Investigating the Scalability of Parallel DTW Distance Computations of Stock Price Time Series

Daniela P. Rigoli[1], **Cesar A. F. De Rose**[1]

[1]PPGCC - Escola Politécnica - PUCRS
Caixa Postal 1429 – 90.619-900 – Porto Alegre – RS – Brazil

daniela.rigoli@edu.pucrs.br, cesar.derose@pucrs.br

Abstract. *Time-series processing and analysis play a fundamental role in several scientific and industrial domains, particularly in identifying patterns and relationships in temporal data. In the financial domain, similarity analysis between assets can reveal correlated behavior and support decision-making. However, computing Dynamic Time Warping (DTW) distances is computationally expensive when applied to large collections of time series. This work investigates the scalability of parallel implementations based on OpenMP, MPI, and the hybrid MPI+OpenMP model to accelerate DTW distance computation on high-performance computing systems. Different communication strategies, workload distribution schemes, and task granularities are evaluated through experiments conducted on a multiprocessor and a multicore cluster, considering speedup and parallel efficiency as performance metrics. Results demonstrate that in general parallel computing substantially increases the practicality of DTW-based similarity analysis for large financial datasets, with the hybrid MPI+OpenMP implementation obtaining over 90% efficiency. Furthermore, the optimization strategies and scalability insights presented in this work can be extended to other domains that rely on large-scale time-series processing, such as climate modeling, bioinformatics, and sensor networks.*

1. Introduction

Time-series similarity analysis is relevant across several domains, including finance, neuroscience, and medicine, with applications ranging from renewable energy systems and spike-train analysis [Radovanović et al. 2020] to electrocardiographic (ECG) signal interpretation [Miniakhmetov et al. 2015]. In many real-world scenarios, the goal is not only to analyze individual signals but also to compare large collections of time series to identify similar temporal behaviors. However, measuring similarity between temporal sequences is challenging due to temporal misalignments, noise, and variations in scale or phase.

Several approaches have been proposed to measure similarity between time series. Traditional pointwise metrics, such as Euclidean distance, are computationally efficient but often fail to capture temporal distortions between sequences. Elastic similarity measures address this limitation by allowing non-linear alignments between time series. Among these methods, Dynamic Time Warping (DTW) has become one of the most widely adopted techniques because it can effectively identify similar patterns even when they occur at different time scales or positions [de Jesus et al. 2025]. For this reason, DTW is adopted in this work as the core similarity measure.

Despite its advantages, DTW presents a significant limitation: its computational cost. The classical DTW algorithm has quadratic complexity with respect to the lengths of the time series, and when applied to large datasets requiring pairwise comparisons, the overall computational demand becomes substantial. This limitation becomes a barrier in scenarios where results are required within practical time constraints, such as financial analysis workflows.

At the same time, modern computing environments increasingly rely on parallel architectures, including multicore processors and clusters of interconnected machines. Parallel computing has therefore become a natural strategy for addressing the computational demands of DTW-based similarity analysis. This creates opportunities for developing efficient parallel approaches for similarity analysis of large time-series datasets.

This work investigates multiple parallelization strategies for DTW-based financial time-series similarity analysis. Specifically, three computational models are considered: shared-memory parallelism using OpenMP, distributed-memory parallelism using MPI, and a hybrid MPI+OpenMP approach that combines inter-node and intra-node parallelism. This unified perspective enables a comparison of parallel performance, communication overhead, and resource utilization across different high-performance computing architectures.

The main contributions of this work are:

- design and implementation of pure MPI and hybrid MPI+OpenMP master–worker models tailored for large-scale DTW distance matrix computation;
- optimization of workload distribution, task granularities, and communication strategies to minimize overhead in distributed-memory environments;
- a comprehensive comparative performance analysis across shared-memory, distributed-memory, and hybrid execution models using strong scaling on two distinct HPC architectures using a dataset of 1,600 financial time series, demonstrating over 90% parallel efficiency for the hybrid model;
- insights and optimization guidelines for extending these parallel execution patterns to other time-series-intensive domains with similar computational characteristics.

2. Background

Euclidean distance compares observations at identical timestamps and is therefore sensitive to temporal misalignments. Correlation-based measures evaluate the strength of linear or monotonic relationships but do not capture local temporal distortions. DTW addresses these limitations by allowing non-linear alignments between sequences.

DTW is a widely used algorithm for measuring similarity between two time series that may vary in speed or exhibit temporal misalignments. Originally introduced for speech processing [Vintsyuk 1968, Sakoe and Chiba 1978], DTW has since found applications in fields such as bioinformatics [Saklani et al. 2023], neuroscience [Radovanović et al. 2020], medicine [Miniakhmetov et al. 2015, Ninama et al. 2025], and finance [Zhang et al. 2021, Chan et al. 2026, Sigaki et al. 2019, Winarti et al. 2024]. In the financial domain, DTW can be especially useful for identifying assets with similar temporal price patterns, even when those patterns are not aligned in time.

Given two time series

$$X = (x_1, \ldots, x_n)$$

and

$$Y = (y_1, \ldots, y_m),$$

DTW computes the cumulative alignment cost through dynamic programming. Each element of the accumulated cost matrix is defined as

$$D(i,j) = |x_i - y_j| + \min\left\{D(i-1,j), D(i,j-1), D(i-1,j-1)\right\},$$

where the three predecessor cells represent insertion, deletion, and matching operations. The DTW distance corresponds to the value stored at $D(n,m)$ after the optimal warping path has been computed.

While Euclidean distance compares samples only at identical timestamps, DTW allows one point of a sequence to be aligned with multiple points of the other sequence, producing a more meaningful comparison when similar patterns occur at different times.

The flexibility provided by DTW comes at a relatively high computational cost. The classical algorithm requires $\mathcal{O}(nm)$ time and memory when comparing two sequences of lengths n and m. When computing all pairwise distances among N time series with average length L, the overall complexity becomes approximately

$$\mathcal{O}(N^2 L^2),$$

which rapidly becomes the dominant cost for large datasets.

To reduce unrealistic alignments and improve performance, DTW is commonly combined with global path constraints, such as the Sakoe–Chiba band, which restricts the maximum deviation from the main diagonal of the accumulated cost matrix. These constraints reduce both execution time and excessive warping while preserving alignment quality.

Because pairwise DTW computations are independent, they naturally expose task-level parallelism. This characteristic makes DTW particularly suitable for shared-memory, distributed-memory, and hybrid parallel implementations, which are investigated in this work.

3. Implementation

3.1. OpenMP Baseline and Optimization Load Balance

The proposed implementation builds upon the OpenMP-based DTW distance matrix algorithm introduced by Meert et al. [Meert et al. 2020], which was adopted as the computational baseline in this work. The original implementation computes the pairwise DTW distance matrix by exploiting its symmetry, evaluating only the upper triangular portion of the matrix. Each matrix element corresponds to an independent DTW computation between two time series and can therefore be processed concurrently.

For shared-memory execution, the original OpenMP implementation was adapted to use dynamic scheduling instead of the guided scheduling strategy proposed by Meert et al. [Meert et al. 2020]. The objective was to evaluate the impact of scheduling policies on load balancing and overall parallel performance.

3.2. MPI

To support distributed-memory execution, three MPI implementations were developed. All adopts a *master–worker* execution model, in which a single master process is responsible for scheduling tasks and collecting results, while the remaining processes execute DTW computations.

Version 1 employs a communication protocol based on five messages for each task exchange. Specifically, for every pair of time series, the master sends five messages: the indices of the pair, the lengths of both series, and the corresponding data arrays. Because the batch size is fixed at one pair per message, this communication pattern is repeated for every DTW computation. As a result, communication overhead becomes significant, especially when processing a large number of small tasks. This implementation therefore serves as an MPI proof of concept for evaluating subsequent optimizations.

Version 2 reduces the protocol to two messages per task, reducing communication overhead. As in the previous implementation, each task corresponds to the computation of the DTW distance for a single pair of time series (r, c), making the parallelization axis the set of all pairwise comparisons. Consequently, the grain size also remains unchanged, with one pairwise comparison assigned to each task.

The communication protocol was therefore redesigned to only the raw time-series data is transmitted. The worker determines the message sizes dynamically using `MPI_Probe` and `MPI_Get_count`, eliminating the need to explicitly communicate metadata.

`MPI_Probe` avoids sending explicit message sizes, simplifying the communication protocol while allowing variable-length time series to be received safely. Since task identifiers are no longer transmitted explicitly, the master maintains a local mapping between workers and their most recently assigned task.

Version 3 further optimizes communication by using MPI derived datatypes combined with batched task transfers, reducing both the number of messages and the synchronization overhead.

The task granularity is controlled by the parameter `BATCH_SIZE`, which specifies the number of DTW comparisons grouped into a single work assignment. Increasing the grain size reduces the communication frequency because several DTW computations are executed before a worker communicates with the master again.

In this strategy, the master process generates all pairwise combinations of time series and dynamically distributes batches of tasks to available workers. Each batch contains multiple pairs of time series, reducing the number of communication operations required during execution.

The worker receives the batch, computes all DTW distances for the assigned pairs, and returns the results as a single message. After completing a batch, workers request new tasks, allowing the system to maintain load balancing even when DTW execution times vary according to time series lengths.

3.3. Hybrid

A hybrid MPI+OpenMP version was also implemented. In this configuration, one MPI process is assigned to each compute node, while multiple OpenMP threads are used within each process to exploit intra-node parallelism. This approach combines distributed-memory execution across multiple nodes with efficient utilization of shared-memory resources within each node.

This design extends the batch-based MPI implementation presented in the previous section by introducing a second level of parallelism inside each worker process. MPI is responsible for distributing work among processes running on different nodes, while OpenMP exploits the shared-memory parallelism available within each node.

From a modeling perspective, the application is organized into two levels of execution. At the inter-node level, a master–worker model is employed, where the master distributes batches of pairwise DTW comparisons to the MPI workers. At the intra-node level, each worker uses OpenMP threads to process the DTW comparisons contained in its assigned batch concurrently. The pairwise comparisons within a batch are parallelized using an OpenMP `for` construct with `schedule(dynamic)`, allowing threads to dynamically request new comparisons as they complete previous ones. This strategy provides automatic load balancing inside each node and helps mitigate workload irregularities caused by variations in DTW execution time among different sequence pairs.

The master process remains essentially identical to the batch-based MPI implementation described in the previous section. The main difference is that each assigned batch is processed by a multithreaded worker rather than by a single MPI process.

The task granularity at the MPI level is defined by the configurable parameter `BATCH_SIZE`, where each MPI assignment contains multiple pairwise comparisons. At the OpenMP level, the grain size corresponds to a single DTW comparison, since each thread computes one pairwise comparison at a time.

Compared with the pure MPI implementation, the hybrid approach substantially reduces the number of communicating entities. Instead of launching multiple MPI processes per node, only a small number of worker processes participate in the communication. Consequently, fewer messages are exchanged with the master process, reducing communication overhead and network contention.

An additional advantage of the hybrid approach is memory efficiency. OpenMP threads share the same address space and therefore access a common copy of the received batch data. In contrast, a pure MPI implementation requires each MPI process to maintain its own private memory space, potentially increasing memory consumption.

4. Experiments

The proposed OpenMP, MPI, and hybrid MPI+OpenMP implementations were evaluated using pairwise Dynamic Time Warping (DTW) distance computations on financial time series. The DTW calculations used squared Euclidean distance as the local cost function, following the default configuration of the DTAIDistance implementation. No global warping constraint was applied, allowing the alignment path to explore the full cost matrix. The parallel implementations were evaluated in terms of execution time, speedup, parallel efficiency, and scalability.

The experiments were conducted on the *Cerrado* and *Pantanal* clusters of the High Performance Laboratory (LAD-IDEIA/PUCRS). The *Cerrado* cluster comprises 14 nodes equipped with dual Intel Xeon E5-2620 processors (12 physical cores and 24 hardware threads per node), 24 GB of RAM, Ubuntu 20.04.6 LTS, and OpenMPI 1.10.7, interconnected through Gigabit Ethernet and InfiniBand QDR (32 Gbit/s). The *Pantanal* is Dell EMC PowerEdge R740 server with two sockets, each having a Xeon Gold 5118 processor with 12 2.30 GHz physical cores with 12 MB of L2 cache per core and 16.5 MB of L3 cache per processor, and 272 GB of main memory. In total, there are 24 physical cores that are able to run 48 threads due to Hyper-threading. Operating system is Ubuntu 20.04 with gcc-9.4.0 and OpenMPI-4.1.1. These platforms enabled the evaluation of the OpenMP, MPI, and hybrid implementations under both shared- and distributed-memory environments.

The evaluation focused on investigating the performance of the proposed parallel implementations. A strong-scaling methodology was adopted, in which the workload remained fixed while the number of processing units increased. Performance was assessed using execution time, speedup, and parallel efficiency, computed relative to the sequential implementation.

GPU-based implementations were not considered in the experimental evaluation. The evaluated dataset is smaller than 1 GB and contains time series with lengths below 5,000 samples. Previous studies indicate that GPU acceleration is most advantageous for substantially longer sequences, whereas for shorter time series the overhead of CPU–GPU data transfers may limit the achievable gains [Yang et al. 2022, Atanasovska et al. 2024]. Since successful GPU-based DTW implementations are commonly reported for much larger datasets and longer sequences [Huang et al. 2013, Zhu et al. 2021], this work focuses on exploiting multicore CPU architectures through OpenMP and MPI.

The sequential baseline used for speedup calculations corresponds to the original single-threaded DTW implementation provided by the DTAIDistance library. The same DTW configuration adopted in all parallel experiments was used in the baseline execution, including the default Sakoe–Chiba settings. Therefore, speedup values were computed by comparing each parallel execution against an equivalent sequential execution performing exactly the same set of pairwise DTW computations.

The dataset was collected from Yahoo Finance and contains historical daily closing prices of assets from the IBOV, S&P 500, and NASDAQ indices. After preprocessing to remove incomplete series, the final dataset comprised 1,600 financial assets. Each asset contains approximately ten years of historical data. All time series were normalized using Min–Max scaling before computing DTW distances. The resulting workload consists of 1,279,200 pairwise DTW comparisons.

For the dataset used in this research, the average execution time of an individual pairwise distance computation on the *Cerrado* cluster was 92.29 ms, with a standard deviation of 2.81 ms. The dataset consisted of 1,600 financial assets, each represented by a time series containing 2,000 daily observations.

Among the tested configurations, the results section reports the configurations that achieved the highest speedup while maintaining high parallel efficiency, allowing a direct comparison with the pure OpenMP and MPI implementations. To fa-

cilitate reproducibility, the source code, scripts for executing the models and acquiring the dataset, and the documentation used in this work are publicly available at:https://github.com/DeltaRig/Master .

4.1. Statistical Validation

To ensure the reliability of the measurements, each experimental configuration was executed at least five times. The reported execution time corresponds to the arithmetic mean of the measured execution times. In addition, the standard deviation, coefficient of variation (CV), and margin of error of the 95% confidence interval were computed to quantify the variability of the results.

Overall, the proposed implementations exhibited low execution-time variability. Most OpenMP, MPI, and hybrid configurations presented coefficients of variation below 2%, with several experiments remaining below 1%, indicating stable execution times. Likewise, the corresponding 95% confidence interval margins were small compared with the average execution times, demonstrating good repeatability.

Higher variability was observed only in a few isolated MPI configurations executed with a small number of processes, where external system effects had a greater influence on execution time. Since these configurations do not correspond to the highest-performing results and the observed speedup and efficiency trends remained consistent across repeated executions, these variations do not affect the conclusions of this study.

4.2. Performance Overview

Table 1 summarizes the best-performing configurations for each parallel strategy evaluated in this work. The table provides an overview of the corresponding execution times, speedups, and parallel efficiencies and serves as a reference for the detailed analyses presented in the following sections.

Table 1. Representative configurations selected for the performance analysis.

Approach	Configuration	Speedup	Efficiency
OpenMP	24 threads	20.22	84.3%
MPI	12 nodes / 144 processes	93.41	64.9%
MPI + hyper-threading	12 nodes / 288 processes	127.40	44.2%
Hybrid MPI+OpenMP	10 nodes / 10 MPI + 12 threads	112.14	93.5%

4.3. OpenMP Performance

The OpenMP implementation was evaluated on a single compute node using 6, 12, 24, and 48 execution threads. The last configuration exploits hardware-supported simultaneous multithreading (hyper-threading), allowing its impact on runtime, speedup, and parallel efficiency to be evaluated. In addition to the proposed implementation using dynamic scheduling, the original implementation based on OpenMP guided scheduling was also evaluated to compare the effects of the two scheduling strategies.

Figure 1 shows that dynamic scheduling consistently outperforms the original guided implementation across all evaluated thread counts. With six threads, the execution time decreased from 22,771.87 seconds to 12,886.58 seconds, increasing the speedup

from 3.18 to 5.62 and improving parallel efficiency from 53.04% to 93.74%. Similar improvements were observed at higher thread counts.

At 12 threads, the proposed implementation achieved a speedup of 10.88, compared with 5.94 for the baseline, representing an increase in parallel efficiency from 49.5% to 90.6%. With 24 threads, the speedup increased from 11.10 to 20.22, while parallel efficiency improved from 46.3% to 84.2%. The largest improvement was observed with 48 threads, where the execution time was reduced from 4,515.56 seconds to 3,311.87 seconds, increasing the speedup from 16.05 to 21.88.

4.4. MPI Performance

The MPI implementation was evaluated using multiple cluster nodes. Experiments were performed with configurations ranging from one to twelve nodes, considering both executions restricted to physical cores and executions using hyper-threading. Additional experiments compared the three MPI communication strategies implemented in this work (MPI with five messages per task, MPI with two messages per task, and MPI with an adaptive batch size) to evaluate the impact of communication optimizations on execution time, speedup, and parallel efficiency. Different task batch sizes were also evaluated to analyze the trade-off between computation and communication overhead.

As the number of nodes increased from two to twelve, the execution time decreased from 1,696.77 s to 319.21 s, corresponding to an increase in speedup from 17.57 to 93.41. Parallel efficiency remained above 64% for all evaluated configurations, ranging from 73.22% with 24 MPI processes to 64.87% with 144 MPI processes. Although efficiency gradually decreased as the number of processes increased, the reduction in execution time followed the expected strong-scaling trend. With twelve nodes (144 MPI processes), the proposed implementation reduced the execution time by approximately a factor of 93 compared with the sequential execution.

With hyper-threading enabled, the execution time decreased from 1,028.26 s using 48 MPI processes (2 nodes) to 234.06 s using 288 MPI processes (12 nodes), achieving a maximum speedup of 127.40. Without hyper-threading, the best result was obtained with 144 MPI processes (12 nodes), reducing the execution time to 319.21 s and achieving a speedup of 93.41.

4.5. MPI Communication Optimization

Figure 2 compares the three MPI implementations evaluated on the Cerrado cluster. Initially, the optimization that reduced the communication protocol from five messages to two messages produced virtually identical results. With 24 processes, the execution time decreased only from 1,616.88 seconds to 1,616.24 seconds, corresponding to speedups of 18.44 and 18.45, respectively. Likewise, with 48 processes distributed across four nodes, the execution time remained practically unchanged (797.63 seconds versus 797.60 seconds), resulting in a speedup of approximately 37.4 and a parallel efficiency close to 78%. Even in the hyper-threading configuration, with 48 MPI processes executed on two nodes using 24 hardware threads per node, the difference was negligible, with execution times of 1,372.78 seconds and 1,371.35 seconds.

A different behavior was observed in the third implementation, which employs MPI derived datatypes and transmits multiple DTW tasks in a single communication oper-

ation. Using 48 MPI processes on two nodes, the execution time decreased to 1,028.26 s, increasing the speedup from 21.74 to 29.00 and improving parallel efficiency from approximately 45% to 60%. This represents a reduction of approximately 25% in execution time compared with the previous versions.

4.6. Hybrid MPI+OpenMP

Figure 4 compares the hybrid implementation with the pure MPI implementation. Since both implementations were evaluated on the same cluster, this comparison isolates the impact of the programming model and communication strategy from hardware-related effects. The hybrid model combines MPI across compute nodes with OpenMP within each node, reducing the number of MPI processes while exploiting the shared-memory resources available on each node.

The hybrid implementation, using 12 OpenMP threads per node and a batch size of 24 DTW comparisons, achieved a speedup of 112.14 while maintaining a parallel efficiency of 93.45%. Configurations using 72 and 96 processing elements also maintained parallel efficiencies above 90%.

This behavior can be attributed to the reduction in communication overhead. Only one MPI process per node exchanges messages with the master process, while the DTW computations are executed by OpenMP threads using shared memory. As a result, the computation-to-communication ratio increases compared to the pure MPI implementation.

Execution time for batch sizes of 24, 48 and 96 DTW comparisons using 8 nodes. The results indicate that larger batches lead to slightly better performance, with speedup increasing from 88.81 to 91.02 and parallel efficiency increasing from 92.52% to 94.81% as the batch size grows from 24 to 96 tasks. This behavior can be explained by the reduction in the frequency of communication between the master and worker processes. When larger batches are used, the communication cost is amortized over a greater amount of computation, decreasing the relative overhead associated with task distribution and result collection.

Furthermore, the coefficient of variation remained below 2.1% for all configurations, indicating highly stable execution times and strong experimental reproducibility. The standard deviation also decreased as the batch size increased, suggesting that larger task groups reduce runtime variability. Although excessively large batches can potentially introduce load imbalance in heterogeneous workloads, the tested configurations maintained a good balance between communication cost and workload distribution. Consequently, a batch size of 96 tasks provided the best overall performance among the evaluated configurations, demonstrating that coarse-grained task assignment is beneficial for the proposed hybrid MPI+OpenMP DTW implementation.

5. Discussion

The experiments also highlight that the computational cost of DTW is the main obstacle to its application to large financial datasets. Since pairwise DTW computation has quadratic complexity with respect to both the number and the length of the time series, sequential execution becomes impractical for datasets containing thousands of assets. This char-

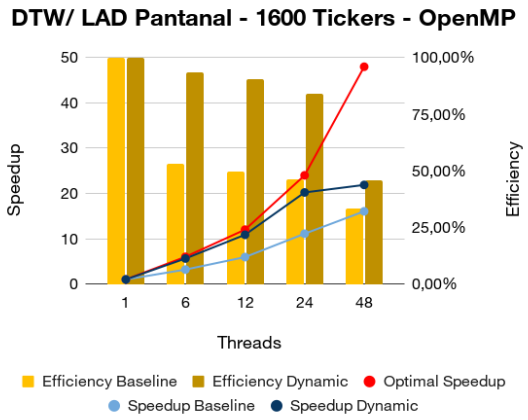

Figure 1. Speedup and parallel efficiency of the OpenMP implementation on the Pantanal cluster using 1,600 financial assets.

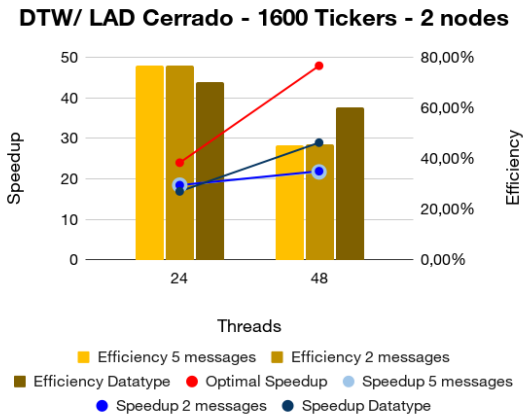

Figure 2. Performance comparison of the three MPI implementations on the Cerrado cluster using 1,600 financial time series.

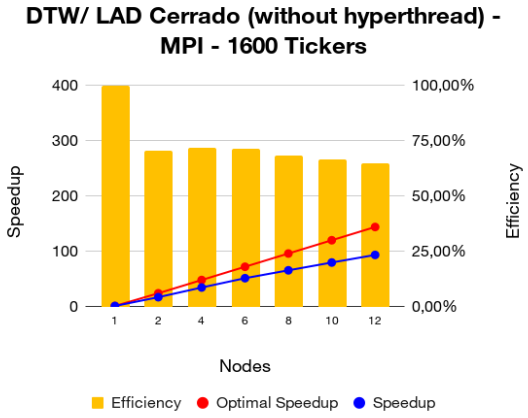

Figure 3. Speedup as a function of the number of MPI processes for the DTW implementation on the Cerrado cluster without hyper-threading.

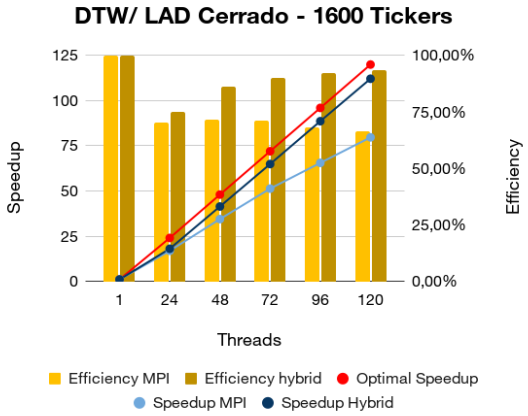

Figure 4. Comparison of speedup and parallel efficiency between the pure MPI and hybrid MPI+OpenMP implementations without hyper-threading.

acteristic motivates the use of parallel computing to make large-scale similarity analysis feasible.

Regarding shared-memory execution, replacing the guided scheduling policy adopted in the baseline implementation with dynamic scheduling improved load balancing. Although the number of DTW comparisons decreases across the upper triangular distance matrix. Assigning one DTW comparison at a time allowed idle threads to immediately receive new work, reducing processor idle time and improving resource utilization.

The MPI implementation successfully extended the computation beyond a single compute node. Since DTW comparisons are independent, distributing them among pro-

cesses provides a naturally parallel workload. Communication optimizations, including the use of MPI derived datatypes and task batching, reduced communication overhead without affecting the correctness of the computation. Experimental results showed that increasing the batch size produced only marginal improvements, indicating that communication was no longer the dominant execution cost and that computation became the primary performance bottleneck.

Within each experimental platform, the results indicate that dynamic scheduling improved the scalability of the OpenMP implementation, while communication optimizations and task batching enhanced the MPI performance. The hybrid MPI+OpenMP implementation achieved high parallel efficiency on Cerrado by combining inter-node workload distribution with shared-memory parallelism inside each node, reducing communication overhead while maintaining effective utilization of computational resources.

Overall, the results demonstrate that combining distributed-memory and shared-memory parallelism makes DTW-based similarity analysis practical for large financial datasets, reducing execution time from several hours to only a few minutes. Although the experiments considered only historical stock prices and the DTW similarity measure, the proposed parallelization strategies are general and can be adapted to other time-series applications and elastic distance measures.

6. Related Work

The computational cost of DTW has motivated the development of parallel and distributed implementations. Miniakhmetov et al. [Miniakhmetov et al. 2015] presented a parallel subsequence matching algorithm using OpenMP and Intel Xeon Phi coprocessors. By dynamically offloading DTW computations, their method achieved significant speedups, including up to $3\times$ on real ECG datasets, while maintaining high coprocessor utilization for long queries. This study demonstrates that parallelism can effectively accelerate DTW computations.

Hamerski et al. [Hamerski et al. 2017] presented a publish-subscribe middleware designed for multiprocessor systems-on-chip (MPSoCs) and evaluated its performance using a DTW-based application. The study compared three task communication models: MPI with full pre-mapping (MPI-all), MPI with on-demand mapping (MPI-dem), and a custom PUB-SUB model across 1:N and N:1 communication patterns. The results showed that, although MPI exhibited lower latency for small communication volumes, the PUB-SUB middleware reduced execution time by up to 29.9% as the workload increased. These gains were attributed to reduced system-call overhead and more efficient message exchanges. Despite requiring an additional 7 kB of memory compared with MPI, the PUB-SUB model proved advantageous for DTW computations.

Yang et al. [Yang et al. 2022] provide a comprehensive evaluation of parallel implementations of DTW and its variants, comparing CPU and GPU approaches. The study shows that GPU acceleration can achieve speedups of one to two orders of magnitude for very long sequences (approximately 50,000–100,000 samples or more). However, for shorter sequences, the overhead associated with CPU–GPU data transfers often outweighs the computational benefits, making optimized CPU implementations more efficient. Furthermore, the quadratic memory requirements of DTW limit the maximum sequence length that can be processed on GPUs. These findings suggest that CPU-based

parallelization remains a suitable approach for datasets composed of numerous short or medium-length time series.

Sigaki et al. [Sigaki et al. 2019] analyzed the informational efficiency of 437 cryptocurrencies by computing permutation entropy and statistical complexity over sliding windows and measuring the similarity between the resulting time series using DTW. Hierarchical clustering with average linkage was employed, and cluster quality was evaluated using silhouette coefficients. The study illustrates the effectiveness of shape-based similarity combined with unsupervised learning for financial pattern discovery and highlights the use of practical libraries such as *DTAIDistance* [Meert et al. 2020].

To the best of our knowledge, previous studies have investigated OpenMP- and MPI-based DTW parallelization strategies in various application domains. However, we found no study that jointly evaluates OpenMP, MPI, and hybrid MPI+OpenMP implementations for financial time series analysis.

7. Conclusion

This work investigated the use of parallel computing to accelerate DTW for financial time-series similarity analysis. Three parallel models were developed and evaluated: OpenMP for shared-memory systems, MPI for distributed-memory environments, and a hybrid MPI+OpenMP approach. Experimental results showed that all implementations significantly reduced execution time compared with the sequential baseline while preserving the correctness of the DTW computation.

Among the evaluated approaches, the OpenMP implementation achieved high efficiency on a single multicore node, whereas MPI enabled the computation to scale across multiple nodes. Communication optimizations and task batching reduced the impact of message passing, thereby improving scalability without affecting load balance. The hybrid MPI+OpenMP implementation provided the best overall performance by combining distributed-memory parallelism with efficient shared-memory execution, achieving the highest speedup and resource utilization among the evaluated configurations.

Overall, the results demonstrate that parallel computing substantially increases the practicality of DTW-based similarity analysis for large financial datasets, reducing execution times from hours to minutes. Future work includes investigating adaptive load-balancing strategies, online DTW algorithms for streaming financial data, and extending the proposed approach to larger HPC systems while also exploring alternative similarity measures.

Acknowledgments

This paper was supported by the Ministry of Science, Technology, and Innovations, with resources from Law No. 8248, dated October 21, 1991, within the scope of PPI-SOFTEX, coordinated by Softex, and published in Residência em TIC - Aditivo, Official Gazette 01245.012095/2020-56.

The authors would like to thank the High Performance Computing Laboratory (LAD-IDEIA) at the Pontifical Catholic University of Rio Grande do Sul (PUCRS), Brazil, for providing the computational infrastructure and technical support that made this research possible and contributed to the results presented in this paper.

References

Atanasovska, A. M., Zdraveski, V., and Spasov, D. (2024). Parallelizing dynamic time warping algorithm for hotel occupancy time series similarity measurement. In *2024 47th MIPRO ICT and Electronics Convention (MIPRO)*, pages 1974–1978.

Chan, G. Y.-Y., Nonato, L. G., Palpanas, T., Silva, C. T., and Freire, J. (2026). Tivy: Time series visual summary for scalable visualization. *IEEE Transactions on Visualization and Computer Graphics*, 32(1):232–242.

de Jesus, L. C., Fernández-Navarro, F., and Carbonero-Ruz, M. (2025). Enhancing financial time series forecasting through topological data analysis. *Neural Comput Applications*, 37:6527–6545.

Hamerski, J. C., Abich, G., Reis, R., Ost, L., and Amory, A. (2017). Publish-subscribe programming for a noc-based multiprocessor system-on-chip. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4.

Huang, S., Dai, G., Sun, Y., Wang, Z., Wang, Y., and Yang, H. (2013). Dtw-based subsequence similarity search on amd heterogeneous computing platform. In *2013 IEEE 10th International Conference on High Performance Computing and Communications 2013 IEEE International Conference on Embedded and Ubiquitous Computing*, pages 1054–1063.

Meert, W., Hendrickx, K., Van Craenendonck, T., Robberechts, P., Blockeel, H., and Davis, J. (2020). Dtaidistance.

Miniakhmetov, R., Movchan, A., and Zymbler, M. (2015). Accelerating time series subsequence matching on the intel xeon phi many-core coprocessor. In *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1399–1404.

Ninama, K., Patel, J., K K, G., Reddy, M. R. V. S. R. S., and Bhowmik, B. (2025). Performance analysis of hybrid mpi and openmp on smith-waterman algorithm. In *2025 3rd International Conference on Intelligent Systems, Advanced Computing and Communication (ISACC)*, pages 887–892.

Radovanović, A., Ye, X., Milanović, J. V., Milosavljević, N., and Storchi, R. (2020). Application of the k-medoids partitioning algorithm for clustering of time series data. In *2020 IEEE PES Innovative Smart Grid Technologies Europe (ISGT-Europe)*, pages 645–649.

Saklani, R., Purohit, K., Vats, S., Sharma, V., Kukreja, V., and Yadav, S. P. (2023). Multicore implementation of k-means clustering algorithm. In *2023 2nd International Conference on Applied Artificial Intelligence and Computing (ICAAIC)*, pages 171–175.

Sakoe, H. and Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49.

Sigaki, H. Y. D., Perc, M., and Ribeiro, H. V. (2019). Clustering patterns in efficiency and the coming-of-age of the cryptocurrency market. *Scientific Reports*, 9(1):1440.

Vintsyuk, T. K. (1968). Speech discrimination by dynamic programming. *Cybernetics*, 4(1):52–57.

Winarti, T., Widodo, E., Handayni, S., and Nugroho, A. (2024). Utilizing pearson correlation matrix to identify negative correlations among cryptocurrencies. In *2024 3rd International Conference on Creative Communication and Innovative Technology (ICCIT)*, pages 1–7.

Yang, D., Shaw, T., and Tsai, T. (2022). A study of parallelizable alternatives to dynamic time warping for aligning long sequences. *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, 30:2117–2127.

Zhang, R., Li, H., and Wang, J. (2021). Index tracking based on dynamic time warping and constrained k-medoids clustering. In *2021 11th International Conference on Intelligent Control and Information Processing (ICICIP)*, pages 352–359.

Zhu, B., Jiang, Y., Gu, M., and Deng, Y. (2021). A gpu acceleration framework for motif and discord based pattern mining. *IEEE Transactions on Parallel and Distributed Systems*, 32(8):1987–2004.