Pigeonhole Sort MPI Program

Algorithm

- Initialize:
  Import necessary libraries.
  Define the pigeonhole sort function.
  Main Function:

- Initialize MPI.
  Get process rank and size.
  If rank is 0, input total number of elements.
  Broadcast total elements to all processes.
  Divide data among processes using Scatter.

- Local Sorting:
  Find local minimum and maximum values.
  Reduce to get global minimum and maximum values.
  Calculate range size for pigeonhole sorting.

- Pigeonhole Sort:
  Perform pigeonhole sorting on local arrays.

- Gather and Print:
  Gather sorted sub-arrays from all processes.
  If rank is 0, print the sorted array.

- Finalize:
  Finalize MPI.

- Return:
  Return from main function.

pseudocode

- Import necessary libraries

    Define pigeonhole_sort function
    Initialize local_pigeonholes array
    For each element in local_arr
        Increment corresponding pigeonhole
    Construct sorted_arr from pigeonholes

- Main Function
    Initialize MPI
    Get process rank and size
    If rank is 0
        Input total number of elements
        Broadcast total elements to all processes
        Divide data among processes using Scatter

- Find local minimum and maximum values
    Reduce to get global minimum and maximum values
    Calculate range size for pigeonhole sorting

- Perform pigeonhole sort on local arrays
    Gather sorted sub-arrays from all processes
        If rank is 0
            Print the sorted array

- Finalize MPI
    Return from main function

Open

pigeonhole_mpi.c
~/Documents/HPCS MINI PROJECT

Save

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

void pigeonhole_sort(int local_arr[], int local_n, int min_val, int range_size, int* sorted_arr) {
    int local_pigeonholes[range_size];
    for (int i = 0; i < range_size; i++) {
        local_pigeonholes[i] = 0;
    }

    for (int i = 0; i < local_n; i++) {
        local_pigeonholes[local_arr[i] - min_val]++;
    }

    int index = 0;
    for (int i = 0; i < range_size; i++) {
        while (local_pigeonholes[i] > 0) {
            sorted_arr[index++] = i + min_val;
            local_pigeonholes[i]--;
        }
    }
}

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);

    int world_rank, world_size;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    int n;
    if (world_rank == 0) {
        printf("Enter the number of elements: ");
        scanf("%d", &n);
    }

    MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);

    int local_n = n / world_size;
    int local_arr[local_n];

    int* sendbuf = NULL;
    if (world_rank == 0) {
        sendbuf = (int*)malloc(n * sizeof(int));
        printf("Enter %d elements: ", n);
        for (int i = 0; i < n; i++) {
            scanf("%d", &sendbuf[i]);
        }
    }

    MPI_Scatter(sendbuf, local_n, MPI_INT, local_arr, local_n, MPI_INT, 0, MPI_COMM_WORLD);

    if (world_rank == 0) {
        free(sendbuf);
    }

    int min_val = local_arr[0];
    int max_val = local_arr[0];
    for (int i = 1; i < local_n; i++) {
        if (local_arr[i] < min_val) {
            min_val = local_arr[i];
        }
        if (local_arr[i] > max_val) {
            max_val = local_arr[i];
        }
    }

    int global_min, global_max;
    MPI_Allreduce(&min_val, &global_min, 1, MPI_INT, MPI_MIN, MPI_COMM_WORLD);
    MPI_Allreduce(&max_val, &global_max, 1, MPI_INT, MPI_MAX, MPI_COMM_WORLD);

    int range_size = global_max - global_min + 1;
    int sorted_arr[n];
    pigeonhole_sort(local_arr, local_n, global_min, range_size, sorted_arr);

    int* gathered_sorted_arr = NULL;
    if (world_rank == 0) {
        gathered_sorted_arr = (int*)malloc(n * sizeof(int));
    }

    MPI_Gather(sorted_arr, local_n, MPI_INT, gathered_sorted_arr, local_n, MPI_INT, 0, MPI_COMM_WORLD);

    if (world_rank == 0) {
        printf("Sorted array: ");
        for (int i = 0; i < n; i++) {
            printf("%d ", gathered_sorted_arr[i]);
        }
        printf("\n");

        free(gathered_sorted_arr);
    }

    MPI_Finalize();

    return 0;
}
```

```
rohithsaidatta@rohithsaidatta-VirtualBox:~/Documents/HPCS MINI PROJECT$ mpicc -o pigeonhole_mpi pigeonhole_mpi.c
rohithsaidatta@rohithsaidatta-VirtualBox:~/Documents/HPCS MINI PROJECT$ mpirun -np 4 ./pigeonhole_mpi
Enter the number of elements: 8
Enter 8 elements: 8 3 2 7 4 6 8 5
Sorted array: 3 8 2 7 4 6 5 8
```

- The compiled the code using mpicc and created an executable named pigeonhole_mpi.
- user used mpirun to run the program with 4 processes.
- The program asked you to enter the number of elements, and user typed 8.
- Then, user entered the array elements: 8 3 2 7 4 6 8 5.
- The program sorted the array in parallel and displayed the sorted result: 3 8 2 7 4 6 5 8.
- The program finished execution.

So, the output shows that the program sorted the input array, and the sorted array is displayed at the end. Keep in mind that the order of equal elements might vary due to the sorting algorithm and parallel processing.