National Institute of Technology Karnataka Surathkal
Department of Information Technology

IT 301 Parallel Computing

Shared Memory Programming Technique (5)

OpenMP : *Synchronization and other clauses*

Dr. Geetha V

Assistant Professor

Dept of Information Technology

NITK Surathkal

Index

Course Outline

Course Plan: Theory:

Part A: Parallel Computer Architectures

Week 1,2,3: ***Introduction to Parallel Computer Architecture:*** Parallel Computing, Parallel architecture, bit level, instruction level , data level and task level parallelism. Instruction level parelllelism: pipelining(Data and control instructions), scalar and superscalar processors, vector processors. Parallel computers and computation.

Week 4,5: Memory Models: UMA, NUMA and COMA. Flynns classification, Cache coherence,

Week 6,7: Amdahl's Law. Performance evaluation, Designing parallel algorithms : Divide and conquer, Load balancing, Pipelining.

Week 8 -11: ***Parallel Programming techniques like Task Parallelism using TBB, TL2, Cilk++ etc. and software transactional memory techniques.***

Course Outline

Part B: OpenMP/MPI/CUDA

Week 1,2,3 : ***Shared Memory Programing Techniques:*** Introduction to OpenMP : Directives: *parallel, for, sections, task, master, single, critical, barrier, taskwait, atomic. Clauses: private, shared, firstprivate, lastprivate, reduction, nowait, ordered, schedule, collapse, num_threads, if(), threadprivate, copyin, copyprivate*

Week 4,5: ***Distributed Memory programming Techniques:*** MPI: Blocking, Non-blocking.

Week 6,7 : CUDA : OpenCL, Execution models, GPU memory, GPU libraries.

Week 10,11,: ***Introduction to accelerator programming using CUDA/OpenCL and Xeon-phi. Concepts of Heterogeneous programming techniques.***

Practical:

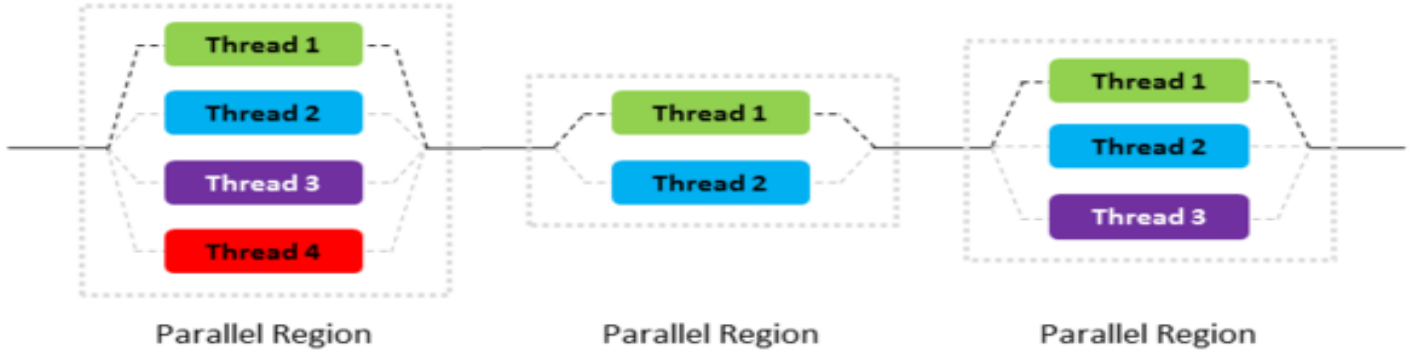
Implementation of parallel programs using OpenMP/MPI/CUDA.

Assignment: Performance evaluation of parallel algorithms (in group of 2 or 3 members)

1. OpenMP

FORK – JOIN Parallelism

- OpenMP program begin as a single process: the master thread. The master thread executes sequentially until the first parallel region construct is encountered.
- When a parallel region is encountered, master thread
  - Create a group of threads by FORK.
  - Becomes the master of this group of threads and is assigned the thread id 0 within the group.
- The statement in the program that are enclosed by the parallel region construct are then executed in parallel among these threads.
- JOIN: When the threads complete executing the statement in the parallel region construct, they synchronize and terminate, leaving only the master thread.



2. OpenMP Programming: Directives : Parallel, For

#pragma omp parallel *[clause[,]clause…] new-line*

Structured-block

Clause: **if**(*scalar-expression*)

num_threads(*integer-expression*)

default(*shared|none*)

private(*list*)

firstprivate(*list*)

shared(*list*)

copyin(*list*)

reduction(*operator:list*)

#pragma omp for *[clause[,]clause…] new-line*

for-loops

Clause: **private**(*list*)

firstprivate(*list*)

lastprivate(*list*)

reduction(*operator:list*)

schedule(*kind[,chunk_size]*)

collapse(*n*)

ordered

nowait

2. OpenMP Programming: Clauses : master

```
#pragma omp master
        Structured block
```

Master :

- It specifies a structured block that is executed by the master thread of the team
- Other threads in the team do not execute the associated structured block.
- There is no implied barrier either on entry to, or exit from, the master construct.

2. OpenMP Programming: Clauses : single

```
#pragma omp single [clause [[,] clause…]
Structured block
Clause:
        Private(list)
        Firstprivate(list)
        Copyprivate(list)
        nowait
```

Single :

- It specifies that the associated structured block is executed by only one thread in the team (not necessarily the master thread).
- The other threads in the team do not execute the block, and wait at an implicit barrier at the end of single construct, unless nowait clause is specified
- *nowait* : The other threads need not wait for synchronization point.
- *Copyprivate* clause must not be used with the *nowait* clause

2. OpenMP Programming: Clauses : critical

critical

- The critical construct restricts execution of the associated structured block to a single thread at a time.
- An optional name may be used to identify the critical construct. All critical constructs without a name are considered to have the same unspecified name.
- A thread waits at the beginning of a critical region until no other thread is executing a critical region with the same name.
- The critical construct enforces exclusive access with respect to all critical constructs with the same name in all threads, not just in the current team.

```
#pragma omp critical [(name)] new-line
Structured block
```

2. OpenMP Programming: Construct: barrier

barrier

- The barrier construct specifies an explicit barrier at the point at which the construct appears.
- The barrier directive may only be placed in the program at a position where ignoring or deleting the directive would result in a program with correct syntax.
- All of the threads of the team executing the binding parallel region must execute the barrier region before any are allowed to continue execution beyond the barrier.
- Each barrier region must be encountered by all threads in a team or by none at all.
- The sequence of work-sharing regions and barrier regions encountered must be the same for every thread in a team.

```
#pragma omp barrier new-line
```

2. OpenMP Programming: Construct: atomic

atomic

- The atomic construct ensures that a specific storage location is updated atomically, rather than exposing it to the possibility of multiple, simultaneous writing threads.
- Expression-stmt is an expression statement with one of the following forms:
  - x binop=expr
  - x++
  - ++x
  - x--
  - --x
- Binop is not an overloaded operator and is one of +,*,-,/,&,^,|,<<, or >>.
- Only the load and store of the object designated by x are atomic; the evaluation of expr is not atomic.

```
#pragma omp atomic new-line
Expression-stmt
```

2. OpenMP Programming: Construct: flush

```
#pragma omp flush [(list)] new-line
```

flush

- The flush construct executes the OpenMP flush operation.
- This operation makes a thread's temporary view of memory consistent with memory and enforces an order on the memory operations of the variables explicitly specified or implied.
- The flush construct with a list applies the flush operation to the items in the list, and does not return until the operation is complete for all specified list items.
- A flush construct without a list, executed on a given thread, operates as if the whole thread-visible data state of the program, as defined by the base language, is flushed.
- If a pointer is present in the list, the pointer itself is flushed, not the object to which the pointer refers.

2. OpenMP Programming: Construct: flush

#pragma omp flush [(list)] new-line

flush

A flush region without a list is implied at the following locations:

- During a barrier region
- At entry to and exit from parallel, critical and ordered regions.
- At exit from work-sharing regions, unless a *nowait* is present.
- At entry to and exit from combined parallel work-sharing regions
- During set and unset lock, test, nest lock etc.

A flush region with a list is implied at the following locations:

At entry and exit from atomic regions, where the list contains only the object updated in the atomic construct.

2. OpenMP Programming: Construct: ordered

```
#pragma omp ordered new-line
Structured block
```

Ordered

The ordered construct specifies a structured block in a loop region which will be executed in the order of the loop iterations.

This sequentializes and orders the code within an ordered region while allowing code outside the region to run in parallel.

When a thread executing other than first iteration, encounters an ordered region, it waits at the beginning of that ordered region until each of the previous iterations that contains an ordered region has completed the ordered region.

2. OpenMP Programming: Construct: ordered

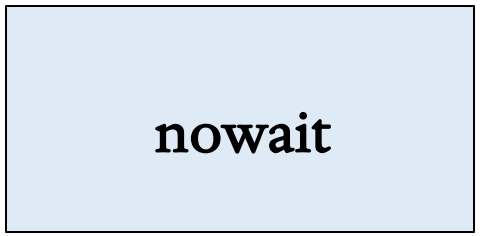
```
#pragma omp ordered new-line
Structured block
```

Ordered

Restrictions to the ordered construct:

- The loop region to which an ordered region binds must have an ordered clause specified on the corresponding loop (or parallel loop) construct.
- During execution of an iteration of a loop within a loop region, the executing thread must not execute more than one ordered region which binds to the same loop region.

2. OpenMP Programming: Clause: nowait



nowait

It there are multiple independent loops within a parallel region, then *nowait* can be used to avoid the implied barrier at the end of the loop construct.

3. OpenMP Programming: Examples

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int main (void) {
int x=10, tid;
printf("x value ouside parallel:%d\n",x);
#pragma omp parallel  num_threads(4) shared(x) private(tid)
    {

    int tid=omp_get_thread_num();
    #pragma omp master
    {
        x=15;
        printf("\n 1. Thread [%d] value of x is %d \n",tid,x);
    }
    x=x+1;

    printf("\n 2. Thread [%d] value of x is %d \n",tid,x);
    }
return 0;
}
```

```
x value ouside parallel:10

 2. Thread [2] value of x is 11

 2. Thread [3] value of x is 16

 1. Thread [0] value of x is 15

 2. Thread [0] value of x is 17

 2. Thread [1] value of x is 12
```

3. OpenMP Programming: Examples

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int main (void) {
int x=10, tid;
printf("x value ouside parallel:%d\n",x);
#pragma omp parallel  num_threads(4) shared(x) private(tid)
    {

    int tid=omp_get_thread_num();
    #pragma omp single
    {
        x=15;
        printf("\n 1. Thread [%d] value of x is %d \n",tid,x);
    }
    x=x+1;

    printf("\n 2. Thread [%d] value of x is %d \n",tid,x);
    }
return 0;
}
```

```
x value ouside parallel:10

 1. Thread [1] value of x is 15

 2. Thread [0] value of x is 16

 2. Thread [2] value of x is 17

 2. Thread [3] value of x is 18

 2. Thread [1] value of x is 19
```

```
x value ouside parallel:10

 1. Thread [2] value of x is 15

 2. Thread [1] value of x is 16

 2. Thread [0] value of x is 17

 2. Thread [3] value of x is 18

 2. Thread [2] value of x is 19
```

3. OpenMP Programming: Examples

```c
int main (void) {
int a[5], i;
    #pragma omp parallel
    {
        #pragma omp for
        for (i = 0; i < 5; i++)
            a[i] = i * i;

        #pragma omp master
        {
            printf("tid of master %d\n",omp_get_thread_num());
            for (i = 0; i < 5; i++)
                printf("a[%d] = %d\n", i, a[i]);
        }
        // Wait.
        //#pragma omp barrier
        printf("tid %d \n",omp_get_thread_num());
           // Continue with the computation.
        #pragma omp for
        for (i=0;i<5;i++)
            a[i]+=i;

         #pragma omp single
        {
            printf("tid of single %d\n",omp_get_thread_num());
            for (i = 0; i < 5; i++)
                printf("a[%d] = %d\n", i, a[i]);
        }
    }
return 0;
}
```

```
tid 2
tid 1
tid of master 0
a[0] = 0
a[1] = 1
a[2] = 6
a[3] = 12
a[4] = 16
tid 0
tid 3
tid of single 2
a[0] = 0
a[1] = 2
a[2] = 6
a[3] = 12
a[4] = 20
```

3. OpenMP Programming: Examples

```c
int main (void) {
int a[5], i;
    #pragma omp parallel
    {
        #pragma omp for
        for (i = 0; i < 5; i++)
            a[i] = i * i;

        #pragma omp master
        {
            printf("tid of master %d\n",omp_get_thread_num());
            for (i = 0; i < 5; i++)
                printf("a[%d] = %d\n", i, a[i]);
        }
        // Wait.
        #pragma omp barrier
        printf("tid %d \n",omp_get_thread_num());
           // Continue with the computation.
        #pragma omp for
        for (i=0;i<5;i++)
            a[i]+=i;

         #pragma omp single
        {
            printf("tid of single %d\n",omp_get_thread_num());
            for (i = 0; i < 5; i++)
                printf("a[%d] = %d\n", i, a[i]);
        }
    }
return 0;
}
```

```
tid of master 0
a[0] = 0
a[1] = 1
a[2] = 4
a[3] = 9
a[4] = 16
tid 1
tid 3
tid 2
tid 0
tid of single 1
a[0] = 0
a[1] = 2
a[2] = 6
a[3] = 12
a[4] = 20
```

3. OpenMP Programming: Examples

```c
#include<stdio.h>
#include<omp.h>
int main(void)
{
int i,n,a[50],b[50],sum;
double t1,t2;
printf("Enter the value of n");
scanf("%d",&n);
t1=omp_get_wtime();
#pragma omp parallel num_threads(4)
{
int id=omp_get_thread_num();
#pragma omp for ordered reduction(+:sum)
for(i=0;i<n;i++)
{
printf("Thread %d: value of i : %d\n",id,i);
sum=sum+i;
#pragma omp ordered
{
b[i]=i+1;
printf("b[%d] value is %d in ORDER\n",i,b[i]);
}
}
}
t2=omp_get_wtime();
printf("Time taken is %f",t2-t1);
return 0;
}
```

```
Enter the value of n5
Thread 3: value of i : 4
Thread 1: value of i : 2
Thread 2: value of i : 3
Thread 0: value of i : 0
b[0] value is 1 in ORDER
Thread 0: value of i : 1
b[1] value is 2 in ORDER
b[2] value is 3 in ORDER
b[3] value is 4 in ORDER
b[4] value is 5 in ORDER
Time taken is 0.001000
```

Index

Reference

Text Books and/or Reference Books:

1. Professional CUDA C Programming – John Cheng, Max Grossman, Ty McKercher, 2014
2. B.Wilkinson, M.Allen, ”Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers”, Pearson Education, 1999
3. I.Foster, ”Designing and building parallel programs”, 2003
4. Parallel Programming in C using OpenMP and MPI – Micheal J Quinn, 2004
5. Introduction to Parallel Programming – Peter S Pacheco, Morgan Kaufmann Publishers, 2011
6. Advanced Computer Architectures: A design approach, Dezso Sima, Terence Fountain, Peter Kacsuk, 2002
7. Parallel Computer Architecture : A hardware/Software Approach, David E Culler, Jaswinder Pal Singh Anoop Gupta, 2011 8. Introduction to Parallel Computing, Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar, Pearson, 2011

Reference

Acknowledgements

1. Introduction to OpenMP https://www3.nd.edu/~zxu2/acms60212-40212/Lec-12-OpenMP.pdf
2. Introduction to parallel programming for shared memory Machines https://www.youtube.com/watch?v=LL3TAHpxOig
3. OpenMP Application Program Interface Version 2.5 May 2005
4. OpenMP Application Program Interface Version 5.0 November 2018

Thank You