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-III

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

Basic Strategies

- Identify the time consuming code sections

- Add OpenMP directive to parallelize most time consuming loops
  #pragma omp

- If a parallelized loop does not perform well check the following
  - Parallel overhead
  - Small loop
  - Coverage & Granularity
  - Load balance
  - Synchronization & Locality

Advance Features of OpenMP

Lecture Outline :

- OpenMP : Use of Different OpenMP Pragmas
- OpenMP Constructs – Synchronization
  - Work sharing - Minimizing Threading Overhead
  - 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]

Example Program using different OpenMP Pragma

- ❖ Example 1 : OpenMP Parallel & work-share directive
  - ➢ Matrix-Matrix Multiplication
- ❖ Example 2 : OpenMP Data scope Clause “threadprivate” clause
- ❖ Example 3 : OpenMP synchronization construct
  - ➢ Prime number calculation
  - ➢ Producer Consumer : Synchronization Issues
- ❖ Example 4 : Performance tuning

OpenMP Prog. : Parallel & Work-share Directive

Example 1 : Matrix- Matrix Multiplication (Outer loop parallelize

Implementation of Matrix into Matrix Multiplication : dim = 128 and the number of threads = 4

```
for(i=0; i < dim; i++){
      for(j=0; j < dim; k++){
         c(i,j) = 0;
         for(k=0; k < dim; k++){
                c(i,j) += a(i,k)*b(k,j);
         }
      }
}
```

Loop Carried Independence : Challenges in Threading a Loop

Source : Reference : [4]

Example 1 : Matrix- Matrix Multiplication (Outer loop parallel)

Implementation of Matrix into Matrix Multiplication : Static Scheduling of loops in matrix Multiplication

dim = 128 and the number of threads = 4

```
#pragma omp parallel for default(private)  \
        shared(a,b,c,dim) num_threads(4)\
        schedule(static)
for(i=0; i < dim; i++){
     for(j=0; j < dim; k++){
       c(i,j) = 0;
       for(k=0; k < dim; k++){
            c(i,j) += a(i,k)*b(k,j);
       }
     }
}
```

OpenMP Prog. : Parallel & Work-share Directive

Example 1 : Matrix- Matrix Multiplication (Inner loop parallel)

Nesting parallel loops in matrix Multiplication

```
#pragma omp parallel for default(private)   \
        shared(a,b,c,dim) num_threads(2)
for (i=0; i < dim;  i++) {
   #pragma omp parallel for default(private)  \
        shared(a,b,c,dim) num_threads(2)
   for(j=0; j < dim; j++){
      c(i,j) = 0;
       #pragma omp parallel for default(private)\
        shared(a,b,c,dim) num_threads(2)
        for(k=0; k < dim; k++){
              c(i,j) += a(i,k)*b(k,j);
        }
     }
}
```

Source : Reference : [4], [6]

OpenMP Prog. : Example : Inner loop parallelize

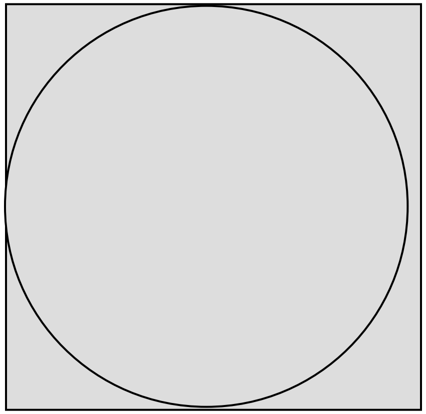
In-Order : The ordered Directive : Example :To compute the cumulative sum of i numbers of a list, we can add the current number to the cumulative sum of i-1 nos. of the list.

```
cumalative_sum[0] = list[0];
#pragma omp parallel for private(I)   \
      shared (cumulative_sum, list, n) ordered
for (i=1; i < n; i++)
{
   /* Other processing on list[I] if needed  */

   #pragma omp ordered;
     {
     cumlative_sum[i] = cumulative_sum[i-1]+list[i];
     }
}
```

Source : Reference : [4], [6]

OpenMP Prog. : Example : Pie Value

Description : Method is based on generating random numbers in a unit length square and counting the number of points that fall within the largest circle inscribed in the square.



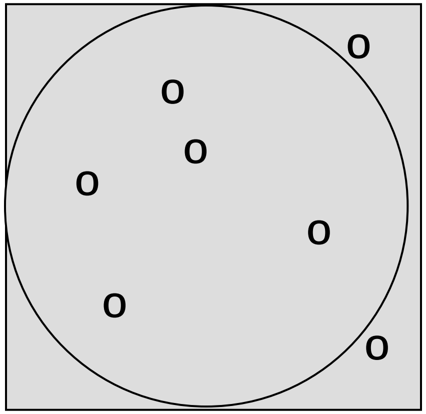
❖Area of the Circle (πr^2) = $\pi/4$

❖Area of Square = 1 X1

The fraction of random points that fall in the circle should approach to $\pi/4$

OpenMP Prog. : Example : Pie Value

1. Assign fixed number of points to each thread.
2. Each thread generates random points and keeps track of the number of points that land in circle locality.
3. After all threads finish execution, their counts are combined to computer the value of π (by calculating the fraction over all threads and multiplying by 4)



Source : Reference : [4], [6]

OpenMP Prog. : Example : Pie Value

❖ Threaded progam to compute PI value

```
#pragma omp parallel default(private)  \
         shared(npoints) reduction(+:sum)
         num_threads(8)
{
 num_threads = omp_get_num_threads();
 sample_points_per_thread=npoints/num_threads;
 sum = 0.0;
 for(i=0; i<sample_points_per_thread; i++){
    rand_no_x=(double)rand_r(&seed))/double((2<<14-1);
    rand_no_y=(double)rand_r(&seed))/double((2<<14-1);
    if((rand_no_x-0.5)*rand_no_x-0.5) +
       (rand_no_y-0.5)*rand_no_y-0.5)) < 0.25
    sum++;
    }
}
```

OpenMP Prog. : Example : Pie Value

❖ Threaded progam to compute PI value

```
#pragma omp parallel default(private)   \
         shared(npoints) reduction(+:sum)
         num_threads(8)
{
 sum = 0.0;
 #pragma omp for
 for(i=0; i < npoints; i++){
     rand_no_x=(double)rand_r(&seed))/double((2<<14-1);
     rand_no_y=(double)rand_r(&seed))/double((2<<14-1);
     if((rand_no_x-0.5)*rand_no_x-0.5) +
        (rand_no_y-0.5)*rand_no_y-0.5)) < 0.25
     sum++;
     }
}
```

OpenMP Directives : Synchronization Constructs

Consider a simple example where two threads on two different processors are both trying to increment a variable x at the same time (Assume x is initially 0) :

THREAD 1 :
Increment (x)
{
X= x+1
}
THREAD 1:
 	
10 LOAD A, (x address)
20 ADD A, 1
30 STORE A, x address)
	THREAD 1 :
	Increment (x)
	{
	X= x+1
	}
	THREAD 1:
	
	10 LOAD A, (x address)
	20 ADD A, 1
	30 STORE A, x address

OpenMP : Synchronization

Contd…

- Atomic is a special case of a critical section that can be used for certain simple statements.
- It applies only to the update of a memory location (the update of X in the following example)

```
C$OMP PARALLEL PRIVATE(B)
        B =  DOIT(I)
C$OMP ATOMIC
        X = X + B

C$OMP END PARALLEL
```

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

OpenMP : Synchronization

Contd…

- The master construct denotes a structured block that is only executed by the master thread. The other threads just skip it (no synchronization is implied).

```
#pragma omp parallel private (tmp)
{
        do_many_things();
#pragma omp master
        {    exchange_boundaries();   }
#pragma barrier
        do_many_other_things();
}
```

OpenMP Synchronization construct

Example 3 : Prime Number calculation

Activity 1: Analysis (Serial Run)

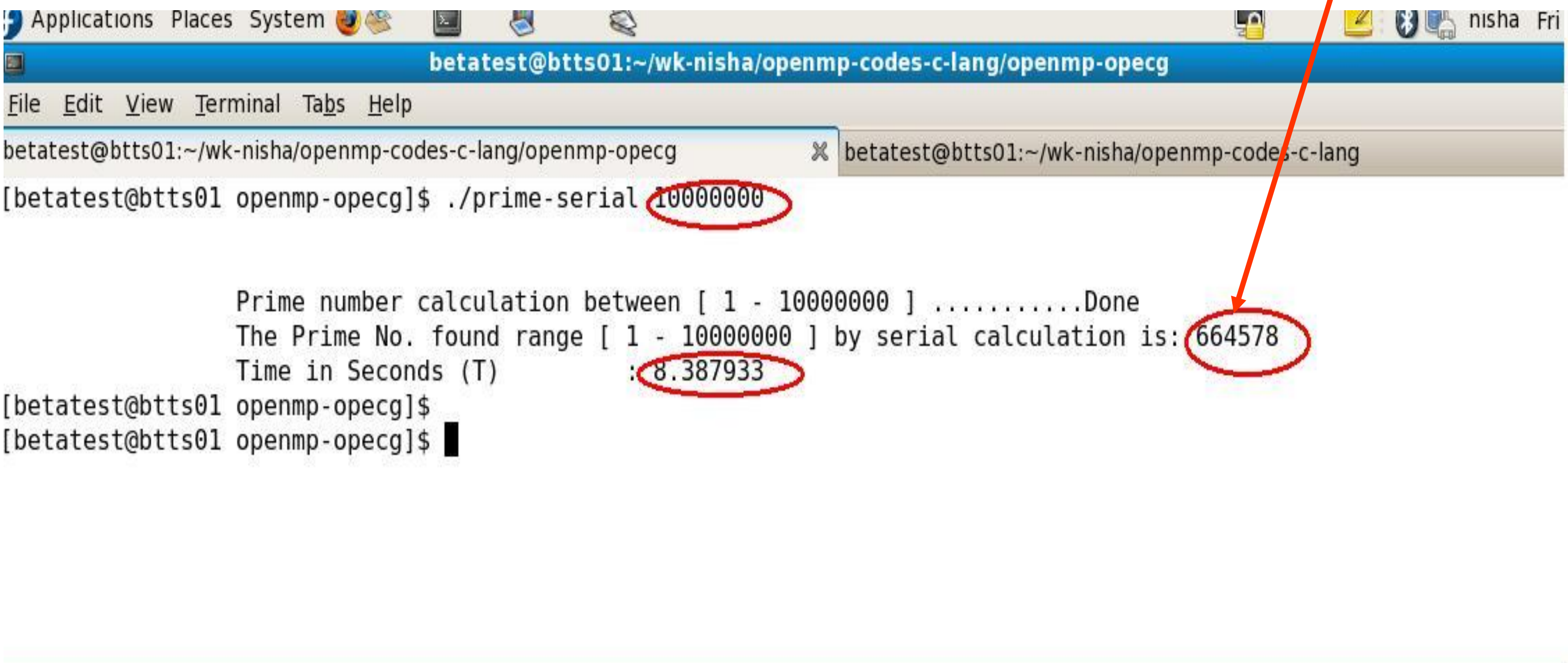
```
for (number =3 ; number < Maxnumber  ; number += 2 )
{
        if (is_prime(number))   {

                Primearray[ Count ] = number;
                Count=Count+1;
        }
 }
```

OpenMP Synchronization construct

Example 3 : Prime Number calculation

Activity 1: Analysis (Serial Run)

Prime no. found 664578



OpenMP Synchronization construct

Example 3 : Prime Number calculation

Activity 2 : Design (Parallel Run)

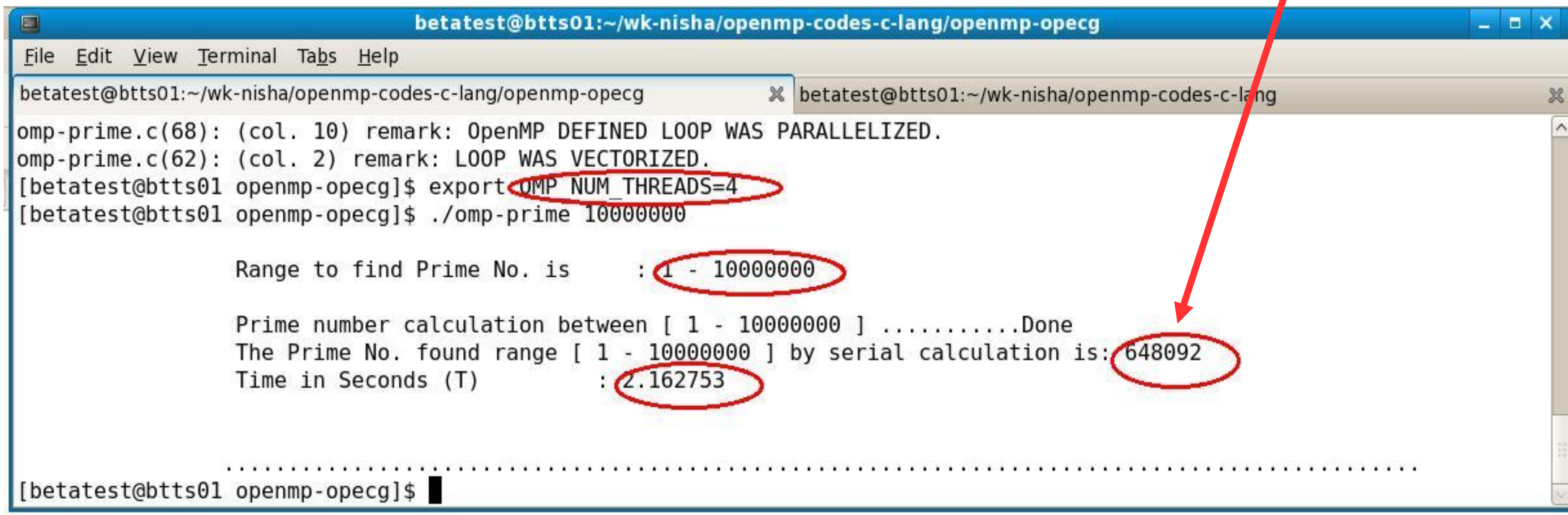
Insert OpenMP parallel for directive

```
#pragma omp parallel for
for (number =3 ; number < Maxnumber  ; number += 2 )
{
        if (is_prime(number))   {

                Primearray[ Count ] = number;
                Count=Count+1;
        }
 }
```

OpenMP Synchronization construct

Example 3 : Prime Number calculation
Activity 2 : Design (Parallel Run)

Prime count is different in serial & parallel run



Is this threaded implementation is right ?

No, we are Getting the different result from the serial computation?

OpenMP Synchronization construct

Example 3 : Prime Number calculation

Activity 3:   Debugging

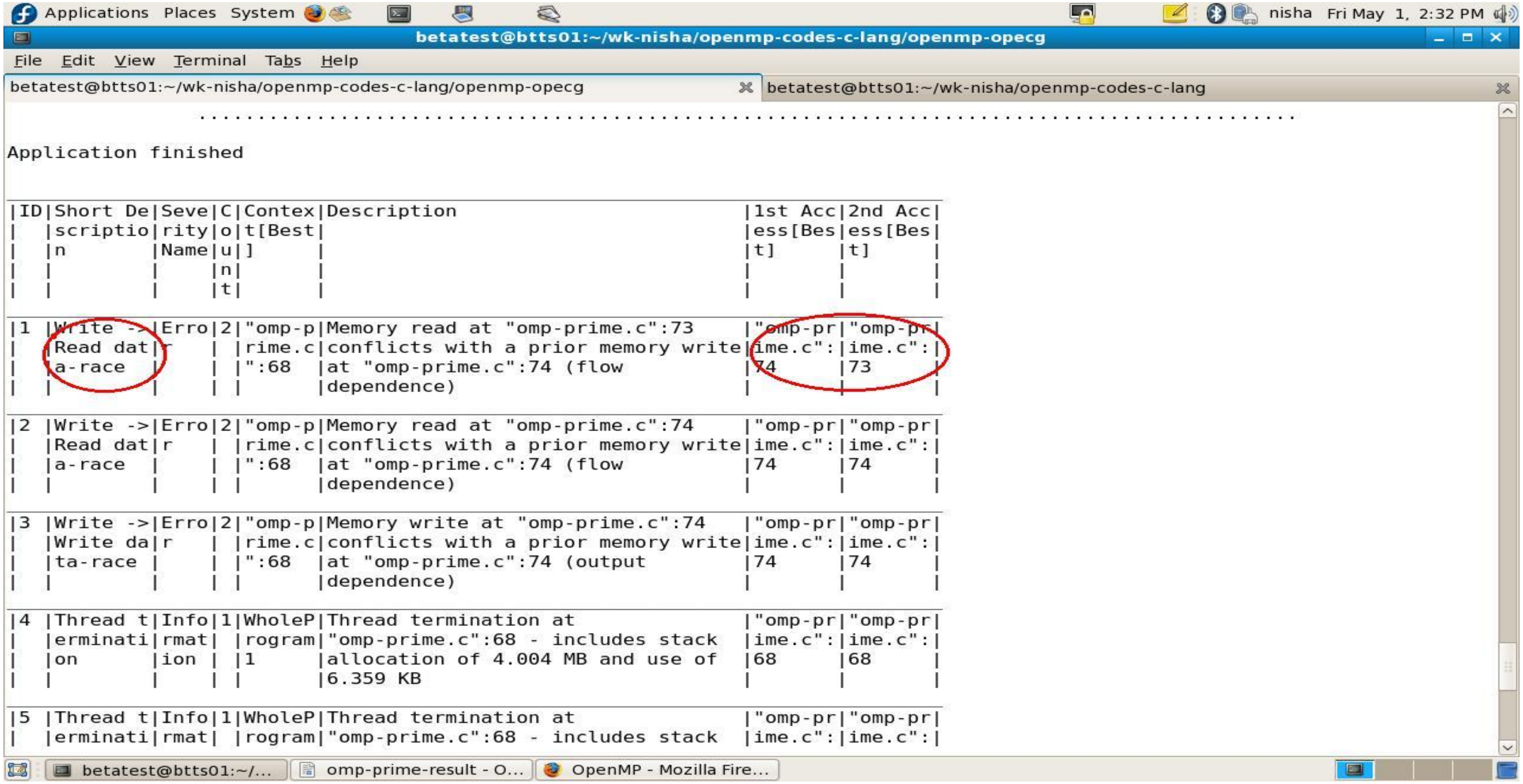
```
#pragma omp parallel for
for (number =3 ; number < Maxnumber  ; number += 2 )
{
        if (is_prime(number))   {

                Primearray[  Count  ] = number;

                Count =  Count  +  1;
        }
 }
```

Synchronization :
Data Race

OpenMP Synchronization construct

Example 3 : Prime Number calculation

Activity 3 : Debugging (Thread Checker)



```
Applications Places System                                nisha  Fri May 1, 2:32 PM
betatest@btts01:~/wk-nisha/openmp-codes-c-lang/openmp-opecg
File Edit View Terminal Tabs Help
betatest@btts01:~/wk-nisha/openmp-codes-c-lang/openmp-opecg    betatest@btts01:~/wk-nisha/openmp-codes-c-lang
                    ..............................................................................................

Application finished

|ID|Short De|Seve|C|Contex|Description                            |1st Acc|2nd Acc|
|  |scriptio|rity|o|t[Best|                                       |ess[Bes|ess[Bes|
|  |n       |Name|u|]     |                                       |t]     |t]     |
|  |        |    |n|      |                                       |       |       |
|  |        |    |t|      |                                       |       |       |

|1 |Write ->|Erro|2|"omp-p|Memory read at "omp-prime.c":73        |"omp-pr|"omp-pr|
|  |Read dat|r   | |rime.c|conflicts with a prior memory write|ime.c":|ime.c":|
|  |a-race  |    | |":68  |at "omp-prime.c":74 (flow              |74     |73     |
|  |        |    | |      |dependence)                            |       |       |

|2 |Write ->|Erro|2|"omp-p|Memory read at "omp-prime.c":74        |"omp-pr|"omp-pr|
|  |Read dat|r   | |rime.c|conflicts with a prior memory write|ime.c":|ime.c":|
|  |a-race  |    | |":68  |at "omp-prime.c":74 (flow              |74     |74     |
|  |        |    | |      |dependence)                            |       |       |

|3 |Write ->|Erro|2|"omp-p|Memory write at "omp-prime.c":74       |"omp-pr|"omp-pr|
|  |Write da|r   | |rime.c|conflicts with a prior memory write|ime.c":|ime.c":|
|  |ta-race |    | |":68  |at "omp-prime.c":74 (output            |74     |74     |
|  |        |    | |      |dependence)                            |       |       |

|4 |Thread t|Info|1|WholeP|Thread termination at                  |"omp-pr|"omp-pr|
|  |erminati|rmat| |rogram|"omp-prime.c":68 - includes stack      |ime.c":|ime.c":|
|  |on      |ion | |1     |allocation of 4.004 MB and use of      |68     |68     |
|  |        |    | |      |6.359 KB                               |       |       |

|5 |Thread t|Info|1|WholeP|Thread termination at                  |"omp-pr|"omp-pr|
|  |erminati|rmat| |rogram|"omp-prime.c":68 - includes stack      |ime.c":|ime.c":|

betatest@btts01:~/...   omp-prime-result - O...   OpenMP - Mozilla Fire...
```

OpenMP Synchronization construct

Example 3 : Prime Number calculation

Activity 4 : Avoid Data-Race Condition

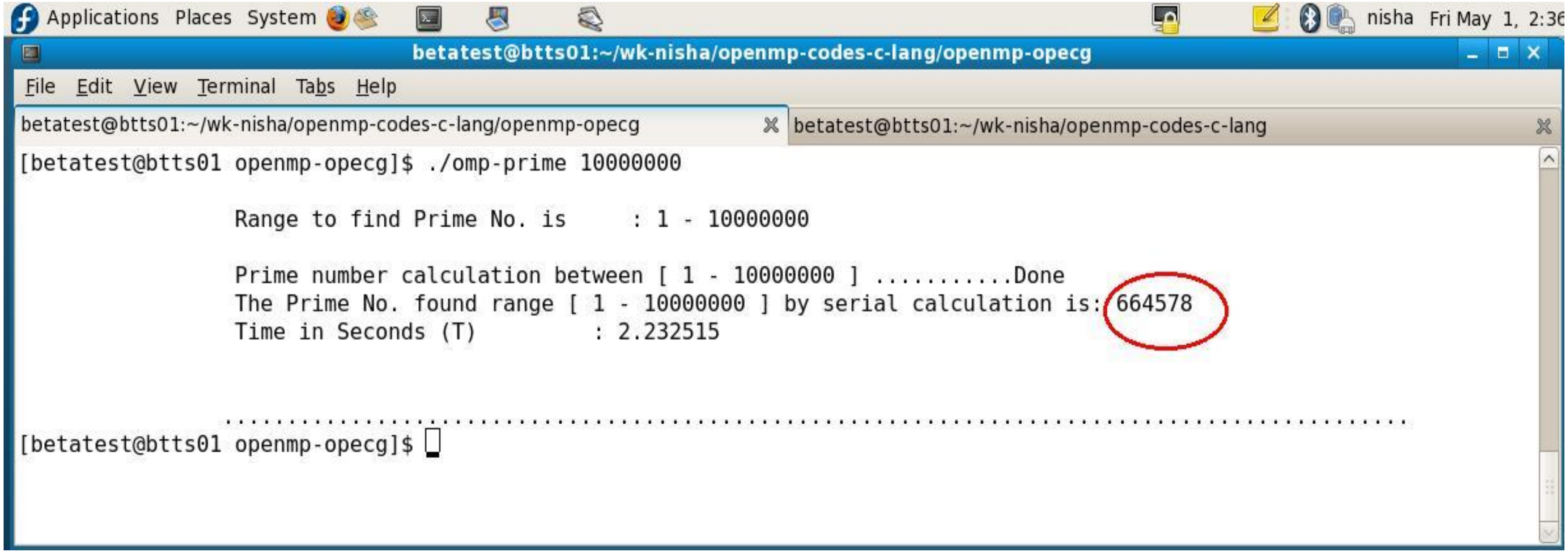
Insert Synchronization Construct

```
#pragma omp parallel for
for (number =3 ; number < Maxnumber  ; number += 2 )
{
    if (is_prime(number))   {
        # pragma omp critical
        {
                        Primearray[ Count ] = number;
                        Count=Count+1;
        }
    }
}
```

OpenMP Synchronization construct

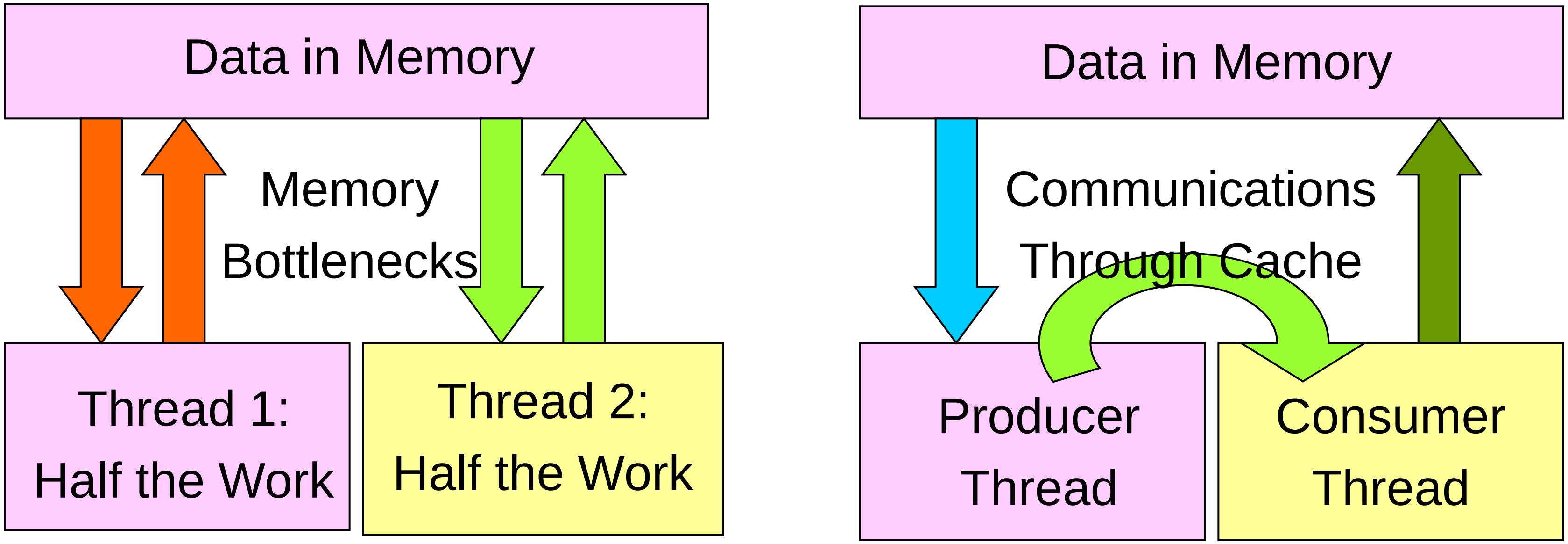
Example 3 : Prime Number calculation

Activity 4 : Parallel Run



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.

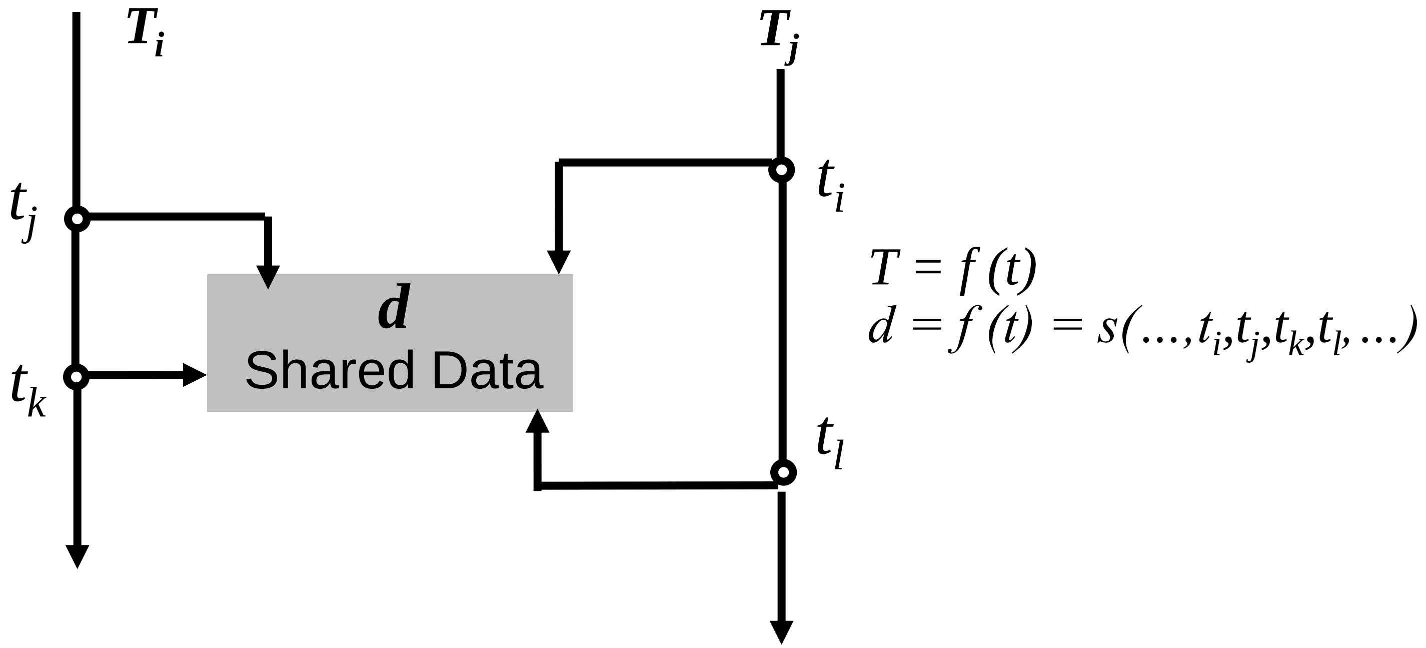


Source : Reference : [4], [6]

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

Synchronization order

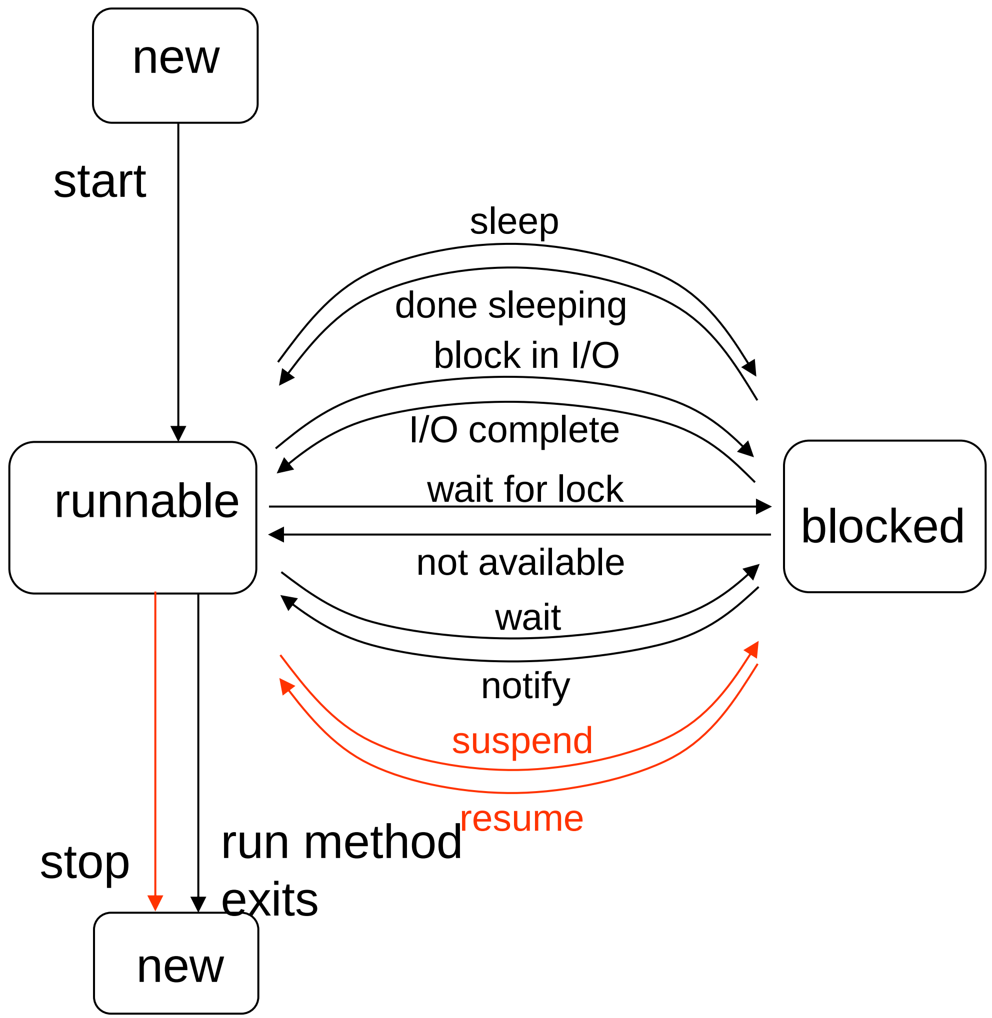


Shared data *d* depends on synchronization functions of time

Shared Data Synchronization, Where Data *d* is protected by a Synchronization Operation

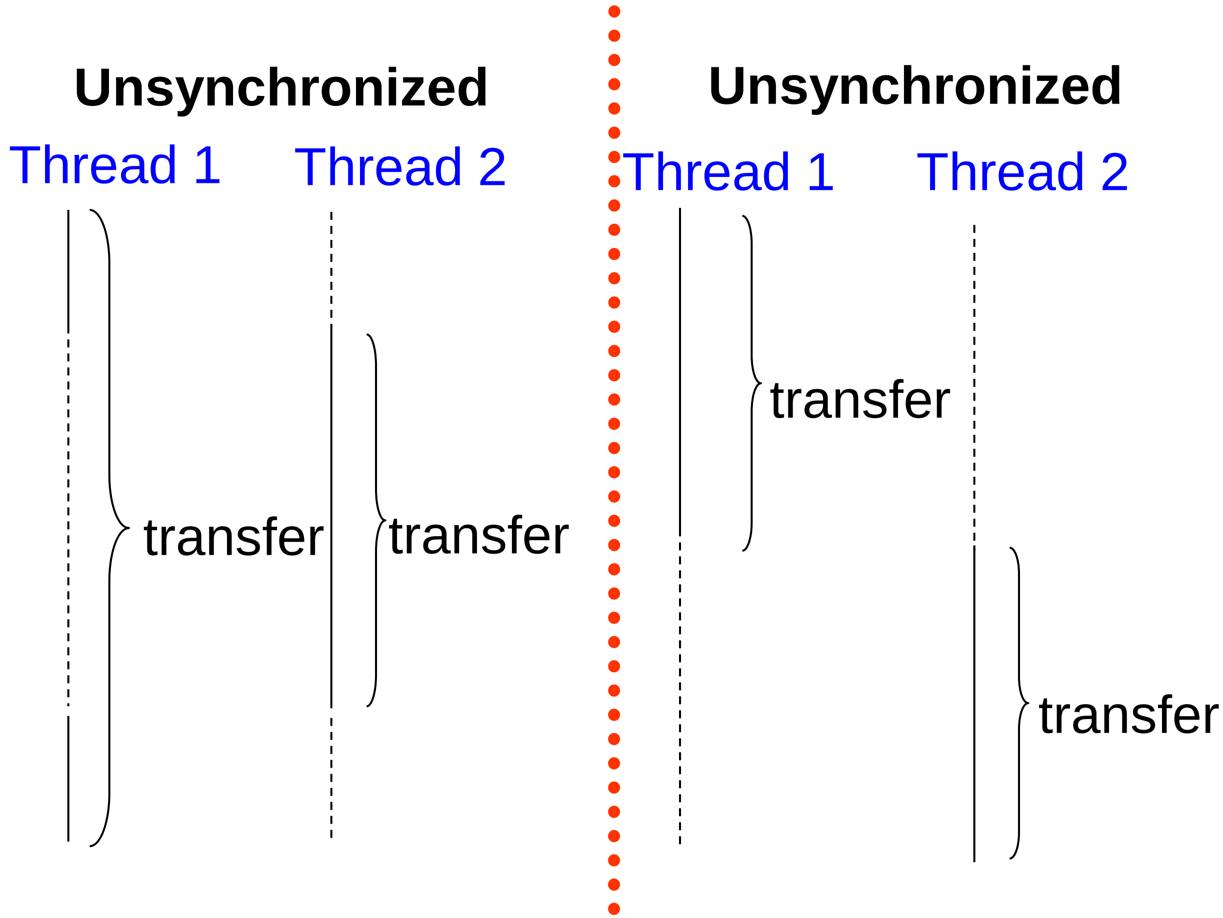
Pthreads:Synchronization & Thread States

- I/O Requests
- Read-Write Locks
- Available CPU
- Release Locks
- Critical Sections

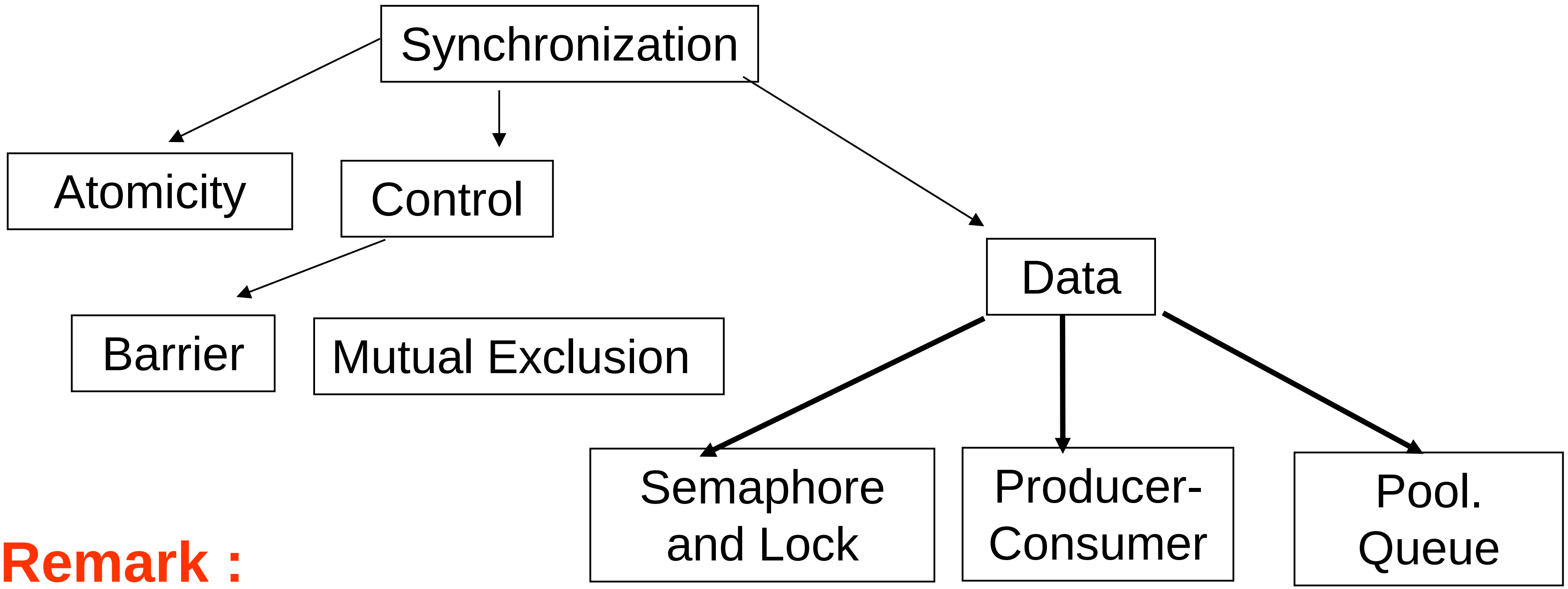


Comparison of unsynchronized / synchronized threads

- ❖ Too little / too much synchronization
  - ➢ In-Correct Results
  - ➢ Performance – Slow done the results



Pthreads : Various types of synchronization



Remark :

- Use of Scheduling techniques as means of Synchronization is not encouraged. – Thread Scheduling Policy, High Priority & Low Priority Threads
- Atomic operations are a fast and relatively easy alternative to mutexes. They do not suffer from the deadlock.

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.

OpenMP Prog. : Synchronization

- In-Order : The ordered Directive
  - Example : Execute on all the Threads

```
cumlative_sum[I]=
          cumulative_sum[I-1]+list [I]
```

- While execute the for loop across the threads cumlative_sum[] can be computed after cumaltive_sum[I-1] has been computed
  - #pragma omp ordered
  - Structured block

OpenMP Prog. : Synchronization

❖ In-Order : The ordered Directive : Example :To compute the cumulative sum of i numbers of a list, we can add the current number to the cumulative sum of i-1 nos. of the list.

```
cumalative_sum[0] = list[0];
#pragma omp parallel for private(I)  \
      shared (cumulative_sum, list, n) ordered
for (i=1; i < n; i++)
{
   /* Other processing on list[I] if needed  */

   #pragma omp ordered;
     {
     cumlative_sum[i] = cumulative_sum[i-1]+list[i];
     }
}
```

OpenMP :Data Handling in OpenMP

- Data Handling in OpenMP
- One of the Critical factors influencing program performance is the manipulation of data by threads.
- How effectively we can use data classes such as **`private`**, **`shared`**, **`firstprivate`**, & **`lastprivate`**

Other data Classes

The **`threadprivate`** and **`copy`** in Directives

- **`#pragma omp threadprivate(variable_list)`**

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

OpenMP :Data Handling in OpenMP

Following Heuristics to guide the process

- If a thread initializes and uses a variable such as loop index ---- specify data as `private`

- If a thread repeatedly reads variable that has been initialized earlier ---- specify data as `firstprivate`

- If multiple threads manipulate a single piece of data, one must explore ways of breaking these manipulations into local operations – use `reduction` clause

- If multiple threads manipulate different parts of a large data structure, the programmer should explore ways of breaking it into smaller data structures and making them `private` to the thread manipulating them.

Performance issues-Synchronization Overhead

- Performance depends on input workload :
  - Increasing clients and contention
    - Number of clients *vs* Ratio of Time to Completion
  - Performance depends on a good locking strategy
    - No locks at all;One lock for the entire data base; One lock for each account in the data base
  - Performance depends on the type of work threads do
    - Percentage of Thread I/O *vs* CPU and Ratio of Time to Completion
  - Performance due to Loop Scheduling and Partitioning

Overheads of OpenMP

Time in Microseconds

Construct	Cost
parallel	1.5
Barrier	1
Schedule (Static)	1
Schedule (guided)	6
schedule (dynamic)	50
ordered	0.5
Single	1
atomic	0.5
Critical	0.5
Lock\Unlock	0.5

Performance & Tuning : Issues

- Coverage & Granularity

– No sufficient parallel work

- Load balance

– Improper distribution of parallel work

- Synchronization & Locality

- Excessive use of global data, contention for the same synchronization object

- Performance depends on a good locking strategy

- No locks at all;One lock for the entire data base; One lock for each account in the data base

- Performance due to Loop Scheduling and Partitioning

Performance & Tuning Examples

Coverage & Granularity : No sufficient parallel work

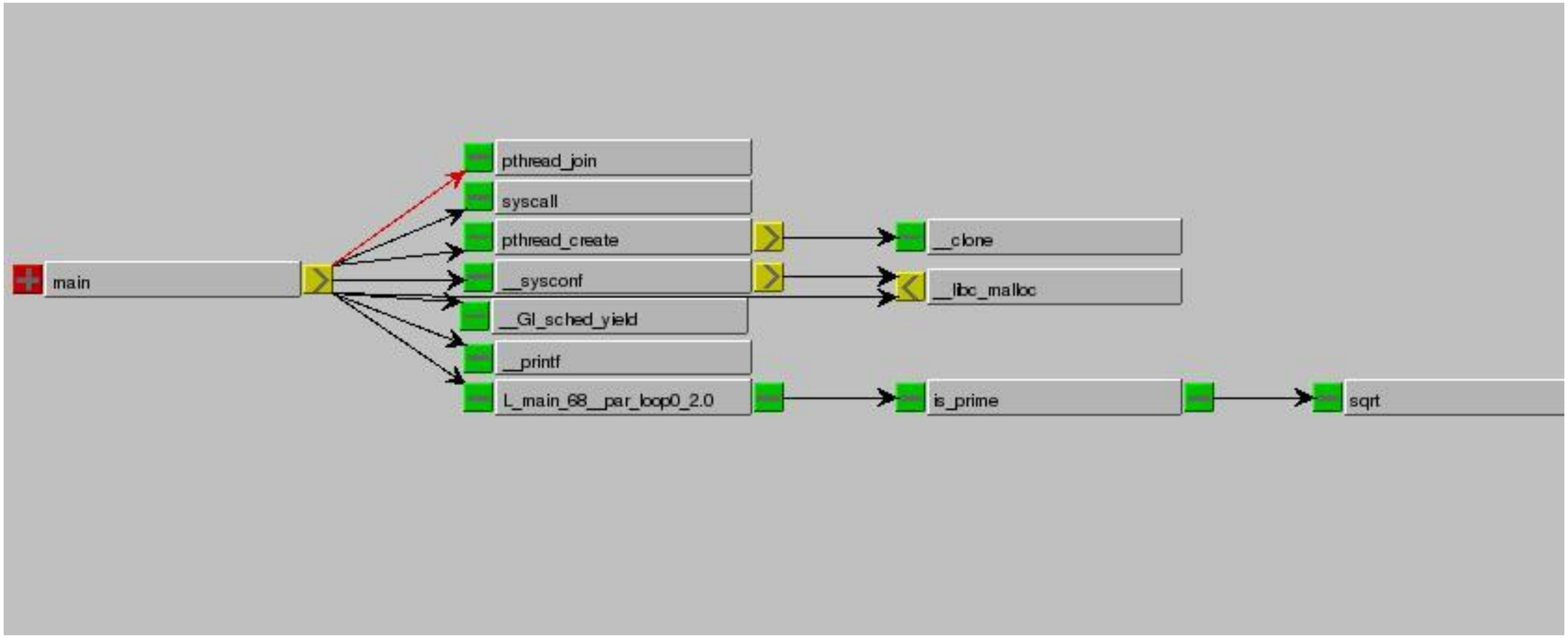
Prime Number calculation :

Range [1-1000]

```
 #pragma omp parallel for
 for (number =3 ; number < Maxnumber  ; number += 2 )
 {
     if (is_prime(number))   {
# pragma omp critical
     {
                         Primearray[ Count ] = number;
                         Count=Count+1;
          }
  }
```

Performance & Tuning Examples

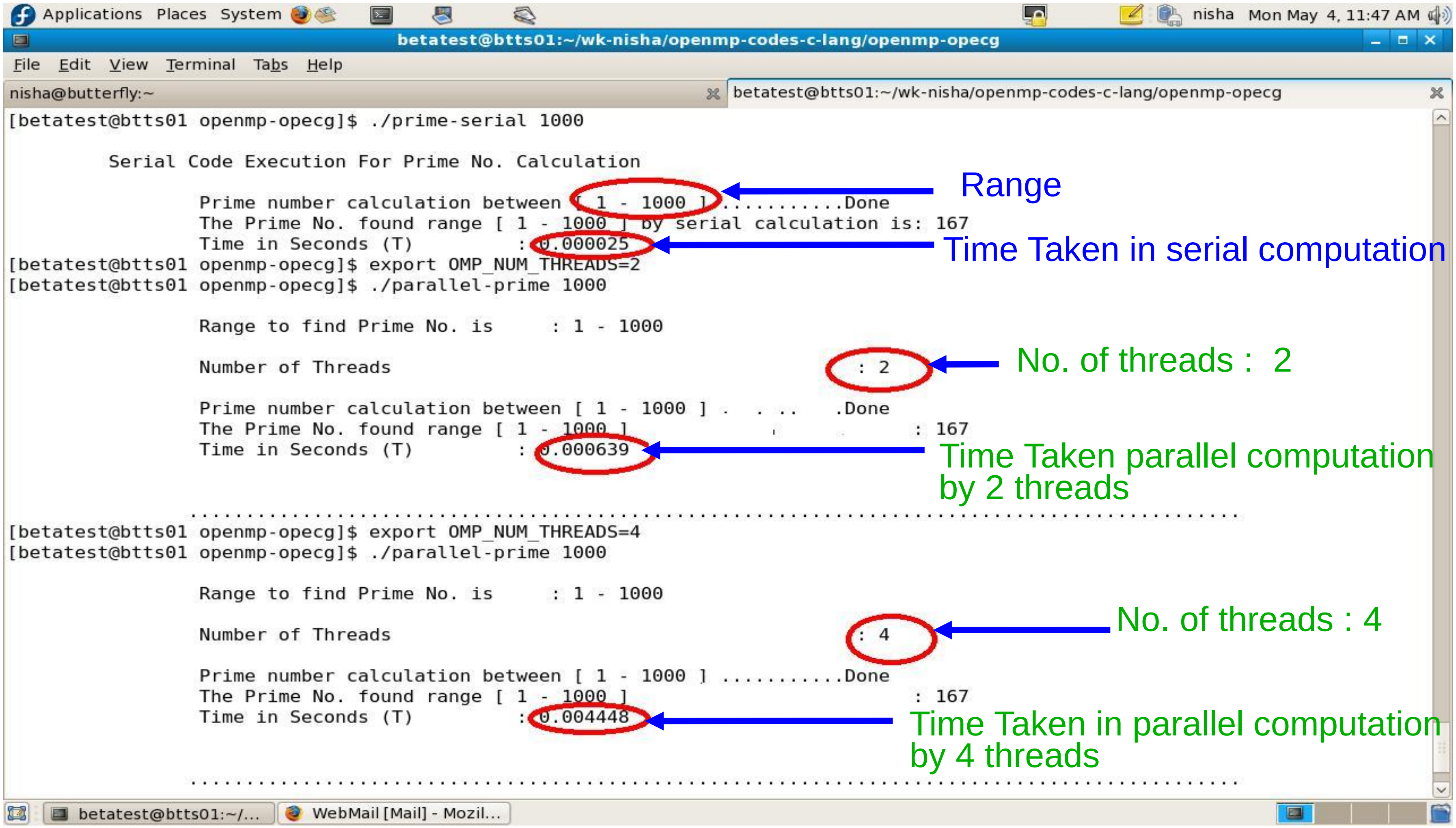
Typical Call-graph tree in openMP libraries



Performance & Tuning Examples

Coverage & Granularity : No sufficient parallel work

Prime Number calculation : Range [1-1000]



Performance & Tuning Examples

Synchronization Issues : Performance depends on the good locking scheme

Example : Find Sum of an Array Elements using Critical/Reduction

a) OpenMP Critical Directive

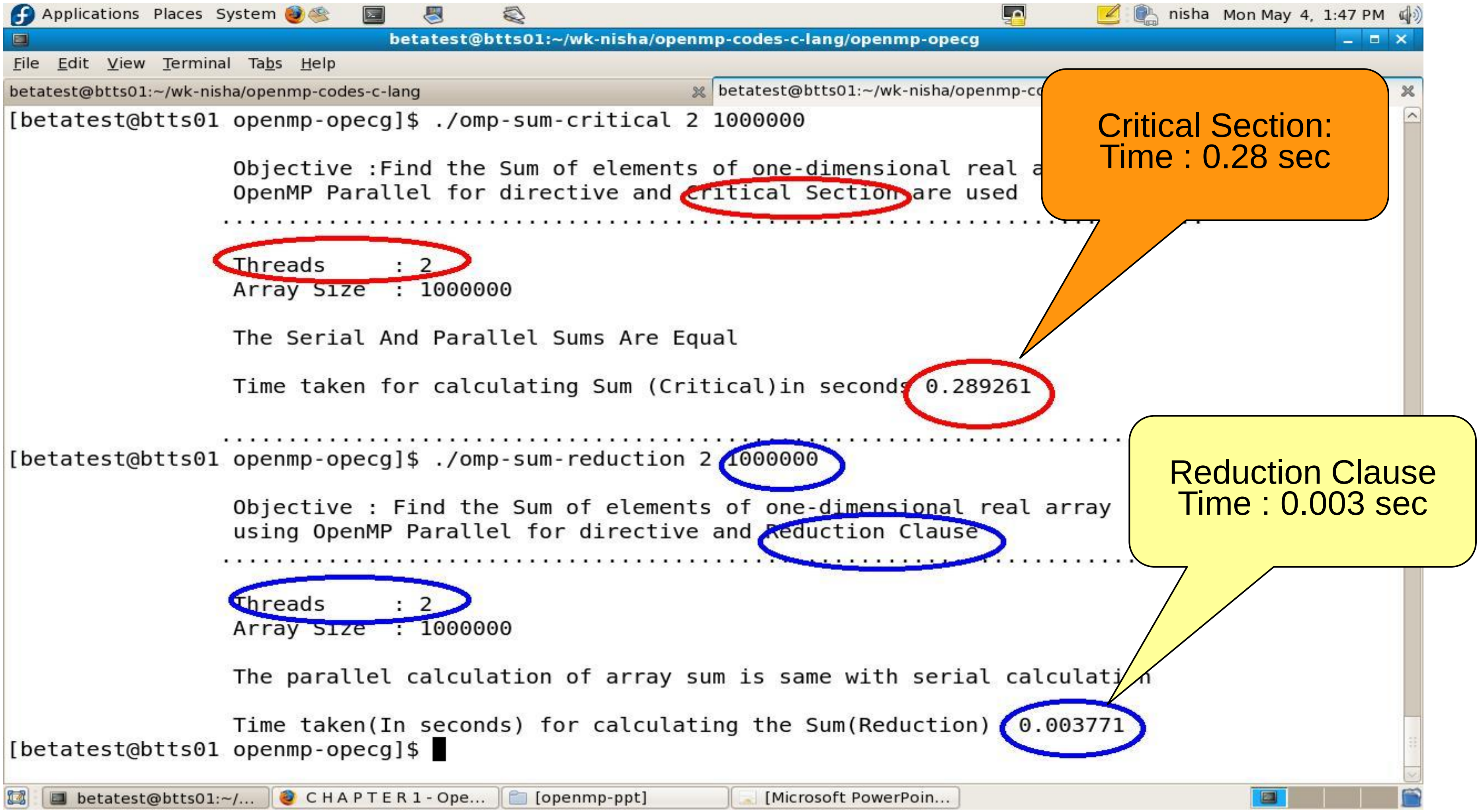
```
#pragma omp parallel for
 for (i = 0; i < array_size; i++)
 {
             #pragma omp critical
               sum = sum + Array[i];

} /* End of parallel region */
```

b) OpenMP Reduction Clause

```
#pragma omp parallel for reduction(+ : sum)
  for (i = 0; i < array_size; i++)
  {
              sum = sum + Array[i];

   } /* End of parallel region */
```

Performance & Tuning Examples

Synchronization Issues : Performance depends on the good locking scheme



Performance issues-Synchronization Overhead

- How do your threads spend their time ?
  - Profiling a program is a good step toward identifying its performance bottlenecks (CPU Utilization, waiting for locks and I/O completion
  - Do the threads spend most of their time blocked, waiting for their threads to release locks ?
  - Are they *runnable* for most of their time but not actually running because other threads are monopolizing the available CPUs ?
  - Are they spending most of their time waiting on the completion of I/O requests ?

Explicit Threads *versus* OpenMP Based Prog.

❖ An Artifact of Explicit threading is that data exchange is more apparent. This helps in alleviating some of the overheads from data movement, false sharing, and contention.

❖ Explicit threading also provides a richer API in the form of condition waits.

❖ Locks of different types, and increased flexibility for building composite synchronization operations

❖ Data-race conditions due to output dependencies

❖ Managing Shared and Private Data – **`OpenMP shared`**, **`private`**, and **`default`** clauses)

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

Explicit Threads *versus* OpenMP Based Prog.

- OpenMP provides a layer on top of naïve threads to facilities a variety of thread-related tasks.

- Using Directives provided by OpenMP, a programmer is get rid of the task of initializing attribute objects, setting up arguments to threads, partitioning iteration spaces etc…. (This may be useful when the underlying problem has a static and /or regular task graph.

- The overheads associated with automated generation of threaded code from directives have been shown to be minimal in the context of a variety of applications.

Explicit Threads *versus* OpenMP Based Prog.

- ❖ Compiler support on Multi-Cores play an important role
- ❖ Issues related to OpenMP performance on Multi cores need to be addressed.
- ❖ Inter-operability of OpenMP/Pthreads on Multi-Cores require attention -from performance point of view
- ❖ Performance evaluation and use of tools and Mathematical libraries play an important role.

Shared Memory Programming :The OpenMP Standard

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
- ❖ OpenMP programming models are covered.
- ❖ OpenMP Synchronization Constructs
- ❖ Using OpenMP Constructs.

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 ?