MPI Algorithmic Performance Analysis

Justin Newman, Andrew Fleming

[a]*James Madison University Department of Computer Science, Harrisonburg, 22801, VA, United States*

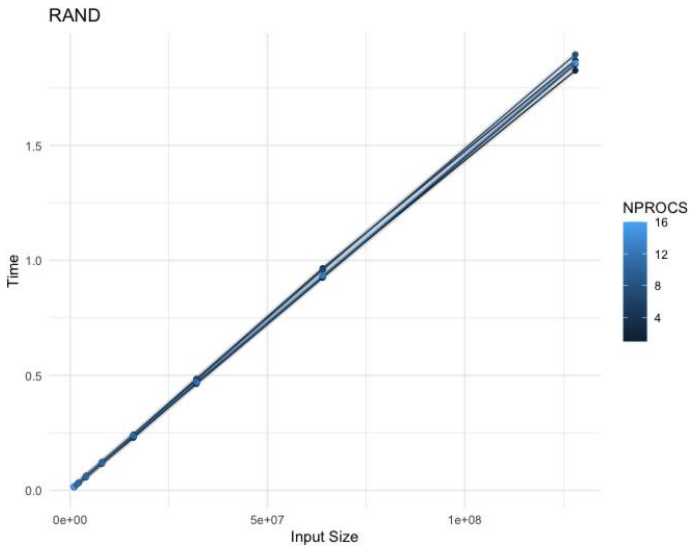
Abstract

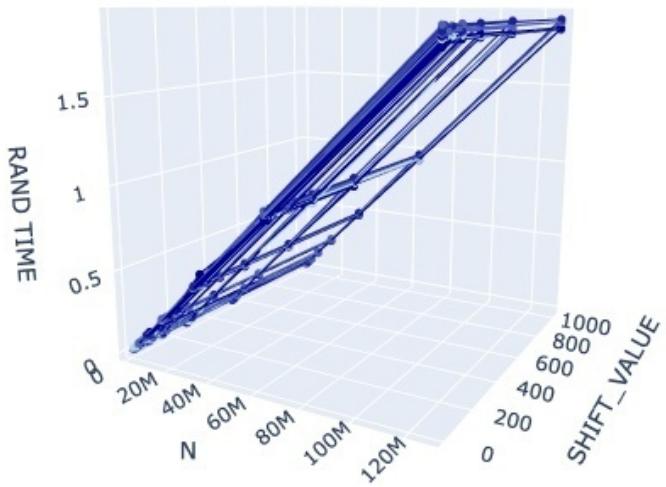
The use of MPI for data decomposition in C projects is a common technique in parallel computing. Data decomposition or data partitioning, is a technique used in parallel computing to divide a large dataset into smaller, manageable chunks that can be processed concurrently by multiple processes or threads. This approach aims to optimize performance and resource utilization by allowing the computation tasks to run in parallel, reducing the overall execution time.

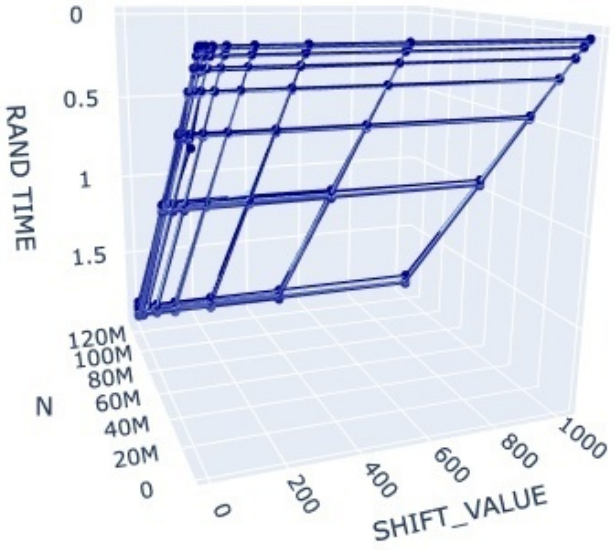
Keywords: Data Decomposition, MPI, C, Parallelism

1. Randomize subroutine

The Randomize function generates random integers for the nums array. First, it seeds the random number generator using srand(42). If the process has rank 0 (i.e., it is the main process), it allocates memory for a temporary array (tempNumberArr) and fills it with random integers between 0 and RMAX - 1. Then, it scatters the temporary array into equal-sized chunks, one for each process, using MPIScatter. Each process stores its chunk in the nums array. The temporary array is freed afterwards. NPROCS and SHIFTVALUE had no effect on the time required to randomize the list.







2. Histogram subroutine

This function calculates the histogram of the nums array. It first initializes a local histogram array (localhist) of size BINS. Then, for each element in the nums array, it increments the corresponding bin in the local histogram. After calculating the local histograms, all processes reduce their local histograms into a single global histogram (hist) using MPIReduce with the MPISUM operation. The Shift value does not effect the run time of the Histogram as seen in Figure 4.

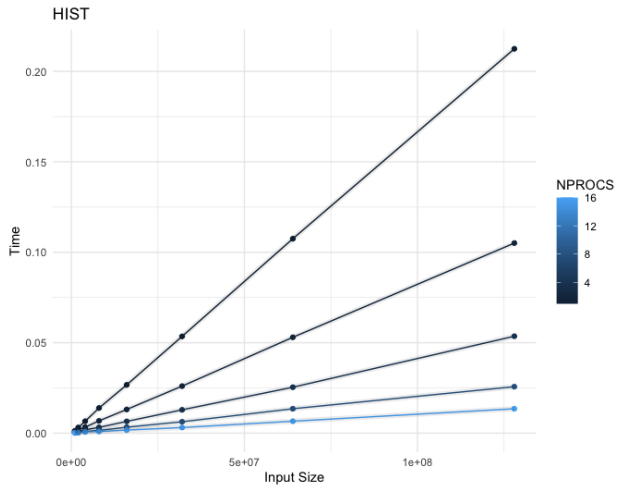


Figure 3: Histogram subroutine time.

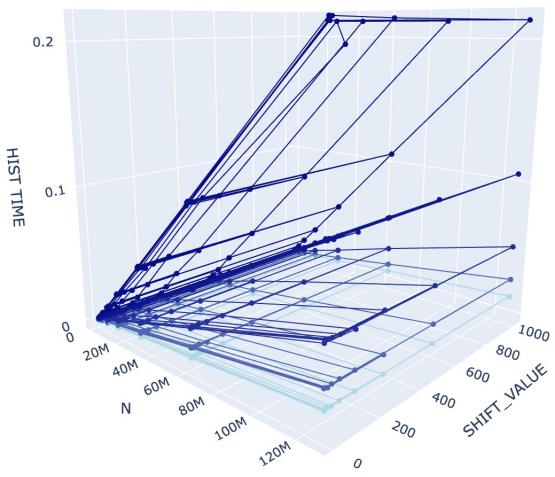


Figure 4: Histogram subroutine time 3D.

3. Shift Left Subroutine

This function shifts the elements of the nums array to the left by a given number of slots (shiftn). Elements that are shifted off the left side rotate around to the end, so no numbers are lost. To achieve this, the function first initializes receive and send buffers. Then, for each slot in the shift, the function sends the leftmost element of the nums array to the left neighbor and receives a new element from the right neighbor using MPISendrecv. It then updates the nums array by moving the elements one position to the left and placing the received element at the end of the array.

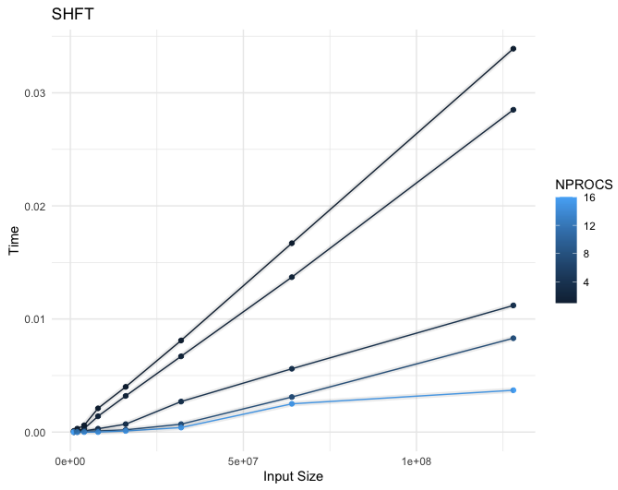


Figure 5: Left shift Subroutine time.

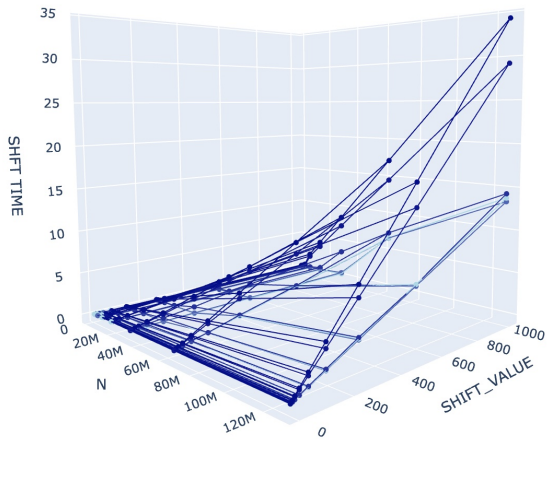


Figure 6: Left shift Subroutine time 3D.

4. MergeSort Subroutine

We used a reverse binary tree structure in order to properly merge between all of the processes. When processes are represented with binary numbers, simple bit shifts can show each process where they are sending and receiving from. When this left bit shift is put in a loop we get an ordered reverse broadcast tree that cuts the amount of processes being used in half every iteration of the loop. When a process receives, it merges the received nums array with its current nums array to produce a new nums array. This process continues until process 0 merges and contains the complete sorted nums array of length global n.

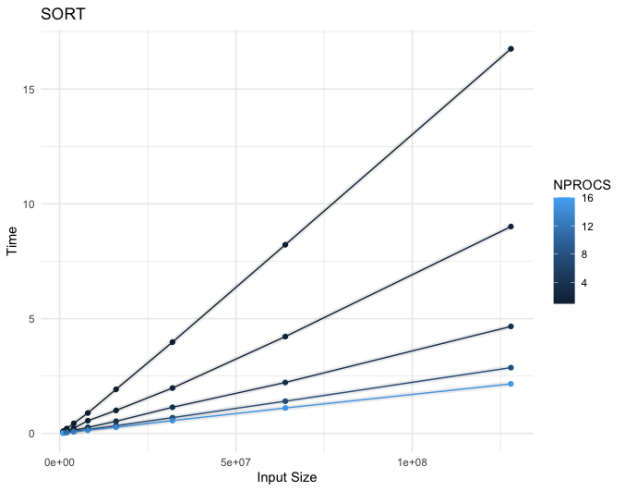


Figure 7: MergeSort Subroutine time.

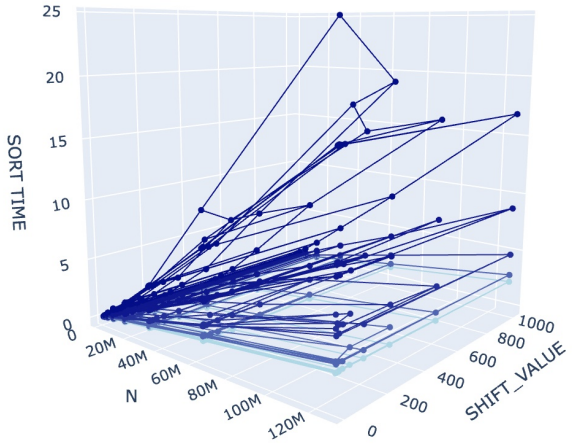
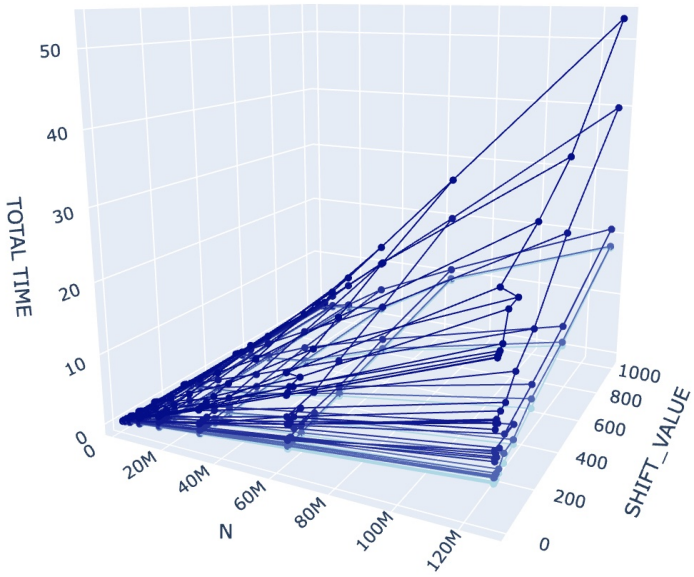


Figure 8: MergeSort Subroutine time 3D.



5. Summary and conclusions

In this paper, we cover MPI for data decomposition in C. We focused on the performance analysis of four subroutines: Randomize, Histogram, Shift Left, and MergeSort. We show various timing improvements and data visualizations. LaTex was used for the general document style. Rstudio and Python were used to perform three dimensional visualizations.

Acknowledgements

Special thanks to Dr. Lam for introducing us to MPI and its benefits. Learning about this powerful tool has been an eye-opening experience that has greatly enhanced our understanding of parallel programming.

References

https://w3.cs.jmu.edu/lam2mo/cs470/cluster.html
https://www.sciencedirect.com/book/9780128046050/an-introduction-to-parallel-programming
https://hpc.llnl.gov/documentation/tutorials/introduction-parallel-computing-tutorial
https://plotly.com/python/3d-scatter-plots/

JMU Computer Science