Bar star Generation sequentially

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

# define max_range 2000 // assumption regarding the array size
# define iter 10 // to run the code 10 times

void generateBarChart(int size, int* values) {
    int max = values[0];

    // Find the maximum value in the dataset
    for (int i = 1; i < size; i++) {
        if (values[i] > max) {
            max = values[i];
        }
    }

    int* frequency = (int*)calloc(max + 1, sizeof(int));

    // Count the frequency of each value in the dataset
    for(int i = 0; i < size; i++) {
        frequency[values[i]]++;
    }

    // Display the bar chart
    printf("--- Bar chart ---\n");
    for(int i = 1; i <= max; i++) {
        if(frequency[i] >= 0) {
            printf("Data Point %d: ", i);
            for(int j = 0; j < frequency[i]; j++) {
                printf("*");
            }
            printf("\n");
        }
    }

    free(frequency);
}
```

```c
int main() {
    int size;
    clock_t t;

int random_values[max_range];
srand(24);

for(int i = 0; i < max_range; i++) {
    random_values[i] = rand() % max_range + 1; // Generates random values between 1 and max_range
}

double sum;
double elapsed_time[iter+1];

for(int i = 0; i< iter ; i++){
    t= clock();
    generateBarChart(max_range, random_values);
    elapsed_time [i]= clock() -t;
}

for (int i = 0 ; i< iter ; i++){
    sum += elapsed_time[i];
}

for (int i = 0 ; i< iter; i++){
    printf("Time taken for iteration (%d) : %f\n", i ,elapsed_time[i]/CLOCKS_PER_SEC);


    }
    // print average time taken over the 10 iterations
    printf("average time taken: %f seconds\n",(sum/iter)/CLOCKS_PER_SEC);

    return 0;

}
```

Bar star Generation Using OpenMP

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <omp.h>
# define max_range 2000 // assumption regarding the array size
# define iter 10 // to run the code 10 times
double generatePBarChart(int size, int* values, int num_threads) {
double end , start;
// Find the maximum value in the dataset
int max_val = values[0];
start = omp_get_wtime();
#pragma omp parallel for reduction(max:max_val) num_threads(num_threads)
for (int i = 0; i < size; i++) {
if (values[i] > max_val) {
       max_val = values[i]; }
}
int* frequency = (int*)calloc(max_val + 1, sizeof(int));
// Count the frequency of each value in the dataset
#pragma omp parallel for num_threads(num_threads)
 for (int i = 0; i < max_val; i++){
    #pragma omp atomic
        frequency[values[i]]++;
}
end = omp_get_wtime(); /* end time */
// show bar chart
printf("\n- - - Bar chart - - -\n");
for(int i = 1; i <= max_val; i++) {
       if(frequency[i] >= 0) {
            printf("\nData Point %d: ", i);
            for(int j = 0; j < frequency[i]; j++)
                printf("*");

       } // end if
}
printf("\n");
free(frequency);
return end - start; }
int main() { double end, start;
int num_threads; printf("Enter the number of threads: "); scanf("%d", &num_threads);
int random_values[max_range];
/* generate seed value static for fair comparison between sequential and parallel*/
srand(24);
for(int i = 0; i < max_range; i++) { random_values[i] = rand() % max_range + 1; }
// generate bar chart for large sizes
double sum;
double elapsed_time[iter+1];
for(int i = 0; i< iter ; i++){
elapsed_time [i] = generatePBarChart(max_range, random_values, num_threads);
}
for(int j = 0 ; j< iter ; j++){
  sum += elapsed_time[j];
}
for (int j = 0 ; j< iter; j++){
printf("Time taken for iteration (%d) : %f\n", j ,elapsed_time[j]); }
// print average time taken over the 10 iterations
 printf("average time taken: %f seconds\n",sum/iter); return 0; }
```

Bar star Generation Using MPI

```c
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#define DEFAULT_SIZE 20
#define ITERATIONS 10

int main(int argc, char* argv[]) {
//initialize the mpi environment
MPI_Init(&argc, &argv);

int max_range = DEFAULT_SIZE;
int rank, size;

//get the number of processes
MPI_Comm_size(MPI_COMM_WORLD, &size);

//get the rank of the processes
MPI_Comm_rank(MPI_COMM_WORLD, &rank);

double Duration;
double iteration_times[ITERATIONS];

for (int iteration = 0; iteration < ITERATIONS; iteration++) {
int values[max_range];
if (rank == 0) {
//printf("Iteration %d - Generated Dataset: ", iteration + 1);
for (int i = 0; i < max_range; i++) {
values[i] = (rand() % max_range) + 1;
//printing the values of array values
//printf("%d ", values[i]);
}
printf("\n");
}
// Broadcast the array values to all processes
MPI_Bcast(values, max_range, MPI_INT, 0, MPI_COMM_WORLD);
//initialize startTime as current time
double startTime = MPI_Wtime();

int elementsPerProcessor = max_range / size; // Assuming
equal distribution
int extra_elements = max_range % size; // remaining
elements
int start_local = rank * elementsPerProcessor + (rank
< extra_elements ? rank : extra_elements );
int end_local = start_local + elementsPerProcessor + (rank
< extra_elements ? 1 : 0);

// Find the local maximum for each processor
int localMax = values[start_local];
for (int i = start_local + 1; i < end_local; i++)
if (values[i] > localMax)
localMax = values[i];

// Reduce all local maximum to get the global maximum
int globalMax;
MPI_Reduce(&localMax, &globalMax, 1, MPI_INT, MPI_MAX, 0,
MPI_COMM_WORLD);

//Broadcast global max to all processes
MPI_Bcast(&globalMax, 1, MPI_INT, 0, MPI_COMM_WORLD);
```

```
//initialize a local counter array
int Lcount[globalMax];
for (int i = 0; i < globalMax; i++)
Lcount[i] = 0;
//count the occurrences of each value in the local range
for (int i = start; i < end; i++)
Lcount[values[i] - 1]++;

//checking if global max is consistent among processors
//initialize receive counter array
int Rcount[size];
for (int i = 0; i < size; i++)
Rcount[i] = 0;
//gather global max values to Rcount on P0
MPI_Gather(&globalMax, 1, MPI_INT, Rcount, 1, MPI_INT, 0,
MPI_COMM_WORLD);

/*for (int i = 0; i < size; i++)
printf("process%d %d\n", rank, Rcount[i]);*/
// End checking



//initialize global counter array
int Gcount[globalMax * size];
printf("process %d size of gcount %d \n", rank, globalMax * size);
for (int i = 0; i < globalMax * size; i++)
Gcount[i] = 0;


// gather Lcount to array Gcount
MPI_Gather(Lcount, globalMax, MPI_INT, Gcount, globalMax, MPI_INT, 0,
MPI_COMM_WORLD);


if (rank == 0) {
// Print the final bar chart from process 0
printf("\n---Bar chart for Iteration %d---\n", iteration + 1);
for (int i = 0; i < globalMax; i++) {
printf("Data Point %d: ", i + 1);
for (int j = 0; j < size; j++)
for (int k = 0; k < Gcount[j * globalMax + i]; k++)
printf("*");
printf("\n");
}
}
```

```
double EndTime = MPI_Wtime();
//calculate time for each iteration
Duration += EndTime - startTime;
iteration_times[iteration] = EndTime - startTime;
MPI_Barrier(MPI_COMM_WORLD);
if (rank == 0) {
for (int iteration = 0;iteration < ITERATIONS;
iteration++) {
printf("Time taken for iteration (%d) : %f seconds\n",
iteration + 1, iteration_times[iteration]);
}
} // end loop iteration
}
if (rank == 0)
printf("\nAverage execution
time: %f seconds\n", Duration / ITERATIONS);

MPI_Finalize();

return 0;
}
```

Performance Comparison of Bar Star Generation Algorithms