National Institute of Technology Karnataka Surathkal
Department of Information Technology

IT 301 Parallel Computing

Shared Memory Programming Technique (6)

OpenMP : *sections, threadprivate, collapse*

Dr. Geetha V

Assistant Professor

Dept of Information Technology

NITK Surathkal

Index

- OpenMP
  - Directives : if, for, master, single, Barrier, atomic, critical,
  - Directives
    - Sections
  - Clauses
    - Threadprivate
    - Collapse
    - Threadwait
    - Copyin, copyprivate

- References

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:

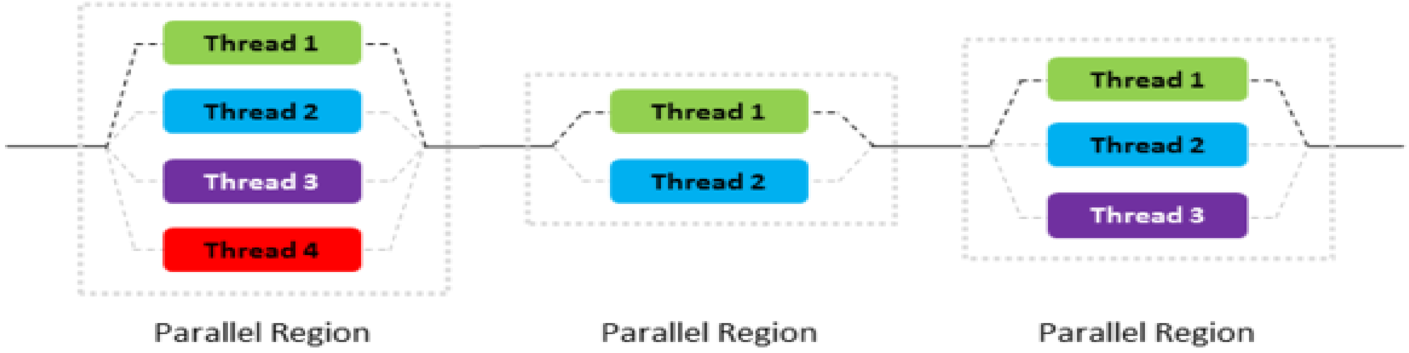
Implementation of parallel programs using OpenMP/MPI/CUDA.

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: Worksharing

Work sharing constructs

- Loop constructs
- Section construct
- Single construct

2. OpenMP Programming: Section

```
#pragma omp sections [clause,….]
{
 [#pragma omp section new-line
   Structured block]
 [#pragma omp section new-line
  Structured-block]
```

Clauses

Private(list)

Firstprivate(list)

Lastprivate(list)

Reduction(operator:list)

nowait

Section :

- It is a non-iterative work-sharing construct that contains a set of structured blocks that are to be divided among, and executed by, the threads in a team.
- Each structured block is executed by one of the threads in the team.
- There is an implicit barrier at the end of sections construct, unless a *nowait* clause is specified.
- Only a single *nowait* clause can appear on a sections directive.

2. OpenMP Programming: Collapse

```
#pragma omp for schedule(static, n)
collapse(2)
for(i=0; i<imax; i++) {
  for(j=0; j<jmax;j ++)
      a[i][j] = b[i][j] + c[i][j]
}
```

Collapse :

- It increases the total number of iterations that will be partitioned across the available number of OMP threads by reducing the granularity of work to be done by each thread.
- If the amount of work to be done by each thread is non-trivial (after collapsing is applied), this may improve the parallel scalability of the OMP applications.

2. OpenMP Programming: threadprivate

```
#pragma omp threadprivate(list)
```

Threadprivate

- Each thread is allowed to have its own temporary view of the shared memory.
- Each thread also has access to another type of memory that must not be accessed by other threads , called threadrivate memory
- **Shared variable**: each thread refers to the original variable.
- **Private variable:** Current thread's private version of the original variable.
- **Threadprivate**: variable appearing in threadprivae directives are threadprivate.

2. OpenMP Programming: threadprivate

```
#pragma omp threadprivate(list)
```

Threadprivate

- It specifies that named global-lifetime objects are replicated, with each thread having is own copy.
- Each copy of the *threadprivate* object is initialized once.
- A thread may not reference another thread's copy of a *threadprivate* object.
- A *threadprivate* object must not appear in any clause except the *copyin*, *copyprivate*, *schedule*, *num_threads*, and *if* clauses.
- The list is a comma-separated list of file-scope, names-scope, or static block-scope variables that do not have incomplete types.

2. OpenMP Programming: threadprivate

```
#pragma omp threadprivate(list)
```

Private vs Threadprivate

- Private variable scope is defined for only specific parallel region. *Threadprivate* is variable scope is declared across the parallel regions.
- *Firstprivate()* is used to copy the values from original variable. *Copyin()* is used to copy the values while entering into parallel region first time.
- Private variables are stored on stack most of the time. *Threadprivate* variables are stored in heap or thread local storage.

2. OpenMP Programming: threadprivate

```
#pragma omp threadprivate(list)


Copyin(list)
Copyprivate(list)
```

Data Copying clauses

- These clauses support the copying of data values from private or *threadprivate* variables on one implicit task or thread to the corresponding variables on other implicit tasks or threads in the team.
- **Copyin(list):** Copies the value of the master thread's *threadprivate* variable to the *threadprivate* variable of each other member of the team executing the parallel region.
- **Copyprivate (list)** : Broadcasts a value from the data environment of one implicit task to the data environments of the other implicit tasks belonging to the parallel region.

2. OpenMP Programming: Examples- sections

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main (void) {
int x=10, y=20,a=0,b=0;
printf("1. x=%d,y=%d,a=%d,b=%d\n",x,y,a,b);
#pragma omp parallel
{
  #pragma omp sections
  {
   #pragma omp section
    {
    a=x+y;
   int tid1=omp_get_thread_num();
    printf("tid=%d,x+y=%d\n",tid1,a);
    }
   #pragma omp section
     {
    b=x*y;
    int tid2=omp_get_thread_num();
    printf("tid=%d, x*y=%d\n",omp_get_thread_num(),b);
     }
    }
   }
printf("2.x=%d,y=%d,a=%d,b=%d\n",x,y,a,b);
return 0;
}
```

```
1. x=10,y=20,a=0,b=0
tid=5,x+y=30
tid=2, x*y=200
2.x=10,y=20,a=30,b=200
```

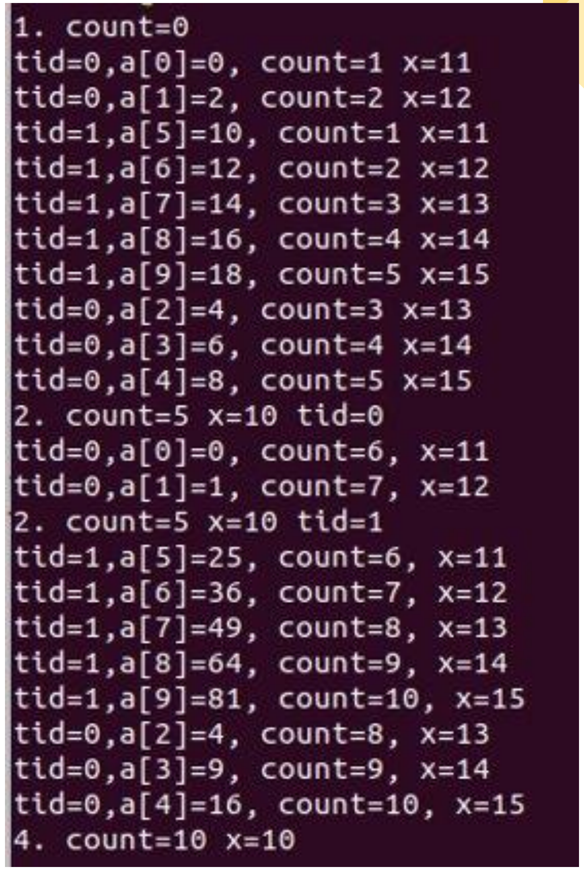
Section 1 executes x+y
Section 2 executes x*y

2. OpenMP Programming: Examples – threadprivate, copyin

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int count=0;
#pragma omp threadprivate(count)

int main (void) {
int x=10, y=20,a[10],b[10],c[10],i;
//int count=0;
for(i=0;i<10;i++)
 b[i]=c[i]=i;

printf("1. count=%d\n",count);
#pragma omp parallel num_threads(2) copyin(count)
{
 #pragma omp for schedule(static,5) firstprivate(x)
  for(i=0;i<10;i++)
   {
    int tid1=omp_get_thread_num();
    a[i]=b[i]+c[i];
    count++;
    x++;
    printf("tid=%d,a[%d]=%d, count=%d x=%d\n",tid1,i,a[i],count,x);
    }

  #pragma omp barrier
  printf("2. count=%d x=%d tid=%d\n",count,x,omp_get_thread_num());
```

```c
#pragma omp for schedule(static,5) firstprivate(x)
  for(i=0;i<10;i++)
   {
    int tid1=omp_get_thread_num();
    a[i]=b[i]*c[i];
    count++;
    x++;
    printf("tid=%d,a[%d]=%d, count=%d, x=%d\n",tid1,i,a[i],count,x);
    }
  }
#pragma omp barrier
 printf("4. count=%d x=%d\n",count,x);
printf("\n");
return 0;
}
```

Thread private and copyin - count value is retained across parallel regions
Firstprivate similar to copyin() , but the difference is that the scope of the variable is private() to parallel region.



```
1. count=0
tid=0,a[0]=0, count=1 x=11
tid=0,a[1]=2, count=2 x=12
tid=1,a[5]=10, count=1 x=11
tid=1,a[6]=12, count=2 x=12
tid=1,a[7]=14, count=3 x=13
tid=1,a[8]=16, count=4 x=14
tid=1,a[9]=18, count=5 x=15
tid=0,a[2]=4, count=3 x=13
tid=0,a[3]=6, count=4 x=14
tid=0,a[4]=8, count=5 x=15
2. count=5 x=10 tid=0
tid=0,a[0]=0, count=6, x=11
tid=0,a[1]=1, count=7, x=12
2. count=5 x=10 tid=1
tid=1,a[5]=25, count=6, x=11
tid=1,a[6]=36, count=7, x=12
tid=1,a[7]=49, count=8, x=13
tid=1,a[8]=64, count=9, x=14
tid=1,a[9]=81, count=10, x=15
tid=0,a[2]=4, count=8, x=13
tid=0,a[3]=9, count=9, x=14
tid=0,a[4]=16, count=10, x=15
4. count=10 x=10
```

2. OpenMP Programming: Examples - Collapse

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main (void) {
int i,j;
#pragma omp parallel
{
   #pragma omp for schedule(static,3) private(i,j)
     for(i=0;i<6;i++)
      for(j=0;j<5;j++)
     {
      int tid2=omp_get_thread_num();
       printf("tid=%d, i=%d  j=%d\n",omp_get_thread_num(),i,j);
     }
    }

return 0;
}
```

```
tid=0,  i=0   j=0
tid=0,  i=0   j=1
tid=0,  i=0   j=2
tid=0,  i=0   j=3
tid=0,  i=0   j=4
tid=0,  i=1   j=0
tid=0,  i=1   j=1
tid=0,  i=1   j=2
tid=0,  i=1   j=3
tid=0,  i=1   j=4
tid=0,  i=2   j=0
tid=0,  i=2   j=1
tid=0,  i=2   j=2
tid=0,  i=2   j=3
tid=0,  i=2   j=4
tid=1,  i=3   j=0
tid=1,  i=3   j=1
tid=1,  i=3   j=2
tid=1,  i=3   j=3
tid=1,  i=3   j=4
tid=1,  i=4   j=0
tid=1,  i=4   j=1
tid=1,  i=4   j=2
tid=1,  i=4   j=3
tid=1,  i=4   j=4
tid=1,  i=5   j=0
tid=1,  i=5   j=1
tid=1,  i=5   j=2
tid=1,  i=5   j=3
tid=1,  i=5   j=4
```

when the loop is executed without collapse ()

2. OpenMP Programming: Examples - Collapse

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>


int main (void) {
int i,j;
#pragma omp parallel
{
   #pragma omp for schedule(static,3) private(i,j) collapse(2)
     for(i=0;i<6;i++)
      for(j=0;j<5;j++)
     {
      int tid2=omp_get_thread_num();
       printf("tid=%d, i=%d  j=%d\n",omp_get_thread_num(),i,j);
     }
   }

return 0;
}
```

```
tid=0,  i=0   j=0
tid=0,  i=0   j=1
tid=0,  i=0   j=2
tid=0,  i=4   j=4
tid=0,  i=5   j=0
tid=0,  i=5   j=1
tid=6,  i=3   j=3
tid=6,  i=3   j=4
tid=6,  i=4   j=0
tid=3,  i=1   j=4
tid=3,  i=2   j=0
tid=3,  i=2   j=1
tid=2,  i=1   j=1
tid=2,  i=1   j=2
tid=2,  i=1   j=3
tid=7,  i=4   j=1
tid=7,  i=4   j=2
tid=7,  i=4   j=3
tid=1,  i=0   j=3
tid=1,  i=0   j=4
tid=1,  i=1   j=0
tid=1,  i=5   j=2
tid=1,  i=5   j=3
tid=1,  i=5   j=4
tid=4,  i=2   j=2
tid=4,  i=2   j=3
tid=4,  i=2   j=4
tid=5,  i=3   j=0
tid=5,  i=3   j=1
tid=5,  i=3   j=2
```

When collapse is used both the for loops are used for scheduling.

2. OpenMP Programming: Examples – copyprivate()

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int count=0;
#pragma omp threadprivate(count)

int main (void) {
int x=10, y=20,a[10],b[10],c[10],i;
//int count=0;
for(i=0;i<10;i++)
 b[i]=c[i]=i;

printf("1. count=%d\n",count);
#pragma omp parallel num_threads(2) copyin(count)
{
 #pragma omp for schedule(static,5) firstprivate(x)
  for(i=0;i<10;i++)
   {
    int tid1=omp_get_thread_num();
    a[i]=b[i]+c[i];
    count++;
    x++;
    printf("tid=%d,a[%d]=%d, count=%d x=%d\n",tid1,i,a[i],count,x);
    }

    #pragma omp barrier
    printf("2. before copyprivate count=%d x=%d tid=%d\n",count,x,omp_get_thread_num());
   #pragma omp single copyprivate(count)
    {
     count=count+20;
   }
   printf("3. after copyprivate count=%d x=%d tid=%d\n",count,x,omp_get_thread_num());

   #pragma omp for schedule(static,5) firstprivate(x)
     for(i=0;i<10;i++)
      {
       int tid1=omp_get_thread_num();
       a[i]=b[i]*c[i];
       count++;
       x++;
       printf("tid=%d,a[%d]=%d, count=%d, x=%d\n",tid1,i,a[i],count,x);
       }
     }
   #pragma omp barrier
    printf("4. count=%d x=%d\n",count,x);
   printf("\n");
   return 0;
   }
```

```
1. count=0
tid=0,a[0]=0, count=1 x=11
tid=0,a[1]=2, count=2 x=12
tid=0,a[2]=4, count=3 x=13
tid=0,a[3]=6, count=4 x=14
tid=0,a[4]=8, count=5 x=15
tid=1,a[5]=10, count=1 x=11
tid=1,a[6]=12, count=2 x=12
tid=1,a[7]=14, count=3 x=13
tid=1,a[8]=16, count=4 x=14
tid=1,a[9]=18, count=5 x=15
2. before copyprivate count=5 x=10 tid=1
2. before copyprivate count=5 x=10 tid=0
3. after copyprivate count=25 x=10 tid=1
tid=1,a[5]=25, count=26, x=11
tid=1,a[6]=36, count=27, x=12
tid=1,a[7]=49, count=28, x=13
tid=1,a[8]=64, count=29, x=14
tid=1,a[9]=81, count=30, x=15
3. after copyprivate count=25 x=10 tid=0
tid=0,a[0]=0, count=26, x=11
tid=0,a[1]=1, count=27, x=12
tid=0,a[2]=4, count=28, x=13
tid=0,a[3]=9, count=29, x=14
tid=0,a[4]=16, count=30, x=15
4. count=30 x=10
```

Copyprivate broadcast the value of the variable to all other threads

Index

- OpenMP
  - Directives : if, for, master, single, Barrier, atomic, critical,
  - Directives
    - Sections
  - Clauses
    - Threadprivate
    - Collapse
    - Threadwait
    - Copyin, copyprivate
- References

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