



UNIVERSIDAD NACIONAL DE SAN AGUSTIN

ESCUELA PROFESIONAL DE
CIENCIA DE LA COMPUTACIÓN
ALGORITMOS PARALELOS

MPI

Alumna :

Rosa Yuliana Gabriela

Paccotacya Yanque

Profesor:

Mg. Alvaro Henry Mamani

Aliaga

Índice

1. Regla Trapezoidal	2
1.1. Algoritmo	2
1.2. Código	2
1.3. Output	4
2. Distribución y lectura de los datos de un vector	4
2.1. Scatter	4
2.2. Gather	5
2.3. Output	6

1. Regla Trapezoidal

Este programa usa MPI para implementar la versión paralela de la regla trapezoidal, estima la integral de a a b de una función $f(x)$ usando n trapezoides.

1.1. Algoritmo

1. Cada proceso calcula su intervalo de integración
2. Cada proceso estima la integral de $f(x)$ sobre su intervalo usando la regla trapezoidal
3. Todos los procesos diferentes de rango 0 mandan su integral al proceso 0.
4. El proceso 0 suma todo lo recibido e imprime el resultado.

1.2. Código

```
1  #include <stdio.h>
2  #include <mpi.h>
3
4  int main(void) {
5      int my_rank, comm_sz, n = 1024, local_n;
6      double a = 0.0, b = 3.0, h, local_a, local_b;
7      double local_int, total_int;
8      int source;
9
10     MPI_Init(NULL, NULL);
11
12     MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
13
14     MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
15
16     h = (b-a)/n;
17     local_n = n/comm_sz;
18     local_a = a + my_rank*local_n*h;
19     local_b = local_a + local_n*h;
20     local_int = Trap(local_a, local_b, local_n, h);
21
22 }
```

```
23     if (my_rank != 0) {
24         MPI_Send(&local_int, 1, MPI_DOUBLE, 0, 0,
25                 MPI_COMM_WORLD);
26     } else {
27         total_int = local_int;
28         for (source = 1; source < comm_sz; source++) {
29             MPI_Recv(&local_int, 1, MPI_DOUBLE, source, 0,
30                     MPI_COMM_WORLD, MPI_STATUS_IGNORE);
31             total_int += local_int;
32         }
33     }
34
35
36     if (my_rank == 0) {
37         printf("Con n = %d trapezoides, nuestra estimación\n", n);
38         printf("de integrar desde %f a %f = %.15e\n",
39               a, b, total_int);
40     }
41
42
43     MPI_Finalize();
44
45     return 0;
46 }
47 double Trap(
48     double left_endpt
49     double right_endpt
50     int    trap_count
51     double base_len
52     double estimate, x;
53     int i;
54
55     estimate = (f(left_endpt) + f(right_endpt))/2.0;
56     for (i = 1; i <= trap_count-1; i++) {
57         x = left_endpt + i*base_len;
58         estimate += f(x);
59     }
60     estimate = estimate*base_len;
61
62     return estimate;
```

```

63 }
64
65
66 double f(double x) {
67     return x*x;
68 }

```

1.3. Output

```

rose@Satellite-S55-A:~/CS_AlgoritmosParalelos/LAB_3_MPI$ mpicc -std=c99 -o trap
trapezoidal.c
rose@Satellite-S55-A:~/CS_AlgoritmosParalelos/LAB_3_MPI$ mpiexec -n 5 ./trap
Con n = 1024 trapezoides, nuestra estimación
de integrar desde 0.000000 a 3.000000 = 8.894946975633502e+00
rose@Satellite-S55-A:~/CS_AlgoritmosParalelos/LAB_3_MPI$ mpiexec -n 10 ./trap
Con n = 1024 trapezoides, nuestra estimación
de integrar desde 0.000000 a 3.000000 = 8.894946975633502e+00
rose@Satellite-S55-A:~/CS_AlgoritmosParalelos/LAB_3_MPI$ mpiexec -n 100 ./trap
Con n = 1024 trapezoides, nuestra estimación
de integrar desde 0.000000 a 3.000000 = 8.381907362490892e+00

```

2. Distribución y lectura de los datos de un vector

Estos programas leen e imprimen un vector entre los procesos usando una distribución de bloques.

2.1. Scatter

MPI_{Scatter} puede ser usada en una función que lee un vector entero en el proceso 0, pero solo manda los componentes necesarios a los otros procesos

```

168     char    prompt[]    /* in */,
169     double  local_vec[] /* out */,
170     int      n           /* in */,
171     int      local_n     /* in */,
172     int      my_rank     /* in */,
173     MPI_Comm comm       /* in */) {
174     double* vec = NULL;
175     int i, local_ok = 1;
176

```

```

177     if (my_rank == 0) {
178         vec = malloc(n*sizeof(double));
179         if (vec == NULL) local_ok = 0;
180         Check_for_error(local_ok, "Read_vector",
181             "No se puede alojar temporary vector", comm);
182         printf("Ingrese vector %s\n", prompt);
183         for (i = 0; i < n; i++)
184             scanf("%lf", &vec[i]);
185         MPI_Scatter(vec, local_n, MPI_DOUBLE,
186             local_vec, local_n, MPI_DOUBLE, 0, comm);
187         free(vec);
188     } else {
189         Check_for_error(local_ok, "Read_vector",
190             "No se puede alojar temporary vector", comm);
191         MPI_Scatter(vec, local_n, MPI_DOUBLE,
192             local_vec, local_n, MPI_DOUBLE, 0, comm);
193     }
194 } /* Read_vector */
195

```

2.2. Gather

Recolecta todos los componentes del vector sobre el proceso 0.

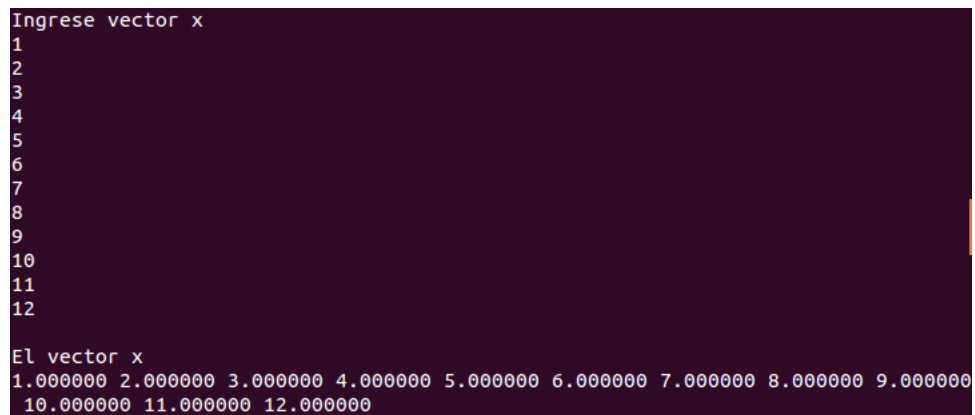
```

232     char        title[]      /* in */,
233     double      local_vec[]  /* in */,
234     int         n            /* in */,
235     int         local_n      /* in */,
236     int         my_rank      /* in */,
237     MPI_Comm    comm         /* in */) {
238     double* vec = NULL;
239     int i, local_ok = 1;
240
241     if (my_rank == 0) {
242         vec = malloc(n*sizeof(double));
243         if (vec == NULL) local_ok = 0;
244         Check_for_error(local_ok, "Print_vector",
245             "No se puede alojar temporary vector", comm);
246         MPI_Gather(local_vec, local_n, MPI_DOUBLE,
247             vec, local_n, MPI_DOUBLE, 0, comm);

```

```
248     printf("\nEl vector %s\n", title);
249     for (i = 0; i < n; i++)
250         printf("%f ", vec[i]);
251     printf("\n");
252     free(vec);
253 } else {
254     Check_for_error(local_ok, "Print_vector",
255                     "Can't allocate temporary vector", comm);
256     MPI_Gather(local_vec, local_n, MPI_DOUBLE,
257               vec, local_n, MPI_DOUBLE, 0, comm);
258 }
259 } /* Print_vector */
260
```

2.3. Output



```
Ingresa vector x
1
2
3
4
5
6
7
8
9
10
11
12

El vector x
1.000000 2.000000 3.000000 4.000000 5.000000 6.000000 7.000000 8.000000 9.000000
10.000000 11.000000 12.000000
```