Parallel and Distributed computing Lab Assignment

## Name: Om Ashish Mishra

## Registration Number: 16BCE0789

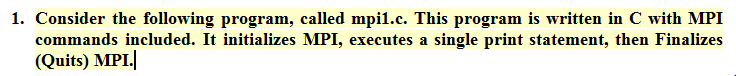
## Slot: B2

The Steps to run MPI

* Create empty project
* Then click Solution Explorer
* Then in Source select item
* In item, write a <projectname>.c , press OK
* Write the code on the screen.
* Click Project on the Menu Bar and go to Properties at the end of the options visible.
* Dropdown C/C++
* Select General
* Double Click on AdditionalIncludeDirectories and add C:\Program Files\Microsoft HPC Pack 2012\Inc
* Then drop down Linker below C/C++
* Select General
* Double Click on AdditionalLibraryDirectories and add C:\Program Files\Microsoft HPC Pack 2012\Lib\i386
* Select Input below General in Linker
* Double Click on AdditionalDependencies and add msmpi.lib to it.
* Click Apply and OK
* Click Build in Menu bar and select Build Solution
* (After Successful execution)Click Debug on the Menu Bar and select Start Debugging
* The output will be shown.

Else

* Go to Command Prompt
* Add the path of the Debug
* Cd C:\Users\16BCE0789\Documents\Visual Studio 2012\Projects\Project1\Debug
* Then type: mpiexec <projectname>
* For more no of threads type: mpi exec –np 20 <projectname>

The Code:

**#include < mpi.h> /\* PROVIDES THE BASIC MPI DEFINITION AND TYPES \*/**

**#include < stdio.h>**

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

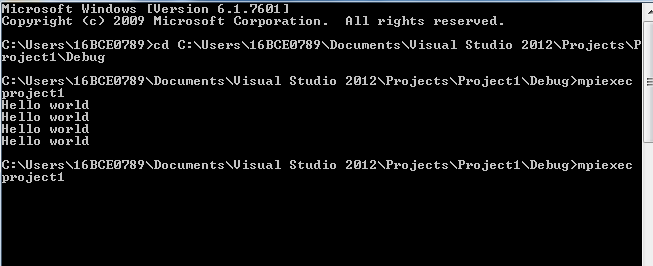
**MPI\_Init(&argc, &argv); /\*START MPI \*/**

**printf("Hello world\n");**

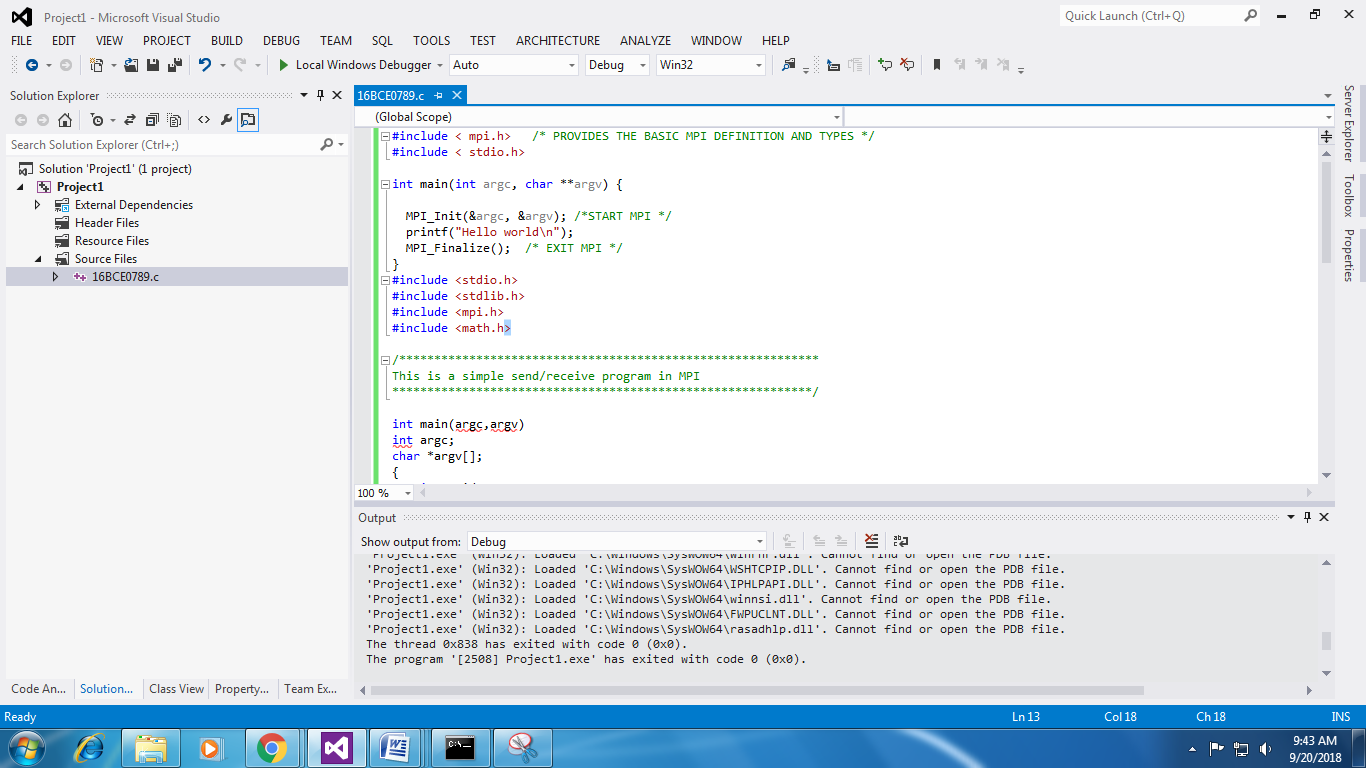
**MPI\_Finalize(); /\* EXIT MPI \*/**

**}**

The Output



The Code:



#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

#include <math.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

This is a simple hello world program. Each processor prints out

it's rank and the size of the current MPI run (Total number of

processors).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void lam\_darwin\_malloc\_linker\_hack(){};

int main(argc,argv)

int argc;

char \*argv[];

{

int myid, numprocs;

FILE \*f1;

int i;

MPI\_Init(&argc,&argv);

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

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

/\* print out my rank and this run's PE size\*/

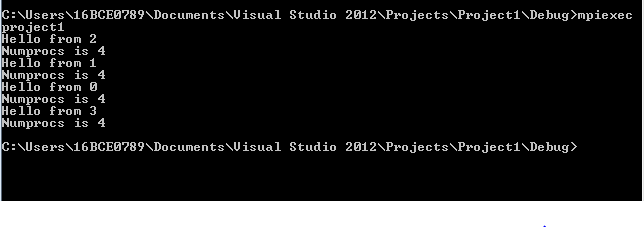
printf("Hello from %d\n",myid);

printf("Numprocs is %d\n",numprocs);

MPI\_Finalize();

}

The Output:



The Code:

#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

#include <math.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

This is a simple send/receive program in MPI

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(argc,argv)

int argc;

char \*argv[];

{

int myid, numprocs;

int tag,source,destination,count;

int buffer;

MPI\_Status status;

MPI\_Init(&argc,&argv);

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

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

tag=1234;

source=0;

destination=1;

count=1;

if(myid == source){

buffer=5678;

MPI\_Send(&buffer,count,MPI\_INT,destination,tag,MPI\_COMM\_WORLD);

printf("processor %d sent %d\n",myid,buffer);

}

if(myid == destination){

MPI\_Recv(&buffer,count,MPI\_INT,source,tag,MPI\_COMM\_WORLD,&status);

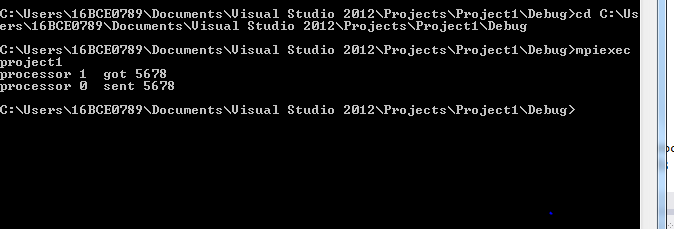
printf("processor %d got %d\n",myid,buffer);

}

MPI\_Finalize();

}

The Output:



**4) Dot Product of Vectors**

The Code:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* FILE: omp\_dotprod\_mpi.c

\* DESCRIPTION:

\* This simple program is the MPI version of a dot product and the third

\* of four codes used to show the progression from a serial program to a

\* hybrid MPI/OpenMP program. The relevant codes are:

\* - omp\_dotprod\_serial.c - Serial version

\* - omp\_dotprod\_openmp.c - OpenMP only version

\* - omp\_dotprod\_mpi.c - MPI only version

\* - omp\_dotprod\_hybrid.c - Hybrid MPI and OpenMP version

\* SOURCE: Blaise Barney

\* LAST REVISED: 06/02/17 Blaise Barney

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

/\* Define length of dot product vectors \*/

#define VECLEN 10

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

{

int i,myid, numprocs, len=VECLEN;

double \*a, \*b;

double mysum, allsum;

/\* MPI Initialization \*/

MPI\_Init (&argc, &argv);

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

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

/\*

Each MPI task performs the dot product, obtains its partial sum, and then calls

MPI\_Reduce to obtain the global sum.

\*/

if (myid == 0)

printf("Starting omp\_dotprod\_mpi. Using %d tasks...\n",numprocs);

/\* Assign storage for dot product vectors \*/

a = (double\*) malloc (len\*sizeof(double));

b = (double\*) malloc (len\*sizeof(double));

/\* Initialize dot product vectors \*/

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

a[i]=1.0;

b[i]=a[i];

}

/\* Perform the dot product \*/

mysum = 0.0;

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

{

mysum += a[i] \* b[i];

}

printf("Task %d partial sum = %f\n",myid, mysum);

/\* After the dot product, perform a summation of results on each node \*/

MPI\_Reduce (&mysum, &allsum, 1, MPI\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);

if (myid == 0)

printf ("Done. MPI version: global sum = %f \n", allsum);

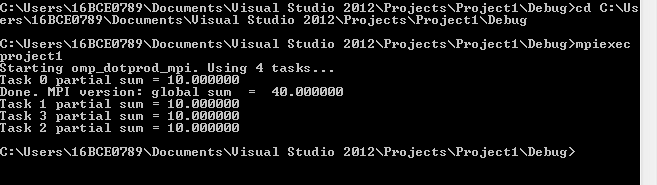
free (a);

free (b);

MPI\_Finalize();

}

The Output:



**2) Most parallel codes assign different tasks to different processors. For example, parts of an input data set might be divided and processed by different processors, or a finite difference grid might be divided among the processors available. This means that the code needs to identify processors. In this example, processors are identified by rank - an integer from 0 to total number of processors - 1.**

**#include < mpi.h> /\* PROVIDES THE BASIC MPI DEFINITION AND TYPES \*/**

**#include < stdio.h>**

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

**int my\_rank;**

**int size;**

**MPI\_Init(&argc, &argv); /\*START MPI \*/**

**/\*DETERMINE RANK OF THIS PROCESSOR\*/**

**MPI\_Comm\_rank(MPI\_COMM\_WORLD, &my\_rank);**

**/\*DETERMINE TOTAL NUMBER OF PROCESSORS\*/**

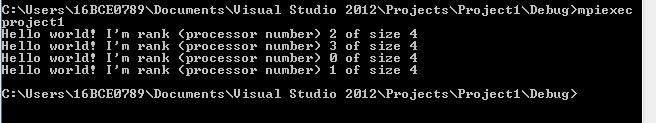
**MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);**

**printf("Hello world! I'm rank (processor number) %d of size %d\n", my\_rank, size);**

**MPI\_Finalize(); /\* EXIT MPI \*/**

**}**

The Output:



1. **Two additional MPI commands may be used to direct traffic (message queuing) during the program execution: MPI\_Send() and MPI\_Recv(). It is safe to say these two commands are at the heart of MPI. Use of these statements makes the program appear more complicated, but it is well worth it if the flow of the program needs to be controlled.**

**#include < mpi.h> /\* PROVIDES THE BASIC MPI DEFINITION AND TYPES \*/**

**#include < stdio.h>**

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

**int my\_rank;**

**int partner;**

**int size, i,t;**

**char greeting[100];**

**MPI\_Status stat;**

**MPI\_Init(&argc, &argv); /\*START MPI \*/**

**MPI\_Comm\_rank(MPI\_COMM\_WORLD, &my\_rank); /\*DETERMINE RANK OF THIS PROCESSOR\*/**

**MPI\_Comm\_size(MPI\_COMM\_WORLD, &size); /\*DETERMINE TOTAL NUMBER OF PROCESSORS\*/**

**sprintf(greeting, "Hello world: processor %d of %d\n", my\_rank, size);**

**/\* adding a silly conditional statement like the**

**following graphically illustrates "blocking" and**

**flow control during program execution**

**if (my\_rank == 1) for (i=0; i<1000000000; i++) t=i;**

**\*/**

**if (my\_rank ==0) {**

**fputs(greeting, stdout);**

**for (partner = 1; partner < size; partner++){**

**MPI\_Recv(greeting, sizeof(greeting), MPI\_BYTE, partner, 1, MPI\_COMM\_WORLD, &stat);**

**fputs (greeting, stdout);**

**}**

**}**

**else {**

**MPI\_Send(greeting, strlen(greeting)+1, MPI\_BYTE, 0,1,MPI\_COMM\_WORLD);**

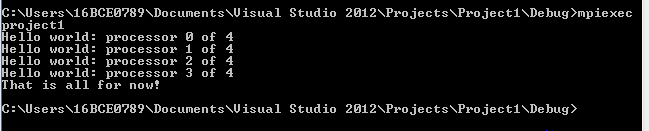
**}**

**if (my\_rank == 0) printf("That is all for now!\n");**

**MPI\_Finalize(); /\* EXIT MPI \*/**

**}**

The Output:



Here is the syntax for MPI\_Send and MPI\_Recv:

The Code:

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**This is a simple send/receive program in MPI**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

#include <math.h>

int main(argc,argv)

int argc;

char \*argv[];

{

int myid, numprocs;

int tag,source,destination,count;

int buffer;

MPI\_Status status;

MPI\_Init(&argc,&argv);

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

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

tag=1;

source=0;

destination=1;

count=1;

if(myid == source){

buffer=100;

MPI\_Send(&buffer,count,MPI\_INT,destination,tag,MPI\_COMM\_WORLD);

printf("processor %d sent %d\n",myid,buffer);

}

if(myid == destination){

MPI\_Recv(&buffer,count,MPI\_INT,source,tag,MPI\_COMM\_WORLD,&status);

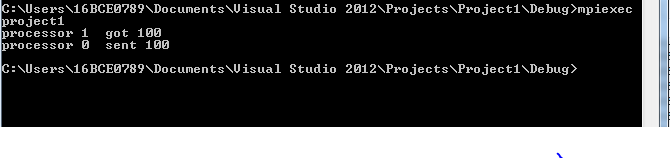
printf("processor %d got %d\n",myid,buffer);

}

MPI\_Finalize();

}

The Output:



5. Program to find Matrix Multiplication.

Ans:

#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

#define ms 2

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

{

    int i,j,k;

    int x,c;

    int matrix\_a[ms][ms];

    int matrix\_b[ms][ms];

    int matrix\_c[ms][ms];

    int myrank, p;

    int NRPE;

    double starttime, endtime;

    MPI\_Init(&argc, &argv);

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

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

    MPI\_Status status;

    NRPE = ms / p;

    if(myrank == 0)

    {

         printf("\nParellel Processing : Matrix Multiplication\n");

         printf("\nby Om Ashish Mishra 16BCE0789\n");

         printf("\n====================================================\n");

        //matrix a

        printf("\n   matrix a \n");

        printf("--------------\n");

        for(i=0; i<ms; ++i)

            for(j=0; j<ms; ++j)

                matrix\_a[i][j] = rand() % 10;

        for(i=0; i<ms; ++i)

        {

            for(j=0; j<ms; ++j)

                printf("%3d", matrix\_a[i][j]);

            printf("\n");

        }

        //matrix b

        printf("\n   matrix b \n");

        printf("--------------\n");

        for(x=0; x<ms; ++x)

            for(c=0; c<ms; ++c)

                matrix\_b[x][c] = rand() % 10;

        for(x=0; x<ms; ++x)

        {

            for(c=0; c<ms; ++c)

                printf("%3d", matrix\_b[x][c]);

            printf("\n");

        }

    }

    //Broadcast Matrix B values to all proccess

    for(i=0; i < ms; i++)

    {

        MPI\_Bcast(matrix\_b[i], ms\*ms, MPI\_INT, 0, MPI\_COMM\_WORLD);

    }

    printf("\n MATRIX B by Process: %d\n", myrank);

    for(x=0; x<ms; ++x)

    {

        for(c=0; c<ms; ++c)

            printf("%3d", matrix\_b[x][c]);

        printf("\n");

    }

    //End of Broadcast

    //Sending of each row of Matrix A to all process

    for(i=0; i<p; i++)

    {

      for(j=0; j<ms; j++)

       {

            MPI\_Reduce(&matrix\_a[j], ms\*NRPE, MPI\_INT, i, 0, MPI\_COMM\_WORLD);

            NRPE++;

       }

    }

    //end of sending

    //Receiving of each row of Matrix A per process

    MPI\_Reduce(matrix\_a, ms\*NRPE, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, &status);

    //Multiplication Area

    starttime = MPI\_Wtime();

    for (k=0; k<ms; k++)

    for (i=0; i<ms; i++) {

      matrix\_c[i][k] = 0;

      for (j=0; j<ms; j++)

        matrix\_c[i][k] = matrix\_c[i][k] + matrix\_a[i][j] \* matrix\_b[j][k];

      }

      endtime   = MPI\_Wtime();

    MPI\_Send(&matrix\_c[i][k], ms\*ms, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);

    //end of multiplication

    //Display area

    if(myrank == 0)

    {

        //MPI\_Recv(matrix\_c, ms\*ms, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, &status);

        printf("\nRESULT: matrix c \n");

        printf("----------------\n");

         for (i=0; i<ms; i++)

         {

          printf("\n");

          for (k=0; k<ms; k++)

           printf("%3d ", matrix\_c[i][k]);

         }

         printf("\n\nParellel Time %f seconds\n",endtime-starttime);

    }

  printf ("\n");

    return 0;

    MPI\_Finalize();

}

