Parallel & Distributed Computing
CSE525

Assignment **#4** - to be submitted to **Dr. Masroor Hussain**

Numerical Integration of vectors with MPI using Trapezoidal Rule

Submitted by,
Quswar Mahmood Abid, CS2003

Report: MPI based Trapezoidal Rule

As discussed in the class please solve the trapezoidal rule using MPI.

In this assignment, we are required to implement a numerical integral method of trapezoidal rule. Trapezoidal rule uses following formulation to calculate numerical integration for two subsequent inputs.

$$\int_a^b f(x)\,dx \approx (b-a)\cdot\tfrac{f(a)+f(b)}{2}$$

Figure 1. Source: Wikipedia

For an input vector, following calculation must be made.

$$\int_a^b f(x)\,dx \approx \sum_{k=1}^{N} \frac{f(x_{k-1})+f(x_k)}{2}\Delta x_k = \tfrac{\Delta x}{2}\left(f(x_0)+2f(x_1)+2f(x_2)+2f(x_3)+2f(x_4)+\cdots+2f(x_{N-1})+f(x_N)\right)$$

Figure 2. Source: Wikipedia

This makes our implementation very easy because easily availability of its individual components. We can make some editions in how we calculate this by changing how we implement it in code. For example, instead of multiplying each sample by 2, we will 2 out of SOP brackets and subtracting start and end elements. Look at following code to see how it will work.

```
#include <stdio.h>
#include "mpi.h"
#define SIZE 16
#define P    4

int i;

int main(int argc, char* argv[]) {
  int numtasks, rank, sendcount, recvcount, source;
```

```
  float sendbuf_1[SIZE] = {
    1.0, 1.0, 1.0, 1.0,
    1.0, 1.0, 1.0, 1.0,
    1.0, 1.0, 1.0, 1.0,
    1.0, 1.0, 1.0, 1.0  };

  float recvbuf_1[SIZE/P];

  float result, result_, dZ;

  result  = 0;
  result_ = 0;
  dZ = 1;

  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &numtasks);

  if (numtasks == P) {
    source = 0;
    sendcount = SIZE/P;
    recvcount = SIZE/P;

    MPI_Scatter(sendbuf_1, sendcount, MPI_FLOAT, recvbuf_1, recvcount,
    MPI_FLOAT, source, MPI_COMM_WORLD);

/*Trapezoidal Part*/
    for (i=0; i<SIZE/P; i++){
        printf("%f from rank:%d\n", recvbuf_1[i], rank);
        //result = result + ( recvbuf_1[i] * 2);
           result = result + recvbuf_1[i];
```

```
    }

      result = result * 2;

      if (rank == 0){
            result = result - recvbuf_1[0];
      }
      if (rank == P-1){
            result = result - recvbuf_1[(SIZE/P)-1];
      }


    //printf("Result = %f, from process# %d\n ", result, rank);

    MPI_Reduce(&result, &result_ , 1, MPI_FLOAT, MPI_SUM, 0,
MPI_COMM_WORLD);

      result_ = dZ * result_ / 2;

    //printf("Result = %f, from process# %d\n ", result, rank);


    if (rank == 0)
        printf("\nResult = %f \n\n", result_);
  } else {
    printf("Must specify %d processors. Terminating.\n",SIZE);
  }

  MPI_Finalize();
  return 0;
}
```

```
[u1@hpc ~]$ nano trapezoidal.c
[u1@hpc ~]$ mpicc trapezoidal.c
[u1@hpc ~]$ mpiexec -n 4 ./a.out
1.000000 from rank:1
1.000000 from rank:1
1.000000 from rank:1
1.000000 from rank:1
1.000000 from rank:2
1.000000 from rank:2
1.000000 from rank:2
1.000000 from rank:2
1.000000 from rank:0
1.000000 from rank:0
1.000000 from rank:0
1.000000 from rank:0

Result = 15.000000

1.000000 from rank:3
1.000000 from rank:3
1.000000 from rank:3
1.000000 from rank:3
[u1@hpc ~]$
```

Figure 3. input vector containing 1 at 16 points, giving 100% correct output

Well, the above code is my initial code, but during final exam, I implemented it with a better way.

Following snapshot show the time taken by serial version of trapezoidal rule to integrate to run:

```
[u1@hpc final]$ ls
a.out  cuda.cu
[u1@hpc final]$ pwd
/home/u1/final
[u1@hpc final]$ nano trap_seq.c
[u1@hpc final]$ gcc trap_seq.c
[u1@hpc final]$ ./a.out
With n = 1024 trapezoids, our estimate
of the integral from 0.000000 to 1.000000 = 0.334310
[u1@hpc final]$ time a.out
-bash: a.out: command not found

real    0m0.001s
user    0m0.000s
sys     0m0.000s
[u1@hpc final]$
```

Figure 4. run time of sequential code, also notice the result of integration

Following snapshot shot shows that same time is spent in parallel implementation:

```
[u1@hpc final]$ nano trap_seq.c
[u1@hpc final]$ gcc trap_seq.c
[u1@hpc final]$ ./a.out
With n = 1024 trapezoids, our estimate
of the integral from 0.000000 to 1.000000 = 0.334310
[u1@hpc final]$ time a.out
-bash: a.out: command not found

real    0m0.001s
user    0m0.000s
sys     0m0.000s
[u1@hpc final]$ nano trap_mpi.c
[u1@hpc final]$ mpicc trap_mpi.c
[u1@hpc final]$ mpiexec a.out
With n = 1024 trapezoids, our estimate
of the integral from 0.000000 to 1.000000 = 0.334310
[u1@hpc final]$ time a.out
-bash: a.out: command not found

real    0m0.001s
user    0m0.000s
sys     0m0.001s
[u1@hpc final]$
```

Figure 5. some parallel implementations do not show good speedups at small programs

Note: I took help from online code posted [at this link](#).

Following is a relevant code snippet for parallelization:

```
MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);


{
      source = 0;
    MPI_Bcast(&a, 1, MPI_DOUBLE, source, MPI_COMM_WORLD);
      MPI_Bcast(&b, 1, MPI_DOUBLE, source, MPI_COMM_WORLD);
      MPI_Bcast(&n, 1, MPI_INT,    source, MPI_COMM_WORLD);
```

```
            double h_local = (b-a)/n;    /* h is the same for all
processes */
            int    n_local = n/numtasks;  /* So is the number of
trapezoids */
            double a_local = a + rank*n_local*h_local;
            double b_local = a_local + n_local*h_local;


            double integral = Trap(a_local, b_local, n_local, h_local);


            /* Add up the integrals calculated by each process */
            MPI_Reduce(&integral, &total, 1, MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);
```

FULL CODE:

```
//Help taken from this code:
http://homepages.math.uic.edu/~hanson/MPI_Reduce_PPMPI.c, and my assignment
code of MPI

//Completed at 07:36PM

#include <stdio.h>

#include "mpi.h"


int i;


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

      int numtasks, rank, sendcount, recvcount, source;


      double a = 0.0; /* Left endpoint */

      double b = 1.0; /* Right endpoint */

      int n = 1024; /* Number of trapezoids */

      double h; /* Trapezoid base length */

      double total; /* Total integral */


      double Trap(double local_a, double local_b, int local_n, double h);


      //h = (b-a)/n; /* base of trapezoids */


      //Initialize MPI

      MPI_Init(&argc,&argv);

      MPI_Comm_rank(MPI_COMM_WORLD, &rank);

      MPI_Comm_size(MPI_COMM_WORLD, &numtasks);


      {

            source = 0;

          MPI_Bcast(&a, 1, MPI_DOUBLE, source, MPI_COMM_WORLD);

            MPI_Bcast(&b, 1, MPI_DOUBLE, source, MPI_COMM_WORLD);

            MPI_Bcast(&n, 1, MPI_INT,    source, MPI_COMM_WORLD);
```

```
            double h_local = (b-a)/n;    /* h is the same for all processes
*/
            int    n_local = n/numtasks;  /* So is the number of trapezoids
*/
            double a_local = a + rank*n_local*h_local;
            double b_local = a_local + n_local*h_local;

            double integral = Trap(a_local, b_local, n_local, h_local);

            /* Add up the integrals calculated by each process */
            MPI_Reduce(&integral, &total, 1, MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);

            /* Print the result */
            if (rank == 0) {
                  printf("With n = %d trapezoids, our estimate\n", n);
                  printf("of the integral from %f to %f = %f\n", a, b,
total);
            }
      }/*
      else{
            printf("Must specify %d processors. Terminating.\n",P);
      }*/

      MPI_Finalize();

      //printf("With n = %d trapezoids, our estimate\n",n);
      //printf("of the integral from %f to %f = %f\n",a,b,total);
      return 0;
}
```

```
double Trap(double local_a, double local_b, int local_n, double h) {
      double integral;
      double x;
      int i;
      double f(double x);
      integral = (f(local_a) + f(local_b))/2.0;
      x = local_a;
      for (i=1; i<=local_n;i++) {
            x = x+h;
            integral = integral + f(x);
      }
      integral = integral*h;
      return integral;
}
double f(double x) {
      return x*x;
}
```