C-DAC Four Days Technology Workshop

ON

Hybrid Computing – Co-Processors/Accelerators
Power-aware Computing – Performance of Applications Kernels

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

Lecture Topic:

Multi-Core Processors : Shared Memory Prog: Pthreads 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 : Shared Memory Prog: Pthreads Part-II

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

Lecture Outline

Following Topics will be discussed :

- What is Thread ?
- What are Pthreads?
- Pthread APIs on Different OS
- Compilation, Linking and Execution of Pthread Programs
- Example codes using Pthreads

Source : Reference [4],[6], [7]

What is Thread ?

What is Thread ?

- A thread is a discrete sequence of related instructions that is executed independently of other instruction sequences.
- It is an entity that can be scheduled by an operating system to run independently.

Threading APIs for Windows

Microsoft Windows using C /C++ languages

- Win 32 / Microsoft Foundation Class Library (**MFC**) wrapped Windows **API** functionality in C++ classes
  - Provides Developers with C/C++ interface for developing windows applications
- Performance Issues – Concept of Virtual Machine op-codes & Overhead Minimization
- Performance Issues – run in a Managed runtime environment
- Legacy Application Support

Source : Reference [4],[6], [7]

Threading APIs for Windows

Microsoft Windows using C /C++ languages

- Creating Threads
  - CreateThread();
- Terminate the Thread
  - ExitThread();
- Managing Threads
- Thread Communication using Windows events
- Thread Synchronization
- Thread Atomic Operations
- Thread Pools; Thread Priority & Thread Affinity

Source : Reference [4],[6], [7]

Threading APIs for Microsoft .NET Framework

- Provide common execution environment for all the major languages : C++ & Visual Basic; C#
  - ThreadStart() – Constructs a new thread
- Microsoft .NET framework Class Library – provides examples of the APIs
- Managing Threads
- Thread Synchronization
- Thread Atomic Operations
- Thread Pools; Thread Affinity
- Thread Priority - .Net framework supports five levels thread priority

Source : Reference [4],[6], [7]

POSIX Threads

What are Pthreads?

- POSIX threads or Pthreads is a portable threading library which provides consistent programming interface across multiple operating systems.
- It is set of C language programming types and procedure calls,implemented with pthread.h file and a thread library.
- Set of threading interfaces developed by IEEE committee in charge of specifying a portable OS Interface.
- Library that has standardized functions for using threads across different platform.

Pthread APIs

- Pthread APIs can be informally grouped into three major classes:
  1. Thread Management
     - thread creation,joining,setting attributes etc.
  2. Thread Synchronization
     - functions that deal with Mutex
  3. Condition Variables
     - functions that deal with condition variables

Pthread APIs

- All identifiers in the thread library begins with **pthread_**

Routine Prefix	Functional Group
pthread_	Threads themselves and misc subroutines
pthread_attr_	Thread Attribute objects
pthread_mutex_	Mutex related routines
pthread_cond_	Condition variable related

Pthread APIs

❖ Thread Management Routines :

pthread_create():

Syntax: pthread_create(thread,attr,start_routine,arg)

where,
thread - an unique identifier for the new thread
attr – an attribute object that is used to set thread attributes
start_routine- the C routine that the thread will execute once created
arg- an argument that may be passed to start_routine.

Pthread APIs

❖ **pthread_exit():**

Syntax: pthread_exit(void *value_ptr)

- is used to terminate a thread. It is called after thread has completed its work and is no longer required to exist.

❖ **pthread_join():**

Syntax: pthread_join(threadId , status)

Blocks the calling thread until specified threadId thread terminates.

Pthread APIs

pthread_detach():

Syntax:

pthread_detach(pthread_t thread_to_detach)

- is used to detach the thread from other threads when it has no need to interact with them.

Pthread APIs

❖ Thread Synchronization Routines :

pthread_mutex_init():

Syntax -

pthread_mutex_init(mutex , attr)

- initializes the mutex and sets its attributes

❖ **pthread_mutex_destroy():**

Syntax -

pthread_mutex_destroy(mutex)

- destroy the mutex

Pthread APIs

- **pthread_mutex_lock():**

  Syntax - : pthread_mutex_lock(mutex)

  - Locks a mutex.

  - If the mutex is already locked, the calling thread blocks until the mutex becomes available.

- **pthread_mutex_trylock():**

  Syntax - : pthread_mutex_trylock(mutex)

  - Tries to lock a Mutex.

  - If the mutex object referenced by mutex is currently locked by any thread, the call returns immediately.

- **pthread_mutex_unlock():**

  Syntax - : pthread_mutex_unlock(mutex)

  - Unlocks a Mutex.

Pthread APIs

- **Condition Variable Routines :** A condition variable is a time mechanism that is tightly bound to a mutex and a data item. It is used when one or more threads are waiting for the value of data item to change.
- Example : The code listed in Producer /Consumer statuaries

pthread_cond_init():

Syntax : pthread_cond_init(condition,attr)

- initializes condition variables.

pthread_cond_destroy():

Syntax : pthread_cond_destroy(condition,attr)

- destroys condition variables.

- Remark : Pthreads has no built in thread pool mechanism

Pthread APIs

- **Semaphores :** A semaphore is a counter that can have any nonnegative value. Threads wait on a semaphore.
- When the semaphore's value is 0, all threads are forced to wait. When the value is non-zero, a waiting thread is released to work.
- Pthreads does not implement semaphores, they are part of a different POSIX specification.
- Semaphores are used to conjunction with Pthreads' thread-management functionality

Usage : Include <semaphore.h>

- sem_init(*, *, …*);
- sem_post(*, *, …*)
- sem_wait(*, *, …*)

Pthread APIs – Key Points

- Threads can communicate with one another using events
- A care is needed to terminate the Thread while using the C runtime library.
- Thread synchronizations can be accomplished through the use of Mutexes, Semaphores, Critical Sections, and Interlocked functions
- Windows support multiple thread-priority levels
- Processor affinity is a mechanism that allows the programmer to specify which processor a thread should try to run on. – OS play an important role on Multi Core processor
- POSIX Threads (Pthreads) is a portable threading APIs that is supported on a number of platforms.

Compiling and Executing Pthread Programs

- The compilation and execution details of Pthreads programs will vary from one system to another. The essential steps are common to all the systems: In case of gcc the steps are :

  For compiling:

  $ gcc -o < executable name > < name of source file > -lpthread

  or

  make

Note : Pthreads code should include the pthreads.h header file.

Compiling and Executing Pthread Programs

To compile pthread code

pthread-helloworld.c

Use the command

```
gcc -o helloworld pthread-helloworld.c -lpthread
```

In order to execute we need to give,

```
. / name_of_executable
. / helloworld
```

Pthreads : Data Race - Example Program

Example 3.4 (`pthread-demo-datarace.c`) (Illustrate data race condition in a situation which occurs when more than one thread are trying to work with or update global variable

```
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>

int myglobal;                          // declaration of global variable
pthread_mutex_t mymutex=PTHREAD_MUTEX_INITIALIZER;
```

Source : Reference [4],[6], [7]

Pthreads : Data Race - Example Program

Example 3.4
continued …

```
void *thread_function_datarace(void *arg)
{
      int i,j;
      for ( i=0; i<n; i++ )
{
            j=myglobal;
            j=j+1;
            printf("\n In thread_function_datarace..\t");
             sleep(1);
            myglobal=j;
      }
      return NULL;
}
```

Pthreads : Data Race - Example Program

Example 3.4 continued …

```
void *thread_function_mutex(void *arg)
 {
int i,j;
        for ( i=0; i<n; i++ )
        {
             pthread_mutex_lock(&mymutex);
       j=myglobal;
             j=j+1;
             printf("\n In thread_function_mutex..\t");
       sleep(0.1);
             myglobal=j;
       pthread_mutex_unlock(&mymutex);
       }
        return NULL;
 }
```

Pthreads : Data Race - Example Program

Example 3.4 continued …

```
int main(void)
{

      pthread_t mythread;
      int i;

  if ( pthread_create( &mythread, NULL, thread_function_datarace, NULL) )
{
            printf("error creating thread.");
            abort();
      }
```

Pthreads : Data Race - Example Program

Example 3.4 continued …

```
      for ( i=0; i<n; i++)
{
            myglobal=myglobal+1;
            printf("\n In main..\t");
             sleep(1);
       }
       if ( pthread_join ( mythread, NULL ) )
{
            printf("error joining thread.");
            abort();
       }
       printf("\n Value of myglobal in thread_function_datarace is :  %d\n",myglobal);
printf("\n--------------------------------------------------------------------------------\n");
```

Pthreads : Data Race - Example Program

Example 3.4
continued …

```
myglobal = 0;
 if ( pthread_create( &mythread, NULL, thread_function_mutex, NULL) ) /
 {
        printf("error creating thread.");
        abort();
  }
for ( i=0; i<n; i++)
  {
      pthread_mutex_lock(&mymutex);
       myglobal = myglobal+1;
       pthread_mutex_unlock(&mymutex);
       printf("\n In main..\t");
       sleep(1);
  }
```

Pthreads : Data Race - Example Program

Example 3.4
continued …

```
    if ( pthread_join ( mythread, NULL ) )
     {
          printf("error joining thread.");
          abort();
     }

printf("\n");
     printf("\n Value of myglobal in thread_function_mutex is :  %d\n",myglobal);


    exit(0);

}
```

Shared Memory Programming: Pthreads

- An Overview of Threading APIs
- Summary of POSIX Threads (Pthreads) Model
- Threads on Windows and Different platforms is possible and Multi Core processors systems with different OS can be used
- Compilation and Linking of Pthread Programs
- Example codes using Pthreads
- Different platforms support Pthreads capabilities. Features may not be available in all Pthreads environments.

Conclusions

- Important issues in Shared parallel programming -Pthreads
- Common Synchronization problems with Pthreads
- Pthreads Performance issues on Multi Core Processors

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

13. OpenMP C and C++ Application Program Interface, Version 1.0". **(1998),** OpenMP Architecture Review Board. October 1998

22. D. A. Lewine. *Posix Programmer's Guide:* **(1991),** Writing Portable Unix Programs with the Posix. 1 Standard. O'Reilly & Associates, 1991

23. 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/

24. 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].

25. 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*

26. A. Zomaya, editor. Parallel and Distributed Computing Handbook. McGraw-Hill, **(1996)**

27. OpenMP C and C++ Application Program Interface, Version 2.5 (**May 2005**)", From the OpenMP web site, URL **:** **http://www.openmp.org/**

28. Stokes, Jon 2002 Introduction to Multithreading, Super-threading and Hyper threading *Ars Technica*, October **(2002)**

29. Andrews Gregory R. 2000, Foundations of Multi-threaded, Parallel and Distributed Programming, Boston MA : Addison – Wesley (**2000)**

30. 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