Cluster Y Assignments Capitulo 3

Cristhian David Huanca Olazabal

29 de septiembre de 2025

1. Tareas.

1.1. Ejercicio 1

Listing 1: Plantilla básica en C++

```cpp
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

int Find_bin(float value, float min_meas, float max_meas, int bin_count) {
    float bin_width = (max_meas - min_meas) / bin_count;
    int bin = (int)((value - min_meas) / bin_width);
    if (bin == bin_count) bin--;
    return bin;
}

int main(int argc, char* argv[]) {
    int rank, size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    float data[] = {1.3, 2.9, 0.4, 0.3, 1.3, 4.4, 1.7, 0.4, 3.2, 0.3,
                    4.9, 2.4, 3.1, 4.4, 3.9, 0.4, 4.2, 4.5, 4.9, 0.9};
    int data_count = 20;
    float min_meas = 0.0, max_meas = 5.0;
    int bin_count = 5;

    int local_n = data_count / size;
    float* local_data = (float*)malloc(local_n * sizeof(float));

    MPI_Scatter(data, local_n, MPI_FLOAT,
                local_data, local_n, MPI_FLOAT,
                0, MPI_COMM_WORLD);

    int* loc_bin_counts = (int*)calloc(bin_count, sizeof(int));
    for (int i = 0; i < local_n; i++) {
        int bin = Find_bin(local_data[i], min_meas, max_meas, bin_count);
        loc_bin_counts[bin]++;
    }

    int* bin_counts = NULL;
    if (rank == 0) {
        bin_counts = (int*)calloc(bin_count, sizeof(int));
    }

    MPI_Reduce(loc_bin_counts, bin_counts, bin_count, MPI_INT,
               MPI_SUM, 0, MPI_COMM_WORLD);

    if (rank == 0) {
        printf("Histograma global:\n");
        for (int b = 0; b < bin_count; b++) {
            printf("Bin %d: %d\n", b, bin_counts[b]);
        }
    }

    free(local_data);
    free(loc_bin_counts);
    if (rank == 0) free(bin_counts);

    MPI_Finalize();
    return 0;
}
```

```
vagrant@master:~$ ls
ejercicio1.c  frecuencias  frecuencias.c  hosts  main.c  test
vagrant@master:~$ scp frecuencias vagrant@192.168.56.11:~
frecuencias                                   100%   17KB   3.9MB/s   00:00
vagrant@master:~$ scp frecuencias vagrant@192.168.56.12:~
frecuencias                                   100%   17KB   3.0MB/s   00:00
vagrant@master:~$ scp frecuencias vagrant@192.168.56.13:~
frecuencias                                   100%   17KB   4.4MB/s   00:00
vagrant@master:~$ mpirun -np 4 --hostfile hosts ./frecuencias
Histograma global:
Bin 0: 6
Bin 1: 3
Bin 2: 2
Bin 3: 3
Bin 4: 6
vagrant@master:~$
```

Figura 1: Ejercicio 1

1.2. Ejercicio 2

Listing 2: Plantilla básica en C++

```cpp
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#include <time.h>

int main(int argc, char* argv[]) {
    int rank, size;
    long long int tosses, local_tosses;
    long long int local_in_circle = 0;
    long long int total_in_circle = 0;

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

    if (rank == 0) {
        printf("Ingrese el numero total de lanzamientos: ");
        fflush(stdout);
        scanf("%lld", &tosses);
    }

    MPI_Bcast(&tosses, 1, MPI_LONG_LONG_INT, 0, MPI_COMM_WORLD);

    local_tosses = tosses / size;

    unsigned int seed = (unsigned int)(time(NULL) + rank);

    for (long long int i = 0; i < local_tosses; i++) {
        double x = (double)rand_r(&seed) / RAND_MAX * 2.0 - 1.0;
        double y = (double)rand_r(&seed) / RAND_MAX * 2.0 - 1.0;
        double distance_squared = x*x + y*y;
        if (distance_squared <= 1.0) local_in_circle++;
    }

    MPI_Reduce(&local_in_circle, &total_in_circle, 1, MPI_LONG_LONG_INT,
               MPI_SUM, 0, MPI_COMM_WORLD);

    if (rank == 0) {
        double pi_estimate = 4.0 * (double)total_in_circle / ((double)tosses);
        printf("Estimacion de pi = %.10f\n", pi_estimate);
    }

    MPI_Finalize();
    return 0;
}
```

Figura 2: Ejercicio 2

1.3. Ejercicio 3

Listing 3: Plantilla básica en C++

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

int main(int argc, char* argv[]) {
    int rank, size;
    int local_val, partner;
    int step, sum;

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

    local_val = rank + 1;
    sum = local_val;

    for (step = 1; step < size; step *= 2) {
        if (rank % (2*step) == 0) {
            partner = rank + step;
            if (partner < size) {
                int recv_val;
                MPI_Recv(&recv_val, 1, MPI_INT, partner, 0, MPI_COMM_WORLD,
                    MPI_STATUS_IGNORE);
                sum += recv_val;
            }
        } else {
            partner = rank - step;
            MPI_Send(&sum, 1, MPI_INT, partner, 0, MPI_COMM_WORLD);
            break;
        }
    }

    if (rank == 0) {
        printf("Suma global = %d\n", sum);
    }

    MPI_Finalize();
    return 0;
}
```

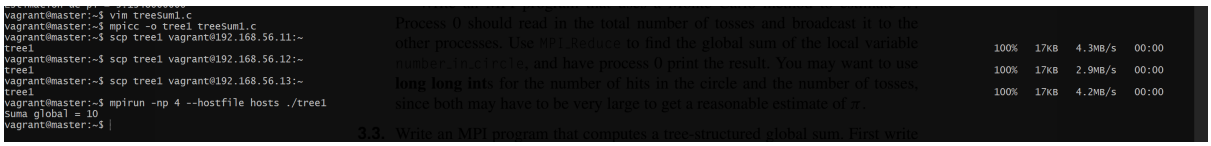

Figura 3: Ejercicio 3 A

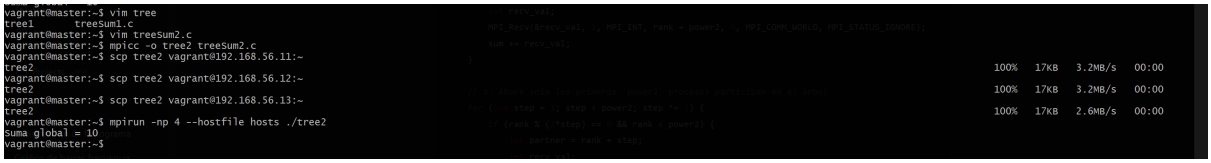

Figura 4: Ejercicio 3 B

1.4. Ejercicio 4

Listing 4: Plantilla básica en C++

```cpp
#include <mpi.h>
#include <stdio.h>
#include <math.h>

int main(int argc, char* argv[]) {
    int my_rank, comm_sz;
    int local_val, global_sum;
    int partner, step;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);

    local_val = my_rank + 1;
    global_sum = local_val;

    int steps = (int) log2(comm_sz);

    for (step = 0; step < steps; step++) {
        partner = my_rank ^ (1 << step);
        int recv_val;

        MPI_Sendrecv(&global_sum, 1, MPI_INT, partner, 0,
                     &recv_val, 1, MPI_INT, partner, 0,
                     MPI_COMM_WORLD, MPI_STATUS_IGNORE);

        global_sum += recv_val;
    }

    printf("Proceso %d: suma global = %d\n", my_rank, global_sum);

    MPI_Finalize();
    return 0;
}
#include <mpi.h>
#include <stdio.h>
#include <math.h>

int main(int argc, char* argv[]) {
    int my_rank, comm_sz;
    int local_val, global_sum;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);

    local_val = my_rank + 1;
    global_sum = local_val;

    int p = 1;
    while (p * 2 <= comm_sz) p *= 2;

    if (my_rank >= p) {
        int dest = my_rank - p;
        MPI_Send(&global_sum, 1, MPI_INT, dest, 0, MPI_COMM_WORLD);
    } else {
        if (my_rank + p < comm_sz) {
            int recv_val;
```

```
            MPI_Recv(&recv_val, 1, MPI_INT, my_rank + p, 0, MPI_COMM_WORLD,
                MPI_STATUS_IGNORE);
            global_sum += recv_val;
        }

        int steps = (int) log2(p);
        for (int step = 0; step < steps; step++) {
            int partner = my_rank ^ (1 << step);
            int recv_val;

            MPI_Sendrecv(&global_sum, 1, MPI_INT, partner, 0,
                         &recv_val, 1, MPI_INT, partner, 0,
                         MPI_COMM_WORLD, MPI_STATUS_IGNORE);

            global_sum += recv_val;
        }
    }

    if (my_rank < p)
        printf("Proceso %d: suma global = %d\n", my_rank, global_sum);

    MPI_Finalize();
    return 0;
}
```

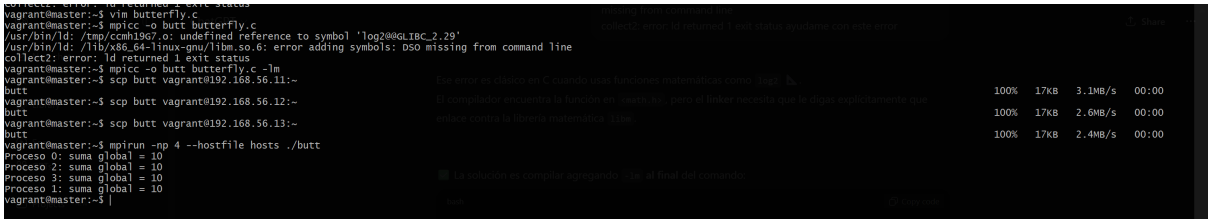

Figura 5: Ejercicio 4

1.5. Ejercicio 5

Listing 5: Plantilla básica en C++

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[]) {
    int n, comm_sz, my_rank;
    double *A = NULL, *x = NULL, *y = NULL;
    double *local_A, *local_x, *local_y;
    int local_cols;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);

    if (my_rank == 0) {
        printf("Ingrese el orden de la matriz (n, divisible por %d): ", comm_sz);
        fflush(stdout);
        scanf("%d", &n);

        A = malloc(n * n * sizeof(double));
        x = malloc(n * sizeof(double));
        y = malloc(n * sizeof(double));

        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                A[i*n + j] = 1.0;

        for (int i = 0; i < n; i++)
            x[i] = 1.0;
    }
```

```
31
32      MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
33
34      local_cols = n / comm_sz;
35
36      local_A = malloc(n * local_cols * sizeof(double));
37      local_x = malloc(local_cols * sizeof(double));
38      local_y = calloc(n, sizeof(double));
39
40      if (my_rank == 0) {
41          for (int p = 0; p < comm_sz; p++) {
42              if (p == 0) {
43                  for (int j = 0; j < local_cols; j++)
44                      for (int i = 0; i < n; i++)
45                          local_A[i*local_cols + j] = A[i*n + j];
46                  for (int j = 0; j < local_cols; j++)
47                      local_x[j] = x[j];
48              } else {
49                  double *send_blockA = malloc(n * local_cols * sizeof(double));
50                  double *send_blockx = malloc(local_cols * sizeof(double));
51
52                  for (int j = 0; j < local_cols; j++)
53                      for (int i = 0; i < n; i++)
54                          send_blockA[i*local_cols + j] = A[i*n + p*local_cols + j];
55
56                  for (int j = 0; j < local_cols; j++)
57                      send_blockx[j] = x[p*local_cols + j];
58
59                  MPI_Send(send_blockA, n*local_cols, MPI_DOUBLE, p, 0, MPI_COMM_WORLD);
60                  MPI_Send(send_blockx, local_cols, MPI_DOUBLE, p, 0, MPI_COMM_WORLD);
61
62                  free(send_blockA);
63                  free(send_blockx);
64              }
65          }
66      } else {
67          MPI_Recv(local_A, n*local_cols, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD,
                MPI_STATUS_IGNORE);
68          MPI_Recv(local_x, local_cols, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD,
                MPI_STATUS_IGNORE);
69      }
70
71      for (int i = 0; i < n; i++) {
72          for (int j = 0; j < local_cols; j++) {
73              local_y[i] += local_A[i*local_cols + j] * local_x[j];
74          }
75      }
76
77      if (my_rank == 0) {
78          MPI_Reduce(MPI_IN_PLACE, local_y, n, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
79      } else {
80          MPI_Reduce(local_y, NULL, n, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
81      }
82
83      if (my_rank == 0) {
84          printf("Resultado␣y␣=␣[␣");
85          for (int i = 0; i < n; i++)
86              printf("%.1f␣", local_y[i]);
87          printf("]\n");
88      }
89
90      free(local_A);
91      free(local_x);
92      free(local_y);
93      if (my_rank == 0) {
94          free(A);
95          free(x);
96          free(y);
97      }
98
99      MPI_Finalize();
100     return 0;
101 }
```

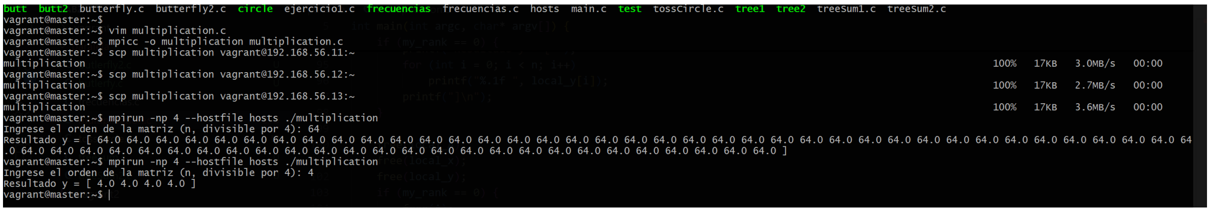

Figura 6: Ejercicio 5

1.6. Ejercicio 7

Listing 6: Plantilla básica en C++

```
#include <mpi.h>
#include <stdio.h>
#include <time.h>

int main(int argc, char* argv[]) {
    int rank;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    int msg = 42;
    int n_pingpong = 100000;
    clock_t start_clock, end_clock;
    double start_wtime, end_wtime;

    MPI_Barrier(MPI_COMM_WORLD);

    if(rank==0) start_clock = clock();
    if(rank==0) start_wtime = MPI_Wtime();

    for(int i=0;i<n_pingpong;i++){
        if(rank==0){
            MPI_Send(&msg, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
            MPI_Recv(&msg, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        } else if(rank==1){
            MPI_Recv(&msg, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            MPI_Send(&msg, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
        }
    }

    MPI_Barrier(MPI_COMM_WORLD);

    if(rank==0){
        end_clock = clock();
        end_wtime = MPI_Wtime();
        printf("Tiempo con clock(): %f s\n", (double)(end_clock - start_clock)/
            CLOCKS_PER_SEC);
        printf("Tiempo con MPI_Wtime(): %f s\n", end_wtime - start_wtime);
    }

    MPI_Finalize();
    return 0;
}
```

Figura 7: Ejercicio 7

1.7. Ejercicio 8

Listing 7: Plantilla básica en C++

```c
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int compare_ints(const void* a, const void* b){
    return (*(int*)a - *(int*)b);
}

int* merge(int* a1, int n1, int* a2, int n2){
    int* result = malloc((n1+n2)*sizeof(int));
    int i=0, j=0, k=0;
    while(i<n1 && j<n2){
        if(a1[i] <= a2[j]) result[k++] = a1[i++];
        else result[k++] = a2[j++];
    }
    while(i<n1) result[k++] = a1[i++];
    while(j<n2) result[k++] = a2[j++];
    return result;
}

int main(int argc, char* argv[]){
    int rank, comm_sz;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);

    int n;
    if(rank==0){
        printf("Ingrese n: "); fflush(stdout);
        scanf("%d",&n);
    }
    MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);

    int local_n = n / comm_sz;
    int* local_keys = malloc(local_n * sizeof(int));

    srand(time(NULL) + rank);
    for(int i=0;i<local_n;i++) local_keys[i] = rand() % 1000;

    qsort(local_keys, local_n, sizeof(int), compare_ints);

    int* gather = NULL;
    if(rank==0) gather = malloc(n*sizeof(int));
    MPI_Gather(local_keys, local_n, MPI_INT, gather, local_n, MPI_INT, 0, MPI_COMM_WORLD
        );

    if(rank==0){
        printf("Listas locales iniciales:\n");
        for(int i=0;i<n;i++){
            printf("%d ", gather[i]);
            if((i+1)%local_n==0) printf("\n");
        }
    }

    free(gather);

    int step = 1;
    int* current_keys = local_keys;
    int current_n = local_n;

    while(step < comm_sz){
        if(rank % (2*step) == 0){
            int partner = rank + step;
            if(partner < comm_sz){
                int recv_n;
                MPI_Recv(&recv_n, 1, MPI_INT, partner, 0, MPI_COMM_WORLD,
                    MPI_STATUS_IGNORE);
                int* recv_keys = malloc(recv_n * sizeof(int));
                MPI_Recv(recv_keys, recv_n, MPI_INT, partner, 0, MPI_COMM_WORLD,
                    MPI_STATUS_IGNORE);

                int* merged = merge(current_keys, current_n, recv_keys, recv_n);
```

```
                free(current_keys);
                free(recv_keys);
                current_keys = merged;
                current_n += recv_n;
            }
        } else {
            int partner = rank - step;
            MPI_Send(&current_n, 1, MPI_INT, partner, 0, MPI_COMM_WORLD);
            MPI_Send(current_keys, current_n, MPI_INT, partner, 0, MPI_COMM_WORLD);
            break;
        }
        step *= 2;
    }

    if(rank==0){
        printf("Lista global ordenada:\n");
        for(int i=0;i<current_n;i++){
            printf("%d ", current_keys[i]);
        }
        printf("\n");
        free(current_keys);
    } else {
        free(current_keys);
    }

    MPI_Finalize();
    return 0;
}
```

```
vagrant@master:~$ vim mergeSort.c
vagrant@master:~$ mpicc -o merge mergeSort.c
vagrant@master:~$ scp merge vagrant@192.168.56.11:~
merge                                         100%   17KB   3.0MB/s   00:00
vagrant@master:~$ scp merge vagrant@192.168.56.12:~
merge                                         100%   17KB   3.6MB/s   00:00
vagrant@master:~$ scp merge vagrant@192.168.56.134:~
ssh: connect to host 192.168.56.134 port 22: No route to host
lost connection
vagrant@master:~$ scp merge vagrant@192.168.56.13:~
merge                                         100%   17KB   2.8MB/s   00:00
vagrant@master:~$ mpirun -np 4 --hostfile hosts ./merge
Ingrese n: 1000
468 930
428 582
623 935
-138586304 22095
Lista global ordenada:
118 428 448 468 582 623 930 935
vagrant@master:~$ mpirun -np 4 --hostfile hosts ./merge
Ingrese n: 100
Listas locales iniciales:
9 14 30 36 99 137 207 231 304 351 364 393 399 409 417 432 464 494 753 758 815 889 892 960 969
8 95 124 136 207 212 226 261 327 366 419 420 426 437 445 471 664 666 767 784 845 899 914 924 954
7 78 97 126 130 169 186 187 205 229 284 322 447 513 537 603 610 678 757 771 795 838 844 904 929
7 78 97 126 130 169 186 187 205 229 284 322 447 513 537 603 610 678 757 771 795 838 844 904 929
Lista global ordenada:
7 7 8 9 14 30 36 78 78 95 97 97 99 124 126 126 130 130 136 137 169 169 186 186 187 187 205 205 207 207 212 226 229 229 231 261 284 284 304 322 322 327 351 364 366 393 399 409 417 419 420 426 432 437 445
447 447 464 471 494 513 513 537 537 603 603 610 610 664 666 678 678 753 757 757 758 767 771 771 784 795 795 815 838 838 844 844 845 889 892 899 904 904 914 924 929 929 954 960 969
vagrant@master:~$
```

Figura 8: Ejercicio 8