**PRACTICAL # 06**

**OBJECT:**

Calculating sum of array elements in parallel using OpenMPI

**THEORY:**

Dividing a big problem into smaller ones and working on each of those smaller problems makes the solution calculation time faster.

If we have several parallel processors available, we can divide the array into the parts of equal sizes. The master processor can assign each of the sub-arrays to the available processors to perform some operation over it. The worker processors will process the partial sub-arrays and send the results back to the master processor. Finally the master processor will collect the sums calculated by each CPU and find the grand sum of the array. This will make the calculation much faster.

There are three portions of this solution.

1. Divide the array into equal parts and assign them to each CPU.
2. Each CPU calculates individual sum in parallel and send it to the master CPU.
3. The master CPU calculates the grand sum from the individual partial sums.

**Code:**

Below is the code to calculate the sum of the array in parallel using OpenMPI.

*#include <mpi.h>*

*#include <stdio.h>*

*#include <string.h>*

*#define BUFSIZE 8//128*

*#define TAG 0*

*int main(int argc, char \*argv[])*

*{*

*char idstr[32]; //char buff[BUFSIZE];*

*int buff[] = {1,2,3,4, 6,7,8,9};*

*int numprocs, slaveProcs, myid, i;*

*int grandSum = 0;*

*MPI\_Status stat;*

*/\* MPI programs start with MPI\_Init; all 'N' processes exist thereafter \*/*

*MPI\_Init(&argc,&argv);*

*/\* find out how big the SPMD world is \*/*

*MPI\_Comm\_size(MPI\_COMM\_WORLD, &numprocs);*

*/\* and this processes' rank is \*/*

*MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myid);*

*slaveProcs = numprocs-1;*

*/\* At this point, all programs are running equivalently, the rank*

*distinguishes the roles of the programs in the SPMD model, with*

*rank 0 often used specially...*

*\*/*

*if(myid == 0)*

*{*

*printf("%d: We have %d processors\n", myid, numprocs);*

*for(i=1;i<numprocs;i++)*

*{*

*//printf("Hello %d! ", i);*

*MPI\_Send(buff, BUFSIZE, MPI\_INT, i, TAG, MPI\_COMM\_WORLD);*

*/\* buf - initial address of send buffer (choice)*

*count - number of elements in send buffer (nonnegative integer)*

*datatype - datatype of each send buffer element (handle)*

*dest - rank of destination (integer)*

*tag - message tag (integer)*

*comm - communicator (handle)\*/*

*}*

*for(i=1;i<numprocs;i++)*

*{*

*MPI\_Recv(buff, BUFSIZE, MPI\_INT, i, TAG, MPI\_COMM\_WORLD, &stat);*

*grandSum += buff[0];*

*}*

*printf("\n Grand sum = %d\n", grandSum);*

*}*

*else*

*{*

*/\* receive from rank 0: \*/*

*int src = 0, k, sum=0;*

*int buffer[1];*

*MPI\_Recv(buff, BUFSIZE, MPI\_INT, src/\*0\*/, TAG, MPI\_COMM\_WORLD, &stat);*

*//sprintf(idstr, "Processor %d ", myid);*

*printf("Processor %d reporting for duty. ", myid);*

*for(k=0; k<BUFSIZE/slaveProcs ; k++){*

*//printf("arr[i] = %d ", buff[BUFSIZE/slaveProcs\*(myid-1)+ k]);*

*sum+=buff[BUFSIZE/slaveProcs\*(myid-1)+ k];*

*}*

*printf("Partial sum = %d \n", sum);*

*buffer[0] = sum;*

*//MPI\_Send(buff, BUFSIZE, MPI\_INT, 0, TAG, MPI\_COMM\_WORLD);*

*MPI\_Send(buffer, 1, MPI\_INT, 0, TAG, MPI\_COMM\_WORLD);*

*}*

*/\* MPI programs end with MPI Finalize; this is a weak synchronization point \*/*

*MPI\_Finalize();*

*return 0;*

}

Save the code as arrsum.c

To compile the program type:

*mpicc arrsum.c -o arrsum*

To run the program:

*mpirun --oversubscribe -np 3 arrsum*

**ACTIVITIES**

**Activity 1**

Execute the program given above and observe the results. Also calculate the average of these elements.

**REVIEW QUESTIONS**

1. What is the mechanism of working of a parallel algorithm?
2. How is a sequential algorithm converted to the parallel algorithm?
3. What is the function of master processor in a parallel computing environment?