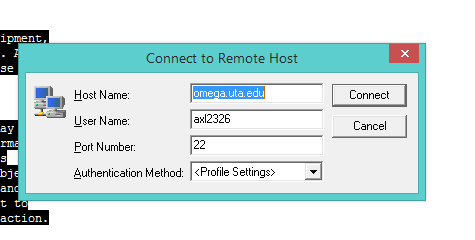
**Homework 3**

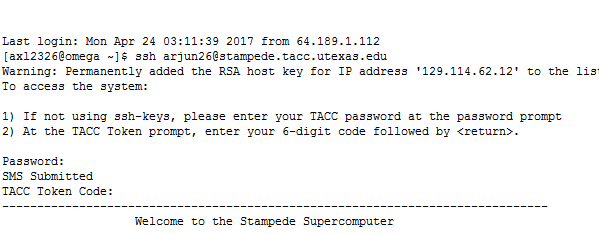
**Arjun lakshmikanth (1001192326)**

1. **First Question**

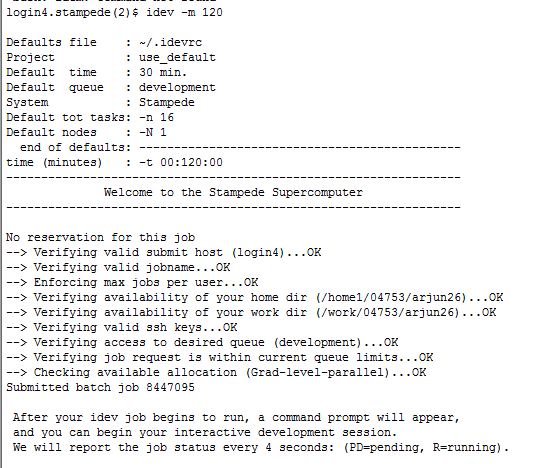
Connect through SSH



Login to Stampede



Login to Compute node using the following command.



Code:

#include <stdio.h>

#include <math.h>

#include "mpi.h"

int main(argumentc, argumentv)

int argumentc;

char \*\*argumentv;

{

MPI\_Init(&argumentc, &argumentv);

int rankc;

int sizec;

float tvalue = 0;

float lowboundry;

float upboundry=0;

int i=0;

float mul = 0;

int gap=0;

float pi=0;

float sumofpi=0;

float answer = 0;

float exactpi = 3.14159265358;

MPI\_Status status;

float total;

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

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

if(rankc !=0)

{

MPI\_Recv(&gap, 1, MPI\_INT, 0, rankc, MPI\_COMM\_WORLD, &status);

for(i=rankc; i<gap; i=(i-1)+sizec)

{

tvalue = ((float)i/gap);

pi = ((float)4/(1+(tvalue\*tvalue)));

if(i%2!=0)

{

pi = pi\*4;

}

else

{

pi = pi\*2;

}

sumofpi = sumofpi + pi;

}

MPI\_Send(&sumofpi,1,MPI\_FLOAT,0,rankc,MPI\_COMM\_WORLD);

}

else if(rankc==0)

{

printf("\nNumber of gaps(>400000): ");

scanf("%d", &gap);

lowboundry=((float)4/(1+0));

upboundry=((float)4/(1+1));

float temp\_pi\_res[gap];

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

{

MPI\_Send(&gap,1,MPI\_INT,i,i,MPI\_COMM\_WORLD);

MPI\_Recv(&temp\_pi\_res[i], 1, MPI\_FLOAT, i, i, MPI\_COMM\_WORLD, &status);

answer = answer+temp\_pi\_res[i];

}

total = lowboundry + upboundry;

answer = answer+total;

mul = (3\*gap);

answer = answer/mul;

printf("\nPI value=% .16f\n Error is % .16f\n", answer, fabs(answer - exactpi));

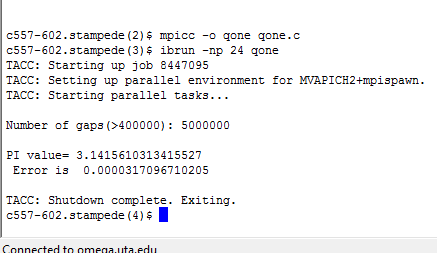
}

MPI\_Finalize();

return 0;

}

Compiling and Running the code:



1. **Second Question**

Code:

#include <stdio.h>

#include "mpi.h"

int sizeval=0;

int values=0;

void Arrayshift(int original\_array[], int new\_array[],int factorforshift, int shift\_portion[] );

void Right(int original\_array[], int new\_array[], int factorforshift, int shift\_portion[]);

void Left(int original\_array[], int new\_array[], int factorforshift, int shift\_portion[]);

int main( int cvalue, char \*valuev[] )

{

int factorforshift;

int new\_position;

int element;

int i= 0;

int procrank;

int sizeval;

MPvaluestatus status ;

MPI\_Init(&cvalue, &valuev);

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

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

if ( (sizeval == 2) || (sizeval == 4) || (sizeval == 8) || (sizeval == 16))

{

}

else

{

MPI\_Abort(MPI\_COMM\_WORLD, i);

exit(0);

}

if(procrank == 0)

{

printf("Input Array Size:\n");

scanf("%d", &values);

printf("\nShift factor ");

scanf("%d", &factorforshift);

int input\_array[values];

int new\_array[values];

int shift\_portion[values];

printf("Enter processor rank: %d\n", procrank);

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

{

printf("\nA[%d]:",i);

scanf("%d", &input\_array[i]);

}

if(factorforshift < 0)

{

while(factorforshift < (-values))

{

factorforshift = factorforshift + values;

}

}

else if(factorforshift > 0)

{

while(factorforshift > values)

{

factorforshift = factorforshift - values;

}

}

if(sizeval > 1)

{

for (element = 1; element < sizeval; element++)

{

MPvaluesend(&factorforshift, 1, MPI\_INT, element, element, MPI\_COMM\_WORLD);

MPvaluesend(&values, 1, MPI\_INT, element, element, MPI\_COMM\_WORLD);

printf("Enter pricessor rank : %d\n", element);

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

{

printf("\nA[%d]:",i);

scanf("%d", &new\_array[i]);

}

MPvaluesend(&new\_array, values, MPI\_INT, element, element, MPI\_COMM\_WORLD);

}

}

shiftArray(input\_array, new\_array, factorforshift,shift\_portion );

if(factorforshift < 0)

{

new\_position = (-factorforshift );

}

else

{

new\_position = factorforshift;

}

if(sizeval > 1)

{

if(factorforshift >= 0)

{

MPvaluesend(&shift\_portion, new\_position, MPI\_INT, 1, 1, MPI\_COMM\_WORLD);

MPI\_Recv(&shift\_portion, new\_position, MPI\_INT, (sizeval - 1), (sizeval - 1), MPI\_COMM\_WORLD, &status);

}

else

{

MPvaluesend(&shift\_portion, new\_position, MPI\_INT, (sizeval - 1), (sizeval - 1), MPI\_COMM\_WORLD);

MPI\_Recv(&shift\_portion, new\_position, MPI\_INT, 1, procrank, MPI\_COMM\_WORLD, &status);

}

}

if(factorforshift >= 0)

{

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

{

new\_array[i] = shift\_portion[i];

}

}

if(factorforshift < 0)

{

for(i = 0, element = (values + factorforshift); element < values; i++, element++)

{

new\_array[element] = shift\_portion[i];

}

}

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

{

printf("Original Array : %d, New Array is : %d \n",input\_array[i], new\_array[i]);

}

if(sizeval > 1)

{

for (element = 1; element < sizeval; element++)

{

MPvaluesend(&factorforshift, 1, MPI\_INT, element, element, MPI\_COMM\_WORLD);

MPI\_Recv(&factorforshift, 1, MPI\_INT, element, procrank, MPI\_COMM\_WORLD, &status);

}

}

}

else if (procrank !=0)

{

MPI\_Recv(&factorforshift, 1, MPI\_INT, 0, procrank, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&values, 1, MPI\_INT, 0, procrank, MPI\_COMM\_WORLD, &status);

int input\_array[values];

int new\_array[values];

int shift\_portion[values];

int inter\_array[values];

if(factorforshift < 0)

{

new\_position = factorforshift \* -1;

}

else

{

new\_position = factorforshift;

}

MPI\_Recv(&input\_array, values, MPI\_INT, 0, procrank, MPI\_COMM\_WORLD, &status);

shiftArray(input\_array, new\_array, factorforshift,shift\_portion);

if(factorforshift >= 0)

{

MPI\_Recv(&inter\_array, new\_position, MPI\_INT, (procrank - 1), procrank, MPI\_COMM\_WORLD, &status);

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

{

new\_array[i] = inter\_array[i];

}

}

if(factorforshift < 0)

{

if(procrank == (sizeval - 1))

{

MPI\_Recv(&inter\_array, new\_position, MPI\_INT, 0, procrank, MPI\_COMM\_WORLD, &status);

}

else

{

MPI\_Recv(&inter\_array, new\_position, MPI\_INT, (procrank + 1), procrank, MPI\_COMM\_WORLD, &status);

}

for(i = 0, element = (values + factorforshift); element < values; i++, element++)

{

new\_array[element] = inter\_array[i];

}

}

if(procrank == (sizeval - 1))

{

if(factorforshift >= 0)

{

MPvaluesend(&shift\_portion, new\_position, MPI\_INT, 0, procrank, MPI\_COMM\_WORLD);

}

else

{

MPvaluesend(&shift\_portion, new\_position, MPI\_INT, (procrank - 1), (procrank - 1), MPI\_COMM\_WORLD);

}

}

else

{

if(factorforshift >= 0)

{

MPvaluesend(&shift\_portion, new\_position, MPI\_INT, (procrank + 1), (procrank + 1), MPI\_COMM\_WORLD);

}

else

{

MPvaluesend(&shift\_portion, new\_position, MPI\_INT, (procrank - 1), (procrank - 1), MPI\_COMM\_WORLD);

}

}

MPI\_Recv(&factorforshift, 1, MPI\_INT, 0, procrank, MPI\_COMM\_WORLD, &status);

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

{

printf("Input array: %d, Array shift : %d \n",input\_array[i], new\_array[i]);

}

MPvaluesend(&factorforshift, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);

}

MPI\_Finalize();

return 0;

}

void shiftArray(int original\_array[], int new\_array[], int factorforshift, int shift\_portion[])

{

int inter\_array[values];

if(factorforshift >= 0)

{

Right(original\_array, new\_array, factorforshift, shift\_portion);

}

if(factorforshift < 0)

{

Left(original\_array, new\_array, factorforshift, shift\_portion);

}

}

void Right(int original\_array[], int new\_array[], int factorforshift, int shift\_portion[])

{

int i = 0, j = 0;

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

{

new\_array[j] = original\_array[i];

}

for(i = 1; i < ( values - factorforshift); i++, j++)

{

new\_array[j] = original\_array[i];

}

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

{

shift\_portion[i-1] = original\_array[values - factorforshift + i - 1];

}

}

void Left(int original\_array[], int new\_array[], int factorforshift, int shift\_portion[])

{

int i = 0, j = 0;

i = -1;

for(j = 0; j <= (values + factorforshift); j++)

{

new\_array[j] = original\_array[(factorforshift \* i) + j];

}

for(j = 0; j <= (factorforshift \* i); j++)

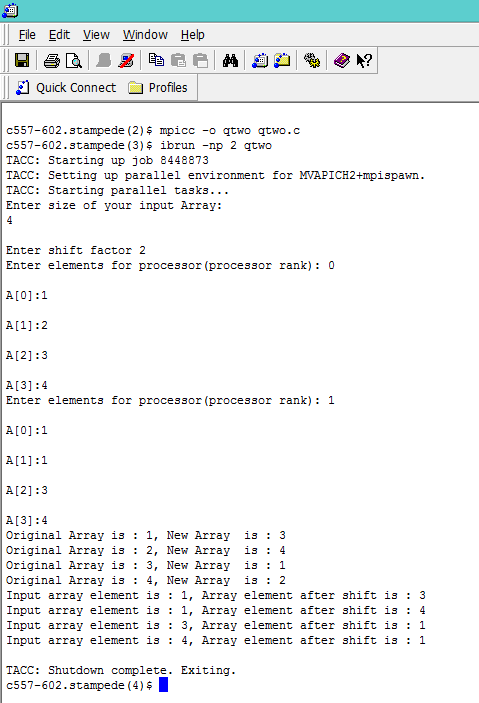
{

shift\_portion[j] = original\_array[j];

}

}

Code compilation and output:



1. **Question three**

**MPI\_scatter CODE**

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

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

int size, rank;

MPI\_Init(&argc, &argv);

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

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

int \*globaldata=NULL;

int localdata;

if (rank == 0) {

globaldata = malloc(size \* sizeof(int) );

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

globaldata[i] = 2\*i+1;

printf("Processor %d has data: ", rank);

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

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

printf("\n");

}

MPI\_Scatter(globaldata, 1, MPI\_INT, &localdata, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

printf("Processor %d has data %d\n", rank, localdata);

localdata \*= 2;

printf("Processor %d doubling the data, now has %d\n", rank, localdata);

MPI\_Gather(&localdata, 1, MPI\_INT, globaldata, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

if (rank == 0) {

printf("Processor %d has data: ", rank);

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

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

printf("\n");

}

if (rank == 0)

free(globaldata);

MPI\_Finalize();

return 0;

}

**MPI\_Gather Code:**

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

int main(int cvalue, char \*\*vvalue)

{

int size, rank;

MPI\_Init(&cvalue, &vvalue);

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

struct timeval tvBegin,tvEnd,tvDiff;

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

int \*globalvar=NULL;

int localvar;

if (rank == 0) {

globalvar = malloc(size \* sizeof(int) );

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

globalvar[i] = 2\*i+1;

gettimeofday(&tvBegin,NULL);

printf("%d data on Processor : ", rank);

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

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

printf("\n");

}

MPI\_Send(globalvar, 1, MPI\_INT, &localvar, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

printf("Processor %d has data %d\n", rank, localvar);

MPI\_Gather(&localvar, 1, MPI\_INT, globalvar, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

gettimeofday(&tvEnd,NULL);

if(rank==0){

printf("The Point-to-point communication time is %f\n",(float)( (tvEnd.tv\_usec-tvBegin.tv\_usec)/100));

}

if (rank == 0)

{

printf("Processor %d has data: ", rank);

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

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

printf("\n");

}

if (rank == 0)

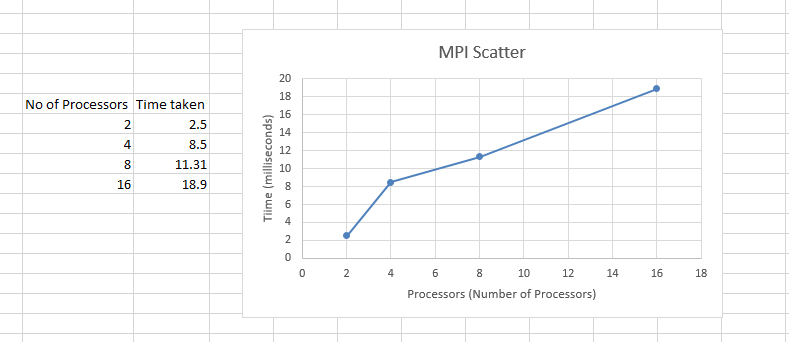
free(globalvar);

MPI\_Finalize();

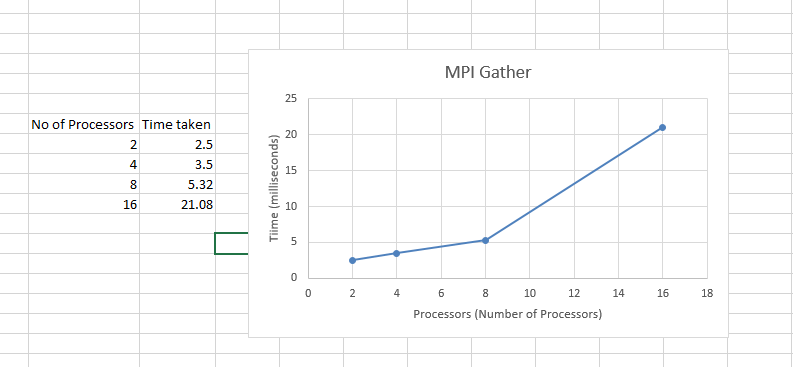
return 0;

}

MPI Scatter data plotted:



MPI Gather data plotted:



----------------- End of Document--------------------