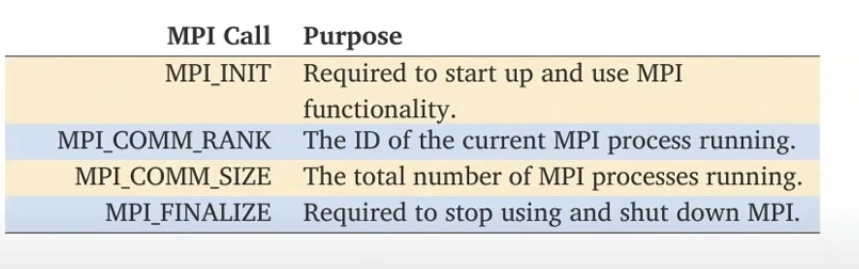
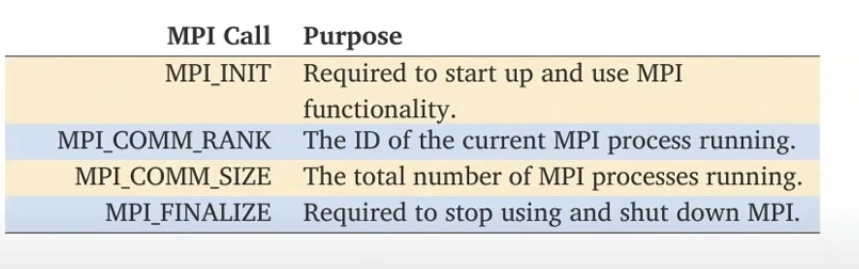
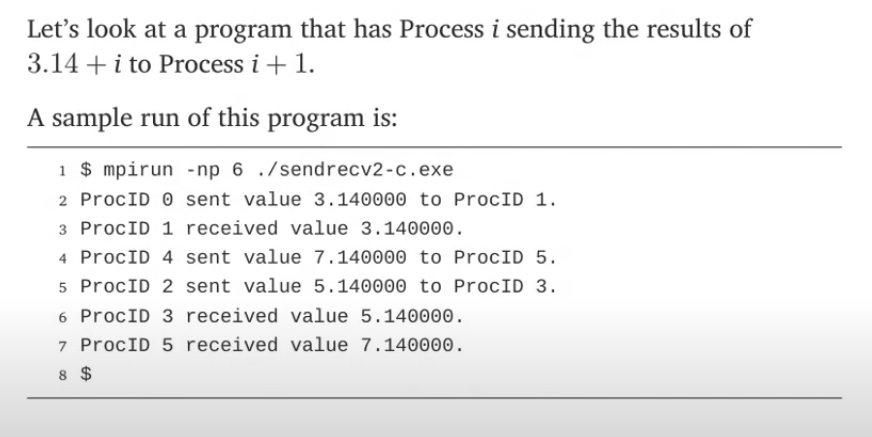
<https://www.google.com/search?q=openmpi+in+hindi+language&biw=1517&bih=694&tbm=vid&ei=m7dMZPUXr5ix4w_j7IVg&oq=OPENMPI+in+hindi&gs_lcp=Cg1nd3Mtd2l6LXZpZGVvEAEYADIFCCEQoAE6BQgAEIAEOgcIABCKBRBDOgYIABAWEB46CggAEBYQHhAPEAo6CAgAEBYQHhAPOggIIRAWEB4QHToHCCEQoAEQClC4BFikJGC8M2gAcAB4AIABsAKIAdILkgEHMC45LjAuMZgBAKABAcABAQ&sclient=gws-wiz-video#fpstate=ive&vld=cid:635e7236,vid:pxwbuA26yrk>

https://www.google.com/search?q=openmpi+in+hindi+language&biw=1517&bih=694&tbm=vid&ei=m7dMZPUXr5ix4w\_j7IVg&oq=OPENMPI+in+hindi&gs\_lcp=Cg1nd3Mtd2l6LXZpZGVvEAEYADIFCCEQoAE6BQgAEIAEOgcIABCKBRBDOgYIABAWEB46CggAEBYQHhAPEAo6CAgAEBYQHhAPOggIIRAWEB4QHToHCCEQoAEQClC4BFikJGC8M2gAcAB4AIABsAKIAdILkgEHMC45LjAuMZgBAKABAcABAQ&sclient=gws-wiz-video#fpstate=ive&vld=cid:f8364483,vid:RoQJNx5npF4





Experiment :3

//Aim :. Develop a distributed system, to find sum of N elements in an array by distributing N/n elements to n number of processors MPI or OpenMP. Demonstrate by displaying the intermediate sums calculated at different processors.

[**https://www.open-mpi.org/software/ompi/v4.1/**](https://www.open-mpi.org/software/ompi/v4.1/)

**https://edu.itp.phys.ethz.ch/hs12/programming\_techniques/openmpi.pdf**

**Installation of OPENMPI**

1. Download openmpi-4.1.4.tar.bz2 from http://www.open-mpi.org

2. Goto the terminal (Command prompt)

3. update using

sudo apt-get update

sudo apt install gcc {if not already installed}

4. Goto the directory which contains the downloaded file (download folder)

5. Extract the files using (download folder)

tar -jxf openmpi-4.1.4.tar.bz2

6. The directory openmpi-4.1.4 is created

7. Go into the source directory through terminal

Command prompt > cd /home/it/download/openmpi-1.4.4

7. Configure, compile and install by executing the following commands

Command prompt//home/it/download/openmpi-1.4.4 > ./configure --prefix=$HOME/opt/openmpi

make all

make install

8. Now openmpi folder is created in ‘opt‘ folder of Home directory.

9. Now the folder LP5 can be deleted (optional)

echo "export PATH=\$PATH:\$HOME/opt/openmpi/bin" >> $HOME/.bashrc

echo "export LD\_LIBRARY\_PATH=\$LD\_LIBRARY\_PATH:\$HOME/opt/openmpi/lib">>$HOME/.bashrc

10. Update the PATH and LD\_LIBRARY\_PATH environment variable using

echo "export PATH=\$PATH:\$HOME/it/opt/openmpi/bin" >> $HOME/it.bashrc

echo "export LD\_LIBRARY\_PATH=\$LD\_LIBRARY\_PATH:\$HOME/it/opt/openmpi/lib">>$HOME/it.bashrc

11. Compile the program using

mpicc name of the program

example : mpicc hello1.c

12. Execute the program using

mpirun -np N ./a.out

example : mpirun -np 2 ./a.out

example : mpirun -np 3 ./a.out

example : mpirun -np 4 ./a.out

/\*\*\*\*\*\* The cores 2 and will be in waiting mode … Press Ctrl+z to end the execution \*\*\*\*\*\*\*/

Assignment program: Add 20 numbers in an array using 4 cores

C:\>cd /opt/openmpi/bin

C:\opt/openmpi/bin> edit program\_name.c

Example :

C:\opt/openmpi/bin> edit add.c

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <stdio.h>

#include "mpi.h"

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

{

int rank, size;

int num[20]; //N=20, n=4

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

for(int i=0;i<20;i++)

num[i]=i+1;

if(rank == 0){ // (Determine the label of calling process )( i.e. Label all the process)

int s[4]; // Declares an array s of size 4 to store partial sums from other processes.

//  Prints a message indicating the distribution is happening at rank 0.

printf("Distribution at rank %d \n", rank);

for(int i=1;i<4;i++)

MPI\_Send(&num[i\*5], 5, MPI\_INT, i, 1, MPI\_COMM\_WORLD); //N/n i.e. 20/4=5

int sum=0, local\_sum=0;

for(int i=0;i<5;i++)

{

local\_sum=local\_sum+num[i]; // **Local Sum Calculation**:

* Calculates the local sum of the first 5 elements of the num array.
* This local sum is stored in a variable local\_sum.

}

for(int i=1;i<4;i++)

{

MPI\_Recv(&s[i], 1, MPI\_INT, i, 1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

} // **Partial Sum Reception**:

* Utilizes MPI\_Recv to receive partial sums from other processes.
* Stores the received partial sums in the array s.

printf("local sum at rank %d is %d\n", rank,local\_sum);

sum=local\_sum;

for(int i=1;i<4;i++)

sum=sum+s[i];

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

} // **Final Sum Calculation**:

* Calculates the final sum by summing up the local sum and the received partial sums.
* Prints the local sum of the root process and the final sum.

else

{

int k[5]; //This loop for root block

MPI\_Recv(k, 5, MPI\_INT, 0, 1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

int local\_sum=0;

for(int i=0;i<5;i++)

{

local\_sum=local\_sum+k[i];

}

printf("local sum at rank %d is %d\n", rank, local\_sum);

MPI\_Send(&local\_sum, 1, MPI\_INT, 0, 1, MPI\_COMM\_WORLD);

}

MPI\_Finalize();

return 0;

}

Distribution at rank 0

local sum at rank 1 is 40

local sum at rank 2 is 65

local sum at rank 3 is 90

local sum at rank 0 is 15

final sum = 210

/\*\*\*\*\*\* students can be asked to take dynamic values for N, n and array \*\*\*\*\*\*\*\*\*\*\*\*/

**Run : Compile the program using**

C:mpicc name of the program

example : opt/openmpi/bin> mpicc add.c

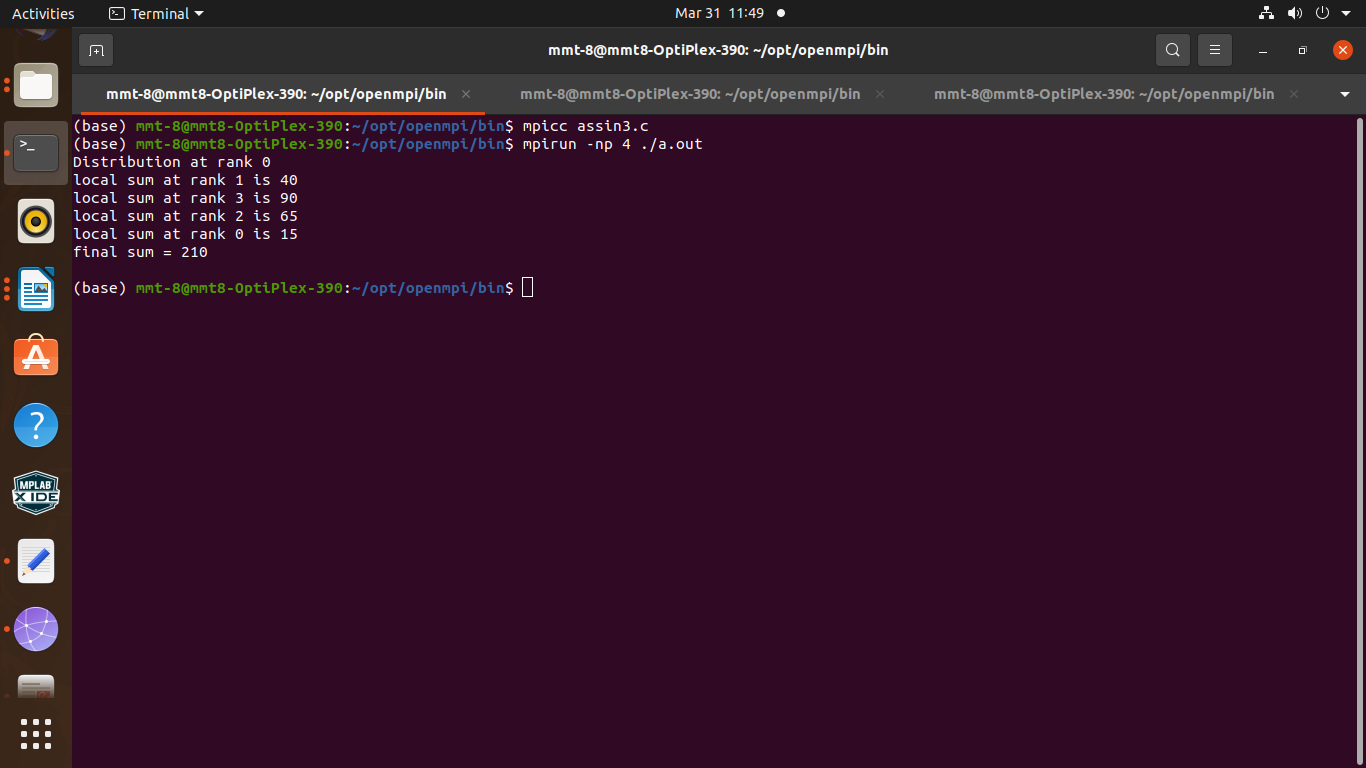
12. Execute the program using

opt/openmpi/bin> mpirun -np N ./a.out

example : opt/openmpi/bin> mpirun -np 2 ./a.out

example : opt/openmpi/bin> mpirun -np 3 ./a.out

example : opt/openmpi/bin> mpirun -np 4 ./a.out



1. Explanation : **Initialization**: The program starts by including the necessary header files (stdio.h and mpi.h) and defining the main function. It initializes MPI using MPI\_Init and retrieves the rank and size of the MPI communicator (MPI\_COMM\_WORLD).
2. **Data Initialization**: An array num of size 20 is declared to hold integers. Each element of the array is initialized with values from 1 to 20.
3. **Root Process (Rank 0)**:
   * The program checks if the current process has rank 0. If it is the root process (rank 0), it proceeds with distributing data to other processes.
   * It prints a message indicating the distribution is happening at rank 0.
   * Using MPI\_Send, it sends blocks of 5 integers (num[i\*5] to num[i\*5 + 4]) to each of the other processes (ranks 1 to 3).
   * It calculates the local sum of the first 5 elements of the array num.
   * It receives the local sums computed by other processes using MPI\_Recv.
4. **Non-root Processes (Ranks 1 to 3)**:
   * If the process is not the root (rank 0), it receives 5 integers sent by the root process using MPI\_Recv.
   * It calculates the local sum of the received 5 integers.
   * It sends the local sum back to the root process using MPI\_Send.
5. **Finalization**: After all communication is complete, the program finalizes MPI using MPI\_Finalize and returns.
6. **Output**:
   * Each process prints its rank and the local sum it calculated.
   * The root process also prints the final sum, which is the sum of all local sums calculated by each process.

Explanation of the data distribution:

* The array num holds integers from 1 to 20.
* The root process (rank 0) divides this array into 4 equal parts, each containing 5 integers.
* These 4 parts are then sent to the other processes (ranks 1 to 3) using MPI\_Send.
* Each non-root process receives 5 integers from the root process and calculates the local sum of these integers.

Explanation of the summation process:

* After receiving the local sums from all processes, the root process sums up all the local sums to calculate the final sum.
* This final sum represents the sum of all integers from 1 to 20, distributed among the processes.

This program demonstrates a parallel computing example where a large data array is distributed among multiple processes, and each process computes a partial sum. The root process collects these partial sums and computes the final sum.