**Program-8**

**Write MPI- C program which approximates an integral using a quadrature rule.**

# include <math.h>

# include <mpi.h>

# include <stdio.h>

# include <stdlib.h>

# include <time.h>

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

double f ( double x );

void timestamp ( );

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

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

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

/\*

Purpose:

MAIN is the main program for QUAD\_MPI.

Licensing:

This code is distributed under the GNU LGPL license.

Modified:

19 July 2010

Author:

John Burkardt

\*/

{

double a;

double b;

double error;

double exact;

int i;

int master = 0;

double my\_a;

double my\_b;

int my\_id;

int my\_n;

double my\_total;

int n;

int p;

int p\_num;

int source;

MPI\_Status status;

int tag;

int target;

double total;

double wtime;

double x;

a = 0.0;

b = 10.0;

n = 10000000;

exact = 0.49936338107645674464;

/\*

Initialize MPI.

\*/

MPI\_Init ( &argc, &argv );

/\*

Get this processor's ID.

\*/

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

/\*

Get the number of processes.

\*/

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

if ( my\_id == master )

{

/\*

We want N to be the total number of evaluations.

If necessary, we adjust N to be divisible by the number of processes.

\*/

my\_n = n / ( p\_num - 1 );

n = ( p\_num - 1 ) \* my\_n;

wtime = MPI\_Wtime ( );

timestamp ( );

printf ( "\n" );

printf ( "QUAD\_MPI\n" );

printf ( " C/MPI version\n" );

printf ( " Estimate an integral of f(x) from A to B.\n" );

printf ( " f(x) = 50 / (pi \* ( 2500 \* x \* x + 1 ) )\n" );

printf ( "\n" );

printf ( " A = %f\n", a );

printf ( " B = %f\n", b );

printf ( " N = %d\n", n );

printf ( " EXACT = %24.16f\n", exact );

printf ( "\n" );

printf ( " Use MPI to divide the computation among\n" );

printf ( " multiple processes.\n" );

}

source = master;

MPI\_Bcast ( &my\_n, 1, MPI\_INT, source, MPI\_COMM\_WORLD );

/\*

Process 0 assigns each process a subinterval of [A,B].

\*/

if ( my\_id == master )

{

for ( p = 1; p <= p\_num - 1; p++ )

{

my\_a = ( ( double ) ( p\_num - p ) \* a

+ ( double ) ( p - 1 ) \* b )

/ ( double ) ( p\_num - 1 );

target = p;

tag = 1;

MPI\_Send ( &my\_a, 1, MPI\_DOUBLE, target, tag, MPI\_COMM\_WORLD );

my\_b = ( ( double ) ( p\_num - p - 1 ) \* a

+ ( double ) ( p ) \* b )

/ ( double ) ( p\_num - 1 );

target = p;

tag = 2;

MPI\_Send ( &my\_b, 1, MPI\_DOUBLE, target, tag, MPI\_COMM\_WORLD );

}

total = 0.0;

my\_total = 0.0;

}

/\*

Processes receive MY\_A, MY\_B, and compute their part of the integral.

\*/

else

{

source = master;

tag = 1;

MPI\_Recv ( &my\_a, 1, MPI\_DOUBLE, source, tag, MPI\_COMM\_WORLD, &status );

source = master;

tag = 2;

MPI\_Recv ( &my\_b, 1, MPI\_DOUBLE, source, tag, MPI\_COMM\_WORLD, &status );

my\_total = 0.0;

for ( i = 1; i <= my\_n; i++ )

{

x = ( ( double ) ( my\_n - i ) \* my\_a

+ ( double ) ( i - 1 ) \* my\_b )

/ ( double ) ( my\_n - 1 );

my\_total = my\_total + f ( x );

}

my\_total = ( my\_b - my\_a ) \* my\_total / ( double ) ( my\_n );

printf ( " Process %d contributed MY\_TOTAL = %f\n", my\_id, my\_total );

}

/\*

Each process sends its value to the master process.

\*/

MPI\_Reduce ( &my\_total, &total, 1, MPI\_DOUBLE, MPI\_SUM, master, MPI\_COMM\_WORLD );

/\*

Compute the weighted estimate.

\*/

if ( my\_id == master )

{

error = fabs ( total - exact );

wtime = MPI\_Wtime ( ) - wtime;

printf ( "\n" );

printf ( " Estimate = %24.16f\n", total );

printf ( " Error = %e\n\n", error );

printf ( " Time = %f\n\n", wtime );

}

/\*

Terminate MPI.

\*/

MPI\_Finalize ( );

/\*

Terminate.

\*/

if ( my\_id == master )

{

printf ( "\n" );

printf ( "QUAD\_MPI:\n" );

printf ( " Normal end of execution.\n" );

printf ( "\n" );

timestamp ( );

}

return 0;

}

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

double f ( double x )

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

/\*

Purpose:

F evaluates the function.

\*/

{

double pi;

double value;

pi = 3.141592653589793;

value = 50.0 / ( pi \* ( 2500.0 \* x \* x + 1.0 ) );

return value;

}

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

void timestamp ( void )

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

/\*

Purpose:

TIMESTAMP prints the current YMDHMS date as a time stamp.

Example:

31 May 2001 09:45:54 AM

Licensing:

This code is distributed under the GNU LGPL license.

Modified:

24 September 2003

Author:

John Burkardt

Parameters:

None

\*/

{

# define TIME\_SIZE 40

static char time\_buffer[TIME\_SIZE];

const struct tm \*tm;

time\_t now;

now = time ( NULL );

tm = localtime ( &now );

strftime ( time\_buffer, TIME\_SIZE, "%d %B %Y %I:%M:%S %p", tm );

printf ( "%s\n", time\_buffer );

return;

# undef TIME\_SIZE

}







