Name: Raghav Maheshwari
Roll NO: 53
Panel: A

Lab Assignment -5 (PP)

★ AIM:
To write a C program to use parallel reduction over a cluster of MPI nodes.

★ Objective:
To understand working of a parallel reduction mechanism in MPI.

★ Theory:
Explain cost optimal addition of n numbers over p processor.

⇒ A parallel algorithm is cost Optimal if cost of solving a problem on parallel computer has same asymptotic growth as the fun of input size as fastest known known sequential algorithm on a single processing element.

The runtime of the best known sequential algorithm. ≥ The runtime of the parallel algo is (cost on parallel algorithm)

Addition of n numbers over p processor.

Proof:

Consider problem of adding n numbers on p processing elements such that $p < n$ and both n and p are power of 2.

We will simulate n processing elements on p processing elements think them as virtual processors.

Each of p processor, is not assigned (n/p) virtual processor. the first $(\log n)$ of $1(\log n)$ steps of the original algorithm are simulated in $L(n/n) \log p$ steps on p processing elements subsequent $(\log n - \log p)$ step do not require any communication.

Overall parallel execution time is $((n/p) \log p)$

) Each processing element locally adds its n/p no. in time $\Theta(n/p)$.

) The p partial sums on p processing elements can be added in time $\Theta(n/p)$ so parallel runtime algorithm.

$$T_p = \Theta((n/p) + \log p)$$

$$\text{and cost} = \Theta(p \times [(n/p) + \log n])$$
$$= \Theta(n + p \log n)$$

as long as $n * \Theta(p \log n)$

Hence cost of parallel algorithm $= \Theta(n+n) = \Theta(2n)$
$= \Theta(n)$

PAGE NO. :

FAQ

① Explain cost optimal analysis of addition of n number over n processors?

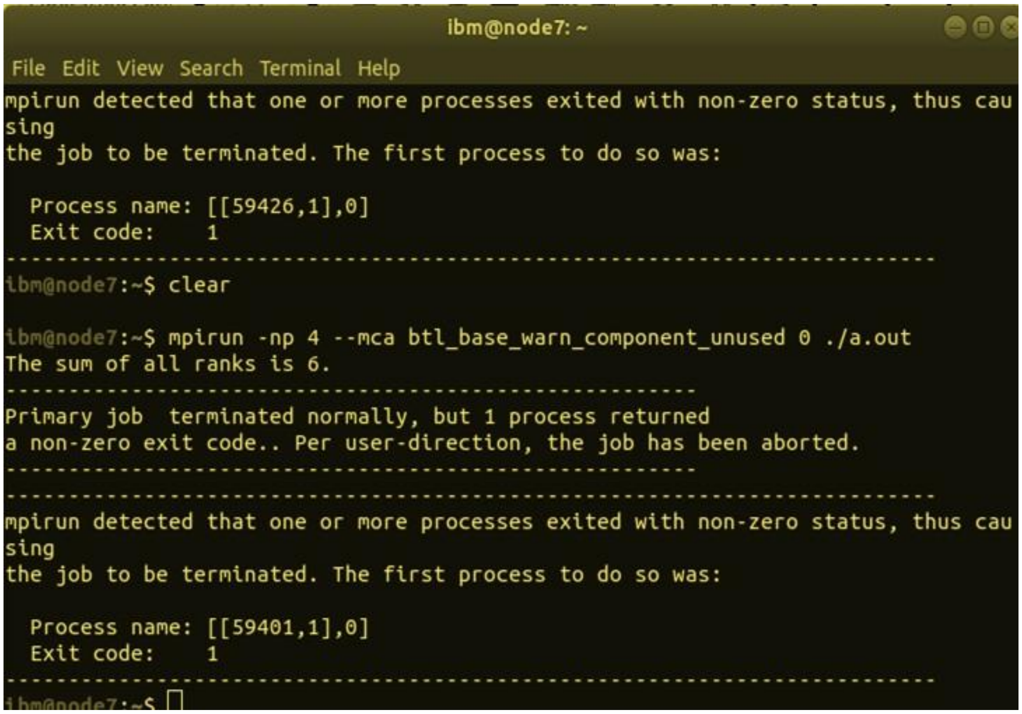
Ans For adding n number on n processing elements has parallel computation time is $T(n, n) = O\log(n))$, so processor time product will yield corresponding cost as

$$C(n, n) = O(n\log n) \text{ (in worst case)}$$

Program:

```
#include<mpi.h>

#include<stdio.h>


/**
 * @brief Illustrates how to use a reduce.
 * @details This application consists of a sum reduction; every MPI process
 * sends its rank for reduction before the sum of these ranks is stored in the
 * root MPI process. It can be visualised as follows, with MPI process 0 as
 * root:
 *
 * +_________+ +_________+ +_________+ +_________+
 * | Process 0 | | Process 1 | | Process 2 | | Process 3 |
 * +-+_____+-+ +-+_____+-+ +-+_____+-+ +-+_____+-+
 *   | Value |     | Value |     | Value |     | Value |
 *   |   0   |     |   1   |     |   2   |     |   3   |
 *   +-------+     +-------+     +-------+     +-------+
 *          \          |          |          /
 *           \         |          |         /
 *            \        |          |        /
 *             \       |          |       /
 *              +-----+-----+-----+-----+
 *                          |
 *                      +---+---+
 *                      |  SUM  |
 *                      +---+---+
 *                          |
 *                      +---+---+
 *                      |   6   |
 *                    +-+-------+-+
 *                    | Process 0 |
 *                    +-----------+
 **/
```

```
int main(int argc,char **argv)

{



MPI_Init(&argc,&argv); //Initialization of MPI Parallelism

int myrank,sum,min,max;

int arr[]={1,2,3,4};

int x;

int i=0;

MPI_Comm_rank(MPI_COMM_WORLD,&myrank); //communication between the cores



MPI_Scatter(arr,1,MPI_INT,&x,1,MPI_INT,0,MPI_COMM_WORLD);       //MPI Scatter function for scattering the data to ranks

for(i=0;i<4;i++)

{

        if(myrank==i)

        {

                printf("Value on scattered %d core is %d\n",myrank,x);

        }

}

MPI_Reduce(&x,&sum,1,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD); //MPI Reduction function for sum

if(myrank==0)

{

        printf("Sum=%d on the core=%d\n",sum,myrank);

}
MPI_Reduce(&x,&min,1,MPI_INT,MPI_MIN,1,MPI_COMM_WORLD);    //MPI Reduction function for min
if(myrank==1)
{
        printf("Minimum=%d on the core=%d\n",min,myrank);
}
MPI_Reduce(&x,&max,1,MPI_INT,MPI_MAX,2,MPI_COMM_WORLD);  //MPI Reduction function for max
if(myrank==2)
{
        printf("Maximum=%d on the core=%d\n",max,myrank);
}
MPI_Finalize();             //Termination of the MPI Parallelism
return 0;
}
```



```
mpirun detected that one or more processes exited with non-zero status, thus cau
sing
the job to be terminated. The first process to do so was:

  Process name: [[59426,1],0]
  Exit code:    1
--------------------------------------------------------------------------
ibm@node7:~$ clear

ibm@node7:~$ mpirun -np 4 --mca btl_base_warn_component_unused 0 ./a.out
The sum of all ranks is 6.
-------------------------------------------------------
Primary job  terminated normally, but 1 process returned
a non-zero exit code.. Per user-direction, the job has been aborted.
-------------------------------------------------------
--------------------------------------------------------------------------
mpirun detected that one or more processes exited with non-zero status, thus cau
sing
the job to be terminated. The first process to do so was:

  Process name: [[59401,1],0]
  Exit code:    1
--------------------------------------------------------------------------
ibm@node7:~$
```