C-DAC Four Days Technology Workshop

ON

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

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

Lecture Topic:

Multi-Core Processors : Mixed Mode Prog: MPI and OpenMP

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

Advance Features of OpenMP

Lecture Outline

Following Topics will be discussed

- Example programs using different OpenMP Pragmas for SPMD and MPMD programs
- Mixing of MPI and OpenMP
- Key factors That impact Performance and Performance Tuning Methodology
- Comparative features of Shared & Distributed Memory Programming Paradigms

OpenMP : Contents

- OpenMP's constructs fall into 5 categories:
  - Parallel Regions
  - Work sharing
  - Data Environment
  - Synchronization
  - Runtime functions/environment variables
- OpenMP is basically the same between Fortran and C/C++

OpenMP : Key Points

- Need OpenMP-complaint compiler

- **OpenMP** consists of a rich-set of pragmas, environment variables and a runtime API for threading

- The environment variables and APIs should be used sparingly because they can affect performance detrimentally.

- OpenMP automatically uses an appropriate number of threads for the target system.

OpenMP : Environment Variables

- Control how “omp for schedule(RUNTIME)” loop iterations are scheduled.
  - OMP_SCHEDULE “schedule[, chunk_size]”
- Set the default number of threads to use.
  - OMP_NUM_THREADS int_literal
- Can the program use a different number of threads in each parallel region?
  - OMP_DYNAMIC TRUE || FALSE
- Do you want nested parallel regions to create new teams of threads, or do you want them to be serialized?
  - OMP_NESTED TRUE || FALSE

OpenMP : Environment Variables

- Environment variables are not propagated by mpirun, so you may need to explicitly set the requested number of threads with OMP_NUM_THREADS().
- OpenMP is:
  - A great way to write parallel code for shared memory machines.
  - A very simple approach to parallel programming.
  - Your gateway to special, painful errors (race conditions).
- OpenMP impacts clusters:
  - Mixing MPI and OpenMP.
  - Distributed shared memory.

Is MPI Large or Small?

Is MPI Large or Small? (use MPI-2.0 on Multi Cores)

- MPI is large (125 Functions)
  - MPI's extensive functionality requires many functions
  - Number of functions not necessarily a measure of complexity
- MPI is small (6 Functions)
  - Many parallel programs can be written with just 6 basic functions
- MPI is just **right** candidate for message passing
  - One can access flexibility when it is required
  - One need not master all parts of MPI to use it

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.

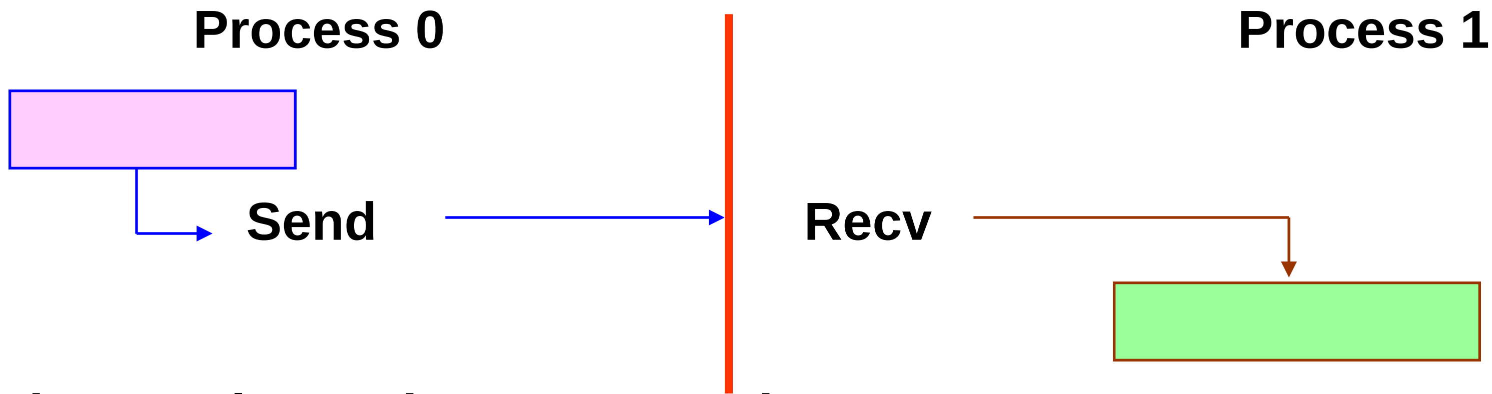
  MPI_Send

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

  MPI_Recv

MPI Send and Receive

Blocking Sending and Receiving messages



Fundamental questions answered

- To whom is data sent?
- What is sent? How does the receiver identify it?

Other Modes of Communication : Non-Blocking

MPI Communication Modes

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.

MPI Collective Communications

(Contd…)

Characteristics of Collective Communication Operations

- Collective action over a communicator
- All processes must communicate
- Synchronization may or may not occur
- All collective operations are blocking. (Refer MPI-2 Enhancements)
- No tags.
- Receive buffers must be exactly the right size

MPI Collective Communications

Type	Routine	Functionality
Data Movement	MPI_Bcast	One-to-all, Identical Message
	MPI_Gather	All-to-One, Personalized messages
	MPI_Gatherv	A generalization of MPI_Gather
	MPI_Allgather	A generalization of MPI_Gather
	MPI_Allgatherv	A generalization of MPI_Allgather
	MPI_Scatter	One-to-all Personalized messages
	MPI_Scatterv	A generalization of MPI_Scatter
	MPI_Alltoall	All-to-All, personalized message
	MPI_Scatterv	A generalization of MPI_Alltoall

MPI Collective Communications

Type	Routine	Functionality
Aggregation	MPI_Reduce MPI_Allreduce MPI_Reduce_scatter MPI_Scan	All-to-one reduction, All-to-One, A generalization of MPI_Reduce A generalization of MPI_Reduce All-to-all parallel prefix

Synchronization	MPI_Barrier MPI_Scatterv	Barrier Synchronization

OpenMP : Mixing OpenMP and MPI

- OpenMP and MPI coexist by default:
  - MPI will distribute work across processes, and these processes may be threaded.
  - OpenMP will create multiple threads to run a job on Multi Core Systems.

- But be careful … it can get tricky:
  - Messages are sent to a process on a system not to a particular thread.
  - Make sure you implementation of MPI is threadsafe.

OpenMP : Dangerous mixing of OpenMP and MPI

- The following will work on some MPI implementations, but may fail for others: MPI libraries are not always thread safe.

```
MPI_Comm_Rank(MPI_COMM_WORLD, &mpi_id) ;
#pragma omp parallel
{
    int tag, swap_neigh, stat, omp_id = omp_thread_num();
    long buffer [BUFF_SIZE], incoming [BUFF_SIZE];
    big_ugly_calc1(omp_id, mpi_id, buffer);
                                                        // Finds MPI id and tag so
    neighbor(omp_id, mpi_id, &swap_neigh, &tag);// messages don't conflict
    MPI_Send (buffer,   BUFF_SIZE, MPI_LONG, swap_neigh,
              tag, MPI_COMM_WORLD);
    MPI_Recv (incoming, buffer_count, MPI_LONG, swap_neigh,
              tag,  MPI_COMM_WORLD, &stat);
    big_ugly_calc2(omp_id, mpi_id, incoming, buffer);
#pragma critical
   consume(buffer, omp_id, mpi_id);}
```

OpenMP : Messages and Threads

- Keep message passing and threaded sections of your program separate:
  - Setup message passing outside OpenMP regions
  - Surround with appropriate directives (e.g. critical section or master)
  - For certain applications depending on how it is designed it may not matter which thread handles a message.
    - Beware of race conditions though if two threads are probing on the same message and then racing to receive it.

OpenMP : Safe mixing of OpenMP and MPI

- Put MPI in sequential regions

```
MPI_Init(&argc, &argv) ; MPI_Comm_Rank(MPI_COMM_WORLD,&mpi_id) ;
// a whole bunch of initializations
#pragma omp parallel for
for (I=0;I<N;I++) {
    U[I] =  big_calc(I);
}
    MPI_Send (U,   BUFF_SIZE, MPI_DOUBLE, swap_neigh,
                tag, MPI_COMM_WORLD);
    MPI_Recv (incoming, buffer_count, MPI_DOUBLE, swap_neigh,
                tag,  MPI_COMM_WORLD, &stat);
#pragma omp parallel for
for (I=0;I<N;I++) {
    U[I] =  other_big_calc(I, incoming);
}
consume(U, mpi_id);
```

OpenMP : Safe mixing of OpenMP and MPI

```
MPI_Init(&argc,&argv) ; MPI_Comm_Rank(MPI_COMM_WORLD,&mpi_id);
// a whole bunch of initializations
#pragma omp parallel
{
#pragma omp for
    for (I=0;I<N;I++)    U[I] =  big_calc(I);
#pragma master
{
 MPI_Send(U, BUFF_SIZE,MPI_DOUBLE,neigh, tag, MPI_COMM_WORLD);
 MPI_Recv(incoming,count,MPI_DOUBLE,neigh, tag, MPI_COMM_WORLD, &stat);
}
#pragma omp barrier
#pragma omp for
    for (I=0;I<N;I++)   U[I] =  other_big_calc(I, incoming);
#pragma omp master
    consume(U, mpi_id);
}
```

Protect MPI calls inside a parallel region

MPI-OpenMP Example

Each process prints Hello World along with its process rank and thread identifier(id)

Header file for MPI

```
include mpif.h
Call MPI_INIT(ierror)
Call MPI_COMM_SIZE(MPI_COMM_WORLD,Numprocs,ierror)
Call MPI_COMM_RANK(MPI_COMM_WORLD,MyRank,ierror)

Call OMP_SET_NUM_THREADS(4)
!$OMP PARALLEL PRIVATE(threadid)
threadid=OMP_GET_THREAD_NUM()
Print *,"Hello World from Process ",MyRank,"Thread",threadid
!$OMP END PARALLEL

Call MPI_FINALIZE(ierror)
```

Shared and Distributed Memory Comparison

Feature	Shared Memory	Distributed Memory
Ability to parallelize small parts of an application at a time	Relatively easy to do. Reward versus effort varies widely	Relatively difficult to do. Tends to require more of an all-or-nothing effort.
Feasibility of scaling an application to a large number of processors	Currently, few vendors provide scalable shared memory systems (e.g., ccNUMA systems)	Most vendors provide the ability to cluster nonshared memory systems with moderate to high-performance interconnects

Shared and Distributed Memory Comparison

(Contd..)

Feature	Shared Memory	Distributed Memory
Additional complexity over serial code (to be addressed by programmer)	Simple parallel algorithms are easy and fast to implement. Implementation of highly scalable complex algorithms is supported.	Significant additional overhead and complexity even for implementing simple and localized parallel constructs.

Shared and Distributed Memory Comparison

(Contd..)

Feature	Shared Memory	Distributed Memory
Impact on code quantity (e.g., amount of additional code required) and code quality (e.g., the read-ability of the parallel code)	Typically requires a small increase in code size (2-25%) depending on extent of changes required for parallel scalability. Code readability requires some knowledge of shared memory constructs, but is otherwise maintained as directives embedded within serial code	Tends to require extra copying of data into temporary message buffers, resulting in a significant amount of message handling code. Developer is typically faced with extra code complexity even in non-performance-critical code segments. Readability of code suffers accordingly.

Shared and Distributed Memory Comparison

(Contd..)

Feature	Shared Memory	Distributed Memory
Availability of application development and debugging environments	Requires a special compiler and a runtime library that supports OpenMP. Well-written code will compile and run correctly on one processor without an OpenMP compiler. Single memory address space simplifies development and support of a right debugger functionality.	Does not require a special compiler. Only a library for the target computer is required, and these are generally available. Debuggers are more difficult to implement because a direct, global view of all program memory is not available.

Conclusions

- Different OpenMP Constructs on Parallel Regions; Work sharing; Data Environment ; Synchronization; Runtime functions and environment variables have been discussed
- Example programs using different OpenMP Pragmas for SPMD and Non-SPMD programs
- Mixing of MPI and OpenMP and thread Safety issues are important for producing correct results
- Performance on Shared Memory machines /Multi Cores for applications involving structured computations is excellent

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 ?