Parallel Histogram Generation using MPI

Author: Frederick Rohn

Abstract

This report presents the design, implementation, and performance analysis of a parallel histogram generation algorithm developed in C using the Message Passing Interface (MPI). The project explores how distributed computation and communication overhead affect scalability and efficiency in multi-process environments. The algorithm distributes data across processes to construct local histograms which are then combined into a global result. Experimental results were obtained on a high-performance computing (HPC) environment to assess runtime and parallel efficiency at varying process counts and problem sizes.

Specification Summary

The objective was to parallelize histogram generation using MPI for two main inputs: the number of data points (up to one billion) and the number of bins. Each process computes a local histogram based on its subset of data, and the results are combined via *MPI_Reduce*. The experiment was repeated for different data sizes (1M and 10M) and process counts (1, 2, 4, 8). Timing data was collected to compute speedup and efficiency, and additional tests were run to evaluate the effect of bin count (4 vs. 10) on performance.

Implementation

The MPI-based algorithm divides the global data array among available processes using *MPI_Scatter*. Each process computes its local histogram by iterating over its assigned subset of random floating-point values. Once local histograms are computed, they are combined using *MPI_Reduce* to form a global histogram on the root process. The correctness of the final histogram is validated against a sequential version, and total execution time is measured using MPI timing functions.

Performance Results

Table 1: Execution Time (s) for 5 Bins

Processes	1	2	4	8
Data Items = 1M	0.003781	0.003056	0.001720	0.001043
Data Items = 10M	0.036516	0.023171	0.019129	0.018335

Table 2: Speedup Results

Processes	1	2	4	8
Speedup (1M)	1	1.237	2.198	3.625
Speedup (10M)	1	1.576	1.909	1.992

Table 3: Parallel Efficiency

Processes	1	2	4	8
Efficiency (1M)	1	0.619	0.550	0.453
Efficiency (10M)	1	0.788	0.477	0.249

Performance Analysis

Speedup increased as the number of processes grew from 1 to 8, demonstrating successful parallelization. However, diminishing returns were observed due to communication overhead and non-parallelizable sections of the program, consistent with Amdahl's Law. Larger datasets (10M) achieved higher absolute speedup but lower efficiency per process, as the cost of MPI communication grew relative to computation. Increasing the number of bins slightly increased runtime due to the larger reduction vectors, though the effect remained small for modest bin counts (4–10).

Conclusion

The parallel histogram algorithm achieved strong performance scaling up to 8 processes, with a measured speedup of approximately 3.6× for 1 million data points. The results demonstrate how effective workload distribution and efficient MPI collective operations can accelerate large-scale computations on HPC systems while highlighting the trade-offs between computation and communication in parallel environments.