PARALLEL SUM REDUCTION

The goal of this work is to obtain the sum of all the values of a large 1D array using Parallel Sum Reduction Operator. There are number of ways for computing the sum of the elements of an array but parallel sum reduction approach seems to be a better solution than sequential approach. Here basically we divide the array into multiple partitions, each thread will calculate the sum of the elements reside in that partition and finally adds up the results. Reduction operator reduce the collection of values into a single value.

PARALLEL SUM REDUCTION USING OpenMP

Implementation: The basic idea behind the working of any OpenMP program is that, a normal sequential program can converted into OpenMP program / parallel program by adding just one line of statement: " *#pragma omp parallel*". This is called *parallel construct*, the following lines of code after parallel construct will be executed by multiple threads. This is a very basic idea of how a simple OpenMP program works.
In this assignment I have made use of two such constructs to compute sum of all the elements of a large array also calculated sum of elements sequentially to compare the efficiency.

Reduction clause: allows programmer to specify one or more variables that are private to individual thread can be reduced at the end of reduction clause construct. Private copies of each thread will be reduced into a single value and combined with the original value which is global/shared value usually. Syntax of the reduction clause : reduction (reduction identifier : variable list) where the variable list is a comma separated list and the reduction identifier is one of the following: +, -, *, &, |, ^, &&, ||, max, min. The source code for this is submitted in the file named *ParallelReduction.c*

Task-Level Parallelism: Tasking concept was introduced in OpenMP 3.0. It consists of code that will be executed concurrently, separate data environment for each task and the location of thread executing it. *"#pragma omp task*" defines a task. Normally a task construct should be placed within a parallel construct. When a thread finds a task construct clause, It depends on thread to decide if to execute the task immediately or to schedule it for later time. If decided to execute later, such tasks will be placed in a conceptual pool of tasks with their current parallel region. The team of threads will execute the tasks in the pool until it becomes empty. There are chances where a thread that executes the task might be different from the thread that placed the task originally into queue. Each task can be executed by a one or more threads. Task synchronization can be achieved using the following constructs *"#pragma omp barrier*" and *"#pragma omp taskwait*". Source code for Task-level parallelism is submitted in a file named *Task-Level-Parallelism.c*

Results: The following table shows the **execution time in seconds** with three different approaches used for parallel sum reduction for various input array size.

	Array size (65536)	**Array size (262144)**	**Array size (1048576)**	**Array size (268435456)**	**Array size (1073741824)**
Sequential approach	0.000215	0.000862	0.002970	0.652130	2.715204
Reduction Clause	0.002075	0.000510	0.001038	0.241379	1.034187
Task-Level Parallelism	0.000475	0.000769	0.002725	0.178938	2.384587