# 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 : Shared Memory Prog:

## OpenMP Part-II

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

# OpenMP Prog. : Parallel Regions

# pragma omp parallel [Clause list]

/* Structured block */

- Conditional Parallelization :The Clause **`If(Scalar expression)`** determines whether the parallel construct in creation of threads. Only one if clause can be used with a parallel directive.

- Degree of Concurrency : The Clause **`num_threads (integer expression)`** specifies the number of threads that are created by the **`parallel`** directive

- Data Handling : The Clause **`private (variable list)`** indicates hat the set of variable specified is local to each thread ~ i.e each thread ha sits own copy of each variable in the list.

# Advance Features of OpenMP

## Lecture Outline :

- OpenMP Constructs Parallel Regions
  - Data Environment
  - Work sharing
  - Runtime functions/environment variables
- Example programs using different OpenMP Pragmas
- Key factors That impact Performance and Performance Tuning Methodology

Source : Reference : [4], [6], [14],[17], ]22], [28]

# OpenMP : Contents

- OpenMP's constructs

  - Data Environment
  - Work Sharing
  - Runtime functions/environment variables

- OpenMP is basically the same between Fortran and C/C++

# OpenMP : Data Environment - Changing storage attributes

- One can selectively change storage attributes constructs using the following clauses*
  - SHARED
  - PRIVATE
  - FIRSTPRIVATE
  - THREADPRIVATE

All the clauses on this page only apply to the *lexical extent* of the OpenMP construct.

- The value of a private inside a parallel loop can be transmitted to a global value outside the loop with:
  - LASTPRIVATE
- The default status can be modified with:
  - DEFAULT (PRIVATE | SHARED | NONE)

All data clauses apply to parallel regions and work sharing constructs except "shared" which only applies to parallel regions.

# OpenMP : Thread private

- Makes global data private to a thread
  - Fortran: COMMON blocks
  - C: File scope and static variables

- Different from making them PRIVATE

  - with PRIVATE global variables are masked.
  - THREADPRIVATE preserves global scope within each thread

- Threadprivate variables can be initialized using COPY-IN CLAUSE

# OpenMP : Thread private Example

Consider two different parallel regions in a code.

```
#include <omp.h>
 int a, b, i, tid; float x;
#pragma omp threadprivate(a, x)

main () {
/* Explicitly turn off dynamic threads */
omp_set_dynamic(0);
 printf("1st Parallel Region:\n");
#pragma omp parallel private(b,tid) {
 tid = omp_get_thread_num();
a = tid;
b = tid;
 x = 1.1 * tid +1.0;
printf("Thread %d: a,b,x= %d %d %f\n",tid,a,b,x); } /* end of parallel section */
printf("Master thread doing serial work here\n");
printf("2nd Parallel Region:\n");
 #pragma omp parallel private(tid) {
 tid = omp_get_thread_num();
printf("Thread %d: a,b,x= %d %d %f\n",tid,a,b,x); }
 /* end of parallel section */ }
```

Because of the threadprivate construct, each thread executing these routines has its own copy of the common block /buf/.

# OpenMP : Thread private Example

Output of the ThreadPrivate example code

```
1st Parallel Region:
 Thread 0: a,b,x= 0 0 1.000000
 Thread 2: a,b,x= 2 2 3.200000
 Thread 3: a,b,x= 3 3 4.300000
 Thread 1: a,b,x= 1 1 2.100000
************************************
Master thread doing serial work here
************************************
2nd Parallel Region:
Thread 0: a,b,x= 0 0 1.000000
Thread 3: a,b,x= 3 0 4.300000
Thread 1: a,b,x= 1 0 2.100000
Thread 2: a,b,x= 2 0 3.200000
```

# OpenMP : Constructs

Contd…

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

# OpenMP : Work-sharing Construct

**OpenMP defines the following work-sharing constructs**.

- ❖ **for** directive
- ❖ **sections** directive
- ❖ **single** directive

# OpenMP Prog. : Parallel Regions

Format of **for** directive

A sequence of **for** directive

- **private (variable list)**
- **firstprivate (Variable List)**
- **Lastprivate**
- **Reduction, Schedule,**
- **nowait, & ordered**

Source : Reference : [4], [6], [14],[17], ]22], [28]

# OpenMP : For/do construct – The schedule clause

❖ The schedule clause effects how loop iterations are mapped onto threads

➢ schedule(static [,chunk])

- Deal-out blocks of iterations of size “chunk” to each thread.

➢ schedule(dynamic[,chunk])

- Each thread grabs “chunk” iterations off a queue until all iterations have been handled.

Source : Reference : [4], [6], [14],[17], ]22], [28]

## OpenMP : For/do construct – The schedule clause

- The schedule clause effects how loop iterations are mapped onto threads
  - schedule(guided[,chunk])
    - Threads dynamically grab blocks of iterations. The size of the block starts large and shrinks down to size “chunk” as the calculation proceeds.
  - schedule(runtime)
    - Schedule  and chunk size taken from the OMP_SCHEDULE environment variable.

# OpenMP* API : The schedule clause

| Schedule Clause | When To Use |
| --- | --- |
| STATIC | Predictable and similar work per iteration |
| DYNAMIC | Unpredictable, highly variable work per iteration |
| GUIDED | Special case of dynamic to reduce scheduling overhead |
| RUNTIME | Uses the OMP_SCHEDULE environment variable at runtime to specify for usage. |

Source : Reference : [4], [6], [14],[17], ]22], [28]

# Example : DO / for Directive

❖ Simple vector-add program

- Arrays A, B, C, and variable N will be shared by all threads.

- Variable I will be private to each thread; each thread will have its own unique copy.

- The iterations of the loop will be distributed dynamically in CHUNK sized pieces.

- Threads will not synchronize upon completing their individual pieces of work (NOWAIT).

# C / C++ – for Directive Example

```
#include <omp.h>
#define CHUNK 100
#define  N          1000
main ()
{
int i, n, chunk;
float a[N], b[N], c[N];
/* Some initializations */
for (i=0; i < N; i++)
        a[i] = b[i] = i * 1.0;
n = N; chunk = CHUNK;
#pragma omp parallel shared(a,b,c,n,chunk) private(i)
        {
        #pragma omp for schedule(dynamic,chunk) nowait
        for (i=0; i < n; i++)
                c[i] = a[i] + b[i];
        } /* end of parallel section */
}
```

# OpenMP : Constructs

Contd…

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

# Environment Variables in OpenMP

- OMP_NUM_THREADS
- OMP_DYNAMIC
- OMP_NESTED
- OMP_SCHEDULE

# OpenMP : Environment Variables

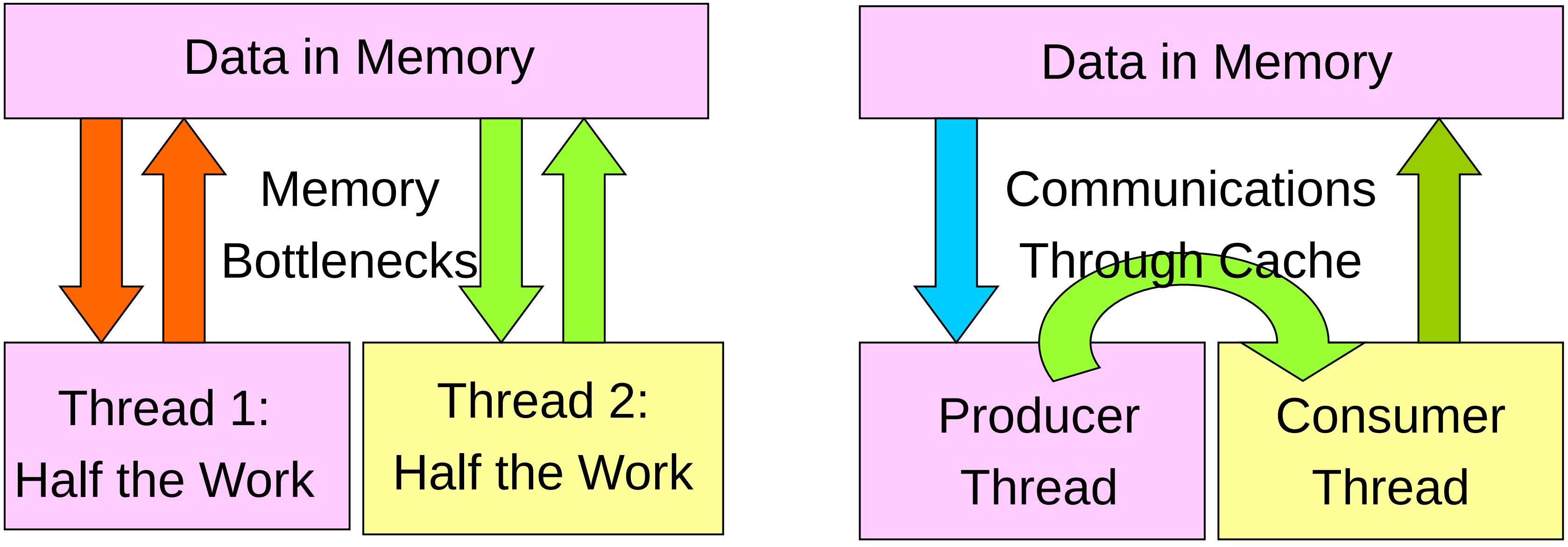
- Control how “omp for schedule(RUNTIME)” loop iterations are scheduled.
  - OMP_SCHEDULE “schedule[, chunk_size]”
- Set the default number of threads to use.
  - OMP_NUM_THREADS int_literal
- Can the program use a different number of threads in each parallel region?
  - OMP_DYNAMIC TRUE || FALSE
- Do 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 and Summary

- Environment variables are not propagated, 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.

# Producer/Consumer Problem : Synchronizing Issues

- ❖ Producer thread generates tasks and inserts it into a work-queue.
- ❖ The consumer thread extracts tasks from the task-queue and executes them one at a time.




Source : Reference : [4], [6], [14],[17], ]22], [28]

# Producer/Consumer Problem : Synchronizing Issues

- **Possibilities & Implementation Issues on Multi cores**
  - The producer thread must not overwrite the shared buffer when the previous task has not been picked up by a consumer thread
  - The consumer threads must not pick-up tasks until there is something present in the shared data structure.
  - Individual consumer threads should pick-up tasks one at a time.
- Implementation can be done mutexes, condition Variables

# Producer & Consumer : Critical Directive

- ❖ Producer thread generates task and inserts it into a task-queue.
- ❖ The consume thread extracts tasks from the queue and executes them one at a time.
  - ➢ There is concurrent access to the task-queue, these accesses must be serialized using critical blocks.
  - ➢ The tasks of inserting and extracting from the task-queue must be serialized.
  - ➢ Define your own "insert_into_queue" and "extract_from_queue" from queue (Note that queue full & queue empty conditions must be explicitly handled)

**Producer & Consumer ; Critical Directive**

```
#pragma omp parallel sections
{
 #pragma parallel section
   {
    /*producer thread */
    tasks =  produce_task();
    #pragma omp critical (task_queue);
      {
      insert_into_queue(task);
      }
   }
#pragma parallel section
   {
    /*Consumer thread */
    tasks =  produce_task();
    #pragma omp critical (task_queue);
      {
      task = extract_from_queue(task);
      }
      consume_task(task);
   }
}
```

# Producer & Consumer : Critical Directive

- Critical Section directive is a direct application of the corresponding mutex function in Pthreads
- Reduce the size of the critical section in Pthreads/OpenMP to get better performance ( Remember that critical section represents serialization points in the program)
- Critical section consists simply of an update to a single memory location.
- Safeguard : Define Structured Block I.e. no jumps are permitted into or out of the block. This leads to the threads wait indefinitely.

## Conclusions

- Simple to use OpenMP on Shared Memory machines
- 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

# 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 ?*