Informe Paralelos 2

David Neyra Gutierrez

David Neyra

Resumen En el presente informe se detalla la tarea de implementacion de algoritmos

Índice general

Introduccion

Los trabajos realizados en el presente informe se han desarrollado con la libreria MPI, para ello se implemento algoritmos como trapezoide y la funcion sort para poder arreglar la informacion.

1. Trapezoide

En el presente caso se detalla la funcion trapezoide realizada

1.1. Implementacion de Trapezoide

Mostramos a continuaciòn una pequeña muestra que se hizo.
include <stdio.h>include <mpi.h>
double Trap(double left$_e ndpt, double right_e ndpt, int trap_c ount, double base_l en) double estimate, x; int i;$
estimate = (left$_e ndpt + right_e ndpt)$/2,0; $for(i = 1; i < trap_c ount - 1; + +$
$i) x = left_e ndpt + i * base_l en; estimate+ = x; estimate = estimate * base_l en; return estimate;$

int main(void) int $my_rank, comm_sz, n = 1024, local_n$; $double a = 0{,}0, b = 3{,}0, h, local_a, local_b$; $double local_int, total_int$; $int source$;
MPI$_Init(NULL, NULL)$; $MPI_Comm_rank(MPI_COMM_WORLD, my_rank)$; $MPI_Comm_size(MPI_C$
h=(b-a)/n; local$_n = n/comm_sz$;
local$_a = a + my_rank * local_n * h; local_b = local_a + local_n * h; local_int = Trap(local_a, local_b, local_n, h)$;
if(my$_rank != 0) MPI_Send(local_int, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD)$;$else$
total$_int = local_int$; $for(source = 1; source < comm_sz; source{+}{+}) MPI_Recv(local_int, 1, MPI_DOUBL$
if (my$_rank == 0) printf("Width\n = printf("of the integral from MPI_Finalize(); return 0$;

2. Sort

Se implementa funciones sort las cuales se encuentran en el libro.

2.1. Muestra de la funcion Sort

Mostramos a continuaciòn una pequeña muestra que se hizo.

```
void Bubble_sort(int a[] ,int n ) {
  int list_length, i, temp;
  for (list_length = n; list_length >= 2; list_length  )
    for (i = 0; i < list_length      1; i++)
      if (a[i] > a[i+1]) {
        temp = a[i];
        a[i] = a[i+1];
        a[i+1] = temp;
      }
}

void Odd_even_sort(int a[],int n) {
  int phase, i, temp;

  for (phase = 0; phase < n; phase++)
    if (phase %2 == 0) {
      for (i = 1; i < n; i += 2)
        if (a[i      1] > a[i]) {
          temp = a[i];
          a[i] = a[i      1];
          a[i      1] = temp;
        }
    }
    else {
      for (i = 1; i < n      1; i += 2)
        if (a[i] > a[i+1]) {
          temp = a[i];
          a[i] = a[i+1];
          a[i+1] = temp;
        }
```

```
31
32      }
33
34  }
```

Listing 1.1. Sort

3. Scatter y Gather

Se implementa la funcion Scatter y Gather

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4  #include <mpi.h>
5  #include <assert.h>
6
7  float *create_rand_nums(int num_elements) {
8    float *rand_nums = (float *)malloc(sizeof(float) *
       num_elements);
9    assert(rand_nums != NULL);
10   int i;
11   for (i = 0; i < num_elements; i++) {
12     rand_nums[i] = (rand() / (float)RAND_MAX);
13   }
14   return rand_nums;
15 }
16
17 float compute_avg(float *array, int num_elements) {
18   float sum = 0.f;
19   int i;
20   for (i = 0; i < num_elements; i++) {
21     sum += array[i];
22   }
23   return sum / num_elements;
24 }
25 int main(int argc, char** argv) {
26   if (argc != 2) {
27     fprintf(stderr, "Uso: usando proceso\n");
28     exit(1);
29   }
30
31   int num_elements_per_proc = atoi(argv[1]);
32   srand(time(NULL));
33
34   MPI_Init(NULL, NULL);
35
36   int world_rank;
37   MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
```

```
38   int world_size;
39   MPI_Comm_size(MPI_COMM_WORLD, &world_size);
40   float *rand_nums = NULL;
41   if (world_rank == 0) {
42     rand_nums = create_rand_nums(num_elements_per_proc *
       world_size);
43   }
44
45   float *sub_rand_nums = (float *)malloc(sizeof(float) *
       num_elements_per_proc);
46   assert(sub_rand_nums != NULL);
47
48   MPI_Scatter(rand_nums, num_elements_per_proc, MPI_FLOAT,
       sub_rand_nums,
49               num_elements_per_proc, MPI_FLOAT, 0,
       MPI_COMM_WORLD);
50
51   float sub_avg = compute_avg(sub_rand_nums,
       num_elements_per_proc);
52
53   float *sub_avgs = NULL;
54   if (world_rank == 0) {
55     sub_avgs = (float *)malloc(sizeof(float) * world_size);
56     assert(sub_avgs != NULL);
57   }
58   MPI_Gather(&sub_avg, 1, MPI_FLOAT, sub_avgs, 1, MPI_FLOAT,
       0, MPI_COMM_WORLD);
59
60   if (world_rank == 0) {
61     float avg = compute_avg(sub_avgs, world_size);
62     printf("Avg of all elements is %f\n", avg);
63
64     float original_data_avg =
65       compute_avg(rand_nums, num_elements_per_proc *
       world_size);
66     printf("Avg computed across original data is %f\n",
       original_data_avg);
67   }
68
69   if (world_rank == 0) {
70     free(rand_nums);
71     free(sub_avgs);
72   }
73   free(sub_rand_nums);
74
75   MPI_Barrier(MPI_COMM_WORLD);
76   MPI_Finalize();
77 }
```

Listing 1.2. Scatter Gather