After installing Open MPI using `sudo apt-get install libopenmpi-dev`, you can compile and run MPI programs using the provided MPI compiler wrapper (`mpicc`) and the MPI runtime (`mpirun`). Here's how you can compile and run your MPI code:

1. \*\*Compile MPI Program\*\*:

- Navigate to the directory containing your MPI program source code (let's assume it's named `mpi\_program.c`).

- Compile the program using `mpicc`:

```

mpicc mpi\_program.c -o mpi\_program

```

- This command compiles `mpi\_program.c` and produces an executable named `mpi\_program`.

2. \*\*Run MPI Program\*\*:

- After successfully compiling your MPI program, you can run it using `mpirun`:

```

mpirun -np N ./mpi\_program

```

- Replace `N` with the number of MPI processes you want to run.

- The `-np` flag specifies the number of processes.

- `./mpi\_program` is the path to your compiled MPI program.

For example, if you want to run your MPI program with 4 processes, you would use:

```

mpirun -np 4 ./mpi\_program

```

This command will execute your MPI program with 4 processes, distributing the workload across them as specified in your program. Make sure that your MPI program is designed to handle the specified number of processes correctly.

#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){

int s[4];

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];

}

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

{

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

}

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);

}

else

{

int k[5];

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;

}

**Output:**

$mpicc prgm.c

$mpirun -np 4 ./a.out

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

This MPI program is designed to add 20 numbers distributed across 4 cores (or processes). Let's break down the program and explain each part with an example.

```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);

// Initializing the array with numbers 1 to 20

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

num[i] = i + 1;

```

Explanation:

- The program starts by initializing MPI and obtaining the rank and size of the current process and the total number of processes.

```c

if(rank == 0){

int s[4];

// Distributing numbers to other processes

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

// Calculating local sum for process 0

int sum = 0, local\_sum = 0;

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

{

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

}

// Receiving local sums from other processes

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

{

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

}

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

sum = local\_sum;

// Adding local sums to get final sum

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

sum = sum + s[i];

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

}

else

{

int k[5];

// Receiving numbers from process 0

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

// Calculating local sum for other processes

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);

// Sending local sum back to process 0

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

}

MPI\_Finalize();

return 0;

}

```

Explanation:

- Process with rank 0 distributes numbers to other processes using `MPI\_Send`.

- Each process calculates its local sum.

- Process 0 receives local sums from other processes and calculates the final sum.

- Other processes (with ranks > 0) receive numbers from process 0, calculate their local sum, and send it back to process 0.

Example:

Let's assume the array `num` contains numbers from 1 to 20.

- Process 0 sends numbers 6 to 10 to process 1, 11 to 15 to process 2, and 16 to 20 to process 3.

- Each process calculates its local sum.

- Process 0 receives local sums from processes 1, 2, and 3 and calculates the final sum.