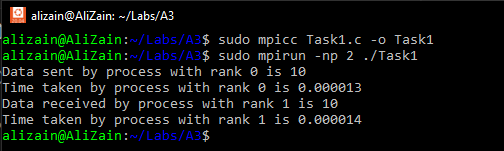
PDC Assignment

21K4653

Task #1:

Output:



Code:

#include <stdio.h>

#include <mpi.h>

#include <stdlib.h>

#include <time.h>

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

{

int rank, size, tag = 4653, i;

double start, end;

MPI\_Status status;

MPI\_Request request;

MPI\_Init(&argc, &argv);

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

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

if (rank == 0)

{

start = MPI\_Wtime();

int a = 10;

MPI\_Isend(&a, 1, MPI\_INT, 1, tag, MPI\_COMM\_WORLD, &request);

MPI\_Wait(&request, &status);

end = MPI\_Wtime();

printf("Data sent by process with rank %d is %d\n", rank, a);

printf("Time taken by process with rank %d is %f\n", rank, end - start);

}

else if (rank == 1)

{

start = MPI\_Wtime();

int b = 0;

MPI\_Irecv(&b, 1, MPI\_INT, 0, tag, MPI\_COMM\_WORLD, &request);

MPI\_Wait(&request, &status);

end = MPI\_Wtime();

printf("Data received by process with rank %d is %d\n", rank, b);

printf("Time taken by process with rank %d is %f\n", rank, end - start);

}

MPI\_Finalize();

return 0;

}

Task #2:



#include<stdio.h>

#include<mpi.h>

#include<stdlib.h>

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

{

int rank, size, i, sum = 0;

MPI\_Request request;

int arr[10];

MPI\_Init(&argc, &argv);

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

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

if (rank == 0)

{

for (i = 0; i < 10; i++)

{

arr[i] = rand() % 10;

printf("%d ", arr[i]);

}

printf("\n");

for (i = 1; i < size; i++)

{

MPI\_Isend(arr, 10, MPI\_INT, i, 0, MPI\_COMM\_WORLD, &request);

}

}

else

{

MPI\_Irecv(arr, 10, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, &request);

}

for (i = 0; i < 10; i++)

{

sum = sum + arr[i];

}

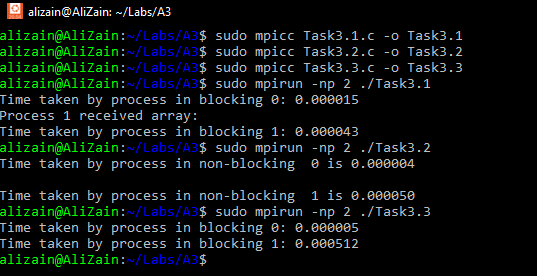
printf("Sum of array is %d\n", sum);

MPI\_Finalize();

return 0;

}

Task #3:



3.1:

#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

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

{

int rank, size, i, j, tag = 4653;

int arr[10];

MPI\_Status status;

double start, end;

MPI\_Init(&argc, &argv);

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

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

if (rank == 0)

{

start = MPI\_Wtime();

for (i = 0; i < 10; i++)

arr[i] = i;

for (i = 1; i < size; i++)

MPI\_Send(arr, 10, MPI\_INT, i, tag, MPI\_COMM\_WORLD);

end = MPI\_Wtime();

printf("Time taken by process in blocking %d: %f\n", rank, end - start);

}

else

{

start = MPI\_Wtime();

MPI\_Recv(arr, 10, MPI\_INT, 0, tag, MPI\_COMM\_WORLD, &status);

printf("Process %d received array: ", rank);

for (i = 0; i < 10; i++)

arr[i] = i;

printf("\n");

end = MPI\_Wtime();

printf("Time taken by process in blocking %d: %f\n", rank, end - start);

}

MPI\_Finalize();

return 0;

}

3.2:

#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

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

{

int rank, size, i, tag = 4653;

int arr[10];

MPI\_Status status;

MPI\_Request request;

double start, end;

MPI\_Init(&argc, &argv);

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

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

if (rank == 0)

{

start = MPI\_Wtime();

for (i = 0; i < 10; i++)

arr[i] = i;

MPI\_Isend(arr, 10, MPI\_INT, 1, tag, MPI\_COMM\_WORLD, &request);

end = MPI\_Wtime();

printf("Time taken by process in non-blocking %d is %f\n", rank, end - start);

}

else if (rank == 1)

{

start = MPI\_Wtime();

MPI\_Irecv(arr, 10, MPI\_INT, 0, tag, MPI\_COMM\_WORLD, &request);

for (i = 0; i < 10; i++)

arr[i] = i;

printf("\n");

end = MPI\_Wtime();

printf("Time taken by process in non-blocking %d is %f\n", rank, end - start);

}

MPI\_Finalize();

return 0;

}

3.3:

#include <stdio.h>

#include <mpi.h>

#include <stdlib.h>

#include <time.h>

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

{

int rank, size;

int sendbuf[10], recvbuf[10];

int i;

double start, end;

MPI\_Init(&argc, &argv);

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

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

for (i = 0; i < 10; i++)

{

sendbuf[i] = i;

}

start = MPI\_Wtime();

MPI\_Sendrecv(sendbuf, 10, MPI\_INT, (rank + 1) % size, 0, recvbuf, 10, MPI\_INT, (rank - 1 + size) % size, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

end = MPI\_Wtime();

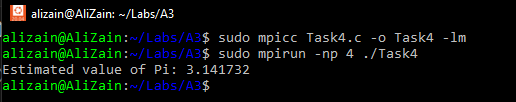
printf("Time taken by process in blocking %d: %f\n", rank, end - start);

MPI\_Finalize();

return 0;

}

Task #4:



#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

#include <time.h>

#define TOTAL\_POINTS 1000000

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

int rank, size;

MPI\_Init(&argc, &argv);

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

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

int points\_inside\_circle = 0;

int points\_total = TOTAL\_POINTS / size;

srand(time(NULL) + rank);

for (int i = 0; i < points\_total; i++) {

double x = ((double)rand() / RAND\_MAX) \* 2 - 1;

double y = ((double)rand() / RAND\_MAX) \* 2 - 1;

if (x \* x + y \* y <= 1) {

points\_inside\_circle++;

}

}

int total\_points\_inside\_circle;

MPI\_Reduce(&points\_inside\_circle, &total\_points\_inside\_circle, 1, MPI\_INT, MPI\_SUM, 0, MPI\_COMM\_WORLD);

if (rank == 0) {

double pi\_estimate = 4.0 \* total\_points\_inside\_circle / TOTAL\_POINTS;

printf("Estimated value of Pi: %f\n", pi\_estimate);

}

MPI\_Finalize();

return 0;

}

Task #5:



#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

#include <time.h>

# define N 10000

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

{

int rank, size, i, max, max\_global;

int a[N];

MPI\_Init(&argc, &argv);

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

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

srand(time(NULL));

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

{

a[i] = rand() % 10000;

}

max = a[0];

for (i = 1; i < N; i++)

{

if (a[i] > max)

{

max = a[i];

}

}

MPI\_Reduce(&max, &max\_global, 1, MPI\_INT, MPI\_MAX, 0, MPI\_COMM\_WORLD);

if (rank == 0)

{

printf("The maximum value in the array is %d\n", max\_global);

}

MPI\_Finalize();

return 0;

}