National Institute of Technology Karnataka Surathkal
Department of Information Technology

IT 301 Parallel Computing

Shared Memory Programming Technique

OpenMP :*parallel, for*

Dr. Geetha V

Assistant Professor

Dept of Information Technology

NITK Surathkal

Index

- Introduction
  - Shared memory and Distributed Memory
  - OpenMP
- OpenMP
  - Program Structure
  - Directives : *parallel* , *for*
- 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, single, critical, barrier, taskwait, atomic. Clauses: private, shared, firstprivate, lastprivate, reduction, nowait, ordered, schedule, collapse, num_threads, shared, 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:

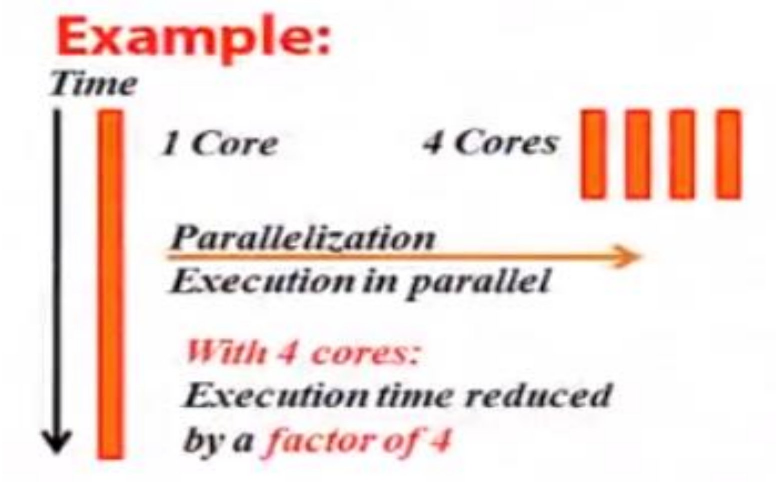
Implementation of parallel programs using OpenMP/MPI/CUDA.

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

1. Introduction

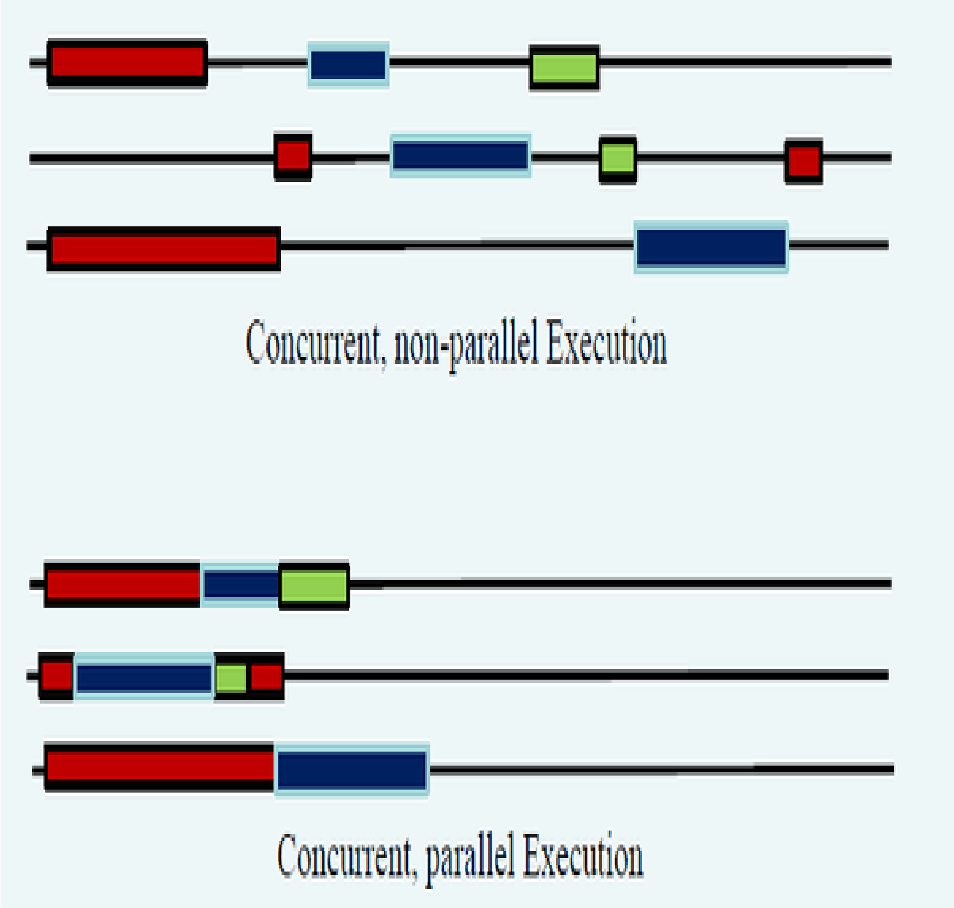
- Serial Programming
  - Develop a serial program and Optimize for performance
- Real World scenario:
  - Run multiple programs
  - Large and Complex problems
  - Time consuming
- Solution:
  - Use parallel machines
  - Use Multi-core Machines
- Why Parallel ?
  - Reduce the execution time
  - Run multiple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

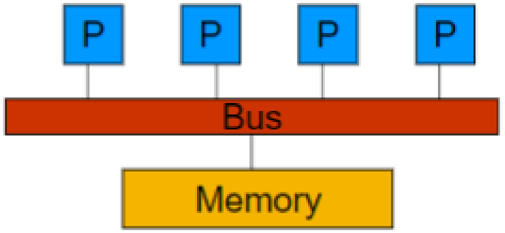


1. Introduction

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

  Multi-threading

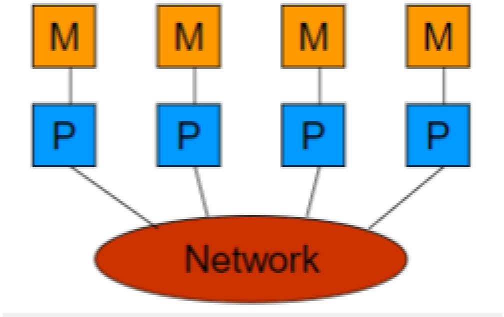
  - Portable, easy to program and use
  - Not very scalable
- OpenMP based Programming



- 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)

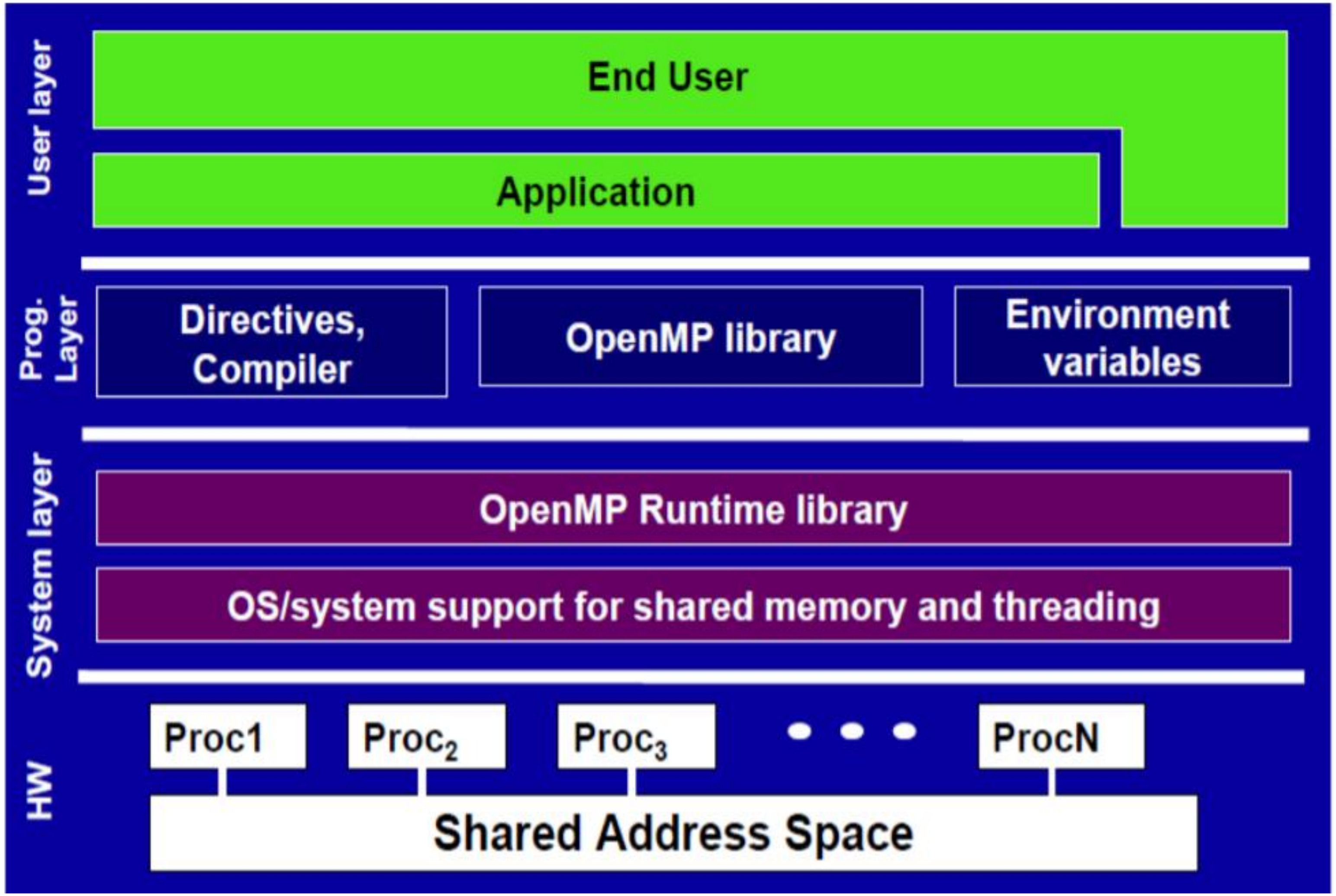
- Library used to divide computational work in a program and add parallelism to serial program (create threads)
- An Application Program Interface (API) that is used explicitly direct multi-threaded, shared memory parallelism
- API Components
  - Compiler Directives
  - Runtime library routines
  - Environment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

1. OpenMP : API

- Open Specification for Multi-Processing (OpenMP)
- History
  - In 1991, Parallel Computing Forum (PCF) group invented a set of directives for specifying loop parallelism in Fortran Programs
  - X3H5, an ANSI subcommittee developed an ANSI standard based on PCF
  - In 1997, the first version of OpenMP for Fortran was defined by OpenMP Architecture Review Board.
  - Binding for C/C++ was introduced later.
  - Version 3.0 is available since 2008.
  - Version 4.0 is available since 2013.
  - Current version is 5.0, released in November 2018

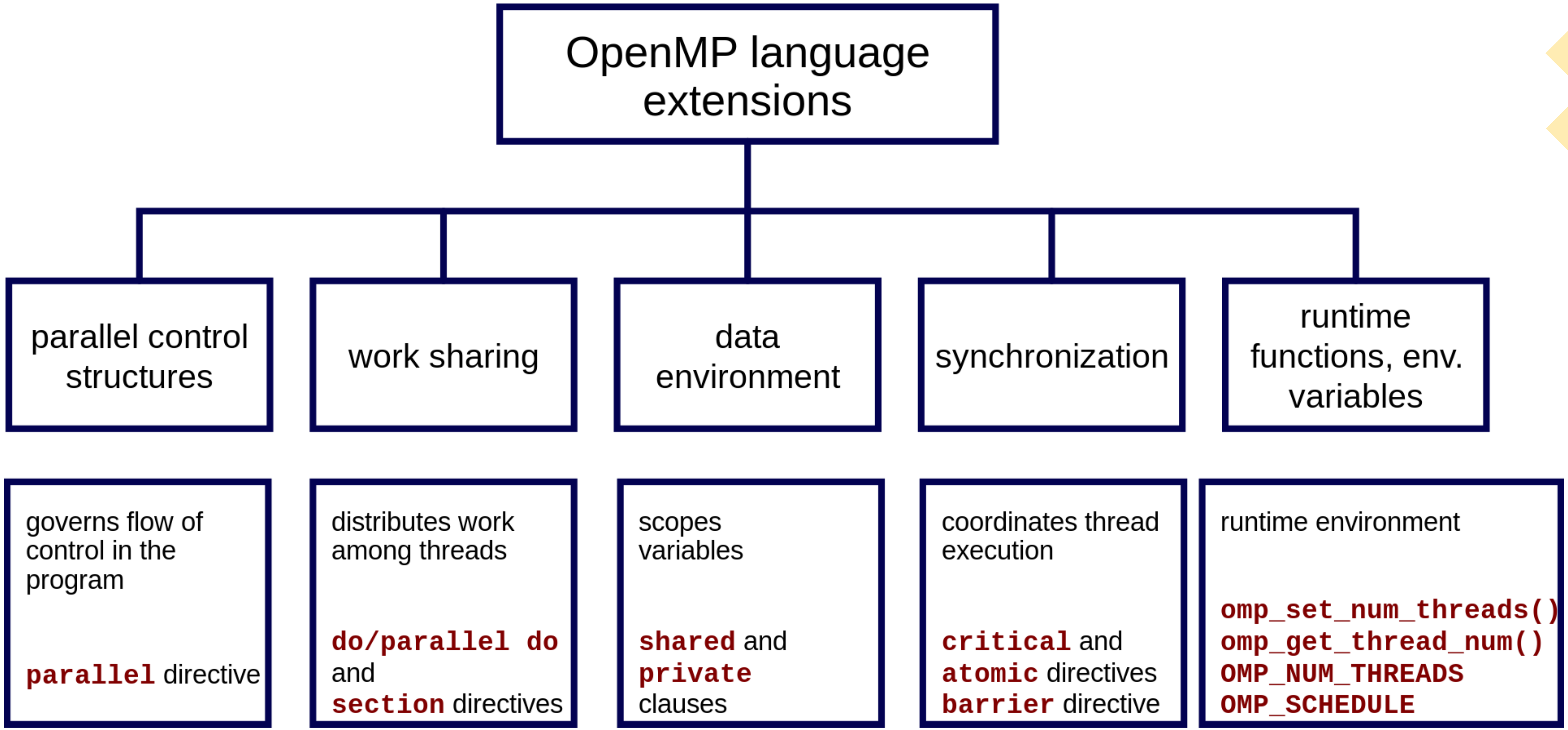
1. OpenMP : API

- Architecture



1. OpenMP : API

- OpenMP Language Extensions



1. OpenMP

Threads and Process

- A process is an instance of a computer program that is being executed. It contains the program code and its current activity.
- A thread of execution is the smallest unit of processing that can be scheduled by an operating system.
- Differences between threads and processes:
  - A thread is contained inside a process.
  - Multiple threads can exist within the same process and share resources such as memory.
  - The threads of a process share the latter's instructions (code) and its context (values that its variables reference at any given moment).
  - Different processes do not share these resources.

http://en.wikipedia.org/wiki/Process_(computing)

1. OpenMP

Process

- A process contains all the information needed to execute the program
  - Process ID
  - Program code
  - Data on run time stack
  - Global data
  - Data on heap

Each process has its own address space.

- In multitasking, processes are given time slices in a round robin fashion.

  – If computer resources are assigned to another process, the status of the present process must be saved, in order that the execution of the suspended process can be resumed later.

1. OpenMP

Threads

- Thread model is an extension of the process model.
- Each process consists of multiple independent instruction streams (or threads) that are assigned computer resources by some scheduling procedure.
- Threads of a process share the address space of this process.
  - Global variables and all dynamically allocated data objects are accessible by all threads of a process
- Each thread has its own run-time stack, register, program counter.
- Threads can communicate by reading/writing variables in the common address space.

1. OpenMP

OpenMP Programming Model

- Shared memory, thread-based parallelism
  - – OpenMP is based on the existence of multiple threads in the shared memory programming paradigm.
  - – A shared memory process consists of multiple threads.

- Explicit Parallelism
  - – Programmer has full control over parallelization. OpenMP is not an automatic parallel programming model.

- Compiler directive based
  - – Most OpenMP parallelism is specified using compiler directives which are embedded in the source code.

1. OpenMP

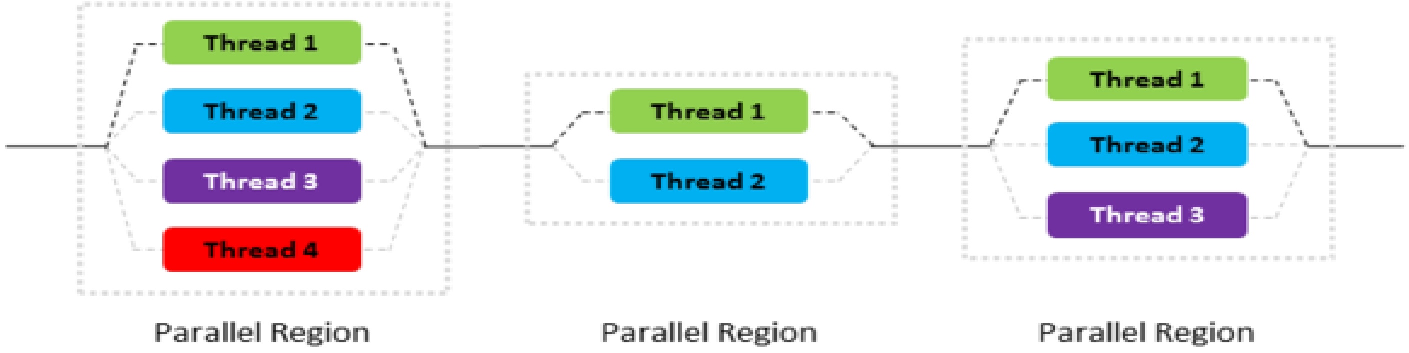
OpenMP is not

- Necessarily implemented identically by all vendors
- Meant for distributed-memory parallel systems (it is designed for shared address spaced machines)
- Guaranteed to make the most efficient use of shared memory
- Required to check for data dependencies, data conflicts, race conditions, or deadlocks
- Required to check for code sequences
- Meant to cover compiler-generated automatic parallelization and directives to the compiler to assist such parallelization
- Designed to guarantee that input or output to the same file is synchronous when executed in parallel.

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)*

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.
- 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

- 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 **<u>#pragma omp</u>**
  - 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 thead 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

```
%%Program hello.c
%%Num_threads() sets nuber of threads

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

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

Compiling the program

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

Output

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

2. OpenMP Programming: Directives - Loop

```
#pragma omp parallel

#pragma omp parallel
{
  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++)
    // do something with i
}
```

```
# pragma omp parallel for

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

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
  }
```

2. OpenMP Programming

- *# pragma omp for* Restrictions
  - The values of the loop control expressions of the loop associated with the loop directive must be the same for all the threads in the team.
  - Only a single **schedule** clause can appear on a loop directive.
  - *chunk_size* must be a loop invariant integer expression with a positive value.
  - The value of the *chunk_size* expression must be the same for all threads in the team.
  - When **schedule(runtime)** is specified, chunk_size must not be specified.
  - Only a single **ordered** clause can appear on a loop directive.
  - The **ordered** clause must be present on the loop construct if any **ordered** region ever binds to a loop region arising from the loop construct.
  - The loop iteration variable may not appear in a **threadprivate** directive.
  - The *for-loop* must be a structured block, and in addition, its execution must not be terminated by a **break** statement.
  - The *for-loop* iteration variable var must have a signed integer type.
  - Only a single **nowait** clause can appear on a **for** directive.

Index

- Introduction
  - Shared memory and Distributed Memory
  - OpenMP
- OpenMP
  - Program Structure
  - Directives : parallel, for
- 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