**7th April, 21**

**Steps to install MPI on Ubuntu**

$ sudo adduser mpi

$ passwd mpi

$ su mpi

$ sudo apt-get update

$ sudo rm /var/lib/dpkg/lock

$sudo rm /var/cache/apt/archives/lock

$ sudo apt-get install mpich

$ gedit hello.c

$mpicc hello.c –o h1

$mpirun -np 2 ./h1 –allow-run-as-root

**8th April,21**

**MPI Hello World Program**

#include <mpi.h>

#include <stdio.h>

int main(int argc, char\*\* argv) {

// Initialize the MPI environment

MPI\_Init(NULL, NULL);

// Get the number of processes

int world\_size;

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

// Get the rank of the process

int world\_rank;

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

// Get the name of the processor

char processor\_name[MPI\_MAX\_PROCESSOR\_NAME];

int name\_len;

MPI\_Get\_processor\_name(processor\_name, &name\_len);

// Print off a hello world message

printf("Hello world from processor %s, rank %d out of %d processors\n",

processor\_name, world\_rank, world\_size);

// Finalize the MPI environment.

MPI\_Finalize();

}

**9th April, 21**

**MPI\_send / MPI\_Recv**

;l

#include <stdio.h>

#include *<mpi.h>*

int main(int argc, char\*\* argv) {

int process\_Rank, size\_Of\_Cluster, message\_Item;

MPI\_Init(&argc, &argv);

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

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &process\_Rank);

**if**(process\_Rank == 0){

message\_Item = 42;

MPI\_Send(&message\_Item, 1, MPI\_INT, 1, 1, MPI\_COMM\_WORLD);

printf("Message Sent: %d**\n**", message\_Item);

}

**else** **if**(process\_Rank == 1){

MPI\_Recv(&message\_Item, 1, MPI\_INT, 0, 1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

printf("Message Received: %d**\n**", message\_Item);

}

MPI\_Finalize();

**return** 0;

}

**Output:**

Message Sent: 42

Message Received: 42

**14th April, 21**

**MPI\_scatter**

#include *<stdio.h>*

#include *<mpi.h>*

.

int main(int argc, char\*\* argv){

int process\_Rank, size\_Of\_Comm;

int distro\_Array[4] = {39, 72, 129, 42};

int scattered\_Data;

MPI\_Init(&argc, &argv);

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

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &process\_Rank);

MPI\_Scatter(&distro\_Array, 1, MPI\_INT, &scattered\_Data, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

printf("Process has received: %d **\n**", scattered\_Data);

MPI\_Finalize();

**return** 0;

}

**Output:**

Process has received: 72

Process has received: 129

Process has received: 42

**15th April, 21**

**Vector Multiplication Scatter, Gather,Bcast**

#define N 4

#include <stdio.h>

#include <math.h>

#include <sys/time.h>

#include <stdlib.h>

#include <stddef.h>

#include "mpi.h"

void print\_results(char \*prompt, int a[N][N]);

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

{

int i, j, k, rank, size, tag = 99, blksz, sum = 0;

int a[N][N]={{1,2,3,4},{5,6,7,8},{9,1,2,3},{4,5,6,7,}};

int b[N][N]={{1,2,3,4},{5,6,7,8},{9,1,2,3},{4,5,6,7,}};

int c[N][N];

int aa[N],cc[N];

MPI\_Init(&argc, &argv);

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

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

//scatter rows of first matrix to different processes

MPI\_Scatter(a, N\*N/size, MPI\_INT, aa, N\*N/size, MPI\_INT,0,MPI\_COMM\_WORLD);

//broadcast second matrix to all processes

MPI\_Bcast(b, N\*N, MPI\_INT, 0, MPI\_COMM\_WORLD);

MPI\_Barrier(MPI\_COMM\_WORLD);

//perform vector multiplication by all processes

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

{

for (j = 0; j < N; j++)

{

sum = sum + aa[j] \* b[j][i];

}

cc[i] = sum;

sum = 0;

}

MPI\_Gather(cc, N\*N/size, MPI\_INT, c, N\*N/size, MPI\_INT, 0, MPI\_COMM\_WORLD);

MPI\_Barrier(MPI\_COMM\_WORLD);

MPI\_Finalize();

if (rank == 0) //I\_ADDED\_THIS

print\_results("C = ", c);

}

void print\_results(char \*prompt, int a[N][N])

{

int i, j;

printf ("\n\n%s\n", prompt);

for (i = 0; i < N; i++) {

for (j = 0; j < N; j++) {

printf(" %d", a[i][j]);

}

printf ("\n");

}

printf ("\n\n");

}

**MPI Allgather**

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <mpi.h>
5. **int** main(**int** argc, **char**\* argv[])
6. {
7. [MPI\_Init](https://www.rookiehpc.com/mpi/docs/mpi_init.php)(&argc, &argv);
9. // Get number of processes and check that 3 processes are used
10. **int** size;
11. [MPI\_Comm\_size](https://www.rookiehpc.com/mpi/docs/mpi_comm_size.php)([MPI\_COMM\_WORLD](https://www.rookiehpc.com/mpi/docs/mpi_comm_world.php), &size);
12. **if**(size != 3)
13. {
14. printf("This application is meant to be run with 3 MPI processes.\n");
15. [MPI\_Abort](https://www.rookiehpc.com/mpi/docs/mpi_abort.php)([MPI\_COMM\_WORLD](https://www.rookiehpc.com/mpi/docs/mpi_comm_world.php), EXIT\_FAILURE);
16. }
18. // Get my rank
19. **int** my\_rank;
20. [MPI\_Comm\_rank](https://www.rookiehpc.com/mpi/docs/mpi_comm_rank.php)([MPI\_COMM\_WORLD](https://www.rookiehpc.com/mpi/docs/mpi_comm_world.php), &my\_rank);
22. // Define my value
23. **int** my\_value = my\_rank \* 100;
24. printf("Process %d, my value = %d.\n", my\_rank, my\_value);
26. **int** buffer[3];
27. [MPI\_Allgather](https://www.rookiehpc.com/mpi/docs/mpi_allgather.php)(&my\_value, 1, [MPI\_INT](https://www.rookiehpc.com/mpi/docs/mpi_int.php), buffer, 1, [MPI\_INT](https://www.rookiehpc.com/mpi/docs/mpi_int.php), [MPI\_COMM\_WORLD](https://www.rookiehpc.com/mpi/docs/mpi_comm_world.php));
28. printf("Values collected on process %d: %d, %d, %d.\n", my\_rank, buffer[0], buffer[1], buffer[2]);
30. [MPI\_Finalize](https://www.rookiehpc.com/mpi/docs/mpi_finalize.php)();
32. **return** EXIT\_SUCCESS;
33. }