**Parallel Processing (CS417L)**

**Previously we looked at:**

* Introduction to OpenMP
* Directives in OpenMP

**Today we’ll be studying:**

* Using the Sections Construct in OpenMP
* Using the Task Directives in OpenMP
* Variable Sharing in Sections & Task Directives

**Sections Construct:**

A construct in OpenMP, used for executing different, independent tasks in parallel within a single parallel region.

* The number of threads must be defined, and sections are assigned to threads at the start of execution.
* All sections must finish before exiting the parallel region.
* Variables are shared by default thus you must specify *private* or *firstprivate* for exclusive copies.

#pragma omp parallel **sections**

{

**#pragma omp section**

**{**

**// section 1**

**}**

**#pragma omp section**

**{**

**// section 2**

**}**

}

**Task Directive:**

A directive in OpenMP that allows for dynamic allocation of tasks to threads, making it easier to manage workload distribution.

* Useful in scenarios where the number of tasks is not fixed or predictable.
* Used to handle Irregular Workloads

#pragma omp parallel

{

#pragma omp single

{

#pragma omp task

{

// Task code

}

#pragma omp task

{

// Another task code

}

}

}

**Variable Sharing:**

* *Shared*

#pragma omp parallel sections

{

#pragma omp section

{

// Access shared variable

}

}

* *Private*

#pragma omp parallel sections private(myVar)

{

#pragma omp section

{

myVar = ...; // Each thread has its own myVar

}

}

* *First Private*

int x = 10;

#pragma omp parallel sections firstprivate(x)

{

#pragma omp section

{

printf("%d\n", x); // Each thread gets its own copy initialized to 10

}

}

* *Last Private*

int lastValue;

#pragma omp parallel sections lastprivate(lastValue)

{

#pragma omp section

{

lastValue = ...; // This value will be captured after the section ends

}

}

**Summary of When to Use Each**

* Use Sections when you have clear, independent tasks that can be executed simultaneously and are part of a fixed workload (e.g., performing three different computations in parallel).
* Use Tasks when dealing with irregular workloads or when the workload cannot be determined at compile time (e.g., recursive algorithms or dynamically created tasks that depend on previous results).

**Importance:**

* Using sections or tasks can improve performance by efficiently utilizing CPU resources.
* Both constructs enable better thread management and reduce overhead compared to traditional threading models.
* Ideal for dividing independent work into sections or handling complex workloads that may vary in size or nature.