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-I)

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

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],[25], [26]

Introduction to Parallel I/O in MPI-2 - Outline

- Why do I/O in MPI?
- Non-parallel I/O from an MPI program
- Non-MPI parallel I/O to shared file with MPI I/O
- parallel I/O to shared file with MPI I/O
- Fortran-90 version
- Reading a file with a different number of processes
- C++ version
- Survey of advanced features in MPI I/O

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

Why MPI is a Good Setting for Parallel I/O

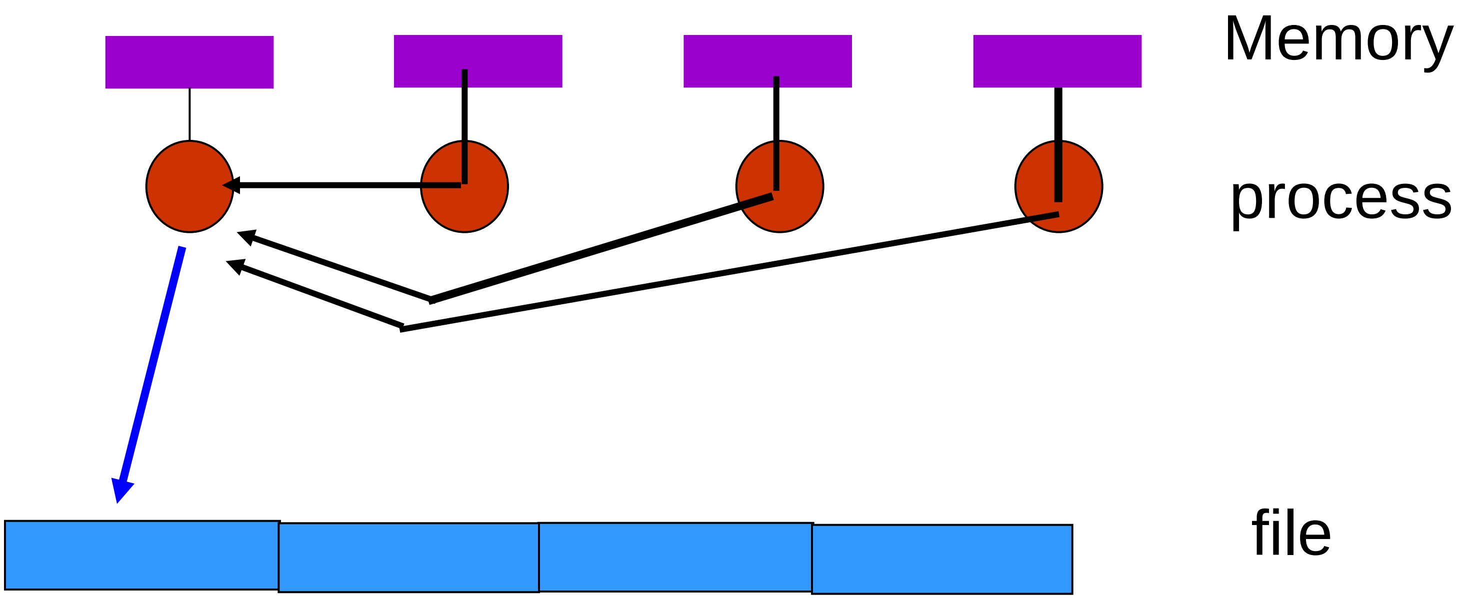
- Writing is like sending and reading is like receiving
- Any parallel I/O system will need:
  - collective operations
  - user-defined datatypes to describe both memory and file layout
  - communicators to separate application-level message passing from I/O-related message passing
  - non-blocking operations
- I.e., lots of MPI-like machinery

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

Introduction to Parallel I/O

- First example: non-parallel I/O (Sequential) from an MPI program



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

Next few examples have:

```
#include "mpi.h"
#include <stdio.h>
#define BUFSIZE 1000
int main(int argc, char *argv[])
{
  int I, myrank, numprocs, buf[BUFSIZE];

  MPI_Init (&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  MPI_Comm_size (MPI_COMM_WORLD, &numprocs);
  ....
  ....
  MPI_Finalize();
  Return 0;
}
```

Non-parallel I/O from MPI Program

```
MPI_Status status;
FILE *myfile;
for (i=0; i<BUFSIZE; i++)
   buf[i] = murank * BUFSIZE + i;
if (myrank ! =0)
   MPI-Send(buf, BUFSIZE, MPI_INT, 0, 99,
            MPI_COMM_WORLD);
else{
   myfile = fopen (“testfile”, “w);
   fwrite(buf, sizeof(int), BUFSIZE, myfile);
   for (i=1, i<numprocs; i++) {
       MPI_Recv(buf, BUFSIZE, MPI-INT, i, 99,
                MPI_COMM_WORLD, &status);
       fwrite(buf, sizeof(int), BUFSIZE, myfile);
   }
   fclose(myfile);
}
```

Pros and Cons of Sequential I/O

- Pros:
  - parallel machine may support I/O from only one process
  - I/O libraries (e.g. HDF-4, SILO, etc.) not parallel
  - resulting single file is handy for `ftp, mv`
  - big blocks improve performance
  - short distance from original, serial code
- Cons:
  - lack of parallelism limits scalability, performance

Sequential Versions of UNIX I/O

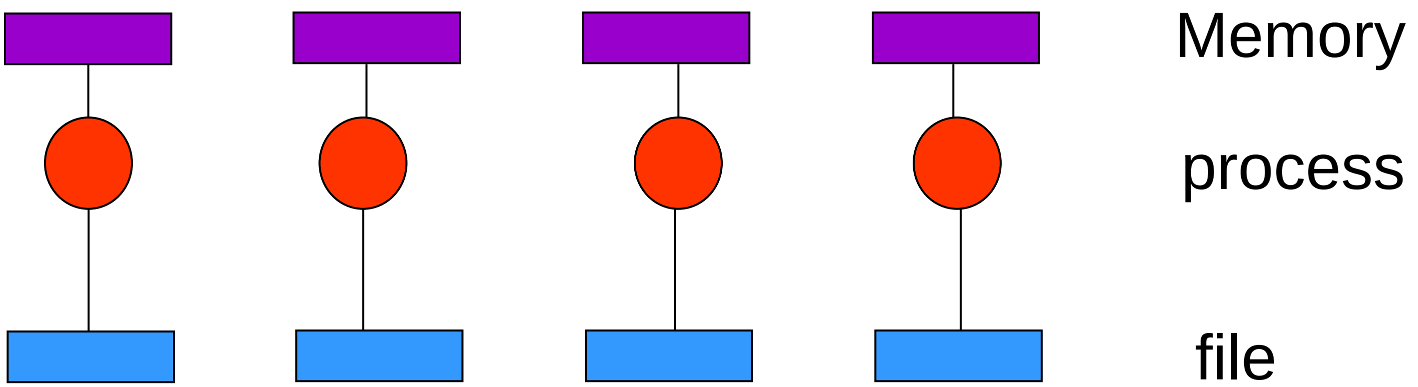
Unix

```
FILE myfile;
myfile =
fopen(...)


fread(...)
fwrite(...)


fclose
```

Memory Allocation on Multiple threads

Non-MPI Parallel I/O

- Each process writes to a separate file



- Pro: parallelism
- Con: lots of small files to manage

Non-MPI Parallel I/O

```
char filename[128];
FILE *myfile;

for (i=0; i<BUFSIZE; i++)
  buf[i] = myrank * BUFSIZE + i;

sprintf(filename, "testfile.%", myrank);
myfile = fopen(filename, "w");
fwrite(buf, sizeof(int), BUFSIZE, myfile);
fclose(myfile);
```

- **Pro:** parallelism
- **Con:** Individual processes may find their data to be in small contiguous chunk, many I/O operations with smaller data.

MPI I/O to Separate Files

- Same pattern as previous example
- MPI I/O replaces Unix I/O in a straightforward way
- Easy way to start with MPI I/O
- Does not exploit advantages of MPI I/O
  - parallel machine may support I/O from only one process
  - I/O libraries need not be parallel
- Note files cannot be read conveniently by a different number of processes

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

MPI I/O to Separate Files

```
char filename[128];
MPI_FILE myfile;

for (i=0; i<BUFSIZE; i++)
  buf[i] = myrank * BUFSIZE + i;
sprintf(filename, "testfile.%", murank);
MPI_File_open(MPI_COMM_SELF, filename,
            MPI_MODE_WRONLY | MPI_MODE_CREATE,
            MPI_INFO_NULL, &myfile);
MPI_File_write(myfile, buf, BUFSIZE, MPI_INT,
             MPI_STATUS_IGNORE);
MPI_File_close(&myfile);
```

MPI Versions of UNIX I/O

Unix

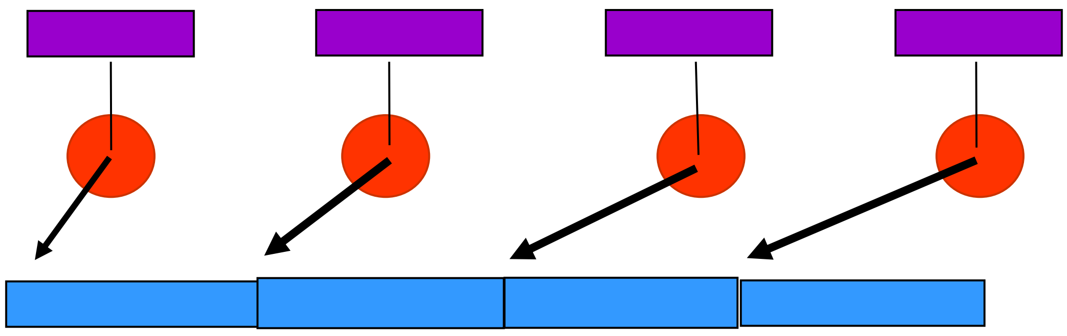
```
FILE myfile;
myfile =
fopen(...)



fread(...)
fwrite(...)


fclose
```

MPI

```
MPI_File myfile;
MPI_File_open(...)
  takes info, comm
  args

MPI_File_read/write(...)
  take(addr, count,datatype)


MPI_File_close
```

MPI Parallel I/O to Single File

❖ Processes write to shared file



- **`MPI_File_set_view`** assigns regions of the file to separate processes

MPI Parallel I/O to Single File

```
MPI_FILE thefile;

for (i=0; i<BUFSIZE; i++)
                    buf[i] = myrank * BUFSIZE + i;
MPI_File_open(MPI_COMM_WORLD, “testfile”,
                           MPI_MODE_CREATE |
                    MPI_MODE_WRONLY,
                           MPI_INFO_NULL, &thefile);
MPI_File_set_view(thefile, myrank * BUFSIZE * sizeof(int),
                               MPI_INT, MPI_INT, “native”,
                               MPI_INFO_NULL);
MPI_File_write(thefile, buf, BUFSIZE, MPI_INT,
                            MPI_STATUS_IGNORE);
MPI_File_close(&thefile);
```

MPI_File_set_view

- Describes that part of the file accessed by a single MPI process.
- Arguments to `MPI_File_set_view:`
  - `MPI_File file`
  - `MPI_offset disp`
  - `MPI_Datatype etype`
  - `MPI_Datatype filetype`
  - `char *datarep`
  - `MPI_Info info`

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

Fortran Issues

- “Fortran” now means Fortran-90 (or –95).
- MPI can’t take advantage of some new Fortran features, e.g. array sections.
- Some MPI features are incompatible with Fortran-90.
  - e.g., communication operations with different types for first argument, assumptions about argument copying.
- MPI-2 provides “basic” and “extended Fortran support.

Using MPI with Fortran

- Basic support:
  - The file **`mpi.h`** must be valid in both fixed-and free-form format.
  - includes some new functions using parameterized types

- Extended support (new in **`MPI-2`**)
  - **`mpi`** module
    - allows function prototypes
    - catches common errors at compile time
      - Status
      - ierr

MPI I/O Fortran

```
PROGRAM main
use mpi

integer ierr, i, myrank, BUFSIZE, thefile
parameter (BUFSIZE=100)
integer buf(BUFSIZE)
integer(kind=MPI_OFFSET_KIND) disp

call MPI_INIT(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD,myrank,
  ierr)
do i = 0, BUFSIZE
  buf(i) = myrank * BUFSIZE + i
Enddo

* in F77, see implementation notes (might be
  integer*8)
```

MPI I/O in Fortran contd.

```
call MPI_File_open(MPI_COMM_WORLD, “testfile”,
  &
           MPI_MODE_WRONLY | MPI_MODE_CREATE,
  &
           MPI_INFO_NULL, thefile, ierr);
call MPI_TYPE_SIZE(MPI_INTEGER, intsize)
disp = myrank * BUFSIZE * Intsize
call MPI_FILE_SET_VIEW(thefile, disp,
  MPI_INTEGER, &
               MPI_INTEGER, ‘native’, &
               MPI_INFO_NULL, ierr);
call MPI_FILE_WRITE(thefile, buf, BUFSIZE,
  MPI_INTEGER, &
            MPI_STATUS_IGNORE, ierr);
call MPI_FILE_CLOSE(thefile, ierr);
call MPI_FINALIZE(ierr)
END PROGRAM main
```

C++ Bindings

- C++ binding alternatives:
  - use C bindings
  - Class library (e.g., OOMPI)
  - “minimal” binding
- Chose “minimal” approach
- Most MPI functions are member functions of MPI classes:
  - example: MPI::COMM_WORLD.send(...)
- Others are in MPI namespace if namespaces are supported, otherwise in MPI class
  - problem with MPI::SEEK_SET
- C++ bindings for both MPI-1 and MPI-2

C++ Version

```
// example of parallel MPI read from single file
#include <iostream.h>
#include “mpi.h”

int main(int argc, char *argv[])
{
  int bufsize, *buf, count;
  char filename[128];
  MPI::Status status;

  MPI::Init();
  int myrabj = MPI::COMM_WORLD.Get_rank();
  int numprocs = MPI::COMM_WORLD.Get_size();
  MPI::FILE thefile =
  MPI::File::Open(MPI::COMM_WORLD,“testfile”,
                           MPI::MODE_RDONLY
                           MPI::INFO_NULL);
```

C++ Version, Part 2

```
MPI::offset filesize = thefile.Get_size();
filesize    = filesize / sizeof(int);
bufsize     = filesize / numprocs + 1;
buf = new int[bufsize];
Thefile.Set_view(myrank * bufsize *
   sizeof(int),
              MPI_INT, MPI_INT, "native",
              MPI::INFO_NULL);
thefile.Read(buf, bufsize, MPI_INT, &status);
count = status.Get_count(MPI_INT);
count << "process" <<myrank << "read" <<count
     << "ints" << endl;
thefile.Close();
delete [] buf;
MPI::Finalize();
return 0;
}
```

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

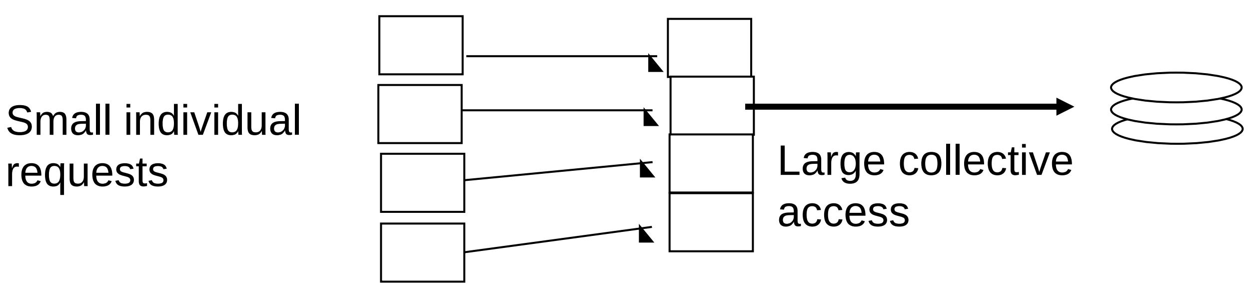
Other Ways to Write to a Shared File

- `MPI_File_seek` — like Unix seek
- `MPI_File_read_at` — combine seek and I/O
- `MPI_File_write_at` — for thread safety
- `MPI_File_read_shared`
- `MPI_File_write_shared` — use shared file pointer

- Collective operations

Collective I/O in MPI

- A critical operation in parallel I/O
- Allows communication of “big picture” to file system
- Framework for 2-phase I/O, in which communication precedes I/O (can use MPI machinery)
- Basic idea: build large blocks, so that reads/writes in I/O system will be large



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],[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 ?