Imp :-

1) Explain OpenMP → from my notes
2) Loop Scheduling → my notes
3) Performance pitfalls → Page 15 ~~21~~ of pdf.
4) Various clauses of OpenMP Worksharing for loops (Page 7-8 of maam pdf).
5) Data Scoping (Page - 6)
7) Serialization vs False Sharing → (my notes)

HPC

Unit-4

Open MP :- (Open ~~Source~~ Multi-Processing).

Open MP is an application programming Interface (API) that supports multi-platform shared memory multiprocessing programming in C, C++ and Fortran.

It is used for multithreading (to get maximum performance we use multithread concept with open MP).

So, OpenMP is a set of compiler directives as well as an API for programs written in C, C++ or FORTRAN. that provides support for parallel programming in shared memory environments.

* #include <omp.h>
↳ header to include openMP features.

We use open MP when we need to execute parallel work.

History :-

1) In 1997, first version of openMP was developed by ~~ANSI~~ for Fortran by openMP Architecture Review Board. Binding for C/C++ was introduced later.
2) Version 3.1 is available since 2011.

OpenMP Architecture Review Board → Combination of various companies (Intel, HP, IBM, Oracle, ARM, etc)

Date
Page

Example Code :-

```
#include<omp.h>
void main () {
int x[5] = [3,2,1,4,6];
int y[5] = [6,7,8,9,5];
int z[5];
int tid;
#pragma omp parallel numthread(5) {
    tid = omp_get_thread_num();
    z[tid] = x[tid] + y[tid];
    printf("z[%d] = %d\n", tid, z[tid])
}
```

O/P :-

```
z[0] = 9
z[1] = 9
z[2] = 9
z[3] = 13
z[4] = 11
```

Order is not fixed.

Format in Fortran :-

```
!$ omp PARALLEL
{


}
```

← end braces is compulsory.

We can use similar code for for loop too.

Steps to create Parallel program :-

1) Include header file
2) Specify parallel region
3) Set no. of threads

↳ We need to figure out most consuming part of code & place parallel statement before this.

Note → In openMP, there is no need to fetch again and again (no need to recreate). But in MPI we need to do it.

Adv. of openMP :-

1) Ease of use We can easily parallelise our code using openMP.

Why we use ?
1) Portable
2) Simple & Quick
3) Support both fine grained & coarse grained
4) Compact API ⟶ Simple & limited set of ~~derivatives~~ directives
   ↳ Not automatic parallelization
5) Incremental parallelisation.

Name → Akash Kumar
Reg. No → RA2011003030001

Assignment - 3

Q.1. Explain about loop scheduling and its types.

Ans → Loop Scheduling is a technique used in parallel computing to distribute the iterations of a loop among multiple threads or processors. The goal of loop scheduling is to balance the workload among threads, reduce contention, and improve the overall performance of parallel program.

Following are various types of loop scheduling :-

(i) Static Scheduling :- In this, the iterations of the loop are divided into a predefined fixed number of chunks and each chunk is assigned to a different thread or processor.

(ii) Dynamic Scheduling :- In this, the iterations of the loop are divided dynamically, based on the current workload of each thread. Each thread is assigned a small no. of iterations at a time, and when it finishes those iterations, it request more from a global work queue.

(iii) Guided Scheduling :- It is a type of dynamic where size of chunk is proportional to the no. of assigned iteration divided by no. of threads and the chunk size will be decreased too.

$$\text{Chunk size} \propto \frac{\text{Remaining iteration}}{\text{No. of threads}}$$

(iv) Runtime Scheduling :- In this ~~chunk~~ size, ICV (Internal Control variable) will decide the chunk size.

(v) Auto scheduling :- In this compiler will decide chunk size automatically based on the characteristics of the loop and the hardware available.

Q.2. Writ about impact of openMP work sharing construct.

Ans → OpenMP is widely used programming model for parallel computing on shared memory architecture. Following are the impact of openMP ~~on~~ work sharing construct :-

(i) Improved Performance :- By distributing the work among multiple threads, the overall execution time can be reduced leading to improved performance.

(ii) Increased scalability :- The work sharing construct can help to improve the scalability of parallel programs by allowing them to use more threads or processor as size of problem increases.

(iii) Load balancing :- It can help to balance the workload among threads, ensuring that each threads is executing a roughly ~~enough~~ equal amount of work.

(iv) Data Sharing :- OpenMP provides a set of ~~derival~~ directives for managing data sharing among threads, allowing them to access shared data in a safe and efficient manner.

(v) Code of complexity :- The use of work sharing construct can sometimes increases the complexity of code.

Q.3. Writ a short not on Wave front parallelization.

Ans → Wavefront parallelization is a technique used in parallel computing to distribute tasks among ~~a~~ multiple processor or computing nodes. It is also known as data parallelism.

In this the computation is divided into a set of independent tasks or data elements that can be executed concurrently. these task are arranged in a wave-like pattern, where each row or wave represents a set of tasks that can be executed in parallel. Each wave is processed by a different processor or computing node, and the results are then combined to obtain the final output.

Q.4. What is the difference between serialization and false sharing?

Ans → Serialization and false sharing are two performance issues that can occur in parallel computing, particularly when multiple threads or process access shared memory.

Serialization refers to the situation where multiple threads or process access a shared resource or a critical section of a code one at a time. In other words, they must take turns accessing the resource and they cannot do so simultaneously. Serialization can significantly reduce the perfomance of a parallel program as it can lead to contention and delays in accessing shared resources.

On the other hand false sharing occur when multiple threads or processes access different variable or data structures that happen to be located on the same cache line or memory block. This can result in cache thrasing, where cache line is constantly invalidated and reloaded by different processors, even though they are not actually accessing the same data. It can cause significant perfomance degradation as it can lead to unnecessary cache eviction and bus traffic.