1. **Execution of Simple Hello world program on MPI platform**

**#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();**

**}**

**Output:**

**Hello World from processor p1 rank 1 out of 4**

**Hello World from processor p2 rank 0 out of 4**

**Hello World from processor p3 rank 2 out of 4**

**Hello World from processor p4 rank 3 out of 4**

**Installation:**

1.apt-get install libopenmpi-dev

2.apt-get install openmpi-bin

**Compilation:**

mpicc filename.c -o filename

**Running:**

**mpirun -np 8 ./filename (mpirun -np 4 ./filename)**

1. **Program to send data and receive data to/from processors using MPI using MPI\_Send() and MPI\_Recv() function**

|  |  |
| --- | --- |
|  | **#include <mpi.h>** |
|  | **#include <stdio.h>** |
|  | **#include <stdlib.h>** |
|  |  |
|  | **int main(int argc, char\*\* argv) {** |
|  | **// Initialize the MPI environment** |
|  | **MPI\_Init(NULL, NULL);** |
|  | **// Find out rank, size** |
|  | **int world\_rank;** |
|  | **MPI\_Comm\_rank(MPI\_COMM\_WORLD, &world\_rank);** |
|  | **int world\_size;** |
|  | **MPI\_Comm\_size(MPI\_COMM\_WORLD, &world\_size);** |
|  |  |
|  | **int number;** |
|  | **if (world\_rank == 0) {** |
|  | **// If we are rank 0, set the number to -1 and send it to process 1** |
|  | **number = -1;** |
|  | **MPI\_Send(** |
|  | **/\* data = \*/ &number,** |
|  | **/\* count = \*/ 1,** |
|  | **/\* datatype = \*/ MPI\_INT,** |
|  | **/\* destination = \*/ 1,** |
|  | **/\* tag = \*/ 0,** |
|  | **/\* communicator = \*/ MPI\_COMM\_WORLD);** |
|  | **} else if (world\_rank == 1) {** |
|  | **MPI\_Recv(** |
|  | **/\* data = \*/ &number,** |
|  | **/\* count = \*/ 1,** |
|  | **/\* datatype = \*/ MPI\_INT,** |
|  | **/\* source = \*/ 0,** |
|  | **/\* tag = \*/ 0,** |
|  | **/\* communicator = \*/ MPI\_COMM\_WORLD,** |
|  | **/\* status = \*/ MPI\_STATUS\_IGNORE);** |
|  | **printf("Process 1 received number %d from process 0\n", number);** |
|  | **}** |
|  | **MPI\_Finalize();** |
|  | **}** |

**Output:**

**Process 1 received number -1 from process 0**

1. **Write a program to find Factorial using MPI. Compare the speedup with the sequential execution.**

**Parallel Program**

#include <stdio.h>

#include <mpi.h>

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

{

int myRank;

int size;

int fact;

int lower, upper;

int i;

double local\_result = 1.0;

double total; //final factorial value

double start\_time, end\_time;

MPI\_Init(&argc, &argv);

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

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

if (myRank == 0)

{

printf("Enter a number : ");

scanf("%d", &fact); //fact =16 No. of processes = 4

}

start\_time = MPI\_Wtime();

MPI\_Bcast(&fact, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

if (myRank == 0)

lower = 1;

else

lower = myRank \* (fact / size) + 1;

if (myRank == (size - 1))

upper = fact;

else

upper = (myRank + 1) \* (fact / size);

/\*

Process = 4 Fact= 16

16 \*15 \*14 \* 13 \* 12 \* 11……… \*1

Process Rank 0 - lower=1 and upper=0+1 \* 16/4 =4

Process Rank 1 - lower=1\*(4)+1=5 upper =2\*4=8

Process Rank 2 - lower=2 \* 4 +1=8+1=9 upper=3\*4=12

Process Rank 3 - lower=3\*4+1=13 upper=16

Process 0 - 1 to 4

Process 1 - 5 to 8

Process 2 - 9 to 12

Process 3 - 13 to 16

\*/

for (i = lower; i <= upper; i++)

{

local\_result = local\_result \* (double)i;

}

MPI\_Reduce(&local\_result, &total, 1, MPI\_DOUBLE, MPI\_PROD, 0, MPI\_COMM\_WORLD);

if (myRank == 0)

{

end\_time = MPI\_Wtime();

printf("The factorial of %d : %lf \nCalculated using %d processes\n", fact, total, size);

printf("Wallclock time elapsed: %.2lf seconds\n", end\_time - start\_time);

}

MPI\_Finalize();

return 0;

}

1. **Write a program to perform addition of two arrays. Compare the speedup with the sequential execution.**

**c[i] = a[i] + b[i]**

**Parallel**

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

#define MASTER 0

#define ARRAY\_SIZE 8000

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

{

int \*a, \*b, \*c;

int \*ap, \*bp, \*cp;

int total\_proc, rank, n\_per\_proc, n = ARRAY\_SIZE, i;

MPI\_Status status;

double start\_time, end\_time;

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &total\_proc);

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

start\_time = MPI\_Wtime();

if (rank == MASTER)

{

a = (int \*)malloc(sizeof(int) \* n);

b = (int \*)malloc(sizeof(int) \* n);

c = (int \*)malloc(sizeof(int) \* n);

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

a[i] = i;

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

b[i] = i;

}

n\_per\_proc = n / total\_proc;

ap = (int \*)malloc(sizeof(int) \* n\_per\_proc);

bp = (int \*)malloc(sizeof(int) \* n\_per\_proc);

cp = (int \*)malloc(sizeof(int) \* n\_per\_proc);

MPI\_Scatter(a, n\_per\_proc, MPI\_INT, ap, n\_per\_proc, MPI\_INT, MASTER, MPI\_COMM\_WORLD);

MPI\_Scatter(b, n\_per\_proc, MPI\_INT, bp, n\_per\_proc, MPI\_INT, MASTER, MPI\_COMM\_WORLD);

for (i = 0; i < n\_per\_proc; i++)

cp[i] = ap[i] + bp[i];

MPI\_Gather(cp, n\_per\_proc, MPI\_INT, c, n\_per\_proc, MPI\_INT, MASTER, MPI\_COMM\_WORLD);

if (rank == MASTER)

{

int good = 1;

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

{

if (c[i] != a[i] + b[i])

{

printf("problem at index %d\n", i);

good = 0;

break;

}

}

if (good)

{

printf("Added %d Elements Successfully\n", n);

end\_time = MPI\_Wtime();

printf("Wallclock time elapsed: %lf seconds\n", end\_time - start\_time);

}

}

if (rank == MASTER)

{

free(a);

free(b);

free(c);

}

free(ap);

free(bp);

free(cp);

MPI\_Finalize();

return 0;

}

**Serial Code:**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define ARRAY\_SIZE 8000

int main()

{

int i, n = ARRAY\_SIZE;

int \*a, \*b, \*c;

struct timeval start, end;

gettimeofday(&start, NULL);

a = malloc(sizeof(int) \* n);

b = malloc(sizeof(int) \* n);

c = malloc(sizeof(int) \* n);

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

{

a[i] = i;

}

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

{

b[i] = i;

}

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

{

c[i] = a[i] + b[i];

}

printf("Added %d Elements Successfully\n", n);

gettimeofday(&end, NULL);

long seconds = (end.tv\_sec - start.tv\_sec);

long micros = ((seconds \* 1000000) + end.tv\_usec) - (start.tv\_usec);

printf("Time elpased is %ld.%ld seconds\n", seconds, micros);

free(a);

free(b);

free(c);

}

**Data Set:**

All numbers in the range of 1 to 8000