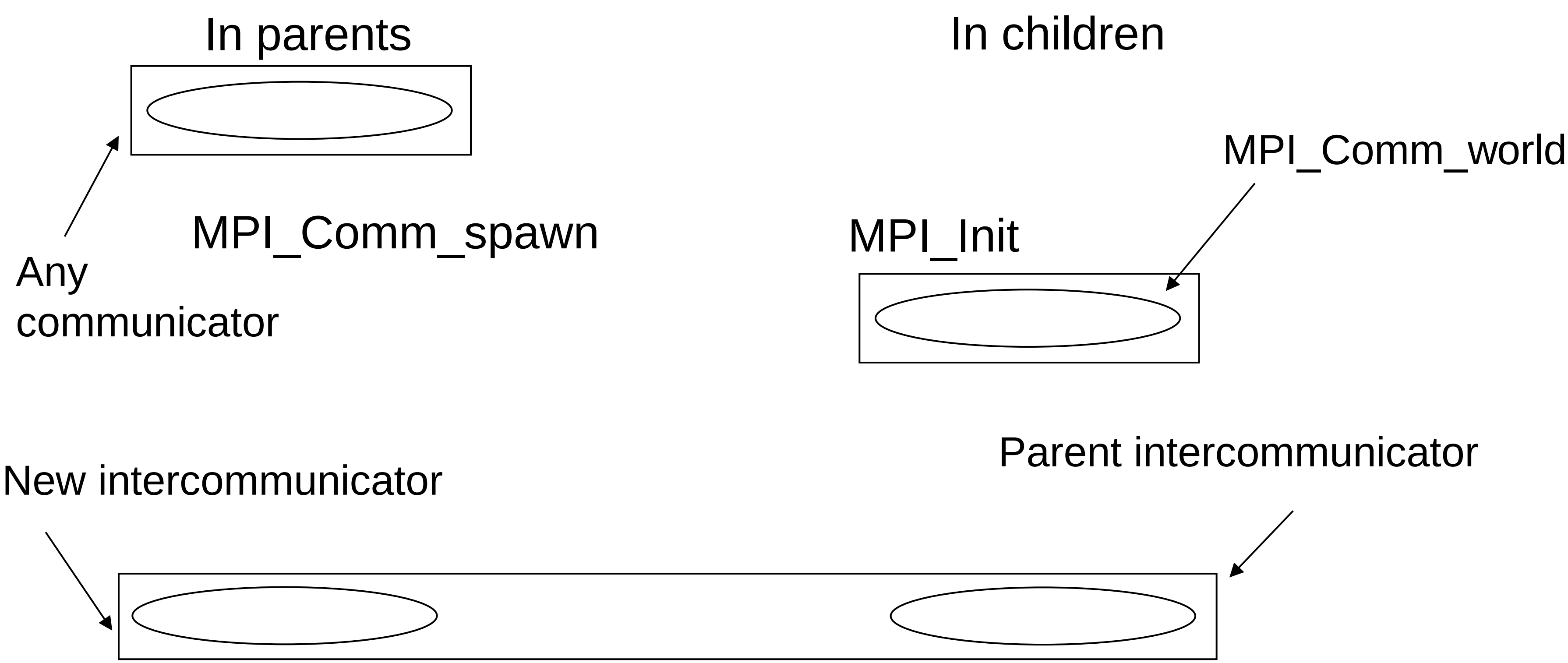
- Contain a local group and a remote group
- Point-to-point communication is between a process in one group and a process in the other.
- Can be merged into a normal (intra) communicator
- Created by `MPI_Intercomm_create` in MPI-1.
- Play a more important role in MPI-2, created in multiple ways.

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

MPI Inter-communicators



Spawning New Processes



Spawning Processes

`MPI_Comm_spawn(command, argv, numprocs, info, root, comm, intercomm, errcodes)`

- Tries to start `numprocs` process running `command`,
- The operation is collective over `comm`.
- Spawnees are in remote group of `intercomm`.
- Errors are reported on a per-process basis in `errcodes`.
- `Info` used to optionally specify hostname, archname wdir, path, file, softness.

Spawning Multiple Executables

- `MPI_Comm_spawn_multiple (...)`
- Arguments `command`, `argv`, `numprocs`, `info` all become arrays,.
- Still collective

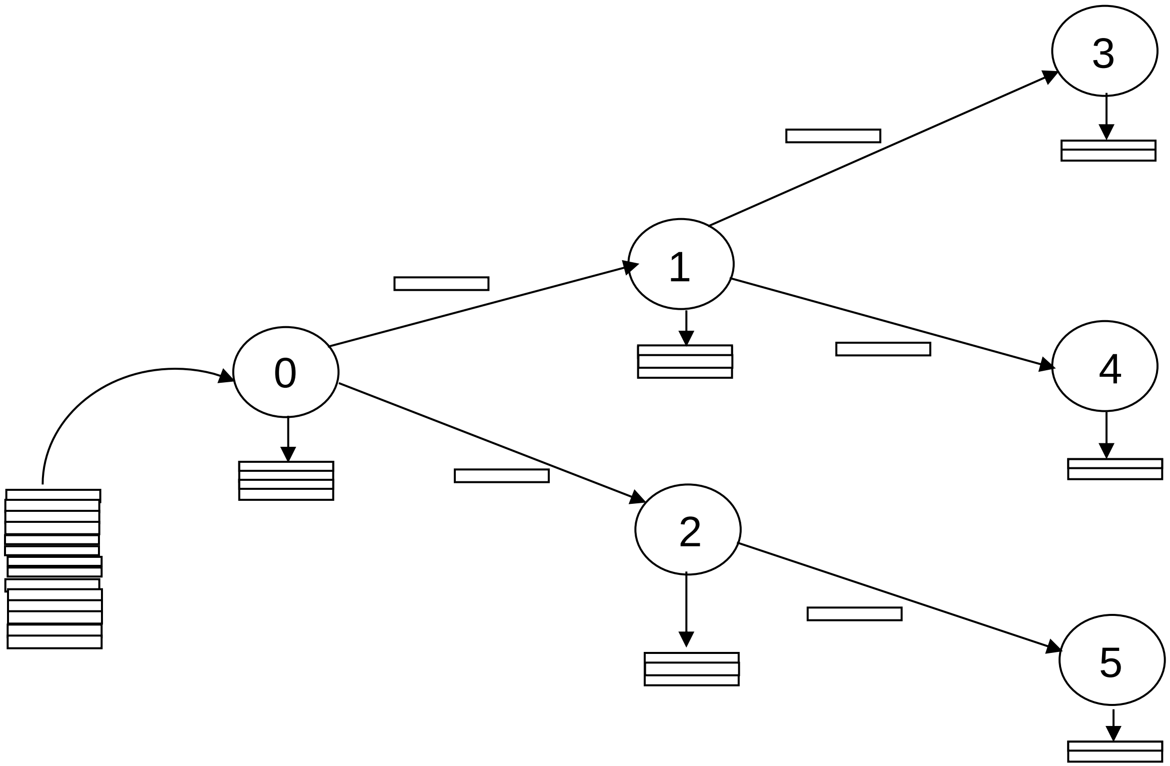
In the Children

- `MPI_Init` (only MPI Programs can be spawned)
- `MPI_COMM_WORLD` is processes spawned with one call to MPI_Comm_spawn.
- `MPI_Comm_get_parent` obtains parent intercommunicator.
  - same as intercommunicator returned by `MPI_Comm_spawn` in parents.
  - Remote group is spawners.
  - Local group is those spawned.

Manager-Worker Example

- Single manager process decides how many workers to create and which executable they should run.
- Manager spawns n workers, and addresses them as 0, 1, 2, …, n-1, in view intercomm.
- Workers address each other as 0, 1, …n-1 in `MPI_COMM_WORLD`, address manager as 0 in parent intercomm.
- One can find out how many processes can usefully be spawned (MPI_UNIVERSE_SIZE attribute of MPI_COMM_WORLD)

Parallel File Copy



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

Parallelism in Distributing Files

- All processes writing in parallel
- Message-passing in parallel (assume scalable implementation of MPI_Bcast)
- Pipeline parallelism with blocks from the file
- Use syntax adopted from cp:

```
Pcp 0-63 /home/progs/cpi /tmp/cpi
```

More Info on `Info`

- An extensible list of key=value pairs
- Used in I/O, One-sided, and Dynamics to package variable, optional types of arguments that may not be standard
- Example use in Fortran, for MPI_Spawn:

```
Numslaves = 10
call MPI_INFO_CREATE( spawninfo, ierr)
call MPI_INFO_SET( spawinfo, 'host', 'home.mcs.anl.gov',ierr)
call MPI_INFO_SET( spawinfo, 'path', '/homekosh/progs',ierr)
call MPI_INFO_SET( spawinfo, 'wdir', '/homekosh/tmp',ierr)
call MPI_COMM_SPAWN( 'slave, MPI_ARGV_NULL, numslaves, &
                     spawinfo, 0, MPI_COMM_WORLD,     &
                     slavecomm, MPI_ERRCODES_IGNORE, ierr)
Call MPI_INFO_FREE( spawinfo, ierr )
```

Code for pcp Master - 1

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#define BUFSIZE   256*1024
#define CMDSIZE   80
int main( int argc, char *argv[] )
{
  int num_hosts, mystatus, allstatus,
  done,numread;
  int infd, outfd;
  char      outfilename(MAXPATHLEN),
  controlmsg[CMDSIZE];
  char buf[BUFSIZE];
  char      soft_limit[20];
  MPI_Info hostinfo;
  MPI_Comm pcpslaves, all_procs;
  MPI_Init( &argc, &argv );
```

pcp Master - 2

```
makehostlise( argv[1], “targets”, &num_hosts );

MPI_Info_create( &hostinfo );
MPI_Info_set( hostinfo, “file”, “targets” );
sprintf( soft_limit, “0:%d”, num_hosts );
MPI-Info_set( hostinfo, “soft”, soft_limit );
MPI_Comm_spawn( “pcp_slave”, MPI_ARGV_NULL,
  num_hosts,
            hostinfo, 0, MPI_COMM_SELF,
  &pcpslaves,
            MPI_ERRCODES_IGNORE );
MPI_Info_free( &hostinfo );
MPI_Intercomm_merge( pcpslaves, 0, &all_procs );
```

pcp Master - 3

```
strcpy( outfilename, argv[3] );
if ( (infd = open( argv[2], o_RDONLY ) ) == -1 ) {
     fprintf( stderr, “%s doesn’t exist\n”, argv[2] );
     sprintf( controlmsg, “exit );
     MPI_Bcast( controlmsg, CMDSIZE, MPI_CHAR, 0,
               all_procs );
     MPI_Finalize();
     return -1 ;
}
else {
     sprintf( controlmsg, “ready” );
     MPI_Bcast( controlmsg, CMDSIZE, MPI_CHAR, 0,
               all_procs );
}
```

pcp Master - 4

```
MPI_Bcast( outfilename, MAXPATHLEN, MPI_CHAR, 0,
         all_procs );
if ( (oufd = open( outfilename,
     O_CREAT|O_WRONLY|O_TRUNC, S_IRWXU ) ) == -1 )
  mystatus = -1;
else
  mystatus = 0;
MPI_Allreduce( &mystatus, &allstatus, 1, MPI_INT,
             MPI_MIN, all_procs );
if ( allstatus( stderr, “Output file %s not opened\n”,
             outfilename );
  MPI_Finalize();
  return 1;
}
```

pcp Master - 5

```
done = 0;
while (!done) {
   numread = read( infd, buf, BUFSIZE );
   MPI_Bcase( &numread, 1, MPI_INT, 0, all_procs );
   if (numread > 0 ) {
       MPI_Bcast) buf, numread, MPI_BYTE, 0,
   all_procs );
       write( outfd, buf, numread );
   }
   else {
       close( outfd );
       done = 1;
   }
}
MPI_Comm_free( &pcpslaves );
MPI_Comm_free( &allprocesses );
MPI_Finalize();
return );
}
```

pcp Slave - 1

```
#include “mpi.h”
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#define BUFSIZE       256*1024
#define CMDSIZE       80
int main( int argc, char *argv[] )
{
   int          mystatus, allstatus, done,numread;
   char         outfilename(MAXPATHLEN), controlmsg[CMDSIZE];
   int          outfd;
   char         buf[BUFSIZE];

   MPI_Comm slavecomm, all_processes;

   MPI_Init( &argc, &argv );

   MPI_Comm_get_parent( &slavecomm );
```

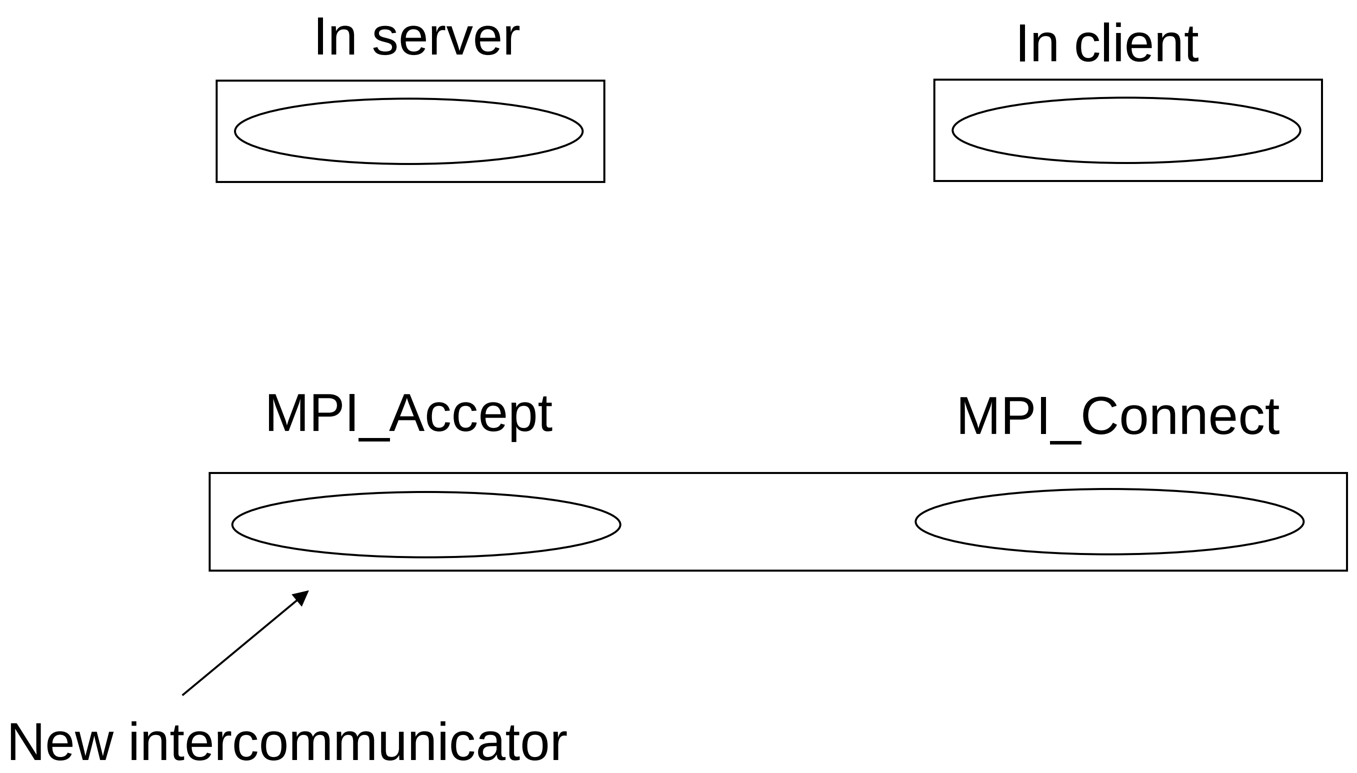
pcp Slave - 3

```
/* at this point all files have been successfully opened */
   done = 0;
   while ( !done ) {
        MPI_Bcast( &numread, 1, MPI_INT, 0, a;;_rpocesses );
        if ( numread > 0) {
            MPI_Bcast( buf, numread, MPI_BYTE, 0, all_procs );
            write( outfd, buf, numread );
        }
        else {
            close( outfd );
            done = 1;
        }
   }
   MPI_Comm_free( &slavescomm );
   MPI_Comm_free( &all_processes );
   MPI_Finalize();
   return 0;
}
```

Establishing Connections

- Two sets of MPI processes may wish to establish connections, e.g.,
  - Two parts of an application started separately.
  - A visualization tool wishes to attach to an application.
  - A server wishes to accept connections from multiple clients.
  - Both server and client may be parallel programs.
- Establishing connections is collective but asymmetric (“Client/Server”).
- Connection results in an intercommunicator.

Establishing Connections Between Parallel Programs



Connecting Processes

- Server:
  - `MPI_Open_port( info, port_name )`
    - System supplies port_name
    - Might be host:num; might be low-level switch#
  - `MPI_Comm_accept( port_name, info, root, comm, intercomm )`
    - Collective over comm
    - Returns intercomm; remote group is clients
- Client:
  - `MPI_Comm_connect( port_name, info root, comm, intercomm )`
    - Remote group is server

Extended Collective Operations

- In MPI-1, collective operations are restricted to ordinary (intra) communicators.
- In MPI-2, most collective operations apply also to intercommunicators, with appropriately different semantics.
- E.g, Bcast/Reduce in the intercommunicator resulting from spawning new processes goes from/to root in spawning processes to/from the spawned processes.
- In-place extensions

External Interfaces

- Purpose: to ease extending MPI by layering new functionality portably and efficiently
- Aids integrated tools (debuggers, performance analyzers)
- In general, provides portable access to parts of MPI implementation internals.
- Already being used in layering I/O part of MPI on multiple MPI implementations.

Components of MPI External Interface Specification

- Generalized requests
  - Users can create custom non-blocking operations with an interface similar to MPI's.
  - MPI_Waitall can wait on combination of built-in and user-defined operations.
- Naming objects
  - Set/Get name on communicators, datatypes, windows.
- Adding error classes and codes
- Datatype decoding
- Specification for thread-compliant MPI

Threads and MPI in MPI-2

- 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]

Language Interoperability

- Singleton MPI_Init
- Passing MPI Objects between languages
- Constant values, error handlers
- Sending in one language; receiving in another
- Addresses
- Datatypes
- Reduce operations

Summary

- MPI-2 provides major extensions to the original message-passing model targeted by MPI-1
- MPI-2 can deliver to libraries and applications portability across a diverse set of environments.

Source : Reference : [4], [6], [11],[12],[24],[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 ?