**National Institute of Technology Karnataka Surathkal**
**Department of Information Technology**

# IT 301 Parallel Computing

## Shared Memory Programming Technique (2)

## OpenMP :*parallel, for 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, single, critical, barrier, taskwait, atomic.* *Clauses: private, shared, firstprivate, lastprivate, reduction, nowait, ordered, schedule, collapse, num_threads, if().*

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:**

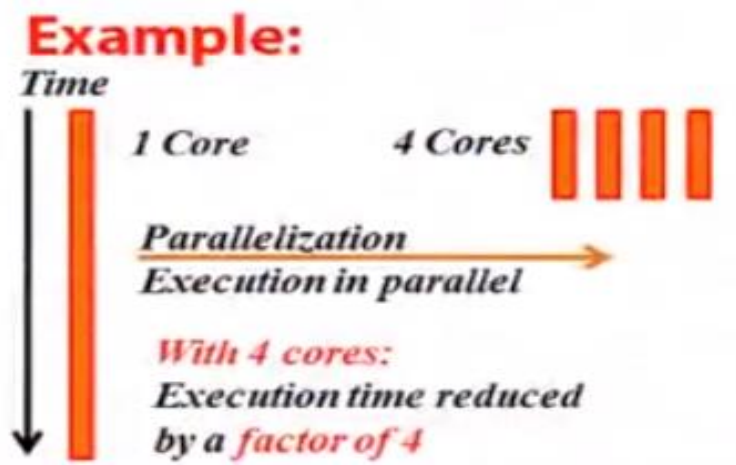
Implementation of parallel programs using OpenMP/MPI/CUDA.

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

# 1. Introduction

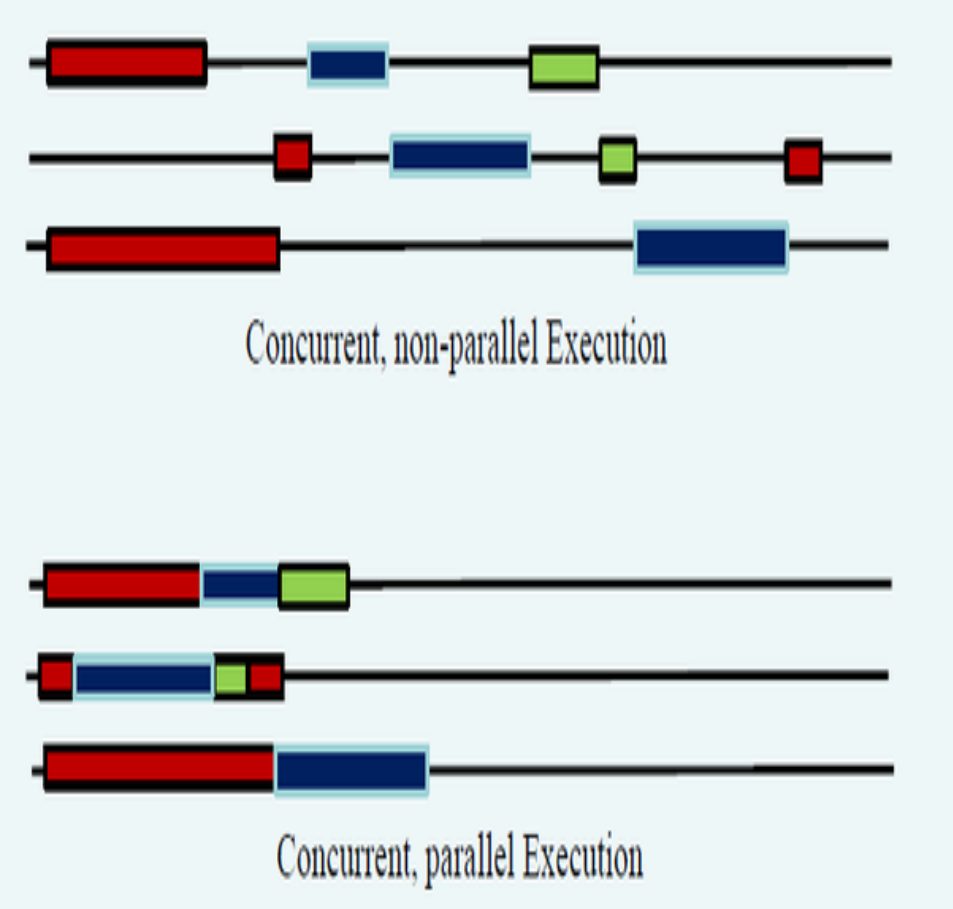
- Serial Programming
  - Develop a serial program and Optimize for performance
- Real World scenario:
  - Run multiple programs
  - Large and Complex problems
  - Time consuming
- Solution:
  - Use parallel machines
  - Use Multi-core Machines
- Why Parallel ?
  - Reduce the execution time
  - Run multiple programs

- What is parallel programming?
  - Obtain the same amount of computation with multiple cores or threads at low frequency (Fast)



# 1. Introduction

- Concurrency
  - Condition of a system in which multiple tasks are logically active at the same time ….but they may not necessarily run in parallel
- Parallelism
  - Subset of concurrency
  - Condition of a system in which multiple tasks are active at the same time and run in parallel

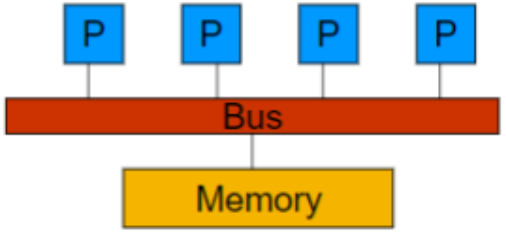


# 1. Introduction

- Shared Memory Machines
  - All processors share the same memory
  - The variables can be shared or private
  - Communication via shared memory

  Multi-threading

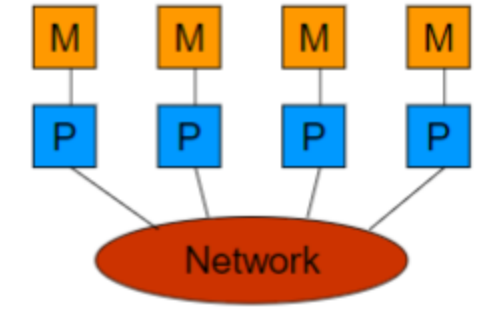
  - Portable, easy to program and use
  - Not very scalable
- OpenMP based Programming




- Distributed Memory Machines
  - Each processor has its own memory
  - The variables are Independent
  - Communication by passing messages (network)

  Multi-Processing

  - Difficult to program
  - Scalable
- MPI based Programming



# 1. OpenMP : API

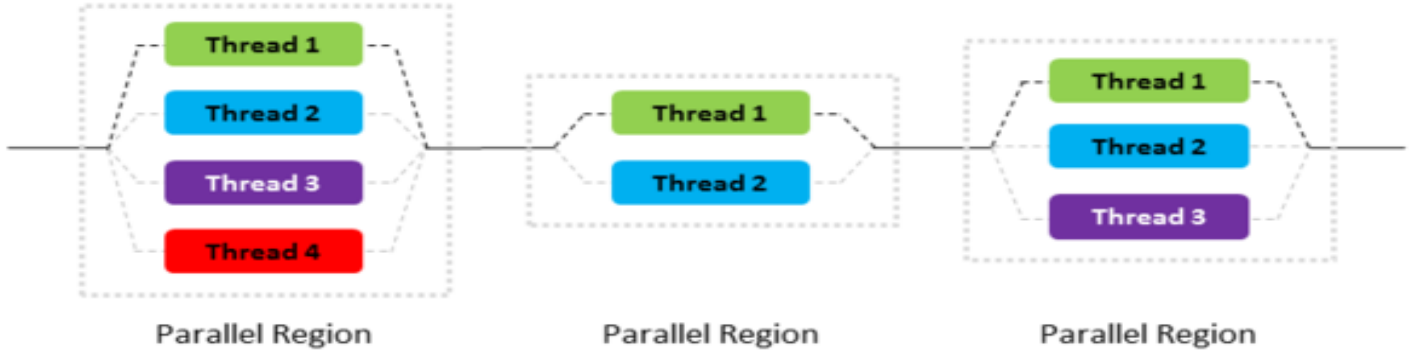
## Open Specification for Multi-Processing (OpenMP)

- Library used to divide computational work in a program and add parallelism to serial program (create threads)
- An Application Program Interface (API) that is used explicitly direct multi-threaded, shared memory parallelism
- API Components
  - Compiler Directives
  - Runtime library routines
  - Environment variables
- Standardization
  - Jointly defined and endorsed by major computer hardware and software vendors

# 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.



# 1. OpenMP

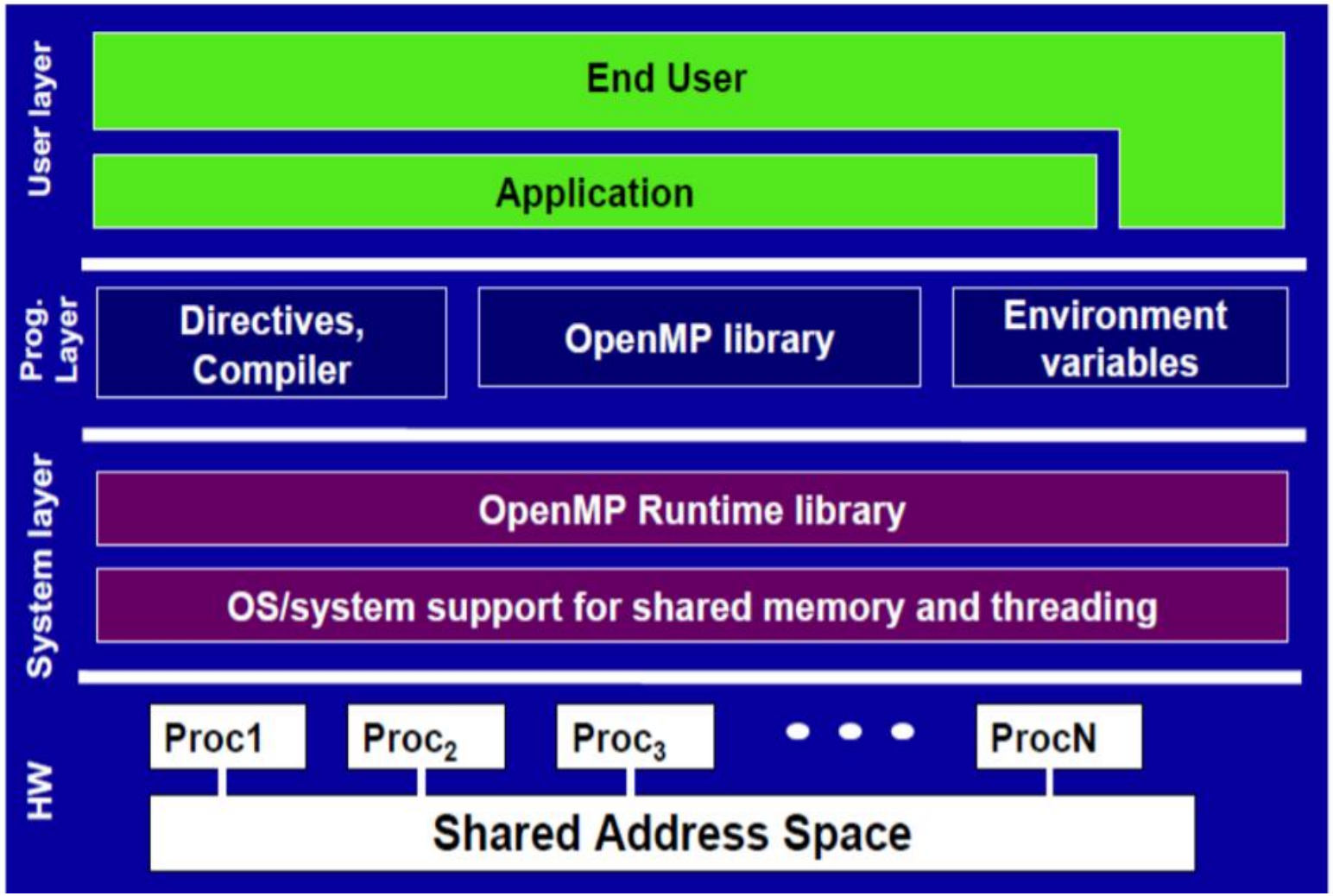
- I/O
  - OpenMP does not specify parallel I/O.
  - It is up to the programmer to ensure that I/O is conducted correctly within the context of a multithreaded program.

- Memory Model
  - Threads can “cache” their data and are not required to maintain exact consistency with real memory all the time.

  - When it is critical that all threads view a shared variable identically, the programmer is responsible for ensuring that the variable is updated by all threads as needed. ***(flush)***

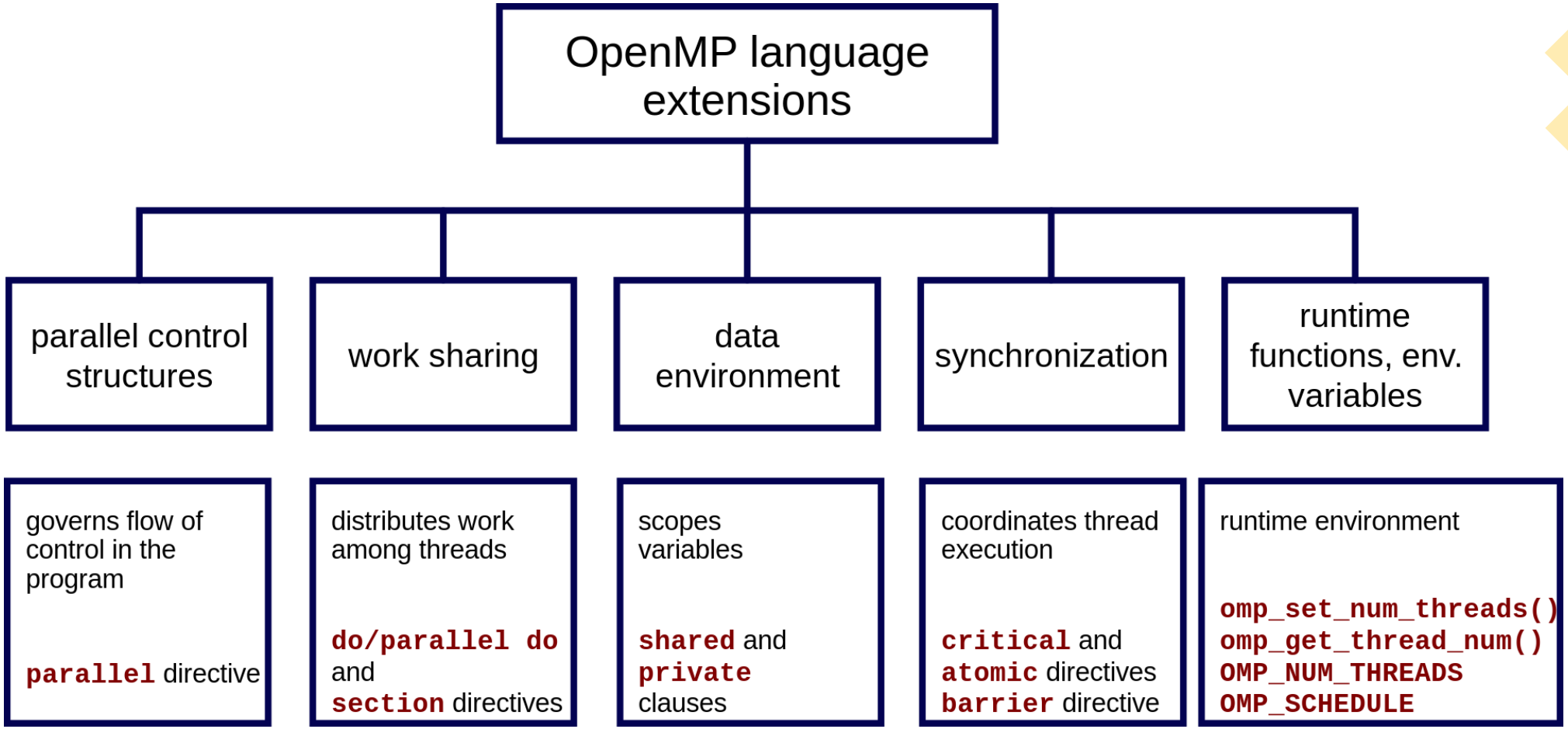
# 1. OpenMP : API

- Architecture



# 1. OpenMP : API

- OpenMP Language Extensions



# 2. OpenMP Programming

- Directives
  - An OpenMP executable directive applies to the succeeding structured block or an OpenMP Construct. A “structured block” is a single statement or a compound statement with a single entry at the top and a single exit at the bottom.

- Clauses
  - Not all the clauses are valid on all directives. The set of clauses that is valid on a particular directive is described with the directive. Most of the clauses accept a comma-separated list of list items. All list items appearing in a clause must be visible.

- Runtime Library Routines
  - Execution environment routines affect and monitor threads, processors, and the parallel environment. Lock routines support synchronization with OpenMP locks. Timing routines support a portable wall clock timer. Prototypes for the runtime library routines are defined in the file “omp.h”.

# 2. OpenMP Programming : *Parallel*

## Directives

- Parallel
- For
- Sections
- Single
- Task
- Master
- Critical
- Barrier
- Taskwait
- Automic
- Flush
- Ordered
- Threadprivate

## Clauses

- Default (shared/none)
- Shared
- Private
- Firstprivate
- Lastprivate
- Reduction
- Copyin
- copyprivate

## Run time variables

- omp_set_num_threads
- omp_get_num_threads
- omp_get_max_threads
- omp_get_thread_num(void)
- …….etc

# 2. OpenMP Programming

```
%%Program hello.c
#include <stdio.h>
#include <omp.h>

int main(void)
{
    #pragma omp parallel
    printf("Hello, world.\n");
    return 0;
}
```

Compiling the program

```
$ gcc -fopenmp hello.c -o hello
```

Output

```
Hello, world.
Hello, world.
```

# 2. OpenMP Programming

- Directives
  - An OpenMP executable directive applies to the succeeding structured block or an OpenMP Construct. A “structured block” is a single statement or a compound statement with a single entry at the top and a single exit at the bottom.
  - They are case sensitive
  - Starts with #pragma omp
  - Directives cannot be embedded in embedded within continued statements, and statements cannot be embedded within directives.
  - Only one directive-name can be specified per directive

# 2. OpenMP Programming: Directives

The parallel construct forms a team of threads and starts parallel execution.

**#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*)

**Restrictions**

- A program which branches into or out of a parallel region is non-conforming.
- A program must not depend on any ordering of the evaluations of the clauses of the parallel directive, or on any side effects of the evaluations of the clauses.
- At most one ***if*** clause can appear on the directive.
- At most one ***num_threads*** clause can appear on the directive. The ***num_threads*** expression must evaluate to a positive integer value.

# 2. OpenMP Programming: Directives

**#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*)

- A team of threads is created to execute the parallel region
- A thread which encounters the parallel construct becomes master thread
- The thread id of master is 0
- All threads including master thread executes parallel region
- **omp_get_thread_num()** provides thread id
- There is implied barrier at the end of a parallel region.
- If a thread in a team executing a parallel region encounters another parallel directive, it creates a new team, and that thread is master of new team.

# 2. OpenMP Programming: Directives

**#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*)

- If execution of a thread terminates while inside a parallel region, execution of all threads in all teams terminates.
- The order of termination of threads is unspecified
- All the work done by a team prior to any barrier which the team has passed in the program is guaranteed to be complete.
- The amount of work done by each thread after the last barrier that it passed and before it terminates is unspecified

# 2. OpenMP Programming: Clauses

**#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*)

**If Clause**

- A structured block is executed in parallel if the evaluation of ***if()*** clause is evaluated as true.
- A missing if clause is equivalent to an if clause that evaluates true.
- At most one ***if ()*** clause can appear on the directive.

# 2. OpenMP Programming : if clause

```
%%Program hello.c
%% if () clause

#include <stdio.h>
#include <omp.h>

int main(void)
{
int par=1;
 #pragma omp parallel  if(par==1)num_threads(4)
   printf("Hello, world.\n");
   return 0;
}
```

What is the result of the program?

Hello, world.

OR

Hello, world.
Hello, world.
Hello, world.
Hello, world.

# 2. OpenMP Programming: clauses num_threads

**#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*)

**num_threads** ( ) Clause

- **num_threads** must evaluate to positive integer.
- It sets number of threads for the execution of parallel region
- Some of the execution environment routines
  - To set number of threads

    **void omp_set_num_threads(int num_threads);**
  - To get number of threads

    **int omp_get_num_threads(void);**
  - To find number of threads which can for a team

    **int omp_get_max_threads(void);**
  - To get thread id

    **int omp_get_thread_num(void);**

# 2. OpenMP Programming : num_threads clause

```
%%Program hello.c
%% if () clause

#include <stdio.h>
#include <omp.h>

int main(void)
{
int par=1;
 #pragma omp parallel  if(par==1) num_threads(6)
   printf("Hello, world.\n");
   return 0;
}
```

What is the result of the program?

Hello, world.

OR

Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.

# 2. OpenMP Programming: num_threads

```
%%Program hello.c
%% if () clause

#include <stdio.h>
#include <omp.h>

int main(void)
{
int par=0;
 #pragma omp parallel  if(par==1) num_threads(6)
   printf("Hello, world.\n");
   return 0;
}
```

What is the result of the program?

Hello, world.

OR

Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.

# 2. OpenMP Programming: Clauses

**#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*)

**Default(shared/none) , shared(list) Clause**

- **default (shared)** clause causes all variables referenced in the construct which have implicitly determined sharing attributes to be shared.
- **default(none)** clause requires that each variable which is referenced in the construct, and that does not have a predetermined sharing attribute, must have its sharing attribute explicitly determined by being listed in a data sharing attribute clause
- **shared(list)** : One or more list items must be shared among all the threads in a team.

# 2. OpenMP Programming: Directives

**#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*)

**private (list) Clause**

- **private (list)** clause declares one or more list items must be private to a thread.
- A list item that appears in the **reduction** clause of a parallel construct must not appear in a **private** clause on a work-sharing construct.

# 2. OpenMP Programming: Clauses – default(shared)

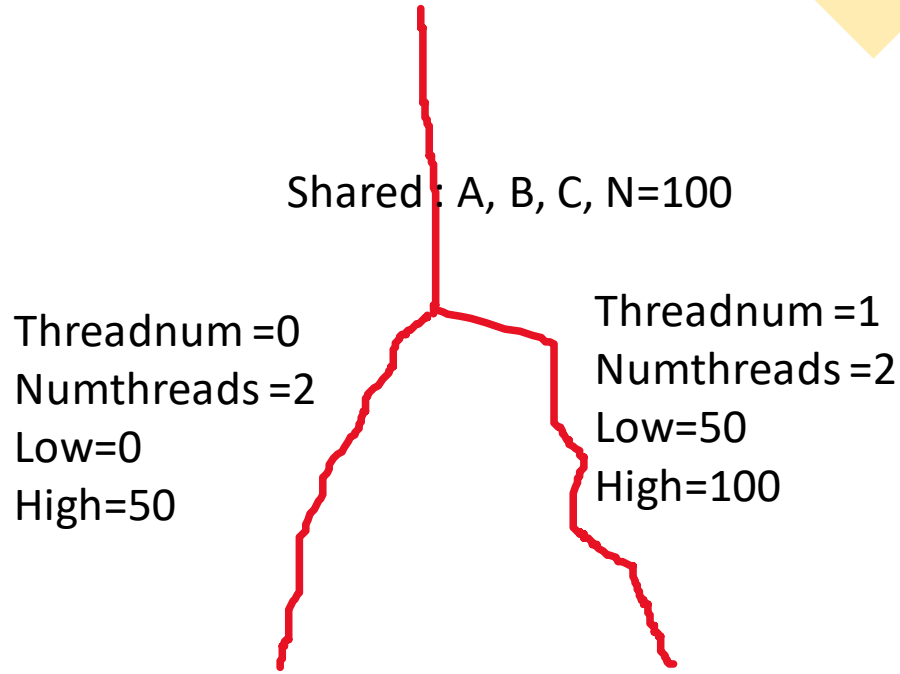
```
// N: total number of iterations
#pragma omp parallel default(shared)
private(threadnum, numthreads, low,high, i)
{
  int threadnum = omp_get_thread_num(),
    numthreads = omp_get_num_threads();

  int low = N*threadnum/numthreads,
    high = N*(threadnum+1)/numthreads;

  for (i=low; i<high; i++)
    a[i]=b[i]+c[i]
    }
```

Shared : A, B, C

Other variables are default



# 2. OpenMP Programming: Directives – Default(none)

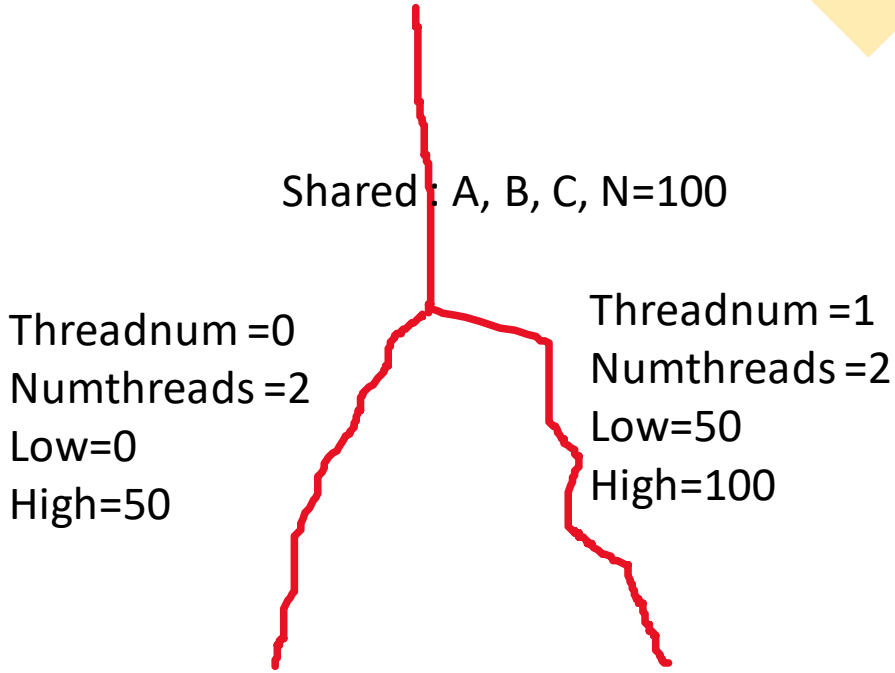
```
// N: total number of iterations
#pragma omp parallel default(none)
 shared(a,b,c,N) private(threadnum, numthreads,
low,high, i)
{
  int threadnum = omp_get_thread_num(),
    numthreads = omp_get_num_threads();

  int low = N*threadnum/numthreads,
    high = N*(threadnum+1)/numthreads;

  for (i=low; i<high; i++)
    a[i]=b[i]+c[i]
    }
```

Shared : A, B, C

Other variables are default



# 2. OpenMP Programming: Directives - shared

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

int main (void) {
int x=0;
#pragma omp parallel shared(x)
    {
    int tid=omp_get_thread_num();
    x=x+1;
    printf("Thread [%d] value of x is %d \n",tid,x);
    }


return 0;
}
```

```
Thread [1] value of x is 1
Thread [0] value of x is 2
Thread [2] value of x is 3
Thread [3] value of x is 4
```

```
Thread [0] value of x is 2
Thread [1] value of x is 1
Thread [2] value of x is 3
Thread [3] value of x is 4
```

Since the x is shared, the change in one thread is visible to all other threads too.

# 2. OpenMP Programming: Directives - private

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

int main (void) {
int x=0;
printf("x value ouside parallel:%d\n",x);
#pragma omp parallel private(x)
    {
    x=10;
    int tid=omp_get_thread_num();
    x=x+1;
    printf("Thread [%d] value of x is %d \n",tid,x);
    }


return 0;
}
```

```
x value ouside parallel:0
Thread [0] value of x is 11
Thread [2] value of x is 11
Thread [1] value of x is 11
Thread [3] value of x is 11
```

The variable x is private here . So the change of value performed in one thread is not visible to other threads.

# 2. OpenMP Programming: Directives - Private

```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) private(x) private(tid)
    {

    int tid=omp_get_thread_num();
    printf("\n 1. Thread [%d] value of x is %d \n",tid,x);
    x=15;
    printf("\n 2. Thread [%d] value of x is %d \n",tid,x);
    x=x+1;
    printf("\n 3. Thread [%d] value of x is %d \n",tid,x);
    }
return 0;
}
```

```
x value ouside parallel:10

 1. Thread [1] value of x is 6683504

 2. Thread [1] value of x is 15

 3. Thread [1] value of x is 16

 1. Thread [3] value of x is 17143736

 2. Thread [3] value of x is 15

 3. Thread [3] value of x is 16

 1. Thread [2] value of x is 6683600

 2. Thread [2] value of x is 15

 3. Thread [2] value of x is 16

 1. Thread [0] value of x is 0

 2. Thread [0] value of x is 15

 3. Thread [0] value of x is 16
```

The variable x is private here. The value of x is 10 before parallel region. Since x is private, the value 10 is not reflected in threads. When the parallel region is entered, the x contains some garbage value as it is not initialed .

# 2. OpenMP Programming: Directives - Shared

```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();
    printf("\n 1. Thread [%d] value of x is %d \n",tid,x);
    x=15;
    printf("\n 2. Thread [%d] value of x is %d \n",tid,x);
    x=x+1;
    printf("\n 3. Thread [%d] value of x is %d \n",tid,x);
    }
return 0;
}
```

```
x value ouside parallel:10

 1. Thread [2] value of x is 10

 2. Thread [2] value of x is 15

 3. Thread [2] value of x is 16

 1. Thread [3] value of x is 10

 2. Thread [3] value of x is 15

 3. Thread [3] value of x is 16

 1. Thread [1] value of x is 10

 2. Thread [1] value of x is 15

 3. Thread [1] value of x is 16

 1. Thread [0] value of x is 10

 2. Thread [0] value of x is 15

 3. Thread [0] value of x is 16
```

X is shared. X is assigned value 15 inside parallel region.
Each thread is assigning value as 15. And updating it with x=x+1; But update is not getting reflected in other threads.

# 2. OpenMP Programming: Directives - Shared

```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();
    printf("\n 1. Thread [%d] value of x is %d \n",tid,x);
    //x=15;

    printf("\n 2. Thread [%d] value of x is %d \n",tid,x);
    x=x+1;

    printf("\n 3. Thread [%d] value of x is %d \n",tid,x);
    }
return 0;
}
```

```
x value ouside parallel:10

1. Thread [1] value of x is 10

2. Thread [1] value of x is 10

3. Thread [1] value of x is 11

1. Thread [3] value of x is 10

2. Thread [3] value of x is 11

3. Thread [3] value of x is 12

1. Thread [0] value of x is 10

2. Thread [0] value of x is 12

3. Thread [0] value of x is 13

1. Thread [2] value of x is 10

2. Thread [2] value of x is 13

3. Thread [2] value of x is 14
```

x is shared. No assignment in the parallel region.
Update is getting reflected in all other threads.
Synchronization is job of programmer.

# 2. OpenMP Programming: Directives

**#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*)

**firstprivate (list) Clause**

- **firstprivate (list)** clause declares one or more list items to be private to a thread and initializes each of them with that the corresponding original item has when the construct is encountered.
- For a **firstprivate** clause on a **parallel** construct, the initial value of the new item is the value of the original list item that exists immediately prior to the **parallel** construct for the thread that encounters the construct.

# 2. OpenMP Programming: Directives - Private

```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) private(x) private(tid)
    {

    int tid=omp_get_thread_num();
    printf("\n 1. Thread [%d] value of x is %d \n",tid,x);
    x=15;
    printf("\n 2. Thread [%d] value of x is %d \n",tid,x);
    x=x+1;
    printf("\n 3. Thread [%d] value of x is %d \n",tid,x);
    }
return 0;
}
```

```
x value ouside parallel:10

 1. Thread [1] value of x is 6683504

 2. Thread [1] value of x is 15

 3. Thread [1] value of x is 16

 1. Thread [3] value of x is 17143736

 2. Thread [3] value of x is 15

 3. Thread [3] value of x is 16

 1. Thread [2] value of x is 6683600

 2. Thread [2] value of x is 15

 3. Thread [2] value of x is 16

 1. Thread [0] value of x is 0

 2. Thread [0] value of x is 15

 3. Thread [0] value of x is 16
```

X is private. While entering parallel region the x gets assigned with some value.

# 2. OpenMP Programming: Directives - Private

```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) private(x) private(tid)
    {

    int tid=omp_get_thread_num();
    printf("\n 1. Thread [%d] value of x is %d \n",tid,x);

    x=x+1;

    printf("\n 3. Thread [%d] value of x is %d \n",tid,x);
    }
return 0;
}
```

```
x value ouside parallel:10

 1. Thread [2] value of x is 6683600

 3. Thread [2] value of x is 6683601

 1. Thread [0] value of x is 0

 3. Thread [0] value of x is 1

 1. Thread [1] value of x is 6683504

 3. Thread [1] value of x is 6683505

 1. Thread [3] value of x is 14195704

 3. Thread [3] value of x is 14195705
```

Shared x. x is already with some garbage value. X is private to each thread.

# 2. OpenMP Programming: Directives - Private

```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) firstprivate(x) private(tid)
    {

    int tid=omp_get_thread_num();
    printf("\n 1. Thread [%d] value of x is %d \n",tid,x);

    x=x+1;

    printf("\n 3. Thread [%d] value of x is %d \n",tid,x);
    }
return 0;
}
```

```
x value ouside parallel:10

1. Thread [2] value of x is 10

3. Thread [2] value of x is 11

1. Thread [1] value of x is 10

3. Thread [1] value of x is 11

1. Thread [3] value of x is 10

3. Thread [3] value of x is 11

1. Thread [0] value of x is 10

3. Thread [0] value of x is 11
```

First private works as follows.

1. assign the value as in main thread before parallel region.
2. thread is private once it enters to parallel region.

# 2. OpenMP Programming: Directives

A loop Construct specifies that the iterations of loops will be distributed among and executed by the encountering team of threads.

**#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***

Example

```
#pragma omp parallel
  #pragma omp for
  for (i=0; i<N; i++) {
    // do something with i
  }
```

# Index

- OpenMP
  - Program Structure
  - Directives : *parallel* , *for*
  - Clauses
    - If
    - Private
    - Shared
    - Default(shared/none)
    - Firstprivate
    - Num_threads

# 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