ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25may1297

A Dynamic MPI-Based Memory-Efficient Framework for Longest Common Subsequence Computation on Massive DNA Sequence

Shubham Kumar Singh[1]; Dharanya T[2]; Aarthi N[3]; Nagadevi S[4]

[1,2,3]Department of Networking and Communications SRM Institute of Science and Technology Chennai, India

Publication Date: 2025/05/22

Abstract—The rapid expansion of genomic datasets, particularly those encompassing entire human chromosomes, presents formidable computational challenges for conventional sequence alignment techniques. Among these, the Longest Common Subsequence (LCS) problem plays a fundamental role in comparative genomics and DNA sequence alignment. However, its inherent time and space complexity—typically quadratic in nature—renders traditional sequential and statically parallelized implementations insufficient for large-scale genomic analysis. In response to this limitation, we propose a dynamic, memory- efficient parallel architecture based on the Message Passing Interface (MPI) framework, specifically optimized for large DNA sequence comparisons. Our approach introduces a master-worker model that dynamically distributes computational workload at runtime. By dividing the input sequences into smaller, manageable segments and assigning these chunks to worker processes on demand, the system ensures effective load balancing across processing units. The architecture leverages a space-optimized dynamic programming technique, where only two rows of the LCS matrix are stored at any time, significantly reducing memory consumption without sacrificing correctness. To evaluate the scalability and performance of our method, we conducted extensive experiments using complete human chromosome datasets across an MPI cluster with eight processes. The results indicate that while the dynamic strategy introduces moderate communication overhead, it consistently outperforms static distribution methods in terms of scalability, adaptability to heterogeneous environments, and memory efficiency. Notably, the proposed solution maintains stability and performance even as sequence sizes grow, making it suitable for deployment in high- performance computing (HPC) environments and cloud-based bioinformatics platforms.

Keywords: *Longest Common Subsequence (LCS), Parallel Computing, Message Passing Interface (MPI), Dynamic Load Balancing, DNA Sequence Alignment, Bioinformatics, High- Performance Computing (HPC), Genome-Scale Processing, Memory Optimization, Master-Worker Architecture.*

How to Cite: Shubham Kumar Singh; Dharanya T; Aarthi N; Nagadevi S. (2025) A Dynamic MPI-Based Memory-Efficient Framework for Longest Common Subsequence Computation on Massive DNA Sequence. *International Journal of Innovative Science and Research Technology*, 10(5), 971-977. https://doi.org/10.38124/ijisrt/25may1297

I. INTRODUCTION

The significant rise in genomic data during the post-genomic era can largely be credited to breakthroughs in next-generation sequencing technologies. These innovations have remarkably accelerated DNA sequencing procedures and substantially reduced expenses. The capacity to sequence an entire human genome in just a few days generates enormous volumes of raw sequencing data, necessitating meticulous processing, comparative assessments, and interpretation. This influx of biological data presents substantial computational hurdles, especially in sequence alignment, which seeks to discover shared characteristics that may unveil functional, structural, or evolutionary relationships among genes or chromosomes.

A considerable challenge in sequence alignment is the Longest Common Subsequence (LCS) dilemma. The LCS is crucial for pinpointing conserved genetic sequences across two different sequences, acting as a key asset in areas such as comparative genomics, mutation evaluation, and evolutionary research. Though the idea is relatively simple, the classic LCS algorithm is resource-intensive, employing a dynamic programming approach with a complexity of $O(m \times n)$, where m and n signify the lengths of the sequences being compared. When this is applied to entire chromosomes comprising hundreds of millions of base pairs, the method necessitates substantial computational resources, often surpassing conventional computing system capabilities regarding memory and processing time.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25may1297

To address these challenges, several parallel computing strategies, especially those leveraging the Message Passing Interface (MPI), have been explored. MPI-centric implementations segment the issue across various processes or computing nodes, thus distributing the workload to boost computational speed. However, many current solutions depend on a static chunk allocation model, where predetermined data segments are assigned to worker processes at the initiation phase. This fixed partitioning can result in load imbalances, particularly when segment computational demands vary, which may impair performance and lead to memory limitations as large sequence segments are processed without adequate optimization.

In our study, we propose a dynamic and memory-efficient MPI- based framework specifically crafted for LCS computation linked to extensive genomic sequences. Our strategy utilizes a master-worker model with dynamic task allocation, enabling the master node to distribute segments of input sequences to available worker nodes based on needs throughout the execution. This adaptable scheduling system improves workload distribution across processing units, enhancing throughput and reducing idle periods. Additionally, to combat memory restrictions, we developed a space-optimized dynamic programming technique that maintains only two rows of the dynamic programming matrix in memory concurrently. This method significantly lowers memory requirements while maintaining the precision of LCS calculations. The primary contributions of this research can be outlined as follows:

- *Dynamic Chunked Parallelization:*
  We introduce a novel MPI- based LCS algorithm that allocates workloads dynamically during execution, which enhances processor effectiveness and scalability across various computing settings.

- *Memory-Efficient Computation:*
  We offer a two-row matrix optimization technique that effectively conserves memory, enabling comprehensive chromosome comparisons on systems with constrained resources.

- *Comprehensive Performance Evaluation:*
  A comparative analysis of static and dynamic parallelization methodologies reveals notable improvements in execution speed, memory efficiency, and adaptability.

The remainder of the document is structured as follows: Section II reviews pertinent research and highlights the limitations of traditional LCS algorithms in large-scale bioinformatics applications. Section III describes the architecture and fundamental design principles of the proposed system. Section IV presents experimental results and performance measures. Section V explores the implications of the findings and proposes directions for future research. Section VI concludes the paper.

II. LITERATURE SURVEY

The Longest Common Subsequence (LCS) problem, foundational in DNA sequence alignment, has been the focus of extensive computational research due to its quadratic time and space complexity. Numerous attempts have been made to accelerate LCS computation using both CPU-based and GPU- based parallel strategies. This section reviews key developments in the field, highlighting existing limitations and motivations for our dynamic, memory-efficient, MPI-based approach. The original LCS challenge was addressed through dynamic programming by Hirschberg, who developed a version that minimizes memory usage with linear space complexity. However, Hirschberg's approach is inherently sequential and does not adapt well to effective parallel execution. Early attempts to implement parallel LCS on shared-memory systems using OpenMP achieved only limited speed enhancements while facing scalability problems. Subsequent studies explored implementations with distributed memory using MPI. For example, Saeed and his team introduced a static MPI parallel LCS algorithm that segments sequences into fixed blocks allocated during initialization. This method is effective for smaller to medium-sized inputs but often faces load imbalance and memory limitations when processing full genome data, due to disparate computational requirements across the divisions. To circumvent CPU limitations, there is an increasing inclination among researchers to leverage GPUs for bioinformatics issues. Liu and Wong presented a CUDA LCS solution that takes advantage of the extensive core architecture of GPUs to accelerate matrix calculations. Their findings indicated significant speed gains compared to CPU methods; however, it required the entire LCS matrix to reside in GPU memory, which limited scalability for longer sequences. Similarly, Liu and colleagues introduced tiled GPU LCS methods aimed at optimizing global memory access. Although these approaches work well, they often confront the limitations of GPU architectures and typically do not generalize effectively across data sets of varying sizes and structures. Additionally, the absence of dynamic task scheduling leads to suboptimal resource utilization in diverse environments. Recent research has explored hybrid approaches. Zhang and his team integrated GPU kernels with MPI to support cross-node communication for large-scale LCS processing. Nonetheless, their method for chunk allocation was rigid, and their system faced load balancing issues due to uneven data distributions. In contrast, dynamic scheduling models, like the one proposed by Bo Peng and his group, demonstrate greater adaptability but have primarily been utilized for sequence alignments via heuristic methods, rather than for accurate LCS computation. Cloud- based solutions such as CloudBurst and Crossbow offer distributed DNA analysis services, prioritizing mapping and alignment over LCS. These platforms do not optimize for memory efficiency or precise sequence comparison and typically depend on Hadoop or Spark, leading to delays that are ill-suited for fundamental dynamic programming tasks.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25may1297

Tackling memory limitations has been a central theme in the research community. Ukkonen's banded alignment and Hirschberg's space optimization represent significant advancements; however, few implementations integrate these techniques with dynamic chunking and distributed scheduling. The framework we propose aims to address this gap by incorporating a two-row dynamic programming method along with on-demand task allocation within MPI, specifically targeting genome-scale data sets. Significant parallel and distributed efforts also include work by Aluru and Jammula, who explored fine-grained parallelism in bioinformatics algorithms while emphasizing cache-aware improvements to reduce memory latency during sequence processing. Although their methods are effective for shorter sequences, they struggle when applied to comparisons involving entire genomes. Bhaskar et al. [12] introduced a method driven by MapReduce for sequence comparison that distributed the LCS problem across multiple nodes within a Hadoop framework. Although their solution provided scalability, it incurred considerable communication costs and was limited by the inherent delays of MapReduce's batch processing. In a more recent investigation, Singh and Kaur [13] examined hybrid tiling techniques across various CPU-GPU clusters. Their adaptive tiling strategy improved the functionality of different devices, but it was deficient in fault tolerance and encountered issues regarding data distribution. Moreover, contemporary implementations driven by FPGAs [14] have demonstrated the benefits of hardware acceleration for LCS computations. Despite their impressive throughput, these methods require specialized development tools and hardware, potentially restricting their use in conventional research environments. Collectively, these studies underscore the trade-offs between performance, memory efficiency, scalability, and ease of deployment. Our proposed dynamic MPI-based framework aims to strike a balance by utilizing minimal memory usage, dynamic load balancing, and applicability at a genomic level without relying on hardware-specific elements.

III. METHODOLOGY

This section describes the architectural framework and core algorithmic strategy behind the proposed Dynamic MPI-Based Chunked Longest Common Subsequence (LCS) Computation System, which is tailored for large-scale DNA sequence comparison. The primary objective is to address the computational inefficiencies and memory constraints that hinder traditional LCS implementations, especially when processing entire human chromosomes. The solution harnesses Message Passing Interface (MPI) to implement a scalable, memory- efficient, and dynamically load-balanced parallel processing model.

A. System Architecture: Dynamic Master-Worker Model

At the core of the system lies a dynamic parallel architecture based on the master-worker paradigm implemented through MPI. Unlike traditional static parallelization models where tasks are assigned at the start of execution, this architecture enables real-time task distribution. The master process assumes the role of orchestrator, initiating sequence loading, dividing the data into computationally manageable segments, and coordinating communication and task assignment with worker processes. Each worker process operates independently, requesting work when available, computing partial results, and returning them asynchronously. This dynamic work assignment model ensures that faster or less burdened processors receive more work over time, thereby improving load balancing and minimizing idle periods. Such flexibility is critical in high-performance computing environments where node performance may vary due to system heterogeneity or transient loads. Moreover, this model avoids memory overload by distributing tasks in small, discrete units rather than requiring the full input sequences to reside in memory on each node. Initial sequence data or metadata is broadcast once, while actual chunk indices are managed centrally by the master during execution. The result is a system that scales effectively across a wide range of processor counts and adapts to the runtime conditions of the computing environment.

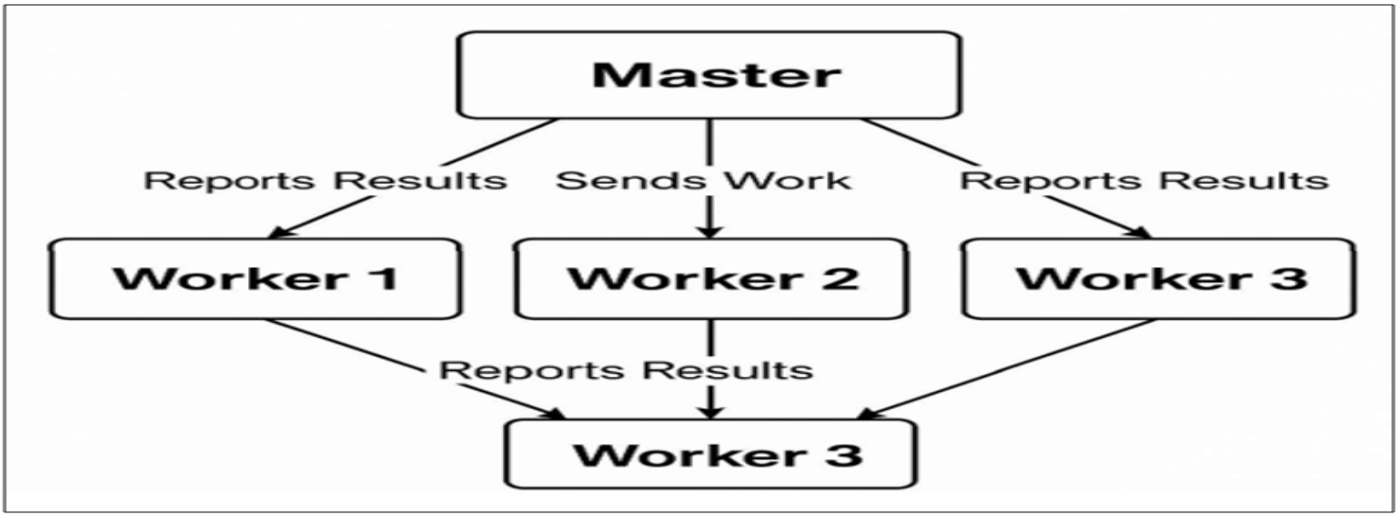

Fig 1 Master process distribution

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25may1297

B. Chunked and Space-Efficient LCS Computation

To address the quadratic space complexity inherent in traditional LCS algorithms, the proposed framework implements a chunk- based decomposition strategy coupled with a memory-efficient dynamic programming approach. Both input sequences are divided into smaller, fixed-length segments—typically ranging from 1000 to 5000 base pairs—forming a grid of chunk pairs, each of which defines a subproblem in the LCS computation space. By processing these chunk pairs individually, the memory footprint of each computation is reduced from O (m × n), where m and n are the full sequence lengths, to O (c × c), where c is the chunk size. Each worker computes the LCS of its assigned chunk pair using a row-wise optimization of the dynamic programming algorithm. Instead of constructing the entire two-dimensional LCS matrix, which becomes infeasible for large input sizes, only two rows—the current and the previous—are stored and updated iteratively during computation. This optimization reduces the space complexity per task from $O(c^2)$ to O(2c), allowing the system to process large data volumes using only modest hardware resources.

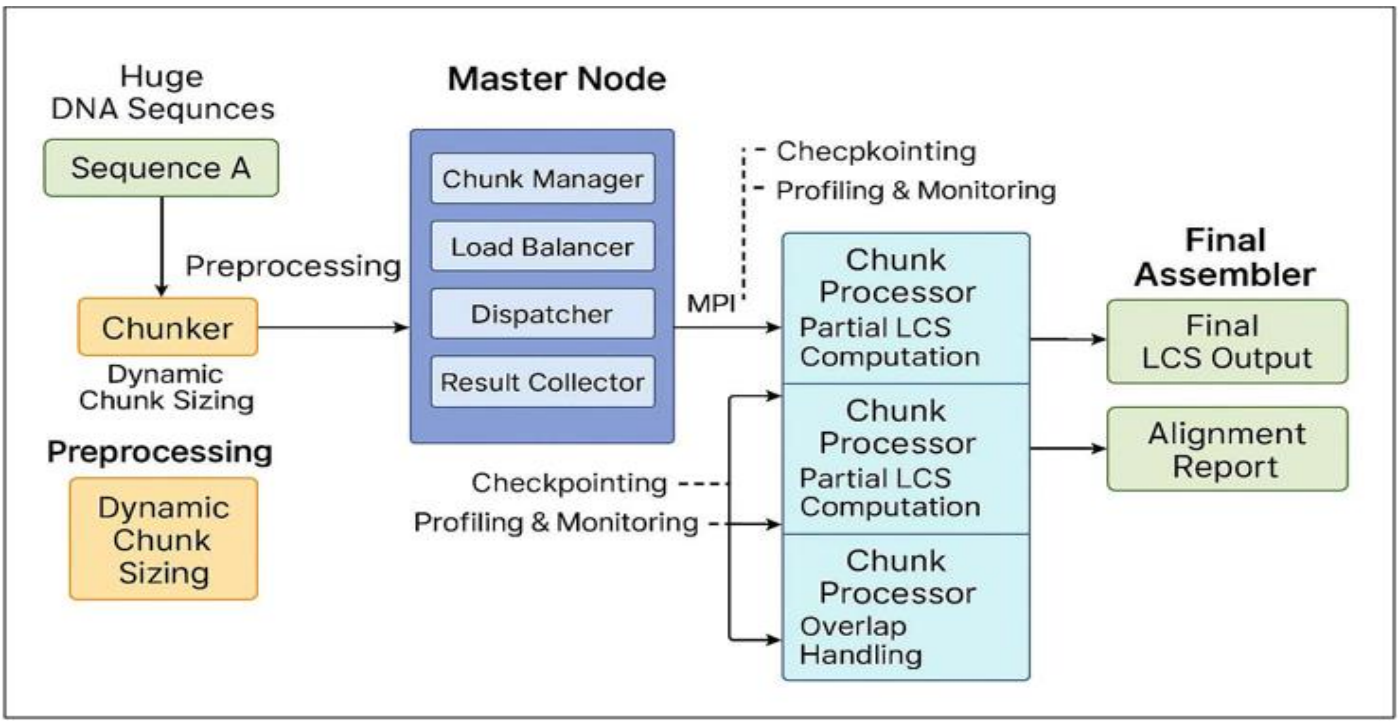

Fig 2 Dynamic MPI-Based Chunked LCS Computation

This diagram outlines a parallel system designed to process large DNA sequences and find their Longest Common Subsequence (LCS). The process starts with a preprocessing step where the input sequence is divided into smaller parts using a method called dynamic chunk sizing. These chunks are then managed by a central Master Node, which assigns work to different processors. Each processor handles part of the LCS computation, with one specifically managing the overlapping regions between chunks. Once all partial results are ready, a final assembly step combines them into the complete LCS result and generates an alignment report. The system also includes features for checkpointing, monitoring, and performance tracking to ensure smooth operation during large-scale computations.

Communication between the master and workers relies on MPI's point-to-point messaging primitives, such as MPI_Send and MPI_Recv, enabling a responsive and lightweight control mechanism. Workers notify the master when ready for a new task, and the master dispatches the next available chunk pair. This decentralized execution model allows the system to scale horizontally: adding more worker processes increases parallel throughput, while the small-memory footprint of each task ensures that data size remains a non-limiting factor. Although the current implementation does not include fault-tolerant mechanisms, the architecture is designed with extensibility in mind.

Versus Chromosome 22), the static strategy encountered significant challenges. In several instances, attempts to compute full LCS matrices led to segmentation faults and memory exhaustion, a consequence of the global allocation of full-size dynamic programming tables. This limitation rendered the static approach impractical for full-genome comparisons on standard hardware configurations. It successfully processed large chromosome pairs without runtime failures, completing alignments that were infeasible under the static model. For example, comparisons such as Chromosome 1 versus Chromosome 22 and Chromosome 2 versus Chromosome 22 completed in 91 and 122 seconds, respectively. While these execution times were longer due to the communication overhead introduced by dynamic task scheduling, the approach provided a reliable and scalable solution suitable for large-scale genomic data analysis.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25may1297

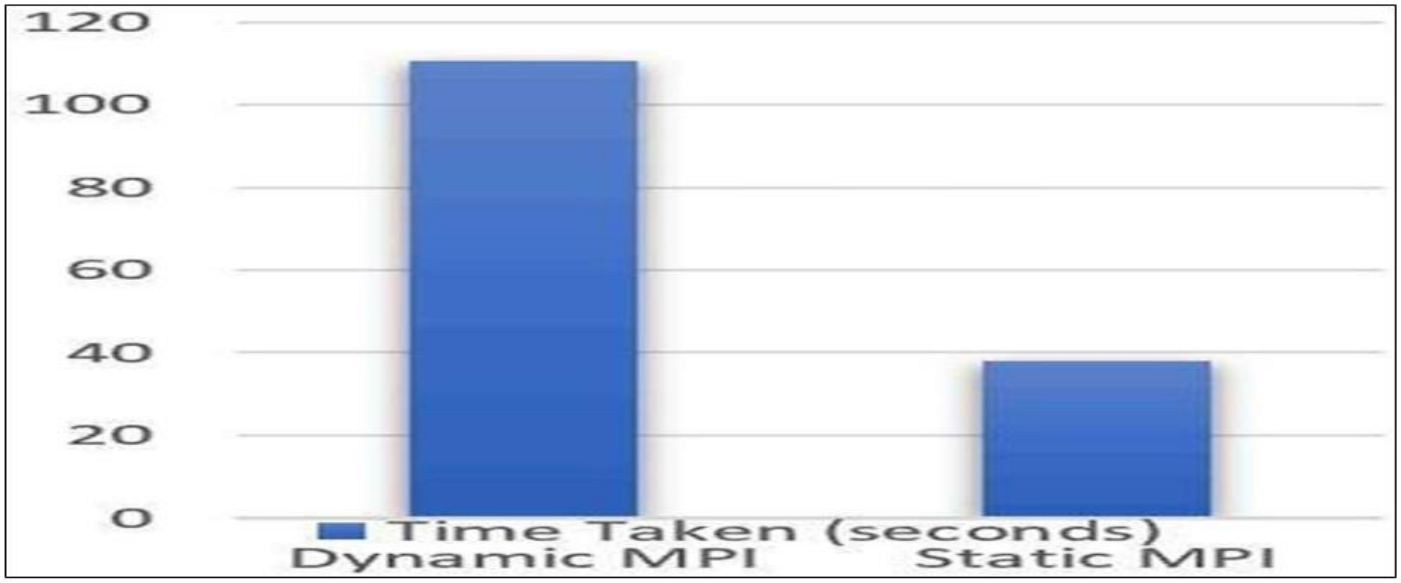

Fig 3 Time taken by Dynamic MPI & Static MPI

IV. RESULTS & DISCUSSION

To assess the practical viability and performance of the proposed dynamic MPI-based LCS computation framework, a series of experiments were conducted using real-world genomic data. Specifically, full-length DNA sequences corresponding to various human chromosomes were obtained from the NCBI GenBank repository. The implementation was developed in C, utilizing the OpenMPI library to support distributed computation across multiple processes.

```
[Worker 4] Chunk 242172000-242174000 | LCS: 1255 | Time: 0.00508653 sec
[Worker 6] Chunk 242166000-242168000 | LCS: 1280 | Time: 0.00934593 sec
[Worker 7] Chunk 242170000-242172000 | LCS: 1249 | Time: 0.00883735 sec
[Worker 1] Chunk 242176000-242178000 | LCS: 1269 | Time: 0.00686694 sec
[Worker 5] Chunk 242174000-242176000 | LCS: 1258 | Time: 0.00734871 sec
[Worker 3] Chunk 242180000-242182000 | LCS: 1262 | Time: 0.00592155 sec
[Worker 4] Chunk 242182000-242184000 | LCS: 1032 | Time: 0.00676056 sec
[Worker 2] Chunk 242178000-242180000 | LCS: 1270 | Time: 0.00904418 sec
[Worker 6] Chunk 242184000-242186000 | LCS: 0 | Time: 0.00653936 sec
[Worker 5] Chunk 242190000-242192000 | LCS: 0 | Time: 0.0064619 sec
[Worker 3] Chunk 242192000-242194000 | LCS: 0 | Time: 0.00882311 sec
[Worker 7] Chunk 242186000-242188000 | LCS: 0 | Time: 0.0116155 sec
[Rank 3] Total Time: 122.763 seconds
[Rank 2] Total Time: 122.763 seconds
[Worker 1] Chunk 242188000-242190000 | LCS: 0 | Time: 0.0114878 sec
[Rank 1] Total Time: 122.763 seconds
[Rank 0] Total Time: 122.763 seconds
[Rank 5] Total Time: 122.763 seconds
[Rank 6] Total Time: 122.763 seconds
[Rank 4] Total Time: 122.763 seconds
[Rank 7] Total Time: 122.763 seconds
swift@SWIFT:~/DNA/Scripts$
```

Fig 4 Execution log of the Dynamic MPI-based Chunked LCS

All tests were executed on a shared-memory system equipped with eight processing cores and 16 GB of RAM. The primary objective of the evaluation was to compare the behaviour of the dynamic master-worker model against a baseline static MPI implementation under varying input sizes and sequence combinations. In both cases, the sequences were partitioned into fixed-size chunks of 1000 base pairs to facilitate controlled memory usage. The static approach involved predefined task allocation at the start of execution, whereas the dynamic version employed runtime chunk scheduling based on process availability, as described earlier. The results revealed a distinct contrast between the two paradigms. The static model exhibited favourable execution times when aligning relatively short sequences, completing certain comparisons such as Chromosome 1 versus Chromosome 2 in approximately three seconds. However, as the input size increased, particularly in scenarios involving large chromosomes (e.g., Chromosome 1

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25may1297

A key observation from the experiments was the improved load balancing achieved through dynamic scheduling. The master process effectively distributed work based on worker availability, allowing faster or underutilized processes to handle a greater portion of the workload. This led to more uniform resource utilization and reduced idle time. Furthermore, the space-optimized implementation at the worker level, which retained only two rows of the LCS matrix at any given time, ensured that the system operated comfortably within memory limits—even for full-length chromosome comparisons.

V. CONCLUSION & FUTURE WORK

This study introduces a dynamic, MPI-based framework tailored for efficient computation of the Longest Common Subsequence (LCS) across large-scale genomic sequences. By adopting a chunked alignment strategy within a master-worker model, the system dynamically assigns computational tasks to available processing units during runtime. This design enables the framework to overcome the scalability and memory limitations that typically hinder traditional or statically parallelized LCS implementations.

Experimental evaluations on full-length human chromosomes validate the robustness and adaptability of the proposed approach. Unlike static task allocation, which often results in

memory exhaustion or inefficient resource use, the dynamic framework demonstrated reliable performance across a range of sequence sizes. The system was able to align entire chromosomes without exceeding memory limits, while also achieving consistent load balancing across multiple cores. These characteristics make the method particularly well-suited for environments with limited memory resources, such as commodity clusters or mid-tier cloud computing instances. Although the dynamic scheduling mechanism introduces a modest increase in communication overhead, particularly for smaller input datasets, this trade-off is offset by significant gains in scalability and memory efficiency. As a result, the framework is a strong candidate for deployment in high-performance computing contexts that require large-scale DNA sequence comparisons, including genomic variation analysis, phylogenetics, and personalized medicine pipelines.

Looking ahead, several extensions are envisioned to enhance the system's capabilities. One promising direction involves implementing adaptive chunk sizing based on the complexity or structure of the input sequences, which could further optimize memory use and execution time. Additionally, a hybrid parallelization scheme combining MPI with thread-level parallelism via OpenMP could leverage multi-core architectures more effectively. Deployment on distributed cloud infrastructures represents another avenue for future exploration, offering opportunities for large-scale comparative genomics across multiple datasets simultaneously. Finally, integration with biological annotation tools and mutation detection frameworks could transform this computational engine into a full-fledged bioinformatics analysis platform capable of real- time genome analysis. Through its combination of scalability, resilience, and efficient resource utilization, the proposed dynamic LCS framework lays a strong foundation for advanced genomic data processing in both research and clinical settings.

The template will number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use “Ref. [3]” or “reference [3]” except at the beginning of a sentence: “Reference [3] was the first ...”

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors’ names; do not use “et al.”. Papers that have not been published, even if they have been submitted for publication, should be cited as “unpublished” [4]. Papers that have been accepted for publication should be cited as “in press” [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols. For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

REFERENCES

[1]. G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz- Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955. (references)
[2]. J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol.
[3]. Oxford: Clarendon, 1892, pp.68–73.
[4]. I. S. Jacobs and C. P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.
[5]. K. Elissa, “Title of paper if known,” unpublished.
[6]. R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand. Abbrev., in press.
[7]. Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].
[8]. M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.
[9]. K. Eves and J. Valasek, “ Adaptive control for singularly perturbed systems examples, ” Code Ocean, Aug. 2023. [Online]. Available: https://codeocean.com/capsule/4989235/tree
[10]. D. P. Kingma and M. Welling, “Auto-encoding variational Bayes,” 2013, arXiv:1312.6114. [Online]. Available: https://arxiv.org/abs/1312.6114

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25may1297

[11]. S. Liu, “Wi-Fi Energy Detection Testbed (12MTC),” 2023, gitHub repository. [Online]. Available: https://github.com/liustone99/Wi-Fi-
[12]. Energy-Detection-Testbed-12MTC
[13]. “Treatment episode data set: discharges (TEDS-D): concatenated, 2006 to 2009.” U.S. Department of Health and Human Services, Substance Abuse and Mental Health Services Administration, Office of Applied Studies, August, 2013, DOI:10.3886 /ICPSR30122.v2