**PRACTICAL # 05**

**OBJECT:**

Designing a parallel version of sequential algorithm based on data

**THEORY:**

A sequential algorithm is the one in which the flow of instructions is in fixed order and an instruction is executed after the execution of the previous instruction. One of the strategies for parallel execution is to redesign the sequential algorithm into parallel with respect to the data. If the design of the parallel algorithm is based on data, the data elements should be independent of each other to make its parallel processing possible. But if data elements are dependent on each other, the parallelism may not be possible.

**Program**:

The program below is the simple sequential program that linearly calculates the sum of the array. The entire array is summed up by a single processor.

*int main(){*

*int arrSize = 10;*

*int arr[] = { 2, 4, 5, 1, 8, 20, 30, 40, 10, 0 };*

*int sum=0;*

*for(int i=0; i<arrSize; i++)*

*{*

*sum+=arr[i];*

*}*

*return 0;*

*}*

The program is calculating the sum sequentially, one iteration at a time. But this algorithm can be slightly modified to exploit parallelism.

Since the array elements are independent of each other and the sum does not require a sequence, so the array can be accessed and summed in parallel. This can be achieved by dividing array into a number of partial sub-arrays and calculating the partial sums of each sub-array. These partial sums can be calculated in parallel. Finally these partial sums are summed up to get the grand sum. The number partial sub-arrays is decided according to the number of the processors available in the system.

The program thus obtained can be made more generic by getting the number of processors available to work from OpenMPI and dividing the array into that number of equal number of the sub-arrays. This makes the each processor to get the same amount of load to work.

The program below is doing the job of redesigning the algorithm to be suitable for parallel execution.

*int main(){*

*int arrSize = 10;*

*int arr[] = { 2, 4, 5, 1, 8, 20, 30, 40, 10, 0 };*

*int cores = 3;*

*int pSums[arrSize/cores];*

*int sum=0;*

*int parts = cores; //parts to keep the number of array parts to be divided in*

*//if the data is not evenly divisible between the available processes,*

*//keep track of the last remaining array part.*

*if(cores\*cores < arrSize)*

*parts++;*

*for(int i=0; i<parts; i++) {*

*pSums[i] = 0;*

*for(int j=0; j<3; j++) {*

*pSums[i] += arr[i\*cores + j];*

*}*

*}*

*for(int k=0; k<arrSize/cores; k++) {*

*sum+=pSums[k];*

*}*

*return 0;*

*}*

Note that in these programs, we are just dividing the algorithm to make it executable in parallel. The performance can be analyzed when it will be implemented in parallel using OpenMPI.

**REVIEW QUESTIONS**

1. What is a sequential algorithm?

1. What is a parallel algorithm?
2. How can a parallel algorithm be faster than its sequential counterpart?

1. How can a sequential algorithm be redesigned to form parallel algorithm?