**Vellore Institute of Technology**

**Vellore**

**Parallel and Distributed Computing**

**NAME: Aman Agarwal(15BCE0600)**

**Submitted to :- Prof . Srimati C**

**Slot:-L55+L56**

**6. MPI – Collective operation with ‘collective computation’**

**6.1**

#include "mpi.h"

#include <stdio.h>

#include <stdlib.h>

/\* A simple test of Reduce with all choices of root process \*/

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

{

int errs = 0;

int rank, size, root;

int \*sendbuf, \*recvbuf, i;

int minsize = 2, count;

MPI\_Comm comm;

MPI\_Init( &argc, &argv );

comm = MPI\_COMM\_WORLD;

/\* Determine the sender and receiver \*/

MPI\_Comm\_rank( comm, &rank );

MPI\_Comm\_size( comm, &size );

for (count = 1; count < 130000; count = count \* 2) {

sendbuf = (int \*)malloc( count \* sizeof(int) );

recvbuf = (int \*)malloc( count \* sizeof(int) );

for (root = 0; root < size; root ++) {

for (i=0; i<count; i++) sendbuf[i] = i;

for (i=0; i<count; i++) recvbuf[i] = -1;

MPI\_Reduce( sendbuf, recvbuf, count, MPI\_INT, MPI\_SUM, root, comm );

if (rank == root) {

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

if (recvbuf[i] != i \* size) {

errs++;

}

}

}

}

free( sendbuf );

free( recvbuf );

}

MPI\_Finalize();

return errs;

}



**6.2**

#include "mpi.h"

#include <stdio.h>

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

{

int count = 1000;

int \*in, \*out, \*sol;

int i, fnderr=0;

int rank, size;

MPI\_Init(&argc, &argv);

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

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

in = (int \*)malloc( count \* sizeof(int) );

out = (int \*)malloc( count \* sizeof(int) );

sol = (int \*)malloc( count \* sizeof(int) );

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

{

\*(in + i) = i;

\*(sol + i) = i\*size;

\*(out + i) = 0;

}

MPI\_Allreduce( in, out, count, MPI\_INT, MPI\_SUM, MPI\_COMM\_WORLD );

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

{

if (\*(out + i) != \*(sol + i))

{

fnderr++;

}

}

if (fnderr)

{

fprintf( stderr, "(%d) Error for type MPI\_INT and op MPI\_SUM\n", rank );

fflush(stderr);

}

free( in );

free( out );

free( sol );

MPI\_Finalize();

return fnderr;

}



**7. MPI – Non-blocking operation**

**7.1**

#include "mpi.h"

#include <stdio.h>

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

{

int myid, numprocs, left, right;

int buffer[10], buffer2[10];

MPI\_Request request, request2;

MPI\_Status status;

MPI\_Init(&argc,&argv);

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

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

right = (myid + 1) % numprocs;

left = myid - 1;

if (left < 0)

left = numprocs - 1;

MPI\_Irecv(buffer, 10, MPI\_INT, left, 123, MPI\_COMM\_WORLD, &request);

MPI\_Isend(buffer2, 10, MPI\_INT, right, 123, MPI\_COMM\_WORLD, &request2);

MPI\_Wait(&request, &status);

MPI\_Wait(&request2, &status);

MPI\_Finalize();

return 0;

}



**7.2**

#include "mpi.h"

#include <stdio.h>

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

{

int myid, numprocs, left, right;

int buffer[10], buffer2[10];

MPI\_Request request;

MPI\_Status status;

MPI\_Init(&argc,&argv);

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

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

right = (myid + 1) % numprocs;

left = myid - 1;

if (left < 0)

left = numprocs - 1;

MPI\_Irecv(buffer, 10, MPI\_INT, left, 123, MPI\_COMM\_WORLD, &request);

MPI\_Send(buffer2, 10, MPI\_INT, right, 123, MPI\_COMM\_WORLD);

MPI\_Wait(&request, &status);

MPI\_Finalize();

return 0;

}

