Li, Y, Li, H, Zhang, C, Zhao, Y and Yang, Z

Incorporation of adaptive compression into a GPU parallel computing framework for analyzing large-scale vessel trajectories

Article

Transportation Research Part C 163 (2024) 104648

Contents lists available at ScienceDirect

Transportation Research Part C

journal homepage: www.elsevier.com/locate/trc

Incorporation of adaptive compression into a GPU parallel computing framework for analyzing large-scale vessel trajectories

Yan Li [a], Huanhuan Li [b,*], Chao Zhang [a], Yunfeng Zhao [b], Zaili Yang [b,*]

[a] State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing, Wuhan University, Wuhan, China
[b] Liverpool Logistics, Offshore and Marine (LOOM) Research Institute, Liverpool John Moores University, Liverpool, UK

A R T I C L E I N F O

Keywords:
Trajectory compression
Parallel computing
Graphics processing unit (GPU)
Unmanned ships
Maritime safety

A B S T R A C T

Automatic Identification System (AIS) offers a wealth of vessel navigation data, which underpins research in maritime data mining, situational awareness, and knowledge discovery within the realm of intelligent transportation systems. The flourishing marine industry has prompted AIS satellites and base stations to generate massive amounts of vessel trajectory data, escalating both data storage and calculation costs. The conventional Douglas-Peucker (DP) algorithm used for trajectory compression sets a uniform threshold, which hampers effective compression. Additionally, compressing and accelerating the computation of large datasets poses a significant challenge in real-world applications. To address these limitations, this paper aims to develop a new Graphics Processing Unit (GPU) parallel computing and compression framework that enables the acceleration of the optimal threshold calculation for each trajectory automatically in maritime big data mining. It achieves this by incorporating a new Adaptive DP with Speed and Course (ADPSC) algorithm, which utilizes the dynamic navigation characteristics of different vessels. It can effectively solve the associated computational time cost concern when using the ADPSC algorithm to compress vast trajectory datasets in the real world. Additionally, this paper proposes a novel evaluation metric for assessing compression efficacy based on the Dynamic Time Warping (DTW) method. Comprehensive experiments encompass vessel trajectory datasets from three representative research areas: Tianjin Port, Chengshan Jiao Promontory, and Caofeidian Port. The experimental results demonstrate that 1) the newly developed ADPSC method outperforms in terms of compression, and 2) the designed GPU parallel computing framework can significantly shorten the compression time for extensive datasets. The GPU-accelerated compression methodology not only minimizes storage and transmission costs for data from both manned and unmanned vessels but also enhances data processing speed, supporting real-time decision-making. From a theoretical perspective, it provides the key to the puzzle of realizing the real-time anti-collision of manned and unmanned ships, particularly in complex waters. It hence makes significant contributions to maritime safety in the autonomous shipping era.

1. Introduction

The rise of economic globalization has catalyzed frequent import and export trade among countries, spurring the continuous growth in the shipping industry (Li et al., 2023a; Tagiltseva et al., 2022). Among various transport sectors enabling international trade,

* Corresponding authors.
E-mail addresses: H.Li2@ljmu.ac.uk (H. Li), Z.Yang@ljmu.ac.uk (Z. Yang).

https://doi.org/10.1016/j.trc.2024.104648
Received 27 September 2023; Received in revised form 12 April 2024; Accepted 30 April 2024
Available online 9 May 2024

Nomenclature

Roman letters

AIS	Automatic Identification System
ADP	Adaptive Douglas-Peucker
ADPSC	Adaptive Douglas-Peucker with Speed and Course
ADPS	Adaptive Douglas-Peucker with Speed
COG	Course Over Ground
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
CR	Compression Ratio
DP	Douglas-Peucker
DTW	Dynamic Time Warping
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
GPU	Graphics Processing Unit
G-PSQUISH-E	GPU-assisted PSQUISH-E
GPGPU	General-Purpose GPU
MPDP	Multi-objective Peak DP
OW	Open Window
Opening Window Time Ratio	OPW-TR
PSQUISH-E	Parallel version of the SQUISH-E
RLL	Rate of Length Loss
SOG	Speed Over Ground
SW	Sliding Window
SPM	Scan-Pick-Move
SQUISH	Spatial QUalIty Simplification Heuristic
SQUISH-E	SQUISH-Extended
SR	Speedup Ratio
TD-TR	Top-Down Time-Ratio
TP-DTW	Trajectory Points-based DTW
US	Uniform Sampling

the maritime industry has emerged as the preferred one for global trade transportation. Its dominance is attributed to cost-effective shipping and the ability to accommodate a diverse range of goods (Li et al., 2022; Li and Lam, 2017). Vessels are pivotal in the maritime industry and can generate massive dynamic data during navigation (Li et al., 2024). This data typically includes details like time stamp, longitude, latitude, Speed Over Ground (SOG), and Course Over Ground (COG). Integrated into an Automatic Identification System (AIS) (Li et al., 2023b; Yang et al., 2019), this data is transmitted to designated servers through base stations and satellites, as illustrated in Fig. 1. Once received, AIS data undergoes processing through algorithms such as mathematical statistics, visual analysis, and anomaly recognition on the server side, yielding valuable information (e.g., traffic intensity, traffic flow distribution characteristics, and collision risk) (Li et al., 2023; Xin et al., 2023). Such information not only aids in monitoring vessel navigation dynamics but also provides critical insights for developing intelligent maritime transportation systems.

The robust growth of the global maritime industry has led to a steady rise in the number of vessels, resulting in a surge of vessel navigation (or AIS) data transmitted to servers. This data expansion strains storage capacity, often containing redundant information (Sun et al., 2020; Zheng et al., 2020). It is crucial to remove such redundancies for accurate data analysis. For instance, in a vessel's straight-line navigation, merely noting the starting and ending points suffices to understand its trajectory, rendering intermediate points unnecessary.

Trajectory compression technology emerges as an effective solution to these challenges in accurate data analysis. Its core idea is to pinpoint feature points in a trajectory to replace the original data, eliminating redundancy and thus reducing storage costs (Liu et al., 2019b; Tang et al., 2021b). Furthermore, trajectory compression offers computational cost savings (Chen et al., 2020b, 2020a). Since the compressed vessel trajectory efficiently captures sailing characteristics while reducing computational time for other research algorithms, numerous scholars employ it for future studies, including trajectory clustering (Bai et al., 2023; Tang et al., 2021a), route extraction (Karataş et al., 2021; Yan et al., 2020), path planning (Gu et al., 2023; Liu et al., 2019a), trajectory anomaly recognition (Dogancay et al., 2021; Liang et al., 2022; Rong et al., 2020), and collision avoidance (Wang et al., 2023, 2024b,a; Xin et al., 2023). It is evident that trajectory compression is a pivotal data preprocessing technique and stands as an essential research component in data mining (Li et al., 2022).

The trajectory compression research predominantly encompasses two approaches: batched compression (Li et al., 2019) and online compression (Liu et al., 2016). Their essential difference is that online compression methods utilize local points for calculation when trajectory data is incomplete, making them suitable for application scenarios that involve compressing during transmission. In contrast, batched compression methods leverage the overall distribution of trajectories to identify feature points. Given that the vessel

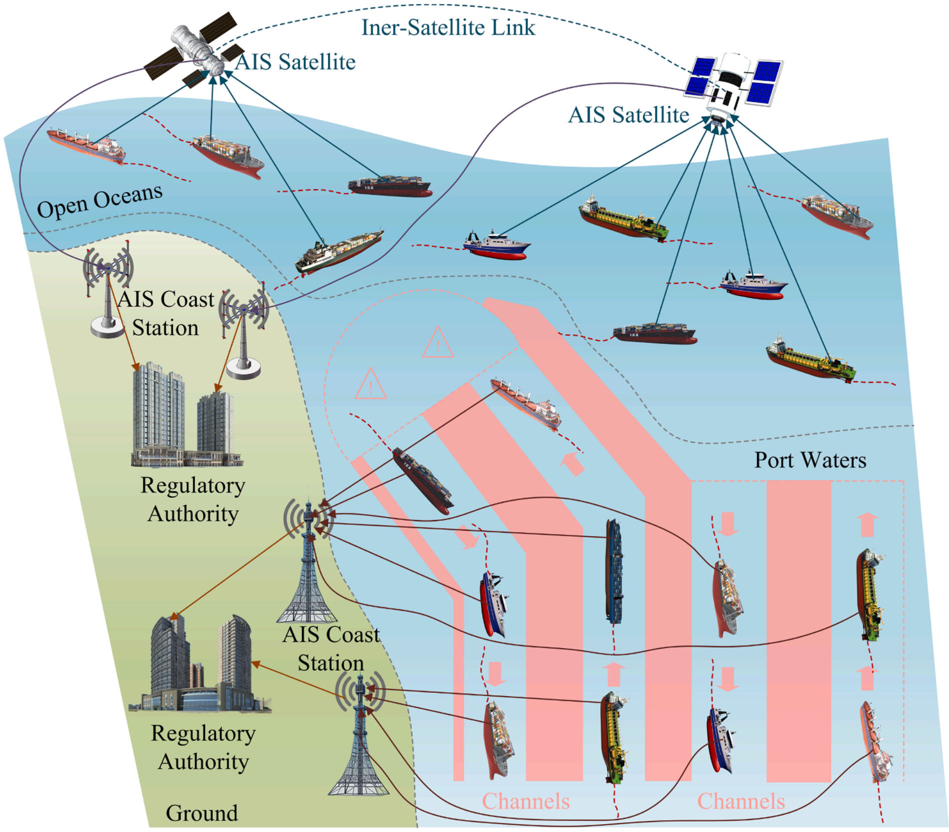


Fig. 1. Visual illustration of satellite and terrestrial AIS networks.

trajectory data acquired in this paper are complete, the focus will be primarily on batched compression techniques. The Douglas-Peucker (DP) algorithm (Saalfeld, 1999) stands out as a renowned, widely adopted, and effective batched compression method, capable of identifying feature points based on the vessel trajectory distribution, subsequently supplanting the original trajectory data. However, the original DP algorithm confronts two notable limitations in handling vessel trajectory compression tasks. First, it mandates the predetermination of a compression threshold, which is conventionally set manually. An excessively high threshold risks omitting vital feature points, considering them redundant, whereas an unduly low threshold may fail to eradicate superfluous data. Additionally, it leads to suboptimal compression results for the majority when applying a uniform compression threshold across all trajectories. Secondly, the original DP algorithm faces challenges in efficiently compressing large-scale data, diminishing its practical applicability.

To address the above challenges, this paper aims to develop a new Graphics Processing Unit (GPU) parallel computing and compression framework, embedding with a novel Adaptive Douglas-Peucker with Speed and Course (ADPSC) algorithm to realize massive trajectory data preprocessing and compression. It can automatically calculate compression thresholds rooted in individual vessel navigation details, including time stamps, longitude and latitude coordinates, SOG, and COG. Moreover, the newly developed ADPSC algorithm has been refined into GPU parallel algorithms (Cheng and Gen, 2019; Kallioras et al., 2015; Świrydowicz et al., 2022; You et al., 2022), which greatly enhances its capability to handle large-scale trajectory data, leading to a significant reduction in algorithm execution time. Finally, a new evaluation index based on the Dynamic Time Warping (DTW) approach is proposed to measure trajectory compression performance. From a theoretical standpoint, this paper provides a crucial piece of the puzzle in achieving real-time anti-collision capabilities for manned and unmanned ships, especially in complex water environments. Consequently, it makes significant contributions to enhancing maritime safety in the era of autonomous shipping.

This paper proposes an adaptive and accelerated compression framework, which can not only accurately simplify vessel trajectories but also process large-scale data quickly to adapt to practical application scenarios in maritime industries. Section 2 provides an overview of vessel trajectory compression methods and GPU parallel computing frameworks, revealing the relevant gaps and contributions. Section 3 serves as the preparation phase for the study, covering two contents. Firstly, it elucidates essential definitions that offer a theoretical overview of this paper. Secondly, it details the preprocessing of experimental data, encompassing tasks such as denoising vessel trajectory data and converting geographical coordinates. Section 4 delves into an in-depth exploration of the relevant theories behind the original DP, optimized ADPSC, and GPU-based parallel acceleration ADPSC algorithms. In Section 5, a comparative experiment is conducted, evaluating the new ADPSC algorithm both qualitatively and quantitatively. The evaluation underscores its superior performance over the original DP algorithm in addressing vessel trajectory data compression challenges. Meanwhile, the speedup ratio is employed to analyze the acceleration benefits of the optimized ADPSC algorithm when compressing large-scale vessel trajectory data using the GPU parallel computing framework, as opposed to traditional Central Processing Unit (CPU) serial approaches. Section 6 sheds light on the relevant research conclusions and outlines potential avenues for future sustainable research.

2. Literature review

Many scholars have recently investigated trajectory compression to reduce data storage costs. Concurrently, there is a growing inclination towards implementing algorithms on GPU parallel computing frameworks to decrease algorithmic computation time. Advanced methods specific to vessel trajectory compression will be presented first in Section 2.1. Subsequently, Section 2.2 delves into the evolutionary development of GPU parallel computing. Section 2.3 summarizes the research contributions of this paper.

2.1. Review of studies on maritime trajectory compression methods

Vessel trajectory refers to how the position of a vessel in space changes over time, with its change function being continuous. Unlike simple function curves, trajectory data contains not only position information but also dynamic details such as time, direction, and velocity. At its core, trajectory compression seeks to identify an approximate trajectory with fewer data points to substitute for the original trajectory. The development of the batched compression methods is elaborated upon in Section 2.1.1, while the evolution of the online compression techniques is detailed in Section 2.1.2.

2.1.1. Batched compression methods

It is necessary to collect the completed trajectory when performing vessel trajectory compression tasks using batched methods to further identify feature points in the data and eliminate redundant information (Arslan et al., 2018; Liu et al., 2019b). This entire trajectory is taken into account in the batched methods, making it easier to achieve global optimization during compression.

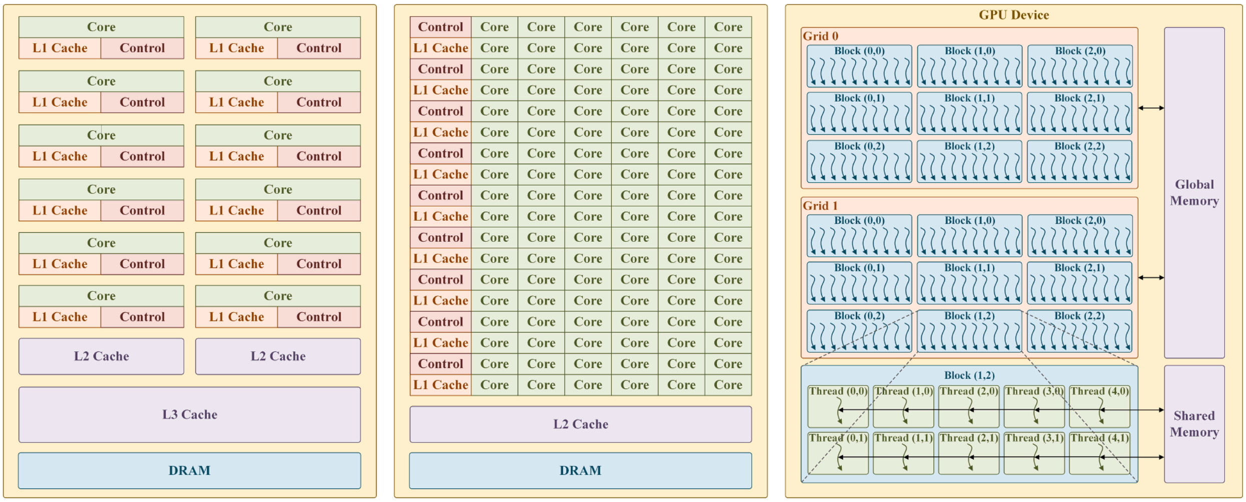
The Uniform Sampling (US) and DP algorithms are the fundamental batched compression techniques frequently utilized in trajectory data compression. These methods have seen extensive refinement over the years, with the DP algorithm garnering particular attention for enhancements. The US algorithm is straightforward, with the core idea of retaining one point out of every W point (Lv et al., 2015; Sun et al., 2016). For instance, given a trajectory with 16 points and deploying the US method to keep one point out of every 5, the resultant trajectory would be formed by the first, sixth, eleventh, and sixteenth points. While the US approach boasts efficiency and reduced computational demand, its shortcoming lies in its inability to effectively conserve crucial trajectory features, leading to disparities between the compressed and the original trajectories.

In contrast, the DP algorithm (Douglas and Peucker, 1973) emerged to address the limitations of the US algorithm. This algorithm, essentially recursive, iteratively identifies feature points based on trajectory distributions and point deviations (Huang et al., 2020). Over time, researchers have improved the DP algorithm to optimize its compression capabilities. As trajectories encapsulate both spatial (longitude and latitude) and temporal (time) aspects, Meratnia and de By (2004) proposed the Top-Down Time-Ratio (TD-TR) algorithm. This approach integrates spatial distance with time ratios, generating more accurate compressed trajectories. To expedite the compression and boost the compression rate, Hansuddhisuntorn and Horanont (2019) put forward an improved version of the TD-TR algorithm called TD-TR Reduce. Liu et al. (2015) utilized data structures like convex hulls to optimize the spatial and temporal complexity of the DP algorithm. Their algorithm could achieve the best compression effect for trajectory data with a space complexity of $O(1)$ and a time complexity of $O(n)$. Zhao and Shi (2018) merged the directional changes of vessel trajectories with the DP algorithm, yielding impressive results under high compression. Zhou et al. (2023) pointed out the shortcomings of the DP algorithm in compressing vessel trajectory data, including three specific points: (1) there is a continuous turning phenomenon in the trajectory, and its compression effect is poor; (2) it does not take into account the impact of vessel speed and course; (3) there's a possibility of errors, such as the compressed vessel trajectory intersecting obstacles. To address the above issues, they introduced a Multi-objective Peak DP algorithm (MPDP), which incorporated a peak sampling strategy. In recent years, scholars identified that universally applying the same manually set compression threshold across all trajectories compromises compression accuracy. Therefore, some adaptive compression algorithms have been proposed by scholars. Liu et al., (2019b) proposed the Adaptive Douglas-Peucker (ADP) algorithm for compressing vessel trajectory data, which employs the average distance from the trajectory point to the baseline (a line linking the starting and ending points) to automatically calculate the compression threshold. Tang et al., (2021b) utilized the threshold change rate to determine the feature points of each trajectory. Li et al. (2022) developed the compression algorithm proposed by Liu et al., (2019b), incorporating not just the distance from the point to the baseline, but also the velocity change rate of each trajectory point when determining the compression threshold. Their method, termed Adaptive Douglas-Peucker with Speed (ADPS), efficiently extracts the featured points for knowledge discovery.

The adaptive compression algorithm autonomously determines the compression threshold for each trajectory, guaranteeing trajectory accuracy even at elevated compression rates. Unlike some existing adaptive compression algorithms, the newly developed ADPSC algorithm takes into account variations in speed and direction for each trajectory point, addressing a longstanding research gap. Furthermore, the ADPSC algorithm is redesigned for a parallelized version tailored for GPU computing frameworks, enabling rapid compression of extensive vessel trajectory datasets for real-world applications.

2.1.2. Online compression methods

The online compression methods employ local features of trajectories to identify critical points (Gao et al., 2019). It is suitable for application scenarios where compression occurs simultaneously with transmission. Keogh et al. (2001) introduced two algorithms,



(a) (b) (c)

Fig. 2. Overview of differences between CPU and GPU computing frameworks. (a) CPU architecture, (b) GPU architecture, and (c) conceptual architecture of the CUDA parallel computing model.

namely Sliding Window (SW) and Open Window (OW). The core idea of SW employs an initialized sliding window to gradually search for feature points in the trajectory. If no feature points are found in the trajectory segment inside the sliding window, it expands, encompassing new trajectory points until the final trajectory point is reached. While OW and SW share similar compression fundamentals, they differ in their approach to assessing feature points. Gao and Shi (2019) extract key feature points from vessel trajectories based on vessel heading angle deviation, position deviation, and spatiotemporal features of AIS data, and then optimize the SW algorithm. This new method is called the vessel spatiotemporal key feature point online extraction algorithm. Sun et al. (2020) proposed a Scan-Pick-Move (SPM) trajectory data compression algorithm based on SW to address the high compression rate and long processing time in existing online trajectory compression algorithms. They used the maximum offset distance reference trajectory point to determine whether the current trajectory point can be compressed, aiming to reduce the storage space. Zhu and Ma (2021) used the trajectory change rate and velocity change rate in SW as the criteria for simplifying trajectory points. Compared with the DP algorithm, SW algorithm, and Opening Window Time Ratio (OPW-TR) algorithm, their method effectively considered vessel behavior patterns and compressed vessel trajectory data. Liu and Yang (2023) proposed an improved opening window trajectory simplification algorithm, analyzing the impact of distance and velocity thresholds on algorithm performance to determine appropriate simplification thresholds. It is worth noting that this algorithm better preserves the position information and spatial features of the original trajectory. Potamias et al. (2006) put forward the threshold-guided sampling algorithm, which employs speed and direction to construct secure regions and then searches for feature points. Muckell et al. (2011) developed an online compression algorithm called the Spatial QUalIty Simplification Heuristic (SQUISH). To compress a trajectory, SQUISH starts by initializing a priority queue, and then progressively adds trajectory points. Once the queue hits its capacity, it removes the point, resulting in the least error. After this, the priority of each trajectory point is updated. SQUISH not only maintains the accuracy of compressed trajectories at high compression rates, but also boasts a relatively low time complexity. Consequently, it is a popular choice for online trajectory data compression research, with several scholars suggesting enhanced versions. For example, Muckell et al. (2014) designed the SQUISH-Extended (SQUISH-E) algorithm, which can achieve the best compression rate within a given error threshold. Han et al. (2018) adopted a multi-core computing framework to accelerate the SQUISH-E algorithm, introducing a Parallel version of the SQUISH-E algorithm called PSQUISH-E. Additionally, they developed the GPU-assisted PSQUISH-E algorithm called G-PSQUISH-E.

Both batched and online compression methods compress trajectory data in an Euclidean space. However, their primary distinction lies in their approach to traversing trajectory points. While batched compression methods extract global feature points from trajectory data, online compression techniques rely on local trajectory information to identify these points, which can result in a higher compression error. Meanwhile, the average time complexity of batched compression technology is generally higher than that of online compression technology.

2.2. Review of studies on GPU computation

Given the increased computational complexity of the optimized ADPSC algorithm compared to the DP algorithm and the vast scale of trajectory datasets that need compression, there is a significant computational demand for trajectory compression. Traditional CPU serial computing frameworks struggle to process optimization algorithms for large-scale dataset compression within a reasonable time frame. It would significantly increase computational costs for engineers to upgrade CPU specifications to meet these demands continuously. New solutions to these challenges are demanded with urgency. GPU offers robust computational capabilities, positioning them as formidable platforms for large-scale data mining (Chen et al., 2018; Heywood et al., 2019; Jurczuk et al., 2021). Currently, while mainstream CPUs typically include several cores, each corresponding to two threads for simultaneous computing, conventional GPUs can run thousands of threads concurrently (Huang et al., 2020; Jeong et al., 2022; Qu and Zhou, 2017). The contrasting structures of CPUs and GPUs are illustrated in Figs. 2 (a) and (b).

The GPU's capacity to support thousands of threads underscores its superior computing prowess compared to the CPU, as depicted in Fig. 2 (c). In the GPU parallel computing architecture, a thread is the foundational calculation unit. One block consists of up to 1,024 threads, and several blocks combine to form a grid (Lin et al., 2023). Within a block, threads can communicate via shared memory. However, inter-block thread communication relies on global memory. Notably, data transfer in shared memory is more efficient than that in global memory (Cagigas-Muñiz et al., 2022; Shanbhag et al., 2022). It is challenging to apply the intricate GPU directly in conventional parallel computing. To simplify the GPU calculation process, the General-Purpose GPU (GPGPU) was proposed, mapping general computing tasks to graphic hardware (Owens et al., 2007). Although GPGPU offers a more accessible entry point than GPU, only professional engineers well-versed in graphic APIs can proficiently master it. As a result, Nvidia launched Compute Unified Device Architecture (CUDA) in 2006, overcoming the drawbacks of both GPU and GPGPU. CUDA provides a versatile programming interface, facilitating data processing and analysis tasks on NVIDIA GPUs (Basnet et al., 2022).

Parallel algorithms can map data across GPU threads, enabling swift execution of tasks like processing large-scale datasets (Manduhu and Jones, 2019; Roberge and Tarbouchi, 2021). Since vessel trajectories consist of distinct points, each being logically independent, it is evident that designing parallel algorithms within the GPU computing structure is suitable for processing extensive vessel trajectory data in the maritime sector. However, the current literature reveals that very few studies relating to the use of GPU for vessel trajectory analysis, and to the authors' best knowledge, no studies have been undertaken to address the design of parallel algorithms in a GPU computing structure in the maritime sector, disclosing the theoretical novelty of this work.

2.3. Contributions of our study

To address the aforementioned limitations and research gaps, this paper proposed an ADPSC method, an innovative DP-based trajectory compression algorithm, harnessing vessel trajectories' distribution characteristics and additional dynamic navigation data such as time stamps, SOG, and COG to determine the compression threshold for each trajectory autonomously. Consequently, it proficiently prunes unnecessary data, addressing the first limitation of the DP algorithm. Furthermore, this paper refines the ADPSC algorithm to expedite the compression of massive vessel trajectories within the GPU parallel computing environment, effectively tackling the DP algorithm's second limitation and research gaps in GPU computation. The paper's core contributions are delineated as follows.

(1) *Propose an adaptive compression algorithm based on multiple factors.*

The new ADPSC algorithm leverages the dynamic navigation information of ships, including location, SOG, and COG. This ensures a unique compression threshold for each trajectory. The proposed approach primarily resolves the problem of employing a universal threshold for all trajectories, which previously led to suboptimal compression outcomes.

(2) *Develop a GPU parallel computing framework that integrates an ADPSC algorithm.*

Given the vast datasets requiring compression in real-world applications, this paper further fine-tunes the ADPSC algorithm, constructing parallel computing strategies to facilitate trajectory data compression on GPUs. This enhancement notably boosts computational speed, making it apt for real-time data processing in practical settings.

(3) *Design a new index for evaluating compression effectiveness.*

Evaluation metrics offer a quantitative assessment of algorithmic compression performance. The DTW approach serves as a metric to gauge the similarity between trajectories pre and post-compression, thereby appraising the algorithm's compression quality. This paper deviates from conventional DTW measurement methods. Instead, it first computes the distance between trajectory points and the baseline both before and after compression, generating two vectors. Subsequently, the DTW method measures the similarity between these vectors, using the resultant value to evaluate compression efficacy. This improved method proves more apt for assessing the compression outcomes of trajectory data.

(4) *Conduct comparative experiments using three substantial and representative datasets.*

To demonstrate the universality of the experiment, this paper collected actual AIS data from three representative regions: Tianjin Port, Chengshan Jiao Promontory, and Caofeidian Port. In the comparative analysis of compression performance, a mix of qualitative and quantitative methods is employed to evaluate the superior performance of the ADPSC algorithm over the original DP algorithm. Additionally, this paper uses the acceleration ratio as a metric to quantitatively gauge the speed-up efficiency of the ADPSC algorithm when compressing trajectory data within the GPU parallel computing framework.

3. Preliminary

This section provides clear definitions essential for understanding and implementing trajectory compression, as detailed in Section 3.1. Additionally, vessel trajectory data should undergo processes such as denoising and coordinate conversion before compression. The denoising step focuses on identifying and rectifying noisy data, with its specifics elaborated in Section 3.2. Meanwhile, coordinate conversion translates trajectory data from the World Geodetic System – 1984 Coordinate System to the Mercator Projection Coordinate System, a procedure that will be described in Section 3.3.

Table 1
List of the notations.

Notations	Definition	Notations	Definition
T_{ori}	The original vessel trajectory data	Eps	The domain radius when defining density
P^n_{ori}	The n-th point in the original vessel trajectory T_{ori}	$Minpt$	The number of minimum point sets within the cluster
t^n_{ori}	The timestamp data in the n-th original vessel trajectory point	lon_{t-1}	The longitude of trajectory points at t - 1
lon^n_{ori}	The longitude data in the n-th original vessel trajectory point	lat_{t-1}	The latitude of trajectory points at t - 1
lat^n_{ori}	The latitude data in the n-th original vessel trajectory point	lon_t	The longitude of trajectory points at t
sog^n_{ori}	The SOG data in the n-th original vessel trajectory point	lat_t	The latitude of trajectory points at t
cog^n_{ori}	The COG data in the n-th original vessel trajectory point	lon_{t+1}	The longitude of trajectory points at $t + 1$
O	The number of original vessel trajectory points	lat_{t+1}	The latitude of trajectory points at $t + 1$
T_{com}	The compressed vessel trajectory data	R	The radius of a parallel circle at the standard latitude
P^m_{com}	The m-th point in the compressed vessel trajectory T_{com}	d	The long radius of the Earth's ellipsoid
t^m_{com}	The timestamp data in the m-th compressed vessel trajectory point	g	The standard latitude in the Mercator projection
lon^m_{com}	The longitude data in the m-th compressed vessel trajectory point	e	The first eccentricity in the Earth's ellipsoid
lat^m_{com}	The latitude data in the m-th compressed vessel trajectory point	S	Isometric latitude
sog^m_{com}	The SOG data in the m-th compressed vessel trajectory point	lon_W	The longitude data of the vessel trajectory points in the World Geodetic System - 1984 Coordinate System
cog^m_{com}	The COG data in the m-th compressed vessel trajectory point	lat_W	The latitude data of the vessel trajectory points in the World Geodetic System - 1984 Coordinate System
C	The number of compressed vessel trajectory points	lon_M	The longitude data of the vessel trajectory points in the Mercator Projection Coordinate System
$AISM$	The matrix includes the primary dynamic AIS information (i.e., timestamp, longitude, latitude, SOG, and COG) of a vessel	lat_M	The latitude data of the vessel trajectory points in the Mercator Projection Coordinate System

3.1. Definitions

This list of notations is presented in Table 1 to enhance clarity throughout the content. Various essential definitions are provided below.

Definition 1. (***The original vessel trajectory.***) A vessel trajectory T_{ori} with the length O includes a series of points collected by the AIS base station. The mathematical expressions are as follows,

$$T_{ori} = \left\{P^1_{ori}, P^2_{ori}, ..., P^n_{ori}, P^{n+1}_{ori}, ..., P^O_{ori}\right\}, \tag{1}$$

$$P^n_{ori} = \left\{t^n_{ori}, lon^n_{ori}, lat^n_{ori}, sog^n_{ori}, cog^n_{ori}\right\}, n = 1, 2, ..., O \tag{2}$$

where P^n_{ori} denotes the n-th point in the original vessel trajectory T_{ori}. t^n_{ori},lon^n_{ori},lat^n_{ori},sog^n_{ori}, and cog^n_{ori} represent the timestamp, longitude, latitude, SOG, and COG data in the nth trajectory point, respectively.

Definition 2. (***Compressed vessel trajectory.***) A sequence of data T_{com} with a length C is compressed from the original vessel trajectory, whose mathematical expressions are represented in Eqs. (3) and (4),

$$T_{com} = \left\{P^1_{com}, P^2_{com}, ..., P^{m-1}_{com}, P^m_{com}, ..., P^C_{com}\right\}, \tag{3}$$

$$P^m_{com} = \left\{t^m_{com}, lon^m_{com}, lat^m_{com}, sog^m_{com}, cog^m_{com}\right\}, m = 1, 2, ..., C. \tag{4}$$

where P^m_{com} is the m-th point in the compressed vessel trajectory. t^m_{com},lon^m_{com},lat^m_{com},sog^m_{com}, and cog^m_{com} represent the timestamp, longitude, latitude, SOG, and COG data in the mth trajectory point, respectively. Specifically, the disparity between O and C indicates the volume of redundant data removed by the compression algorithm.

Definition 3. (***AIS data matrix.***) A matrix of $5 \times O$ contains a primary dynamic AIS information of a vessel, defined as follows,

$$AISM = \begin{bmatrix} t_1 & t_2 & \cdots & t_n & \cdots & t_O \\ lon_1 & lon_2 & \cdots & lon_n & \cdots & lon_O \\ lat_1 & lat_2 & \cdots & lat_n & \cdots & lat_O \\ sog_1 & sog_2 & \cdots & sog_n & \cdots & sog_O \\ cog_1 & cog_2 & \cdots & cog_n & \cdots & cog_O \end{bmatrix} \tag{5}$$

where O denotes the total number of points in all trajectory data, and 5 counts the number of the involved parameters, including *t, lon, lat, sog,* and *cog*. *AISM* is used as the input data of the GPU-accelerated ADPSC algorithm to reduce frequent data transmission in this paper.

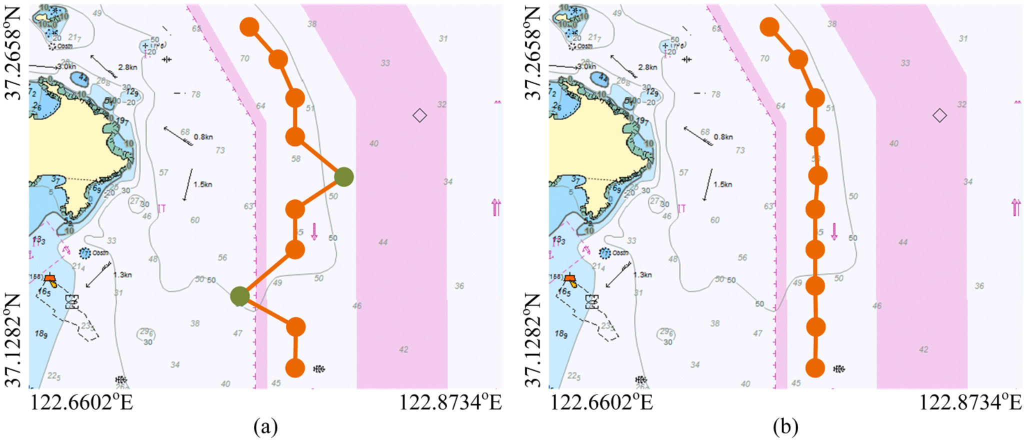


Fig. 3. Visual illustration of vessel trajectory data denoising, (a) trajectory data with noise, and (b) trajectory data after noise removal. In particular, the green dots represent noisy data.

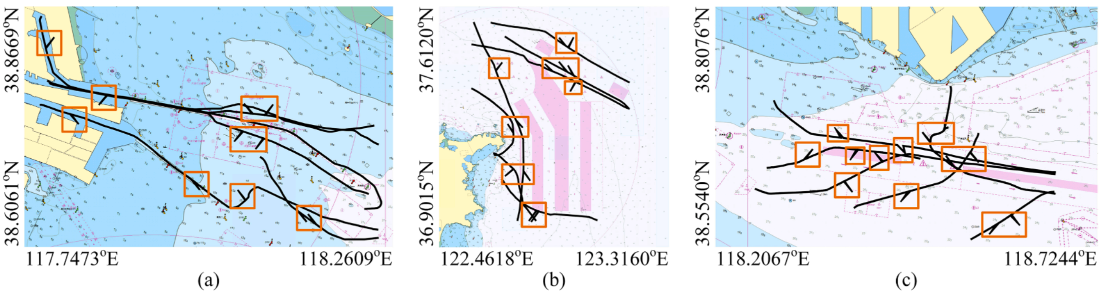


Fig. 4. Visualisation of noisy trajectory data in three research areas: (a) Tianjin Port, (b) Chengshan Jiao Promontory, and (c) Caofeidian Port. In particular, the orange boxes indicate noisy data.

3.2. Trajectory data denoising

During the transmission process of vessel trajectory data between AIS base stations and satellites, there may be noise data, as illustrated in Fig. 3 (a). Unprocessed noise data can impede further research. For instance, when analyzing the compressed content of trajectory data, algorithms may mistakenly assume that noisy data are feature points and preserve them. Therefore, as a preparatory step before conducting specific research in this paper, it is essential to denoise the original trajectory and obtain high-quality trajectory data, as shown in Fig. 3 (b).

The denoising process of original trajectory data mainly includes two aspects: one is to identify noisy data, and the other is to repair noisy data. There are many methods for identifying noisy data, and the most commonly used is to separate noisy data based on clustering methods. This paper uses Density-Based Spatial Clustering of Applications with Noise (DBSCAN) (Bai et al., 2023; Li et al., 2021), the most representative clustering method, to identify noisy data in the original vessel trajectory. This method has two important parameters. One is the domain radius when defining density, abbreviated as *Eps*. The other refers to the threshold for defining core points, which represents the number of minimum point sets within the cluster, abbreviated as *Minpt*. These two parameters determine which cluster each data in the dataset belongs to. In practical clustering calculations, the DBSCAN method divides the trajectory points in the dataset into three categories: core points, boundary points, and noise points. The core point indicates that the number of data points within its radius *Eps* exceeds *Minpt*. The boundary point means that the number of data points contained in its radius *Eps* is less than *Minpt*, and the point falls within the area of the core point. The trajectory points in the dataset that are neither core nor boundary points are classified as noise points. DBSCAN method can accurately identify the noise data in the trajectory. The values of *Eps* and *Minpt* are configured as 0.01 and 3, respectively, as demonstrated in the parameter optimization process detailed in Appendix A. These values represent the optimal parameters for the DBSCAN algorithm to identify outliers across all vessel trajectories within the three study areas.

The next step is to remove these noise data and repair them with the linear interpolation method (Blu et al., 2004). This paper assumes that the trajectory points (lon_t, lat_t) at time t are noise data. The detailed calculation process of using the trajectory coordinates at t - 1 and $t + 1$ to repair the longitude and latitude data of the noise point is shown in Eqs. (6) and (7), respectively,

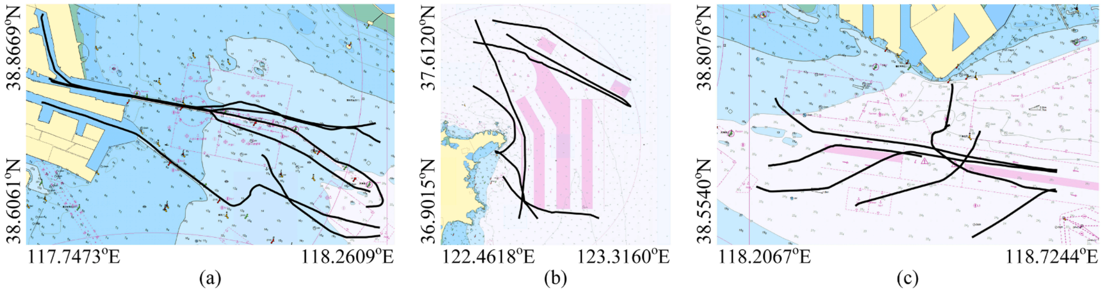


Fig. 5. The distribution of vessel trajectories after denoising in three research areas: (a) Tianjin Port, (b) Chengshan Jiao Promontory, and (c) Caofeidian Port.

Table 2
An example of vessel trajectory points (i.e., longitude and latitude) in two coordinate systems.

World Geodetic System - 1984 Coordinate System		Mercator Projection Coordinate System	
Longitude (°)	Latitude (°)	Longitude (m)	Latitude (m)
118.5658	38.7555	13198682.0677	4686713.5946

$$lon_t = lon_{t-1} + (t - (t-1)) \times \left(\frac{lon_{t+1} - lon_{t-1}}{(t+1) - (t-1)} \right) \tag{6}$$

$$lat_t = lat_{t-1} + (t - (t-1)) \times \left(\frac{lat_{t+1} - lat_{t-1}}{(t+1) - (t-1)} \right) \tag{7}$$

where $lon_{t\text{-}1}$ and lon_{t+1} represent the longitude of trajectory points at t - 1 and $t + 1$, respectively. $lat_{t\text{-}1}$ and lat_{t+1} denote the latitude of trajectory points at t - 1 and $t + 1$, respectively.

Eqs. (6) and (7) take the time information of the noise point and adjacent points as the baseline, and use the coordinates of adjacent points to interpolate the longitude and latitude data of the noise point separately. Figs. 4 and 5 show the distribution of noisy trajectories and preprocessed trajectories in the three study areas on the map, respectively. In summary, DBSCAN and linear interpolation methods can accurately detect and repair noise data, thereby obtaining high-quality trajectory data for subsequent compression research.

3.3. Conversion of geographical coordinates

Calculating the spherical distance between two consecutive points in a trajectory based on the World Geodetic System – 1984 Coordinate System is challenging, potentially leading to significant errors (Huang et al., 2020). Hence, it is not advisable to analyze raw vessel trajectory data directly with compression algorithms. To address the issue, this paper proposes converting the trajectory data coordinates using the Mercator Projection Coordinate System instead of retaining the original system. Meanwhile, the World Geodetic System – 1984 Coordinate System exhibits significant deformation in high latitude areas in two-dimensional space, unlike the Mercator Projection Coordinate System, which does not have this issue. Suppose (lon_W, lat_W) and (lon_M, lat_M) denote the longitude and latitude data of the vessel trajectory points in the original and transformed coordinate systems, respectively. The calculation process of coordinate conversion is as follows:

$$R = \frac{d}{\sqrt{1 - e^2 \sin^2 g}} \times \cos g, \tag{8}$$

$$S = \ln\tan\left(\frac{\pi}{4} + \frac{lat_W}{2}\right) + \frac{e}{2} \ln \frac{1 - e\sin lat_W}{1 + e\sin lat_W}, \tag{9}$$

$$lon_M = lon_W \times R, \tag{10}$$

$$lat_M = S \times R. \tag{11}$$

where R denotes the radius of a parallel circle at the standard latitude, and d represents the long radius of the Earth's ellipsoid. g denotes the standard latitude in the Mercator projection, while e represents the first eccentricity in the Earth's ellipsoid. S denotes isometric latitude.

Table 2 displays the representation of a single point's information from vessel trajectory data in two different coordinate systems.

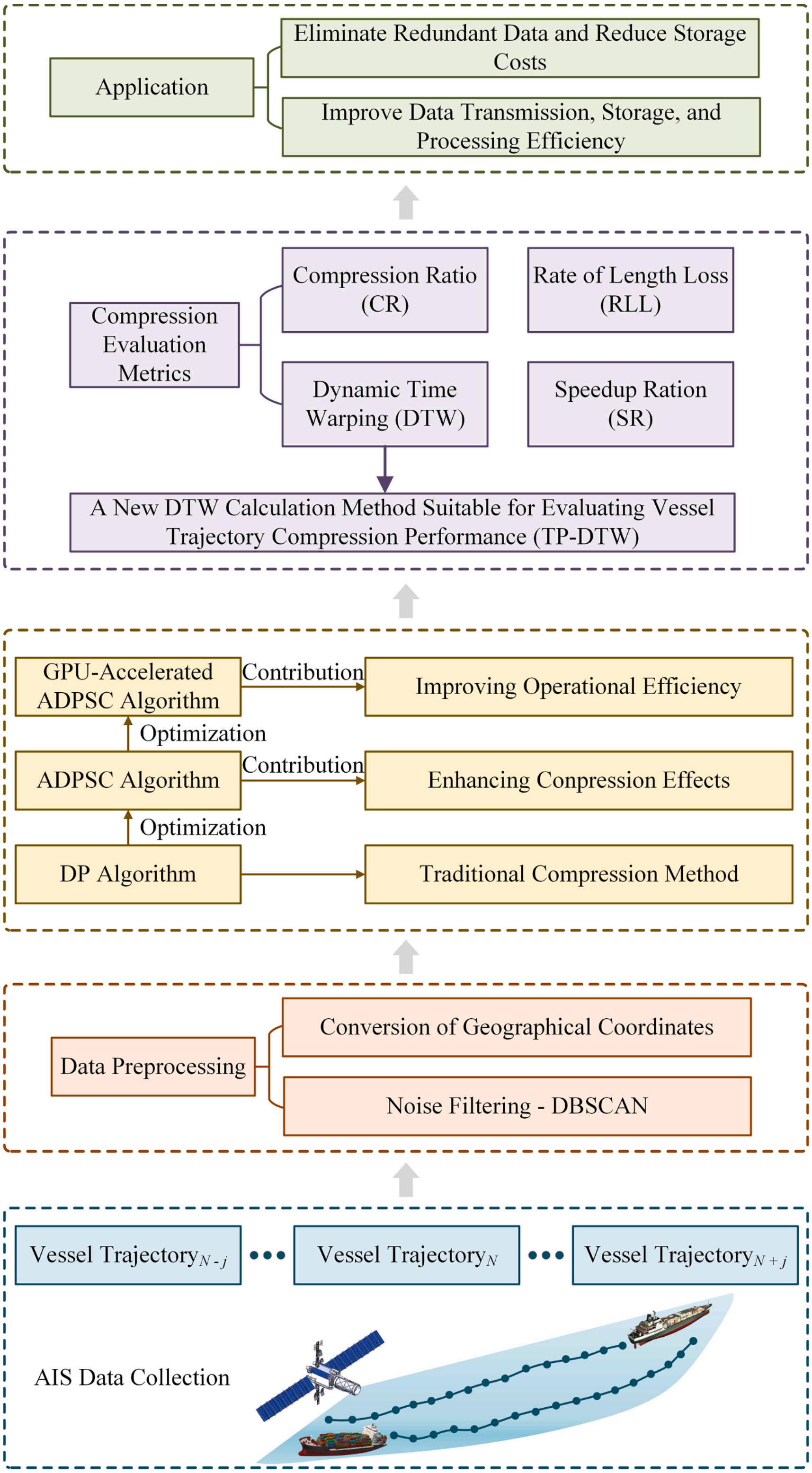


Fig. 6. The whole framework.

Table 3
List of the notations.

Notations	Definition	Notations	Definition
threshold	The compression threshold for each trajectory	α	The angle between the baseline and the *x*-axis of the coordinate system
lt	The total number of trajectory points	*SOG*	The speed over ground
dis_i	The distance from the *i*-th trajectory point to the baseline.	*COG*	The course over ground
lon_S	Longitude of the starting point of vessel trajectory	$SOGY_i$	The velocity component of the *i*-th trajectory point along the *y*-axis in a new coordinate system with the trajectory baseline as the *x*-axis
lat_S	Latitude of the starting point of vessel trajectory	$SOGYV = \{SOGY_1, SOGY_2, \ldots, SOGY_{lt}\}$	The vector is composed of velocity components along the *y*-axis for all trajectory points
lon_E	Longitude of the ending point of vessel trajectory	$TV = \{t_1, t_2, \ldots, t_{lt}\}$	The vector is composed of timestamps for each trajectory point
lat_E	Latitude of the ending point of vessel trajectory	*dataH*	The hour in the timestamp
lon_i	Longitude of the *i*-th intermediate point in the vessel trajectory	*dataM*	The minute in the timestamp
lat_i	Latitude of the *i*-th intermediate point in the vessel trajectory	*dataS*	The second in the timestamp
$DV = \{dis_2, dis_3, \ldots, dis_{lt\text{-}1}\}$	The vector consisting of the distance from each intermediate point to the baseline	$sogyC_i$	The velocity component rate of change of the *i*-th intermediate point along the *y*-axis
Dis	The cumulative distance from the *i*-th trajectory point to the baseline	$SOGYCV = \{sogyC_2, sogyC_3, \ldots, sogyC_{lt\text{-}1}\}$	The vector composed of the *sogyC* of each intermediate point
$rate_i$	The offset weight of the *i*-th intermediate point relative to the baseline.	$sogyCR_i$	The weight change of the *i*-th trajectory point along the *y*-axis SOG component
$RV = \{rate_2, rate_3, \ldots, rate_{lt\text{-}1}\}$	The vector consisting of the offset weight from each intermediate point to the baseline	$SOGYCRV = \{sogyCR_2, sogyCR_3, \ldots, sogyCR_{lt\text{-}1}\}$	The vector composed of the *sogyCR* of each intermediate point

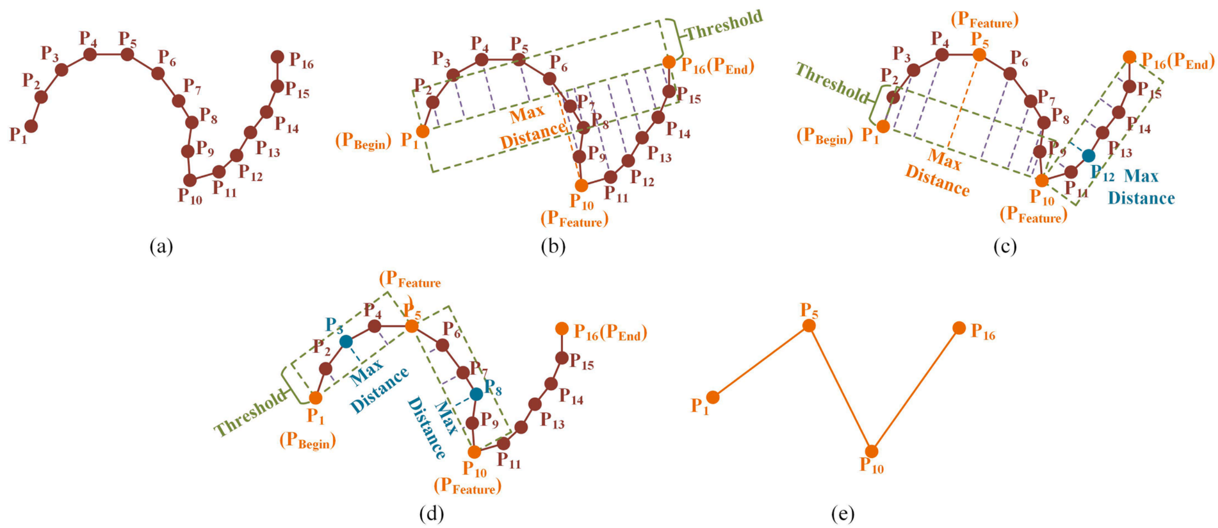


Fig. 7. The schematic of the original DP compression algorithm, (a) an original vessel trajectory, (b) feature points selection based on the threshold, (c) trajectory segmentation and feature points searching within different trajectory segments, (d) identify new feature points iteratively, and (e) the compressed vessel trajectory.

The unit of latitude and longitude coordinates has changed from degrees to meters.

4. Methodology

4.1. A GPU-accelerated compression framework

The entire framework of this paper is illustrated in Fig. 6, encompassing AIS data collection, data preprocessing, GPU-accelerated adaptive compression technique, compression evaluation criteria, and experimental analysis. Embedded within this methodology are the innovative ADPSC approach and its GPU-accelerated counterpart. In particular, AIS data collection and preprocessing are the

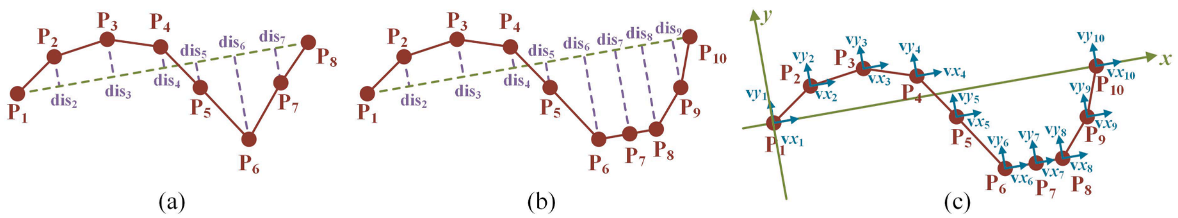


Fig. 8. Visual illustration of trajectory points' offset and SOG decomposition relative to the baseline: (a) and (b) reflect the distance distribution between the midpoint and the baseline for two trajectories, respectively, and (c) displays the SOG decomposition process of trajectory points based on the coordinate system with baseline as the *x*-axis.

preparatory work of the new methodology, aimed at obtaining high-quality vessel trajectory data to avoid the compression algorithm capturing feature points incorrectly. These contents are elaborated in Section 3.2. Sections 4.2 and 4.3 provide comprehensive explanations of the novel ADPSC algorithm and the GPU-accelerated ADPSC method, respectively. They constitute the core content and main contributions of this paper. To verify the effectiveness of the newly proposed algorithms, Sections 5.2, 5.3, and 5.4 introduce evaluation metrics, compression effect, and acceleration performance, respectively. This section introduces new notations for describing the methodology listed in Table 3.

4.2. The proposed ADPSC-based vessel trajectory compression method

The essence of the DP algorithm is to find the feature points in the vessel trajectory data by setting a threshold, illustrated in Fig. 7. These feature points can accurately reflect the features of the original trajectory. A compilation of these features then takes the place of the original trajectory data, achieving the compression objective (Li et al., 2016). Owing to its straightforwardness and efficacy in maritime transportation contexts, it has garnered considerable interest.

The specific steps of the DP algorithm in executing trajectory data compression tasks are as follows:

(1) The starting and ending points of the trajectory serve as feature points and are generated as the baseline, such as points P_1 (P_{Begin}) and P_{16} (P_{End}) in Fig. 7 (b). Using the DP algorithm, the distance from intermediate points (i.e., P_2 to P_{15}) to the baseline is calculated and compared with the threshold. If the distance exceeds the threshold, the point becomes a new feature point, like point P_{10} ($P_{Feature}$) in Fig. 7 (b).

(2) In Fig. 7 (c), the new feature point P_{10} divides the original trajectory into two segments. Subsequently, the feature points P_5 and P_{12} in these segments can be obtained by repeating the calculation process in step (1). Notably, the distance from point P_{12} to the baseline of the second trajectory segment is below the threshold, resulting in only point P_5 becoming a new feature point and splitting the first segment trajectory into two subsets, as depicted in Fig 7 (d).

(3) Although P_3 and P_8 are the farthest from the baseline in new segments, their distances remain below the threshold.

In summary, only points P_1, P_5, P_{10}, and P_{16} are retained as feature points, replacing the original trajectory in Fig. 7 (e). The DP algorithm continues to search for feature points until the maximum distance from any segment point to the baseline is less than the threshold.

The traditional DP algorithm relies on manually establishing a threshold for all trajectory data, typically based on experience rather than theoretical principles. However, this approach results in varied compression thresholds across various vessel trajectories, posing challenges. Utilizing a uniform threshold for all trajectories leads to difficulties in handling redundant data and the potential loss of crucial trajectory feature points. Consequently, the development of adaptive compression algorithms emerges as a crucial research avenue.

Determining a compression threshold is quantifying the extent of deviation (essentially the distance) between trajectory points and the baseline, progressively pinpointing feature points. A trajectory point with a substantial offset from the baseline is more likely to be a feature point. In line with this approach, some researchers compute the average distance from all intermediary points in the trajectory to the baseline, forming a threshold (Li et al., 2022; Liu et al., 2019b). The functional expressions for this threshold are as follows,

$$threshold = \frac{1}{lt-2}\sum_{i=2}^{lt-1} dis_i \tag{12}$$

$$dis_i = \frac{|(lon_S - lon_i) \times (lat_E - lat_i) - (lon_E - lon_i) \times (lat_S - lat_i)|}{\sqrt{(lon_S - lon_E)^2 + (lat_S - lat_E)^2}} \tag{13}$$

where lt represents the total number of trajectory points, and dis_i denotes the distance from the i-th trajectory point to the baseline. (lon_S, lat_S) and (lon_E, lat_E) are the coordinates of the starting and ending points of the trajectory point, respectively. (lon_i, lat_i) represents the longitude and latitude coordinates of the i-th trajectory point. In particular, the denominator in Eq. (13) can not be 0 due to the distinct positions of the collected vessel trajectory data's starting and ending points. Furthermore, the initialization feature points exclude the trajectory's starting and ending points from the calculation, so the value of i is between 2 and lt-1.

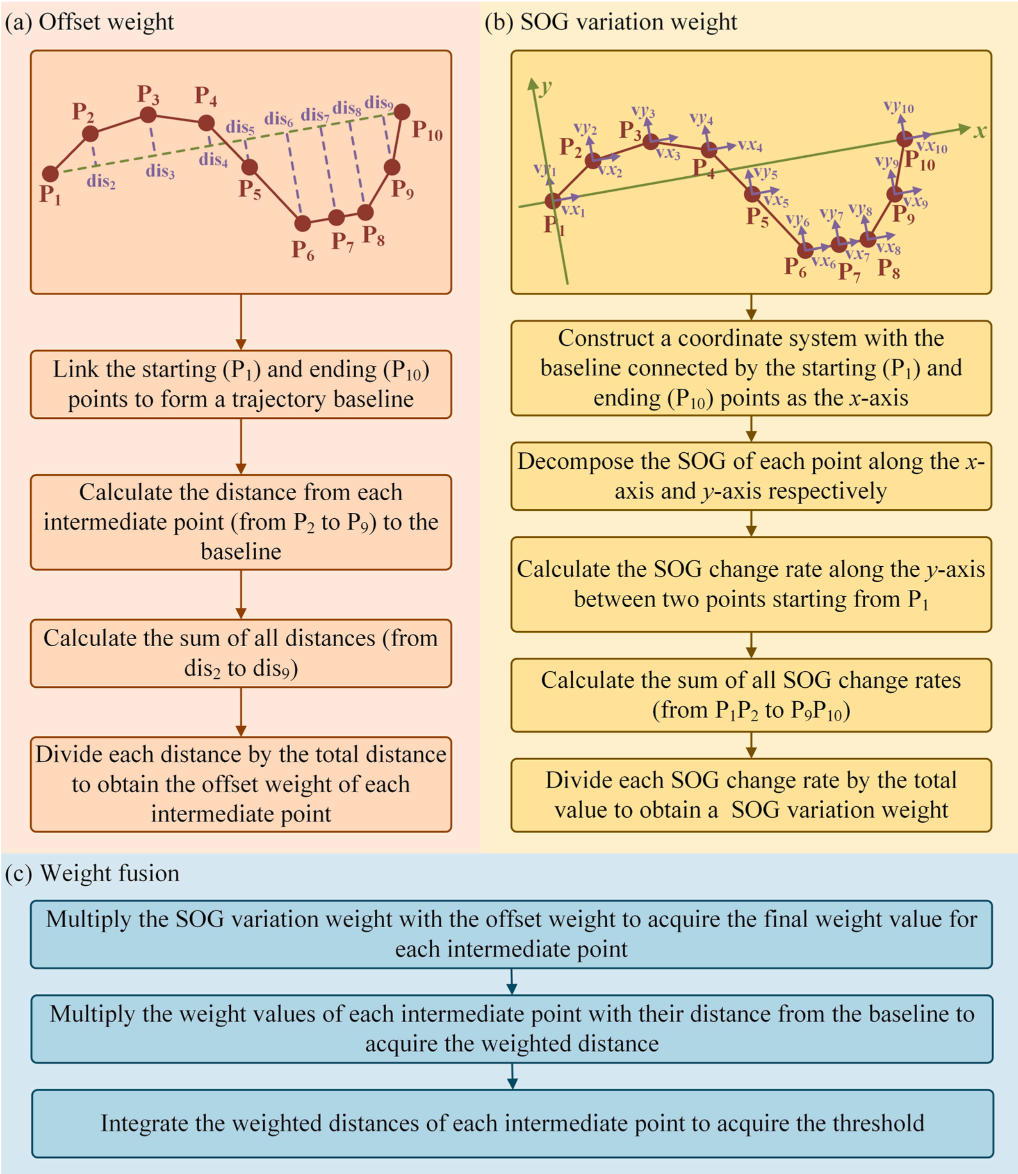


Fig. 9. The scheme of compression threshold calculation: (a) offset weight, (b) SOG variation weight, and (c) weight fusion.

The average calculation method has a limitation, potentially obscuring critical information. As depicted in Fig. 8 (a), the distance from P_6 to the baseline is the largest, implying that its contribution to the threshold calculation is more significant than distances from other trajectory points. Hence, a weighted average calculation method is proposed.

Let a vector $DV = \{dis_2, dis_3, \ldots, dis_{lt-1}\}$ is used to store the distances from all intermediate trajectory points to the baseline. Another vector $RV = \{rate_2, rate_3, \ldots, rate_{lt-1}\}$ denotes the proportion of each distance to the total. The calculation process for each proportional value is outlined as follows,

$$\begin{aligned} Dis &= \sum_{i=2}^{lt-1} dis_i, \\ rate_i &= \frac{dis_i}{Dis}, i = 2, 3, ..., lt-1 \end{aligned} \tag{14}$$

where Dis represents the cumulative distance from the i-th trajectory point to the baseline. dis_i and $rate_i$ represent the distance and proportional weight of the i-th trajectory point to the baseline, respectively.

The threshold calculation based on the weighted average can be expressed by,

$$threshold = \sum_{i=2}^{lt-1} dis_i \times rate_i, i = 2, 3, ..., lt-1 \quad (15)$$

The weighted average calculation method effectively addresses the issue of direct average calculation. However, it also exposes another limitation. This arises when the distance from consecutive n ($n \geq 3$) points to the baseline is identical, and the proportion is the largest. This scenario is exemplified in Fig. 8 (b) by points P_6, P_7, and P_8. These points form a line segment, and according to the compression principle of the DP algorithm, point P_7 becomes redundant and can be removed. If the weighted average calculation method is employed, the distance from P_7 to the baseline also needs to be included, carrying a substantial weight. Consequently, this could result in significant errors in the calculated threshold. To effectively address these issues, this paper incorporates SOG and COG in threshold calculation.

When a vessel navigates in a straight line, even if the SOG changes, retaining the starting and ending points is sufficient to accurately capture the distribution characteristics of the trajectory during compression tasks. Conversely, if a trajectory point quickly deviates from the baseline in a short period, it is highly likely to be a feature point. To address this, the approach taken in this paper involves establishing a coordinate system using the trajectory baseline as the x-axis. This system decomposes the velocity of each trajectory point, as depicted visually in Fig. 8 (c). A substantial change rate of the SOG component perpendicular to the x-axis between two points suggests a rapid deviation from the baseline. It is worth noting that COG assists SOG in its decomposition within this novel coordinate system.

According to the above analysis, the deviation of each intermediate trajectory point from the baseline and the change rate of SOG along the y-axis in the new coordinate system jointly determine whether it qualifies as a feature point. Thus, this paper combines these two aspects of information to calculate the compression threshold, as illustrated in Fig. 9. The core concept involves determining the distance ratio from each intermediate point to the baseline concerning the sum of all distances. Concurrently, the proportion of the change rate of SOG along the y-axis component for each intermediate point is calculated based on the overall change rate. The contribution value of the point to the compression threshold can be obtained by fusing the above two proportional weights and multiplying them with the distance to the baseline. The specific steps of the compression threshold computation are outlined as follows,

Step 1. *Offset weight calculation.* As shown in Fig. 9 (a), this step utilizes Eq. (13) to calculate the distance between each intermediate point and the baseline, which is then stored in vector DV. Furthermore, Eq. (14) is used to obtain a vector RV containing the proportion of each intermediate point's distance to the baseline in relation to the total.

Step 2. *SOG variation weight calculation.* Following Fig. 9 (b), the SOG of each trajectory point is decomposed in a new coordinate system, which is established with the trajectory baseline as the x-axis. In particular, this paper exclusively selects the SOG component along the y-axis to compute the compression threshold because the speed at which a trajectory point deviates from the baseline to some extent determines whether the point is a feature point. The decomposition of SOG requires the assistance of COG. However, it is unfeasible to employ the original COG directly because COG needs to be converted into the new coordinate system. The new coordinate system can be obtained by rotating the old coordinate system clockwise or counterclockwise by a certain degree α ($0 \leqslant \alpha \leqslant 180$) around the trajectory point. The original COG add or subtract α can get the heading value in the new coordinate system. Hence, it is a pivotal step to calculate the angle between the trajectory baseline and the x-axis of the original coordinate system. The details of the calculation process are thoroughly outlined in Algorithm 1.

Algorithm 1: Calculation of included angle

Input: lon_S, lat_S, lon_E, lat_E // (lon_S, lat_S) and (lon_E, lat_E) are the coordinates of the starting and ending points for the trajectory, respectively.
Output: α
1. **if** $lon_E == lon_S$ **then** $\alpha = 90$;
2. **else if** $lat_E == lat_S$ **then** $\alpha = 0$;
3. **else**
4. $\alpha = \arctan\left(\frac{lat_E - lat_S}{lon_E - lon_S}\right) \times \frac{180}{\pi}$;
5. **if** $\alpha < 0$ **then** $\alpha = 180 + \alpha$;
6. **end if**
7. **end if**

For ease of calculation, this paper uniformly establishes that the new coordinate system is achieved by rotating the existing coordinate system counterclockwise by α degrees. The components of SOG along the y-axis in the new coordinate system can be derived based on Algorithm 2.

Algorithm 2: Calculation of SOG components along the y-axis

Input: SOG, COG, α // The definition of COG in the original coordinate system is the angle between it and the positive half-axis of the y-axis, and its value range is [0, 360).
Output: $SOGY$
1. **if** $0 \leqslant \alpha \leqslant 90$ **then**
2. **if** $COG == 0$ **then** $SOGY = SOG \times \sin\left(|90 - \alpha| \times \frac{\pi}{180}\right)$;

(*continued on next page*)

(*continued*)

Algorithm 2: Calculation of SOG components along the *y*-axis

else if $0 < COG \le 90$ **then** $SOGY = SOG \times \sin\left(((90 - COG) - \alpha) \times \frac{\pi}{180}\right)$;
else if $90 < COG \le 180$ **then** $SOGY = -SOG \times \sin\left(((COG - 90) + \alpha) \times \frac{\pi}{180}\right)$;
else if $180 < COG < 270$ **then**
  $cogConvert = 270 - COG$;
  if $cogConvert == \alpha$ **then** $SOGY = 0$;
  else if $cogConvert > \alpha$ **then**
    $SOGY = -SOG \times \sin\left((cogConvert - \alpha) \times \frac{\pi}{180}\right)$;
  else if $cogConvert < \alpha$ **then**
    $SOGY = SOG \times \sin\left((\alpha - cogConvert) \times \frac{\pi}{180}\right)$;
 end if
 else if $COG == 270$ **then** $SOGY = SOG \times \sin\left(\alpha \times \frac{\pi}{180}\right)$;
 else if $270 < COG < 360$ **then** $SOGY = SOG \times \sin\left(((COG - 270) + \alpha) \times \frac{\pi}{180}\right)$;
 end if
else if $90 < \alpha < 180$ **then**
 if $COG == 0$ **then** $SOGY = SOG \times \sin\left((90 - (180 - \alpha)) \times \frac{\pi}{180}\right)$;
 else if $0 < COG \le 90$ **then**
  $SOGY = SOG \times \sin\left(((90 - COG) + (180 - \alpha)) \times \frac{\pi}{180}\right)$;
 else if $90 < COG \le 180$ **then**
  $cogConvert = COG - 90$;
  if $cogConvert == (180 - \alpha)$ **then**
   $SOGY = 0$;
  else if $cogConvert < (180 - \alpha)$ **then**
   $SOGY = SOG \times \sin\left(((180 - \alpha) - cogConvert) \times \frac{\pi}{180}\right)$;
  else if $cogConvert > (180 - \alpha)$ **then**
   $SOGY = -SOG \times \sin\left((cogConvert - (180 - \alpha)) \times \frac{\pi}{180}\right)$;
  end if
 else if $180 < COG < 270$ **then**
  $SOGY = -SOG \times \sin\left(((180 - \alpha) + (90 - (270 - COG))) \times \frac{\pi}{180}\right)$;
 else if $COG == 270$ **then** $SOGY = -SOG \times \sin\left((180 - \alpha) \times \frac{\pi}{180}\right)$;
 else if $270 < COG < 360$ **then**
  $cogConvert = COG - 270$;
  if $cogConvert == (180 - \alpha)$ **then**
   $SOGY = 0$;
  else if $cogConvert < (180 - \alpha)$ **then**
   $SOGY = -SOG \times \sin\left(((180 - \alpha) - cogConvert) \times \frac{\pi}{180}\right)$;
  else if $cogConvert > (180 - \alpha)$ **then**
   $SOGY = SOG \times \sin\left((cogConvert - (180 - \alpha)) \times \frac{\pi}{180}\right)$;
  end if
 end if
end if

According to Algorithm 2, the SOG component along the *y*-axis for each trajectory point can be computed and stored in the vector $SOGYV = \{SOGY_1, SOGY_2, \ldots, SOGY_{lt}\}$. To determine the SOG change rate of each trajectory point along the *y*-axis, the timestamp of each trajectory point is retained in vector $TV = \{t_1, t_2, \ldots, t_{lt}\}$. In particular, the time information in AIS data is expressed in hours, minutes, and seconds. To facilitate calculation, this paper adopts Eq. (16) to convert time into seconds uniformly as follows,

$$t = 3600 \times dataH + 60 \times dataM + dataS \tag{16}$$

where *dataH*, *dataM*, and *dataS* represent the hour, minute, and second in the timestamp, respectively. The velocity change rate along the *y*-axis at each intermediate point can be obtained based on vectors *SOGYV* and *TV* below.

$$sogyC_{i+1} = \frac{SOGY_{i+1} - SOGY_i}{t_{i+1} - t_i}, i = 1, 2, \ldots, lt - 2 \tag{17}$$

where *i* represents the *i*-th value in two vectors. *lt* denotes the total number of points in the trajectory. The velocity change rate component at each intermediate point is stored in the vector $SOGYCV = \{sogyC_2, sogyC_3, \ldots, sogyC_{lt\text{-}1}\}$.

In addition, the proportion of the change rate of SOG along the *y*-axis component for each intermediate point to the total change rate is determined by Eq. (18) and expressed as a vector $SOGYCRV = \{sogyCR_2, sogyCR_3, \ldots, sogyCR_{lt\text{-}1}\}$.

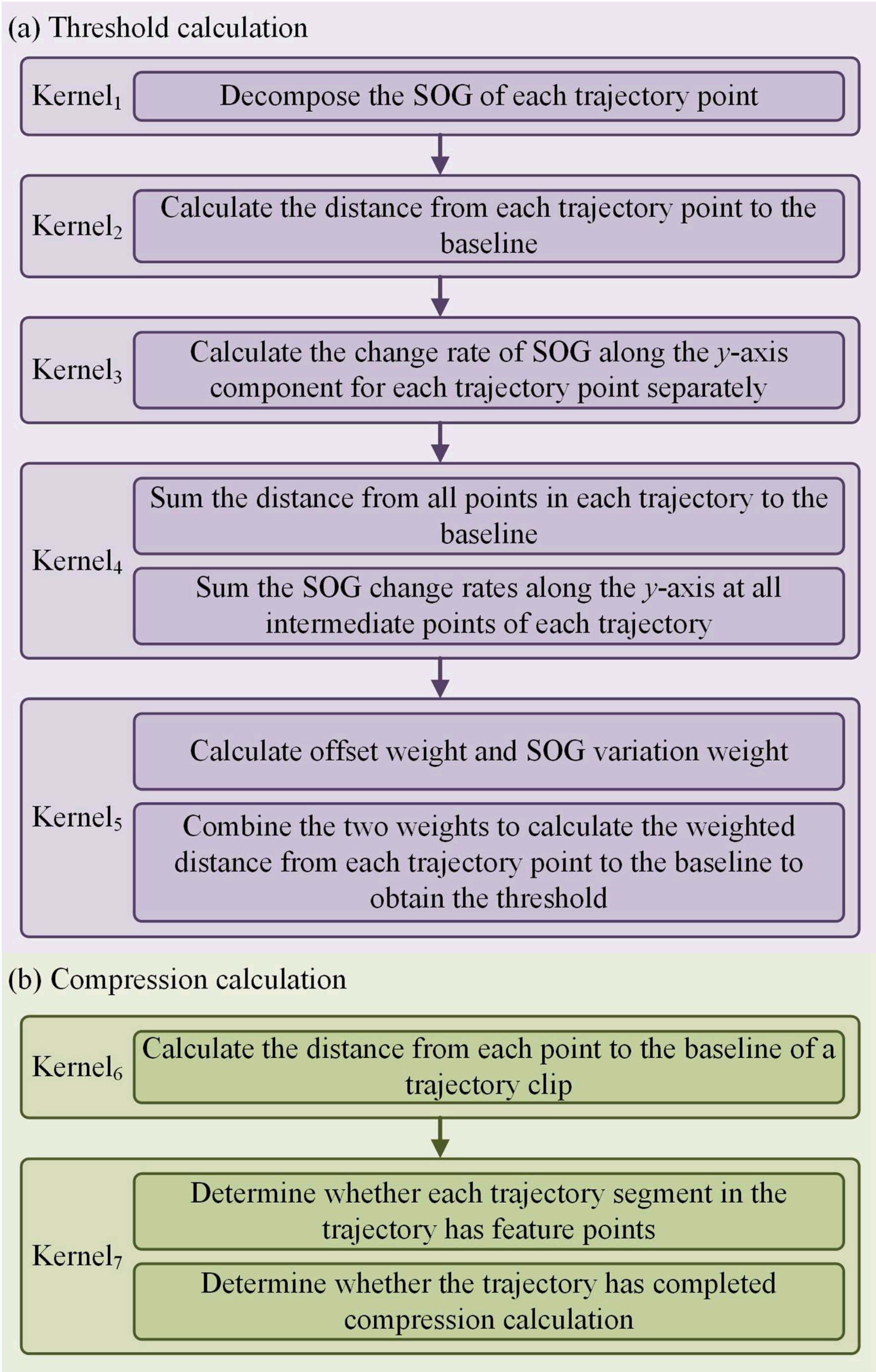


Fig. 10. The flowchart of the proposed GPU-based ADPSC parallel implementation framework: (a) threshold parallel computing process; (b) compression parallel computing processes.

Table 4
The list of notations.

Notations	Definition
Ltpr	A vector is used to match the relationship between points and trajectories.
Lntp	A vector is used to store the number of points in each trajectory.
sogDecL	A vector stores the SOG decomposition results of each trajectory point.
thd	The thread number.
posS	The starting position of the trajectory where the current point being processed by the current thread is located.
posE	The ending position of the trajectory where the current point being processed by the current thread is located.
lonS	Longitude of the starting point.
latS	Latitude of the starting point.
lonE	Longitude of the ending point.
latE	Latitude of the ending point.
lonData	Longitude of the current point.
latData	Latitude of the current point.
sogData	SOG of the current point.
cogData	COG of the current point.
DisL	A vector stores the distance from each trajectory point to the baseline.
disData	The distance from the intermediate point to the trajectory baseline.
posData	Calculate where the *disData* is stored in the vector *DisL*.
length	The length of the vector *DisL*.
poNum	The total number of trajectory points in the dataset.
traNum	The number of vessel trajectories in the dataset.
sogyCL	A vector stores the speed change rate along the y-axis at the intermediate point of each trajectory.
timeP	Time of the previous trajectory point.
timeC	Time of the current trajectory point.
sogyP	SOG component of the previous trajectory point.
sogyC	SOG component of the current trajectory point.
changeR	The rate of change of SOG components.
sumDisL	A vector stores the sum of the distances from all intermediate points of each trajectory to the baseline.
sumSogyCL	A vector stores the sum of the SOG change rates of all intermediate points of each trajectory along the y-axis.
posDataS *posDataE*	Two variables are used to calculate the boundary positions of two vectors (i.e., *DisL* and *sogyCL*) to match the relationship between the data and the trajectory.
thrL	A vector stores the compression threshold for each trajectory.
Lfp	A vector is used to determine which points are the feature points calculated by the ADPSC algorithm.
leftPos *rightPos*	The positions of the starting and ending points of a certain trajectory segment in *AISM*.
labelData	Calculate the value corresponding to the trajectory point in the vector *Lfp*.
lntpData	A variable is used to determine whether a certain trajectory has completed the compression task.
ifFeature	A variable is used to determine whether new feature points are generated in an iterative search. Its initial value is zero. When it becomes one during the calculation process, it indicates that new feature points have been generated in the iterative search.
threshold	The compression threshold for vessel trajectory.
num	A variable is used to assist in searching for feature points in trajectory segments.
begin *end*	Two variables are used to define the positions of the starting and ending points for trajectory segments, respectively.
maxDisData	A variable is defined as the maximum value from a point to the baseline of a trajectory segment.

$$Sum = \sum_{2}^{lt-1} sogyC_i,$$
$$sogyCR_i = \frac{sogyC_i}{Sum}, i = 2, 3, ..., lt - 1 \tag{18}$$

Step 3. *Weight fusion calculation.* The objective is to integrate the offset weights calculated in Steps 1 and 2 with the SOG change weights. The contribution value of each intermediate point to the threshold, essentially a weighted result, is determined by multiplying its fusion weight with the distance from the baseline. In particular, if a certain intermediate point significantly contributes to the threshold, the calculated fusion weight value will be higher. The final threshold is achieved by summing the contribution values of all intermediate points, as expressed by the following function.

$$threshold = \sum_{i=2}^{lt-1} dis_i \times sogyCR_i \times rate_i \tag{19}$$

where dis_i denotes the distance from the i-th trajectory point to the baseline. $sogyCR_i$ is the change weight of the i-th trajectory point along the y-axis SOG component. $rate_i$ represents the offset weight of the i-th trajectory point relative to the baseline.

The purpose of multiplying two weights is to exclude the contribution of a factor if it evaluates to zero during threshold calculation. In such cases, the distance from that particular point to the baseline does not play a role in the threshold calculation. The ADPSC algorithm employs this calculated threshold in iterative searching for feature points, thereby realizing the trajectory compression task.

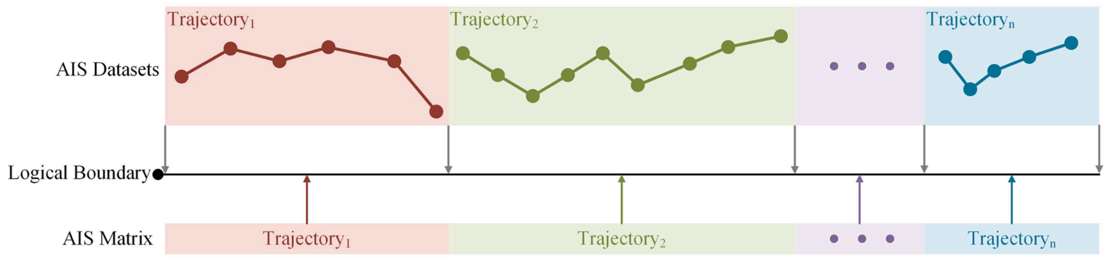


Fig. 11. Coalesced global memory access of AIS data.

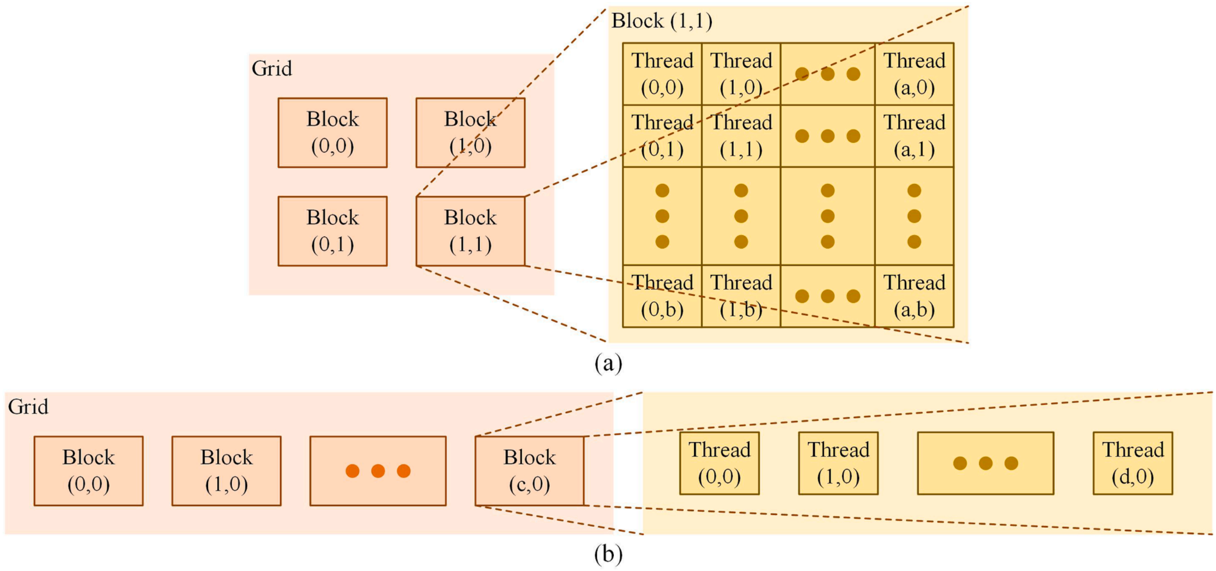


Fig. 12. Visual comparison of trajectory data (or vector) and images (or matrix) in parallel computing framework design. (a) and (b) depict the thread distribution architecture for parallel computing in image and trajectory data, respectively.

4.3. GPU-accelerated ADPSC compression method

The ADPSC algorithm proposed in this paper entails an extended compression threshold calculation process compared to the original DP algorithm. It means that compressing massive vessel trajectory data demands more time. To align with real-world applications, this paper enhances the ADPSC algorithm for large-scale trajectory data compression within the GPU parallel computing framework, substantially diminishing execution time. The ADPSC algorithm predominantly encompasses two calculation processes during trajectory data compression: threshold and compression calculation. Accordingly, the designed parallel algorithm is structured around these two calculation components, as illustrated in Fig. 10. The threshold and compression calculation in the parallel ADPSC algorithm consists of multiple different functional functions, respectively. In the programming framework of CUDA, each function is encapsulated into a kernel, constituting a CUDA parallel computing function executed on the GPU. Each thread within the GPU conducts the kernel function in parallel to process the trajectory points to accomplish compression. Specifically, the operational logic of the kernel functions numbered 1 to 7 in Fig. 10 follows a progressive rather than a random execution order. The necessary notations are listed in Table 4.

When executing parallel algorithms on a GPU, it cannot directly obtain data from memory and needs to transfer the data from memory to video memory. This transfer process is time-consuming. If vessel trajectories in the dataset are compressed one by one in sequence, although it might simplify the design of parallel algorithms, it would still demand substantial computational time. To tackle this challenge, this paper uniformly stores all trajectory data (i.e., time stamp, longitude, latitude, SOG, and COG) in a matrix as represented in Eq. (5). This approach allows copying all necessary data from memory to video memory in a single operation, as displayed in Fig. 11. This helps avoid frequent data copying between memory and video memory, thereby reducing computational costs. In practical parallel computing, it is necessary to set a label vector to determine whether a trajectory point belongs to a specific vessel. Sections 4.3.1 and 4.3.2 explain the threshold and compression parallel computing processes within the ADPSC algorithm, respectively.

4.3.1. Parallelization of the threshold calculation

Each point in the vessel trajectory is discretely distributed, and they exhibit independence from one another. This feature provides convenience for processing each trajectory point by using parallel compression algorithms. Each trajectory point can be systematically linked to threads within the GPU, with each thread executing the compression task by activating the kernel function. GPUs are renowned for their potent computational capabilities, often employed in image processing (or matrix calculation). Parallel algorithms devised for image processing adapt the distribution of threads in GPUs based on the matrix size, as depicted in Fig. 12 (a). The thread arrangement in the parallel algorithm of trajectory data differs from that of image processing, assuming a vector-oriented layout illustrated in Fig. 12 (b). In particular, the total thread count equals the sum of all points in the vessel trajectory dataset.

According to Fig. 10 (a), the parallel calculation of the threshold mainly consists of five functions, which are encapsulated within five kernels. Each thread executes these five kernel functions in turn to obtain the compression threshold of each vessel trajectory. The parallel computing process of these five kernel functions in each thread is as follows:

(1) *Kernel*$_1$. Its purpose is to calculate the SOG components decomposed along the *y*-axis for each trajectory point in the new coordinate system. While all trajectory points are distributed in each thread of the GPU for parallel computing tasks, this presents two challenges. One is the inability to discern the relationship between trajectories and their respective points. Another issue is the inability to match the starting and ending points of trajectories.

To address these two challenges, this paper sets up two label vectors, *Ltpr* and *Lntp*. They are designed to determine the relationship between points and trajectories and identify the starting and ending points in trajectories. For illustration, suppose there are three vessel trajectories in a dataset with track point counts of 4, 5, and 6, respectively. The values of vectors *Ltpr* and *Lntp* would be [1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3] and [0, 4, 9, 15], respectively. The length of vector *Ltpr* is the total number of all trajectory points in the dataset, with each value indicating the specific trajectory to which a point belongs. The value in vector *Lntp* is the accumulation of the number of points in each trajectory. According to the above example analysis, the starting and ending points of the first trajectory are in the 0th and (4–1)th threads.

Based on Algorithm 3, the parallel calculation results of SOG decomposition along the *y*-axis for each trajectory point can be obtained. In particular, the function 'cauSogDecompose' integrates the abilities of Algorithm 1 and Algorithm 2.

Algorithm 3: Parallel computing of SOG decomposition

Input: *AISM*, *Ltpr*, *Lntp, sogDecL* // According to Eq. (5), the first to fifth rows of the *AISM* matrix represent time, longitude, latitude, SOG, and COG, respectively. *sogDecL* is the initialization vector, whose length equals the total number of trajectory points in the dataset. All values in the vector *sogDecL* are 0.
Output: *sogDecL* // A vector stores the SOG decomposition results of each trajectory point.

```
thd = blockIdx.x × blockDim.x + threadIdx.x; // thread number.
posS = Lntp[Ltpr[thd] – 1]; // The starting position of the trajectory where the current point being processed by the current thread is located.
posE = Lntp[Ltpr[thd]] – 1; // The ending position of the trajectory where the current point being processed by the current thread is located.
lonS = AISM[1,:][posS]; // Longitude of starting point.
latS = AISM[2,:][posS]; // Latitude of starting point.
lonE = AISM[1,:][posE]; // Longitude of ending point.
latE = AISM[2,:][posE]; // Latitude of ending point.
sogData = AISM[3,:][thd];
cogData = AISM[4,:][thd];
sogDecL[thd] = cauSogDecompose(lonS, latS, lonE, latE, sogData, cogData);
```

(2) *Kernel*$_2$. This kernel function is used to calculate the distance between each trajectory point and the baseline. During the execution of parallel computing, vectors *Ltpr* and *Lntp* remain essential to determine the relationship between points and trajectories, as well as to identify the starting and ending points. The parallel calculation process is described in Algorithm 4.

Algorithm 4: Parallel computing of distance from trajectory point to baseline

Input: *AISM*, *Ltpr*, *Lntp*, *DisL* // *DisL* is the initialization vector whose all values are 0.
Output: *DisL* // A vector stores the distance from each trajectory point to the baseline.

```
thd = blockIdx.x × blockDim.x + threadIdx.x;
posS = Lntp[Ltpr[thd] – 1];
posE = Lntp[Ltpr[thd]] – 1;
if thd!= posS && thd!= posE then // The starting and ending points in each trajectory do not participate in the calculation.
  lonS = AISM[1,:][posS];
  latS = AISM[2,:][posS];
  lonE = AISM[1,:][posE];
  latE = AISM[2,:][posE];
  lonData = AISM[1,:][thd]; // Longitude of current trajectory point.
  latData = AISM[2,:][thd]; // Latitude of current trajectory point.
  disData = cauDistance(lonS, latS, lonE, latE, lonData, latData);
  posData = thd – (Ltpr[thd] × 2 – 1); // Calculate where the data (or distance) is stored in the vector DisL.
  DisL[posData] = disData;
end if
```

Algorithm 4 reflects the parallel calculation process of the point-to-baseline distance for each trajectory. The function 'cauDistance' integrates the ability of Eq. (13). The length of the vector *DisL* is not equal to the total number of all trajectory points in the dataset. This discrepancy arises primarily because the distance calculation from the trajectory point to the baseline excludes both the starting and ending points. The formula for calculating the length of the vector *DisL* is,

$$length = poNum - 2 \times traNum \tag{20}$$

where *poNum* represents the total number of trajectory points in the dataset, and *traNum* denotes the number of trajectories.

(3) $Kernel_3$. Its core idea is to calculate the SOG change rate of the intermediate point of each trajectory in the *y*-axis direction. The results of the first kernel function (*sogDecL*) are utilized in the calculation process. A detailed description of the parallel computing method is shown in Algorithm 5.

Algorithm 5: SOG change rate along the *y*-axis component

Input: *AISM*, *Ltpr*, *Lntp*, *sogDecL*, *sogyCL* // *sogyCL* is the initialization vector whose all values are 0.
Output: *sogyCL* // A vector stores the speed change rate along the *y*-axis at the intermediate point of each trajectory.
1. $thd = blockIdx.x \times blockDim.x + threadIdx.x$;
2. $posS = Lntp[Ltpr[thd] - 1]$;
3. $posE = Lntp[Ltpr[thd]] - 1$;
4. **if** *thd*!= *posS* && *thd*!= *posE* **then** // The starting and ending points in each trajectory do not participate in the calculation.
5. $timeP = AISM[0,:][thd - 1]$; // Time of the previous trajectory point.
6. $timeC = AISM[0,:][thd]$; // Time of the current trajectory point.
7. $sogyP = sogDecL[thd - 1]$; // SOG component of the previous trajectory point.
8. $sogyC = sogDecL[thd]$; // SOG component of the current trajectory point.
9. $changeR = \frac{|sogyC - sogyP|}{timeC - timeP}$;
10. $posData = thd - (Ltpr[thd] \times 2 - 1)$; // Calculate where the data (or SOG component change rate) is stored in the vector *sogyCL*.
11. $sogyCL[posData] = changeR$;
12. **end if**

Based on Algorithm 5, the SOG change rate along the *y*-axis at all intermediate points of each trajectory can be quickly calculated. Each trajectory's starting and ending points do not participate in calculating the SOG change rate, so the length of vector *sogyCL* is the same as that of vector *DisL*.

(4) $Kernel_4$. This kernel serves two primary purposes. Firstly, it calculates the cumulative distance of all intermediate points of each trajectory from the baseline. Secondly, it determines the collective SOG change rates along the *y*-axis for all intermediate points within each trajectory. Unlike previous kernel functions, where the focus was individual points, this kernel targets entire trajectories, with each trajectory in the dataset being sequentially mapped to the GPU threads. The parallel computation methodology is detailed in Algorithm 6.

Algorithm 6: Calculate the sum of distance and SOG component change rate separately

---

Input: *DisL*, *sogyCL*, *Lntp*, *sumDisL*, *sumSogyCL* // *sumDisL* and *sumSogyCL* are the initialization vectors whose all values are 0.
Output: *sumDisL*, *sumSogyCL* // Two vectors store the sum of the distances from all intermediate points of each trajectory to the baseline and the sum of the SOG change rates of all intermediate points along the *y*-axis, respectively.
1. $thd = blockIdx.x \times blockDim.x + threadIdx.x$;
2. $posDataS = Lntp[thd] - 2 \times thd$;
3. $posDataE = Lntp[thd + 1] - (2 \times thd + 2)$; // Steps 2 and 3 calculate the boundary positions of two vectors (i,e., *DisL* and *sogyCL*) to match the relationship between the data and the trajectory.
4. **for** $i = posDataS$: $posDataE$ **do**
5. $sumDisL[thd] = sumDisL[thd] + DisL[i]$;
6. $sumSogyCL[thd] = sumSogyCL[thd] + sogyCL[i]$;
7. **end for**

Algorithm 6 performs parallel computing tasks based on trajectories. Therefore, the lengths of vectors *sumDisL* and *sumSogyCL* equal the total number of vessel trajectories in the dataset.

(5) $Kernel_5$. This kernel function calculates the offset weight and SOG component variation rate weight of trajectory points according to Eqs. (14) and (18), respectively. It then integrates these two weights, using Eq. (19), to determine the weighted distance of each trajectory point from the baseline, yielding the threshold. This kernel is the same as the fourth kernel because the object processed by each thread is the vessel trajectory. The detailed calculation process is presented in Algorithm 7.

Algorithm 7: Parallel computing of compression threshold

Input: *DisL*, *sogyCL*, *Lntp*, *sumDisL*, *sumSogyCL*, *thrL* // *thrL* is the initialization vector whose all values are 0.
Output: *thrL* // A vector stores the compression threshold for each trajectory.
1. $thd = blockIdx.x \times blockDim.x + threadIdx.x$;
2. $posDataS = Lntp[thd] - 2 \times thd$;
3. $posDataE = Lntp[thd + 1] - (2 \times thd + 2)$; // The functions of steps 2 and 3 are consistent with those in Algorithm 6.
4. **for** i = posDataS: posDataE **do**

(continued on next page)

(*continued*)

Algorithm 7: Parallel computing of compression threshold
5. $thrL[thd] = thrL[thd] + DisL[i] \times \frac{sogyCL[i]}{sumSogyCL[thd]} \times \frac{DisL[i]}{sumDisL[thd]}$; // Refer to Eq. (19).
6. **end for**

Algorithm 7, similar to Algorithm 6, executes parallel computing tasks based on vessel trajectories. As a result, the size of vector ***thrL*** corresponds to the total number of trajectories in the dataset. Following the parallel computing process mentioned above, the compression threshold for each vessel trajectory is determined. This underpins the parallel compression task for vessel trajectory data discussed in Section 4.3.2.

4.3.2. Parallelization of compression calculation

The ADPSC algorithm utilizes the calculated compression threshold to iteratively search for trajectory feature points. It uses these feature points to form a new data sequence to replace the original trajectory, thereby completing the compression task. According to the execution principle of the compression algorithm, the parallel process of compressed computing mainly consists of two parts, shown in Fig. 10 (b). On the one hand, it calculates the distance from each trajectory point to the baseline of the matching trajectory segment. This calculation process is dynamic, as the newly generated feature points will repartition the trajectory. On the other hand, it compares the threshold with the distance from the trajectory point to the baseline to determine the feature points. Meanwhile, a condition must be set to terminate the calculation process to represent that the trajectory has been compressed. The reason is that the compression algorithm is a recursive process requiring a constraint to end the operation. According to Fig. 10 (b), the parallel calculation of the compression mainly consists of two kernel functions. Each thread executes these two kernel functions in sequence to obtain compressed trajectories. The parallel computing process of these two kernel functions in each thread is as follows,

(1) $Kernel_6$. Its purpose is to calculate the distance from each trajectory point to the baseline of the matching trajectory segment and store the results in the vector *DisL*. The length of this vector is equal to the total number of all trajectory points in the dataset, whose all values are 0 as the initialization state. During each iteration of searching for new feature points, a portion of the values in *DisL* is constantly changing. The reason is that each trajectory generates new feature points that divide it into multiple different trajectory segments.

In the specific parallel computing process, four problems must be addressed to complete parallel computing tasks successfully. The initial challenge concerns matching the relationship between points and trajectories. In the parallel algorithm, each trajectory point is assigned to a separate thread for operation, yet the GPU cannot identify which trajectory a point belongs to. This paper solves this issue by setting a label vector ***Lfp***, where each value in the vector can match the logical relationship between each point and the trajectory. For example, suppose there are three trajectories in the dataset. Each trajectory has 4, 5, and 6 points, respectively. The values of vector ***Lfp*** are [1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3]. The second issue is to determine which points are the feature points calculated by the algorithm. The vector ***Lfp*** is utilized to solve this issue. If a point is a feature point of the corresponding trajectory, then its corresponding value in vector ***Lfp*** is negative. Before the compression algorithm executes each trajectory, the starting and ending points are their initialization feature points. Hence, the values of label vector ***Lfp*** are [-1, 1, 1, -1, -2, 2, 2, 2, -2, -3, 3, 3, 3, 3, -3] as an initial state. The third issue is to pinpoint the position of each trajectory's starting and ending points in *AISM*. To address the problem effectively, this paper sets up another label vector ***Lntp***, whose value accumulates the number of points in each trajectory. Based on the above example, the values of vector ***Lntp*** is [0, 4, 9, 15]. The fourth issue is to identify whether a certain trajectory has completed the compression task and does not need to execute this kernel function. The vector ***Lntp*** will also assist in solving this problem. For instance, if the second trajectory mentioned above has already completed the compression task, and the other two trajectories still need to continue searching for new feature points. The values of vector ***Lntp*** is [0, 4, -9, 15]. The detailed calculation process will be introduced in the seventh kernel function. In summary, these four vectors will aid parallel algorithms in calculating the distance from the trajectory point to the baseline and then comparing it with the threshold to search for feature points iteratively.

Each trajectory will generate new feature points during continuous iteration, which will repartition the trajectory and form different trajectory segments. Different trajectory segments in each trajectory have their corresponding starting and ending points, which cannot be identified using the vector *Lntp*. The reason is that the vector *Lntp* can only recognize the positions of the starting and ending points in a trajectory and assist in determining whether the trajectory has completed the compression task. To calculate the baseline distance from each point to the corresponding trajectory segment, this paper uses double pointers to find the starting and ending points of the trajectory segment corresponding to each point, as shown in Algorithm 8.

Algorithm 8: Double-pointer search for starting and ending points
Input: *Lfp*, *thd* // *thd* denotes the thread number and also represents the *thd*-th trajectory point in the dataset that the current thread is processing.
Output: *leftPos, rightPos* // *leftPos* and *rightPos*, respectively, represent the positions of the starting and ending points of a certain trajectory segment in *AISM*.
1. $leftPos = thd - 1$;
2. $rightPos = thd + 1$; // *leftPos* and *rightPos* initialization.
3. $labelData = Lfp[thd]$; // Calculate the value corresponding to the trajectory point in the vector *Lfp*.
4. **while** $Lfp[leftPos] != -labelData \mid\mid Lfp[rightPos] != -labelData$ **do** // When *labelData* is negative, it indicates that the position of the pointer is at the boundary of the trajectory segment (starting or ending point).
5. **if** $Lfp[leftPos] != -labelData$ **then** $leftPos = leftPos - 1$;

(continued on next page)

(*continued*)

Algorithm 8: Double-pointer search for starting and ending points

6. **end if**
7. **if** *Lfp*[*rightPos*]!= −*labelData* **then** *rightPos* = *rightPos* – 1;
8. **end if**
9. **end while**

According to the results of Algorithm 8, this paper can continue to calculate the distance from each trajectory point to the corresponding trajectory segment baseline in parallel, as shown in Algorithm 9.

Algorithm 9: Computing of distance from points to the trajectory segment baseline

Input: *AISM*, *Lfp*, *Lntp*, *DisL*
Output: *DisL*
1. *thd* = *blockIdx.x* × *blockDim.x* + *threadIdx.x*;
2. *lntpData* = *Lntp*[abs(*Lfp*[*thd*])];
3. **if** *lntpData* > 0 **then** // When *lntpData* is a positive number, it indicates that there are still feature points in the current trajectory, and it is necessary to continue calculating the distance between the intermediate point and the corresponding trajectory segment baseline.
4. *labelData* = *Lfp*[*thd*];
5. **if** *labelData* > 0 **then**
6. *leftPos*, *rightPos* = cauStartEndPos(*Lfp*, *thd*);
7. *lonS* = *AISM*[1,:][*leftPos*]; // Longitude of starting point.
8. *latS* = *AISM*[2,:][*leftPos*]; // Latitude of starting point.
9. *lonE* = *AISM*[1,:][*rightPos*]; // Longitude of ending point.
10. *latE* = *AISM*[2,:][*rightPos*]; // Latitude of ending point.
11. *lonData* = *AISM*[1,:][*thd*]; // Longitude of current trajectory point.
12. *latData* = *AISM*[2,:][*thd*]; // Latitude of current trajectory point.
13. *DisL*[*thd*] = cauDistance(*lonS*, *latS*, *lonE*, *latE*, *lonData*, *latData*);
14. **else if** *labelData* < 0 **then**
15. *DisL*[*thd*] = 0; // When *labelData* is negative, it indicates that the current point is a feature point and the distance from the baseline is zero.
16. **end if**
17. **end if**

Algorithm 9 reflects the detailed process of computing the distance between each trajectory point and the baseline of the corresponding trajectory segment in parallel. In particular, the functions 'cauStartEndPos' and 'cauDistance' respectively integrate the capabilities of Eq. (13) and Algorithm 8.

(2) Kernel_7. This kernel function plays two essential roles in parallel computing. One is to judge whether each trajectory segment has new feature points according to the results of the sixth kernel function. If a new feature point is generated, the value matching the point with the vector *Lfp* is changed from a positive number to a negative number. Another function is determining whether a certain trajectory has completed the compression task. When a certain trajectory has completed the compression task, the corresponding value in vector *Lntp* changes from positive to negative. Since the object of this kernel function is a trajectory, the threads in the GPU are mapped to each trajectory in the dataset one by one, which is different from the sixth kernel function. The parallel calculation process is shown in Algorithm 10.

Algorithm 10: Iterative search for feature points

Input: *Lfp*, *Lntp*, *DisL*, *thrL* // Vector *thrL* stores the compression threshold for each vessel trajectory.
Output: *Lfp*, *Lntp*
1. *thd* = *blockIdx.x* × *blockDim.x* + *threadIdx.x*;
2. *lntpData* = *Lntp*[*thd* + 1];
3. **if** *lntpData* > 0 **then** // When *lntpData* is a positive number, it indicates that there are still feature points in the current trajectory which require further iterative search.
4. *ifFeature* = 0; // A variable used to determine whether new feature points are generated in iterative search. Its initial value is zero. When it becomes one during the calculation process, it indicates that new feature points have been generated in the iterative search.
5. *threshold* = *thrL*[*thd*]; // The compression threshold of the current trajectory.
6. *num* = 0; // It assists in searching for feature points in trajectory segments with an initial value of zero.
7. *begin* = 0;
8. *end* = 0; // *begin* and *end* are used to define the positions of the starting and ending points for trajectory segments, respectively. Their initial values are zero.
9. *maxDisData* = 0; // This variable defines the maximum value from a point to the baseline of a trajectory segment. Its initial value is zero.
10. **for** (*i* = *Lntp*[*thd* + 1] – 1; *i* > abs(*Lntp*[*thd*]) – 1; *i*–) **do** // Traverse search for feature points for each trajectory segment.
11. *disData* = *DisL*[*i*];
12. **if** *disData* == 0 && *num* == 0 **then** // Locate the position of the ending point for a trajectory segment.
13. *num* = *num* + 1;
14. *end* = *i*;
15. **else if** *disData*!= 0 && *num* == 1 **then** // Calculate the maximum distance from all intermediate points to the baseline of the trajectory segment.
16. **if** *disData* > *maxDisData* || *disData* == *maxDisData* **then**

(*continued on next page*)

(*continued*)

Algorithm 10: Iterative search for feature points

```
17.           maxDisData = disData;
18.         end if
19.     else if disData == 0 && num == 1 then // Locate the position of the starting point for a trajectory segment.
20.         if maxDisData > threshold || maxDisData == threshold then // Compare the maximum distance value with the threshold to determine if there are new
      feature points and update the variable ifFeature.
21.           ifFeature = 1;
22.           begin = i;
23.           for (j = begin + 1; j < end; j++) do // Determine the position of feature points and change the value corresponding to vector Lfp from positive to
      negative.
24.         if DisL[j] == maxDisData then
25.             labelData = Lfp[j];
26.             Lfp[j] = -labelData;
27.         end if
28.     end for
29.   end if
30.   num = 0;
31.     if disData == 0 && num == 0 then // After completing the feature point search task of the current trajectory segment, redefine the variables num,
      maxDisData and end to search for the feature point of the next trajectory segment.
32.         num = num + 1;
33.         maxDisData = 0;
34.         end = i;
35.     end if
36.   end if
37. end for
38. if ifFeature == 0 then Lntp[thd + 1] = -lntpData; // No new feature points are generated in the iterative search, indicating that the trajectory has completed the
      compression task. Meanwhile, the value in the vector Lntp that matches the current trajectory changes from positive to negative.
39. end if
40. end if
```

In executing parallel algorithms, each thread performs the sixth and seventh kernel functions repeatedly, which reflects that the algorithm is essentially a recursive process. The condition for recursive termination is that all values in the vector *Lntp* are negative. *DisL*, *Lfp*, and *Lntp* are three important vectors that assist parallel algorithms in finding feature points of trajectories through continuous iterations. The vectors *DisL* and *Lfp* length equal the total number of trajectory points. The size of vector *Lntp* is the total number of trajectories plus one. Before executing parallel compression tasks, three vectors need to be assigned initialization values, as illustrated in Fig. 13 (a). The sixth kernel function calculates the distance from all intermediate points to the baseline in parallel and updates the value in the vector *DisL* to store the distance, as shown in Fig. 13 (b). Figs. 13 (c), (d), and (e) reflect the process of two kernel functions iteratively searching for feature points and the changes of all values in the three vectors. When the vessel trajectories meet the termination conditions of the iteration, the final compression result is obtained, as illustrated in Fig. 13 (f). In particular, the trajectory points corresponding to negative values in vector *Lfp* are feature points. The changes of all values in the two label vectors before and after trajectories compression are shown in Figs. 14 (a) and (b), respectively.

5. Experimental results and analysis

To verify the effectiveness of the proposed adaptive and accelerated compression framework, this paper conducts experiments from two perspectives. First, it showcases the superior performance of the ADPSC algorithm in addressing vessel trajectory compression issues compared to the original DP algorithm, considering both qualitative and quantitative perspectives. Second, the acceleration ratio is employed to quantitatively evaluate the enhanced speed of the optimized parallel compression algorithm compared to the original serial algorithm. Concurrently, a GPU parallel computing framework is established as a requisite experimental environment. The required notations are offered in Table 5. Table 6 provides a detailed overview of the hardware and software environments.

5.1. Datasets description

This paper collects AIS data from three different research areas: Tianjin Port, Chengshan Jiao Promontory, and Caofeidian Port. The datasets are used to verify the efficacy of the proposed ADPSC compression algorithm and cover the period from 1 July 2020 to 30 September 2022. The distribution and density visualization effects of vessel trajectories in the three regions are illustrated in Figs. 15 (a), (b), and (c), respectively. Meanwhile, the statistical information related to the above research areas is shown in Table 7. It is well known that vessel trajectory data can be affected during transmission through base stations and satellites, resulting in noisy data. Thus, high-quality trajectory data can be obtained from the newly collected original data after preprocessing in Section 3.3, which can be used for subsequent comparative experiments.

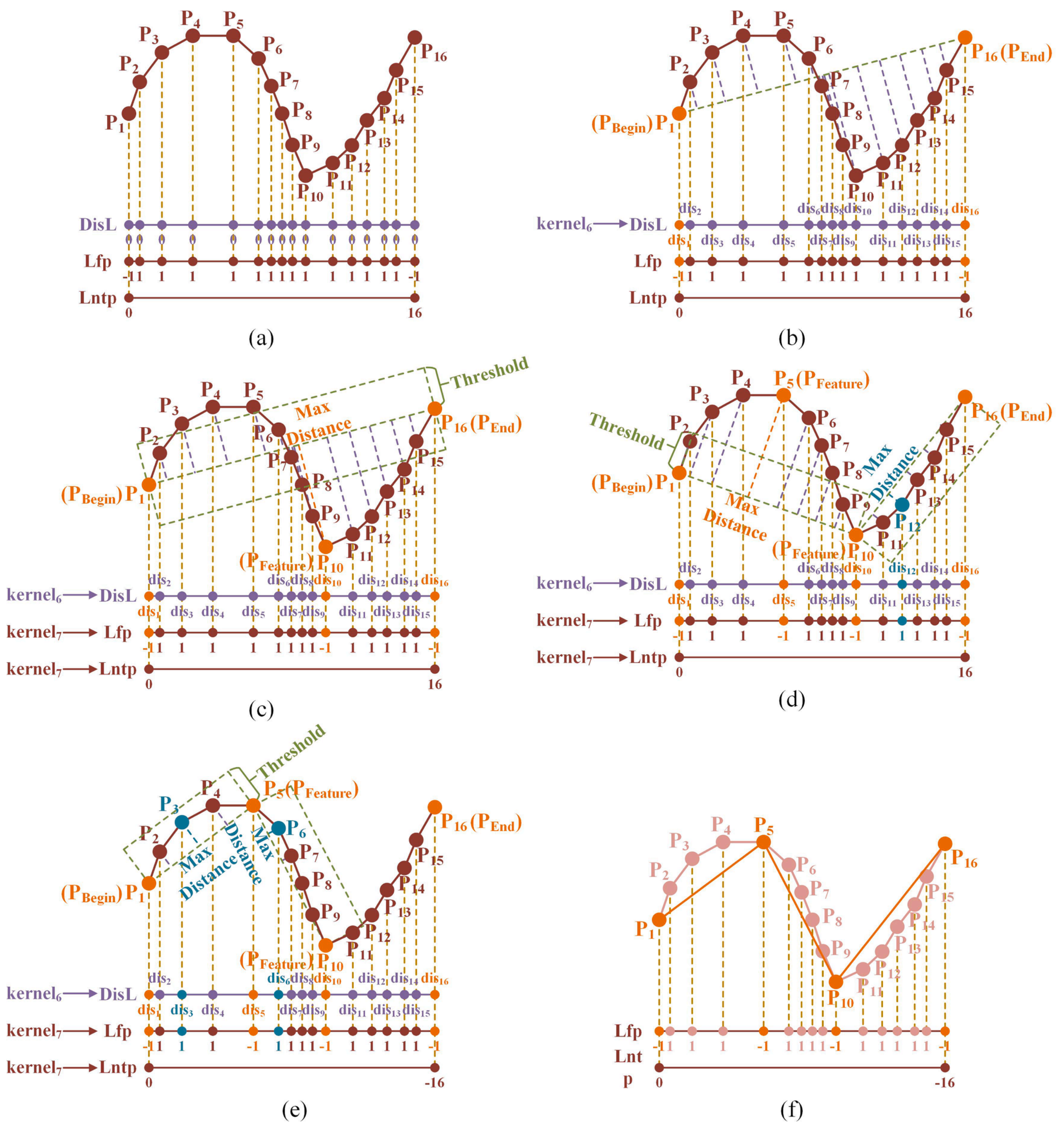


Fig. 13. Procedures of the proposed GPU-based ADPSC parallel compression of vessel trajectory. (a) original vessel trajectory marked with labels *DisL*, *Lfp*, and *Lntp*, (b) parallel computation of *dis*, (c) searching for the maximal *dis* to count the feature trajectory points, (d) segmenting trajectory and updating the new feature points, (e) iteratively finding the feature trajectory points, and (f) obtaining the final compressed vessel trajectory.

5.2. Compression evaluation metrics

To provide a quantitative assessment of the proposed ADPSC algorithm in performing vessel trajectory data compression tasks compared to the original DP algorithm, this paper selects Compression Ratio (CR), Rate of Length Loss (RLL), and DTW as key performance evaluation metrics. In particular, when the CR is high, and both the RLL and DTW are low, it signifies optimal compression performance. Concurrently, the Speedup Ratio (SR) is employed to quantitatively evaluate the acceleration efficiency of the ADPSC algorithm when compressing extensive vessel trajectory data in the GPU parallel computing framework.

5.2.1. Compression ratio

CR is a universal and standard indicator to measure compression efficacy, particularly in illustrating variations in the number of

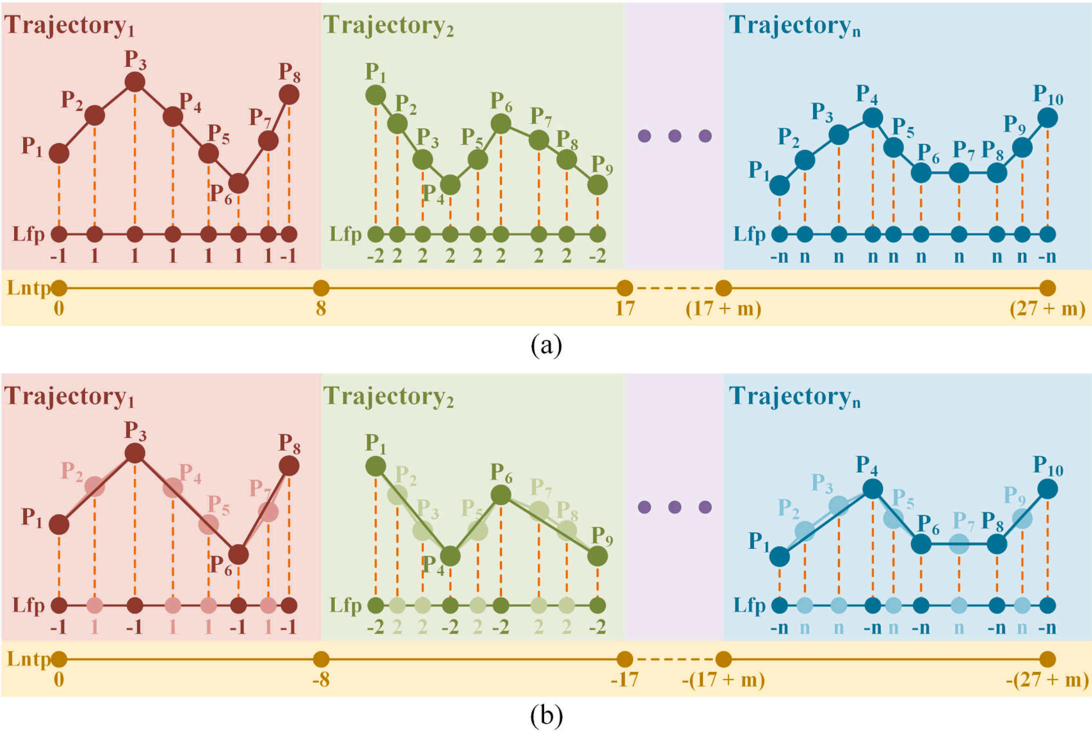


Fig. 14. Visual illustration of changes in two critical labels (i.e., *Lfp* and *Lntp*) before and after performing compression tasks. (a) initialization status of labels *Lfp* and *Lntp* before compression, and (b) data changes in labels *Lfp* and *Lntp* after completing the compression task.

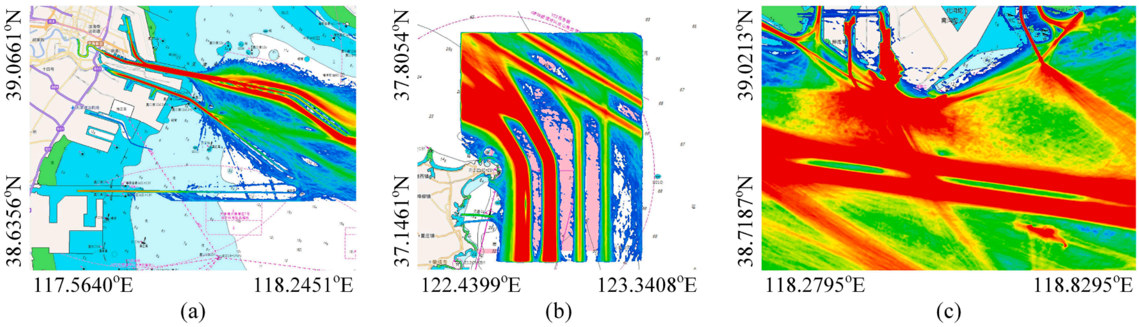


Fig. 15. Density visualisation of realistic vessel trajectory datasets: (a) Tianjin Port, (b) Chengshan Jiao Promontory, and (c) Caofeidian Port.

trajectory points. A higher CR value signifies removing a larger number of trajectory points. Its formula is defined as,

$$CR = \left(1 - \frac{N_{com}}{N_{ori}}\right) \times 100\% \tag{21}$$

where N_{com} and N_{ori} represent the number of points in all compressed and original trajectories, respectively.

5.2.2. Rate of length loss

RLL can reflect the length loss of the vessel's trajectory before and after compression. A lower RLL value suggests minimal distortion in the compressed trajectory, ensuring optimal compression results. The formula for RLL is given by

$$RLL = \frac{\sum_{i=1}^{I} \left|T_{ori}^{i}\right| - \sum_{i=1}^{I} \left|T_{com}^{i}\right|}{\sum_{i=1}^{I} \left|T_{ori}^{i}\right|} \tag{22}$$

where I represents the total number of vessel trajectories in the dataset. $\left|T_{ori}^{i}\right|$ and $\left|T_{com}^{i}\right|$ are the lengths of the i-th original and compressed trajectories, respectively.

Table 5
List of the notations.

Notations	Definition	Notations	Definition
CR	The compression ratio	W_{OC}	The set of all potential warping paths
N_{com}	The number of points in all original vessel trajectories	$d_Q(T_{ori}, T_{com})$	The warping cost of the Q
N_{ori}	The number of points in all compressed vessel trajectories	$d(\cdot,\cdot)$	The squared Euclidean distance
RLL	The rate of length loss	$DTW(T_{ori}, T_{com})$	The minimum result of the warping path based on the DTW algorithm
I	The total number of vessel trajectories in the dataset	$Q*$	The minimum warping cost
$\lvert T^i_{ori}\rvert$	The lengths of the i-th original vessel trajectories	V_{ori}	The vector of the distance from each intermediate point in the original trajectory to the baseline
$\lvert T^i_{com}\rvert$	The lengths of the i-th compressed vessel trajectories	V_{com}	The vector of the distance from each intermediate point in the compressed trajectory to the baseline
M	The patch matrix	k	The slope of the baseline for vessel trajectory
O	The number of original vessel trajectory points	b	The intercept of the baseline for vessel trajectory
C	The number of compressed vessel trajectory points	lon	Longitude of the intermediate point of vessel trajectory
$O \times C$	The size of matrix M	lat	Latitude of the intermediate point of vessel trajectory
$M_{a,b}$	The a-th and b-th values in matrix M	lon_S	Longitude of the starting point of vessel trajectory
P^a_{ori}	The a-th point in the original trajectory	lat_S	Latitude of the starting point of vessel trajectory
P^b_{com}	The b-th point in the compressed trajectory	lon_E	Longitude of the ending point of vessel trajectory
T_{ori}	The original vessel trajectory data	lat_E	Latitude of the ending point of vessel trajectory
T_{com}	The compressed vessel trajectory data	SR	The speedup ratio
$Q = \{q_1, q_2, q_3, \ldots, q_L\}$	The warping path between T_{ori} and T_{com}	T_{CPU}	The execution time of the ADPSC algorithm in the CPU serial computing framework
L	The length of sequence Q	T_{GPU}	The execution time of the ADPSC algorithm in the GPU parallel computing framework

Table 6
Hardware and software environments.

Hardware	Model	Software	Version
CPU	i7-12700KF Dodeca Core	Python	3.8.3
Host Memory	32GB	CUDA	11.7
GPU	GTX 3080	MySQL	8.0
Global Memory	12GB	-	-

Table 7
Statistical information related to three different regions.

Water Areas	Number of Vessel Trajectories	Number of Timestamped Points	Boundary Points	Longitude(^{o})	Latitude(^{o})
Tianjin Port	15720	10056186	Left Top	117.5640	39.0661
			Right Bottom	118.2451	38.6356
Chengshan Jiao Promontory	27738	18501815	Left Top	122.4399	37.8054
			Right Bottom	123.3408	37.1461
Caofeidian Port	63705	26933727	Left Top	118.2795	39.0213
			Right Bottom	118.8295	38.7187

5.2.3. Dynamic time warping

DTW is a prevalent measurement technique adept at calculating the similarity (inversely proportional to distance) between two sequential data (Lahreche and Boucheham, 2021). Its core idea is to find the minimum distance between two sequence data using the dynamic programming method. In this paper, DTW serves as a quantitative evaluation method to assess the similarity between original and compressed trajectory data (Li et al., 2020). Huang et al. (2020) also used DTW as an evaluation indicator when investigating compression algorithms for vessel trajectories. A smaller DTW value between two trajectories suggests that the compression algorithm introduces minimal distortion post-processing, indicating commendable compression efficacy.

A matrix M of size $O \times C$ is created, where each element $M_{a,b}$ along the warping path refers to the cumulative distance between the a-th point P^a_{ori} in the original trajectory and the b-th point P^b_{com} in the compressed trajectory. Meanwhile, let Q represent the warping path between T_{ori} and T_{com}, essentially a sequence $Q = \{q_1, q_2, q_3, \cdots, q_L\}$ with $q_l = (Ol, Cl) \in [1:O] \times [1:C]$. The set of all potential warping paths can be represented by W_{OC}. The warping cost $d_Q(T_{ori}, T_{com})$ of the Q can be defined as follows,

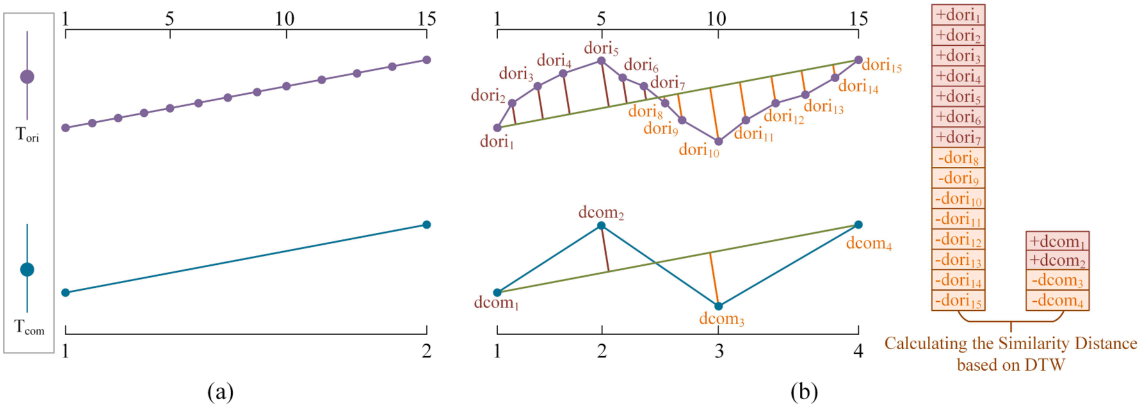


Fig. 16. The display of a new index for evaluating compression effectiveness, (a) shows the morphology of the original (T_{ori}) and compressed (T_{com}) trajectories, respectively; (b) reflects the process of using the optimized measurement method to calculate the similarity values of two trajectories before and after compression.

$$d_Q(T_{ori}, T_{com}) = \sum_{l=1}^{L} d\left(P_{ori}^{Ol}, P_{com}^{Cl}\right) \tag{23}$$

where L represents the length of sequence Q, and its range is $\max(O, C) \leqslant L \leqslant O + C$. $d(\cdot, \cdot)$ is the squared Euclidean distance. The DTW metric between T_{ori} and T_{com} related to the minimum warping cost is given as follows:

$$DTW(T_{ori}, T_{com}) = \sqrt{d_{Q*}(T_{ori}, T_{com})} = \min\left\{\sqrt{d_Q(T_{ori}, T_{com})}\,|Q \in W_{OC}\right\} \tag{24}$$

where $Q*$ is the minimum warping cost. Meanwhile, each element in M can be calculated as

$$M_{a,b} = d\left(P_{ori}^{a}, P_{com}^{b}\right) + \min\{M_{a-1,b-1}, M_{a-1,b}, M_{a,b-1}\} \tag{25}$$

The original DTW method attempts to match all points in the original trajectory with those in the compressed trajectory to determine the best warping path and calculate the similarity value. However, this approach introduces significant errors and inaccuracies in assessing the similarity between the original and compressed trajectories due to the considerable difference in the number of points between them. For example, if a vessel's trajectory forms a straight line and the compressed result only retains the starting and ending points, as illustrated in Fig. 16 (a), the compressed trajectory effectively reflects the original trajectory's distribution characteristics. Consequently, one might expect the similarity measure between these two trajectories to be zero. However, due to the reduction in the number of compressed trajectory points compared to the original trajectory, the similarity measure is not zero, as indicated by the DTW calculation process described earlier.

To address this issue, further optimization of the DTW measurement approach is proposed. The essence of this optimization lies in gauging the compression quality by examining the similarity in distances of each point in the original and compressed trajectories to the baseline. This baseline represents a line connecting the trajectory's starting and ending points, as shown in Fig. 16 (b).

To further elaborate on the computational process of the newly proposed measurement method, this paper breaks it down into three steps.

(1) *Distance calculation.* Eq. (13) is employed to calculate the distance from each point in the original and compressed trajectories to the baseline, respectively. This process yields two distinct distance vectors: V_{ori} for the original and V_{com} for the compressed trajectory.

(2) *Distance symbol judgment.* The core of this step lies in discerning the sign (positive or negative) of values within the two distance vectors, V_{ori} and V_{com}. As illustrated in Fig. 16 (b), this paper characterizes the positional relationship of trajectory points relative to the baseline by allocating a positive value to the distance of points above the baseline and a negative value to those below. Concurrently, if a trajectory point coincides with the baseline, it is assigned a value of zero within the vector. In specific judgments, the coordinates of the starting and ending points of the trajectory are used to generate the equation of the baseline as $y = kx + b$. The calculation method for k (slope) and b (intercept) is shown in Eqs. (26) and (27),

$$k = \frac{lat_E - lat_S}{lon_E - lon_S} \tag{26}$$

$$b = lat_E - \frac{lat_E - lat_S}{lon_E - lon_S} \times lon_E \tag{27}$$

Subsequently, the intermediate trajectory point coordinates (lon, lat) are brought into the linear equation. If inequality $lat - k \times lon - b > 0$ is satisfied, the distance between the trajectory point and the baseline is denoted as positive. In contrast, if inequality $lat - k \times lon - b < 0$ prevails, the distance value is considered negative. An exception arises when lon_E and lon_S are equal,

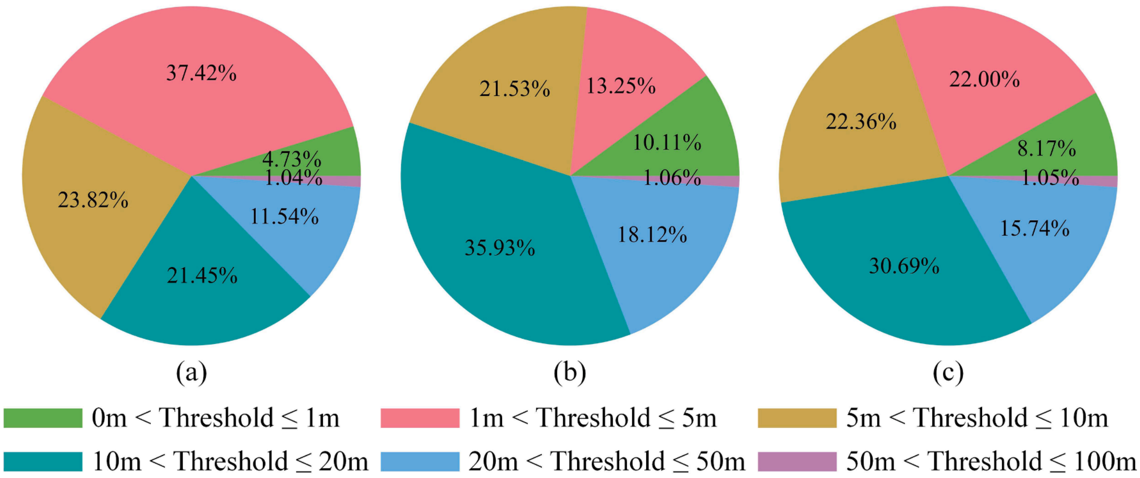


Fig. 17. Visual illustration of the compression threshold distribution calculated by the ADPSC algorithm in three research areas: (a) Tianjin Port, (b) Chengshan Jiao Promontory, and (c) Caofeidian Port.

Table 8
The average and standard deviation results of the ADPSC algorithm for calculating compression threshold in three water areas.

Water Areas	Threshold (m)	
	The Average Value	Standard Deviation
Tianjin Port	9.8550	10.4623
Chengshan Jiao Promontory	12.9139	10.8510
Caofeidian Port	11.8067	10.8123

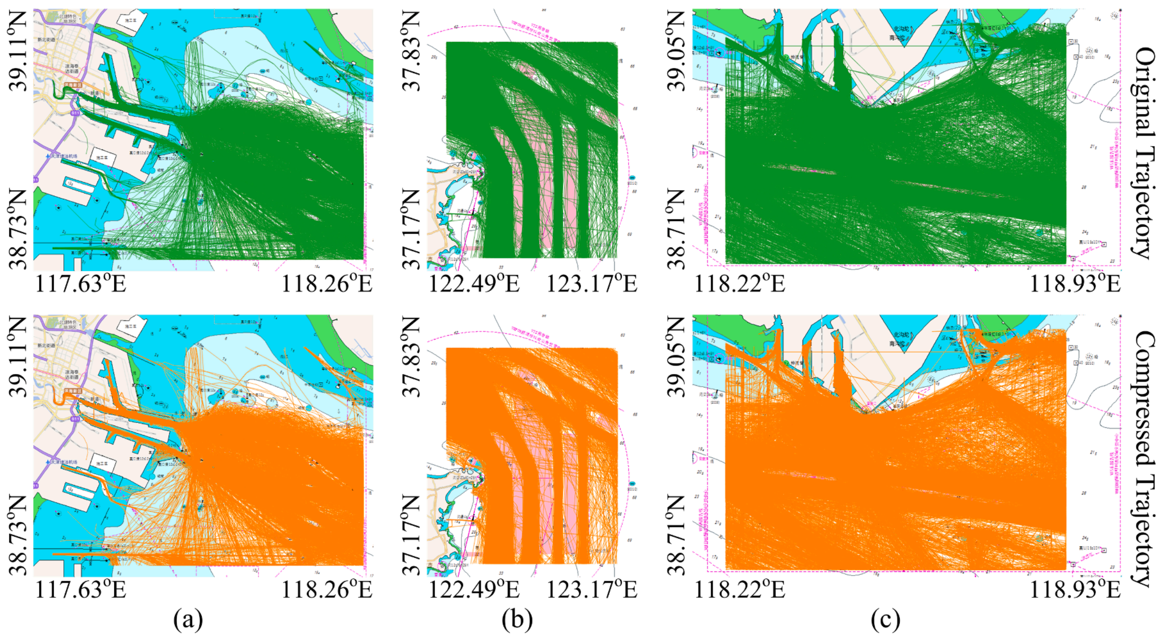


Fig. 18. Visualisation of original and compressed (using ADPSC algorithm) vessel trajectories in three different water areas: (a) Tianjin Port, (b) Chengshan Jiao Promontory, and (c) Caofeidian Port.

Table 9
The evaluation results of all trajectories data compressed based on original DP (i.e., 0.1m, 0.5m, 1.0m, 5.0m, 10.0m, 20.0m, 50.0m, and 100.0m), other adaptive DP (i.e., ADP and ADPS) and our proposed ADPSC algorithms in Tianjin Port.

Method	Threshold (m)	CR (↑%)	RLL (↓%)	DTW (↓)		TP-DTW (↓)	
				The Average Value	Standard Deviation	The Average Value	Standard Deviation
DP	0.1	14.1551	0.0001	0.0488	0.0621	0.0098	0.0171
	0.5	43.7300	0.0010	0.2089	0.2299	0.0406	0.0614
	1.0	60.0844	0.0045	0.3927	0.4280	0.0756	0.1144
	5.0	86.3768	0.0485	1.9654	2.1648	0.3563	0.5801
	10.0	91.7044	0.0913	3.5302	3.8370	0.6184	1.0226
	20.0	95.1449	0.1423	5.5132	5.8175	0.9391	1.5116
	50.0	97.0713	0.4168	8.7089	9.1329	1.4280	2.3826
	100.0	97.9406	0.6533	12.0523	12.8849	1.8978	3.3687
ADP	-	94.9625	0.1396	5.3419	5.6474	0.9123	1.4722
ADPS	-	93.2642	0.1142	4.2447	4.5781	0.7346	1.2006
ADPSC	-	83.4765	0.0311	1.1309	1.4074	0.1577	0.3864

Note: ↑ indicates that a larger CR value corresponds to better compression performance. ↓ denotes that smaller values of RLL, DTW, and TP-DTW result in better compression performance.

Table 10
The evaluation results of all trajectories data compressed based on original DP (i.e., 0.1m, 0.5m, 1.0m, 5.0m, 10.0m, 20.0m, 50.0m, and 100.0m), other adaptive DP (i.e., ADP and ADPS) and our proposed ADPSC algorithms in Chengshan Jiao Promontory.

Method	Threshold (m)	CR (↑%)	RLL (↓%)	DTW (↓)		TP-DTW (↓)	
				The Average Value	Standard Deviation	The Average Value	Standard Deviation
DP	0.1	14.9252	0.0001	0.0604	0.1173	0.0219	0.0553
	0.5	43.9705	0.0009	0.2258	0.2036	0.0796	0.0910
	1.0	60.9943	0.0033	0.4233	0.3459	0.1484	0.1520
	5.0	87.7554	0.0262	2.0797	1.6448	0.7267	0.7192
	10.0	93.1014	0.0455	3.8698	2.9062	1.3465	1.2849
	20.0	96.2251	0.0688	6.3799	4.4978	2.2100	2.0178
	50.0	97.8037	0.0876	10.3528	7.2076	3.5614	3.2201
	100.0	98.4310	0.1110	14.0456	9.9410	4.8001	4.3469
ADP	-	96.0748	0.0675	6.1725	4.3656	2.1396	1.9581
ADPS	-	94.5261	0.0598	5.7395	5.4885	1.9481	1.7544
ADPSC	-	**89.8047**	**0.0434**	**2.9050**	**2.5750**	**1.0577**	**1.0149**

implying the non-existence of k. For instance, the distance value is categorized as positive when the inequality $lon < lon_E(lon_S)$ holds true. Conversely, it is designated as negative.

(3) *Measurement results*. Based on the calculation results in step (2), the DTW method is used to compute the similarity between the newly obtained vectors, V_{ori} and V_{com}, resulting in the final measurement result.

This new measurement method is called Trajectory Points-based DTW (TP-DTW). The conventional similarity calculation is rooted in direct measurements between two trajectories. This original method is designated as DTW herein. Section 5.3 delves deeper into the distinctions between DTW and TP-DTW in assessing compression efficacy.

5.2.4. *Speedup ratio*

SR quantifies the acceleration capabilities of the proposed ADPSC algorithm in performing compression tasks under the GPU parallel computing framework compared to traditional computing methods. A notably higher SR value suggests a more enhanced acceleration yielded by this new computational framework. The underlying equation for SR is as follows,

$$SR = \frac{T_{CPU}}{T_{GPU}} \tag{28}$$

where T_{CPU} and T_{GPU} represent the execution time of the ADPSC algorithm when performing the same task under the CPU serial and GPU parallel computing framework, respectively.

In this paper, an essential aspect to take into account is that the computational expense of the CPU serial framework solely comprises the runtime of the ADPSC algorithm when compressing vessel trajectory data. On the other hand, the computing cost of the GPU parallel framework includes not only the running time of the algorithm but also the time of original data transmission from memory to video memory and the transmission of compressed data from video memory back to memory. If parallel algorithms lack optimal design, they will consume more computational time, especially when dealing with large-scale datasets. Therefore, in designing the GPU parallel algorithms, this paper accounts for the implications of data transfer on computation time, ensuring a more effective integration with the real-world application environment.

Table 11
The evaluation results of all trajectories data compressed based on original DP (i.e., 0.1m, 0.5m, 1.0m, 5.0m, 10.0m, 20.0m, 50.0m, and 100.0m), other adaptive DP (i.e., ADP and ADPS) and our proposed ADPSC algorithms in Caofeidian Port.

Method	Threshold (m)	CR (↑%)	RLL (↓%)	DTW (↓)		TP-DTW (↓)	
				The Average Value	Standard Deviation	The Average Value	Standard Deviation
DP	0.1	14.6540	0.0001	0.0562	0.1010	0.0176	0.0458
	0.5	43.8858	0.0010	0.2197	0.2136	0.0655	0.0837
	1.0	60.6739	0.0037	0.4123	0.3779	0.1221	0.1439
	5.0	87.2700	0.0336	2.0383	1.8507	0.5927	0.6954
	10.0	92.6095	0.0606	3.7470	3.2777	1.0831	1.2468
	20.0	95.8447	0.0930	6.0664	5.0327	1.7503	1.9489
	50.0	97.5458	0.1963	9.7582	7.9970	2.7897	3.1181
	100.0	98.2583	0.2900	13.3246	11.1376	3.7502	4.2556
ADP	-	93.9579	0.0842	5.3059	5.2612	1.4036	1.1356
ADPS	-	92.9724	0.0792	4.2278	4.0239	1.3474	1.2902
ADPSC	-	**87.5761**	**0.0393**	**2.9470**	**2.8563**	**0.8595**	**0.7847**

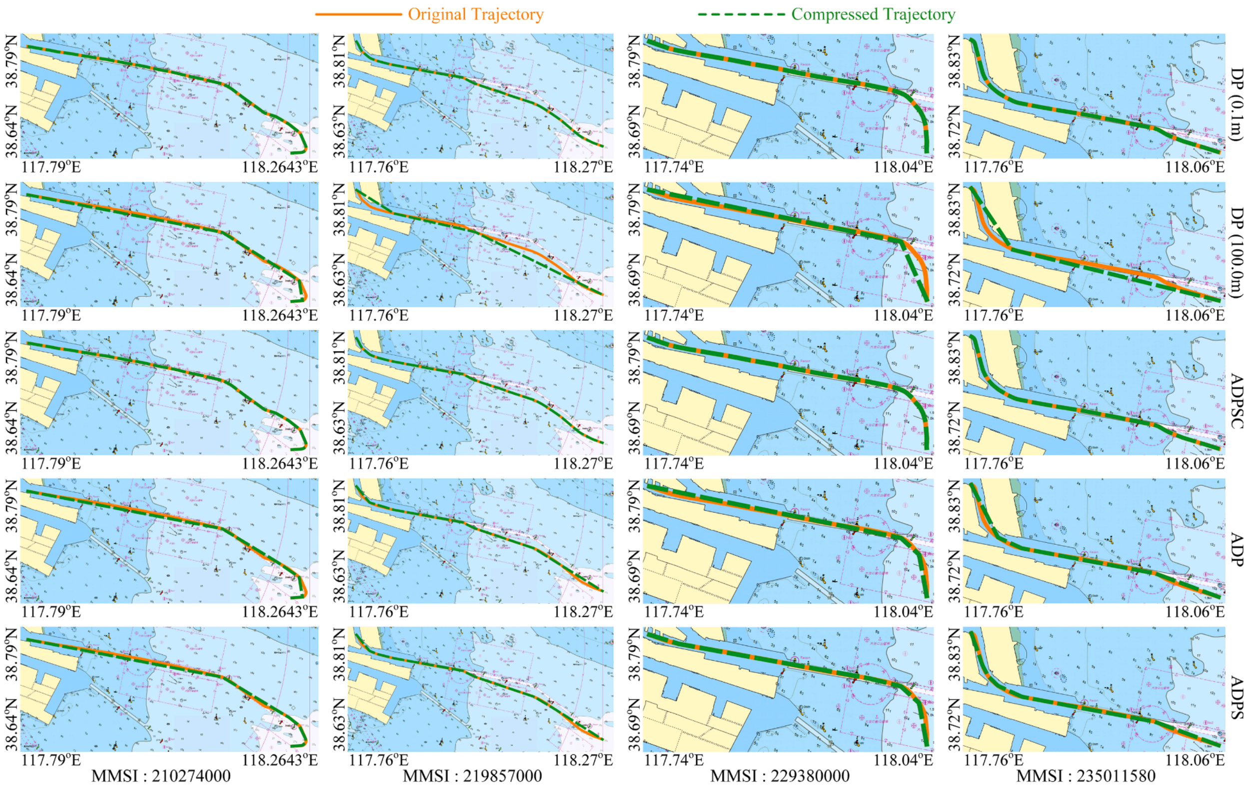


Fig. 19. Visual comparison of compression performance of four representative trajectories based on original DP (i.e., 0.1m and 100m), other adaptive DP (i.e., ADP and ADPS), and the proposed ADPSC algorithms in Tianjin Port.

5.3. Compression performance of the ADPSC method

The primary advantage of the proposed ADPSC algorithm in this paper lies in its ability to calculate compression thresholds based on the characteristics of different trajectory data automatically. Fig. 17 reflects the distribution of compression thresholds calculated based on the ADPSC algorithm for all vessel trajectories in the three research areas. The threshold distribution of the three regions mainly falls within the 1 m to 50 m range. In particular, the trajectory threshold distribution of Tianjin Port is between 1 m and 5 m, which accounts for the most significant proportion. According to the results in Figs. 17 (b) and (c), the threshold distribution of Chengshan Jiao Promontory and Caofeidian Port appear closely assigned, with the most common range being 10 m to 20 m. Concurrently, Table 8 shows the average and standard deviation of all trajectory compression thresholds in three regions. The standard deviations across the regions are consistent. Among them, Chengshan Jiao Promontory has the highest average, Tianjin Port is the lowest, and Caofeidian Port's average is close to that of Chengshan Jiao Promontory.

To provide a clear visual representation of the ADPSC algorithm, this paper projected the original and compressed trajectories of

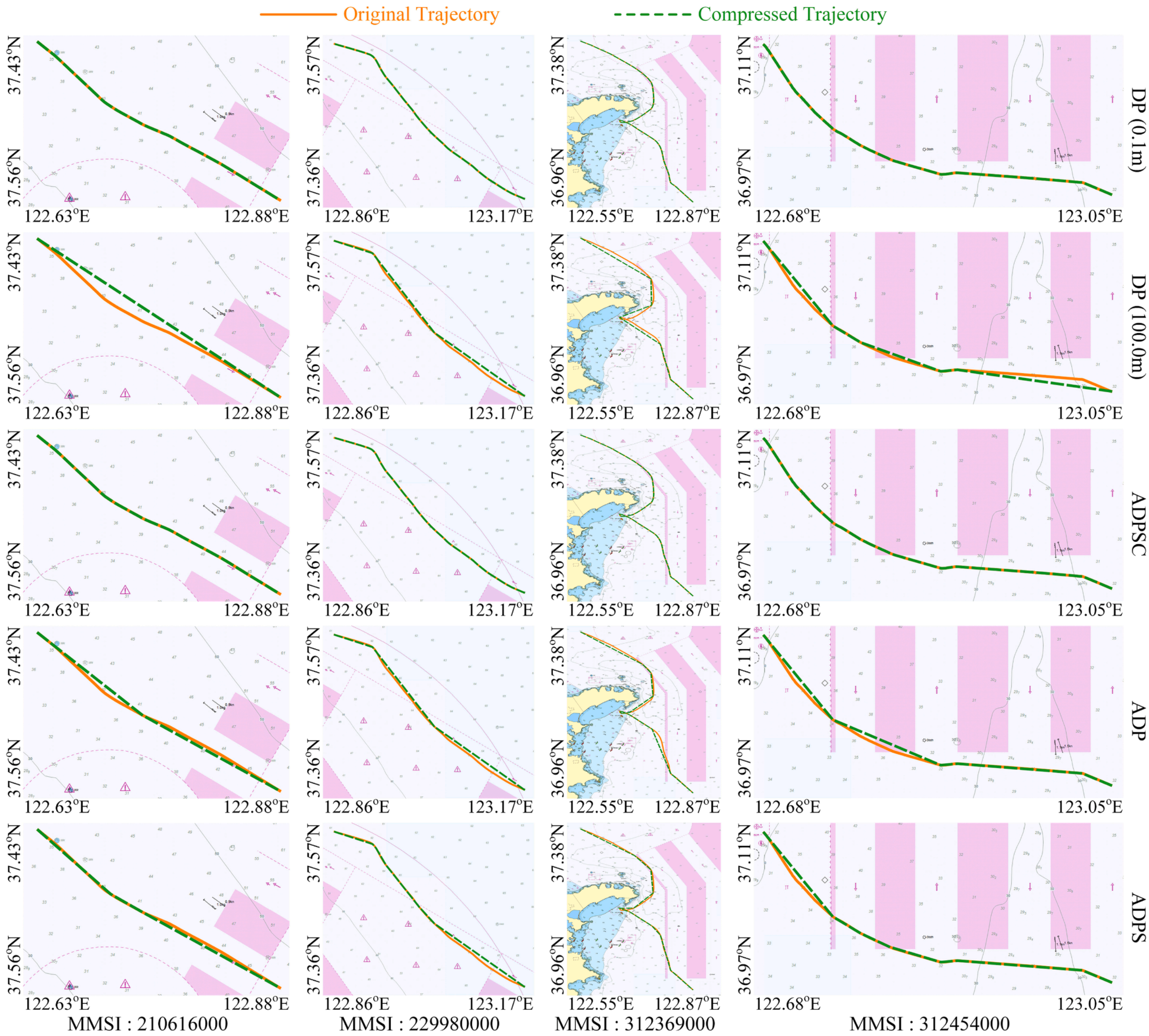


Fig. 20. Visual comparison of compression performance of four representative trajectories based on original DP (i.e., 0.1m and 100m), other adaptive DP (i.e., ADP and ADPS), and the proposed ADPSC algorithms in Chengshan Jiao Promontory.

three regions onto the map, as shown in Fig. 18. At a macro level, this visual comparison effectively illustrates that the distribution of compressed trajectories closely mirrors that of the original ones.

To further quantitatively evaluate the overall effectiveness of the new ADPSC compression algorithm on all trajectory data in each study area, this paper conducts comparative experiments with the original DP and two other classic adaptive compression algorithms, namely ADP (Liu et al., 2019b) and ADPS (Li et al., 2022). Notably, the ADP and ADPS algorithms, along with the ADPSC algorithm proposed in this paper, are essentially optimized based on the original DP algorithm. The advantage of the ADPSC algorithm compared to ADP and ADPS algorithms lies in simultaneous consideration of the SOG and COG of trajectory points, enabling more accurate preservation of feature points. Therefore, the comparative experiment between ADPSC and these two adaptive compression algorithms is highly valuable. This experiment accumulates the CR and RLL values of each trajectory to obtain the consolidated results in Tables 9, 10, and 11. Additionally, the DTW and TP-DTW values of each trajectory are calculated to determine their average value and standard deviation. According to the results in Tables 9, 10, and 11, setting a small threshold (e.g., 0.1 m and 0.5 m) in the DP algorithm yields high similarity between the compressed and original trajectories; however, the CR remains low, retaining redundant data. Conversely, an excessively large threshold (e.g., 50.0 m and 100.0 m) might result in a high CR, but the compressed trajectory will deviate significantly from the original distribution. Conclusively, the three adaptive compression algorithms overcome the conventional DP algorithm's limitations by effectively eliminating redundant data and mirroring the original trajectory's distribution characteristic.

Further examination of the compression effects between ADP, ADPS, and ADPSC reveals that while ADP and ADPS can compress more trajectory points compared to ADPSC, the trade-off is the destruction of the original trajectory's data structure. According to the

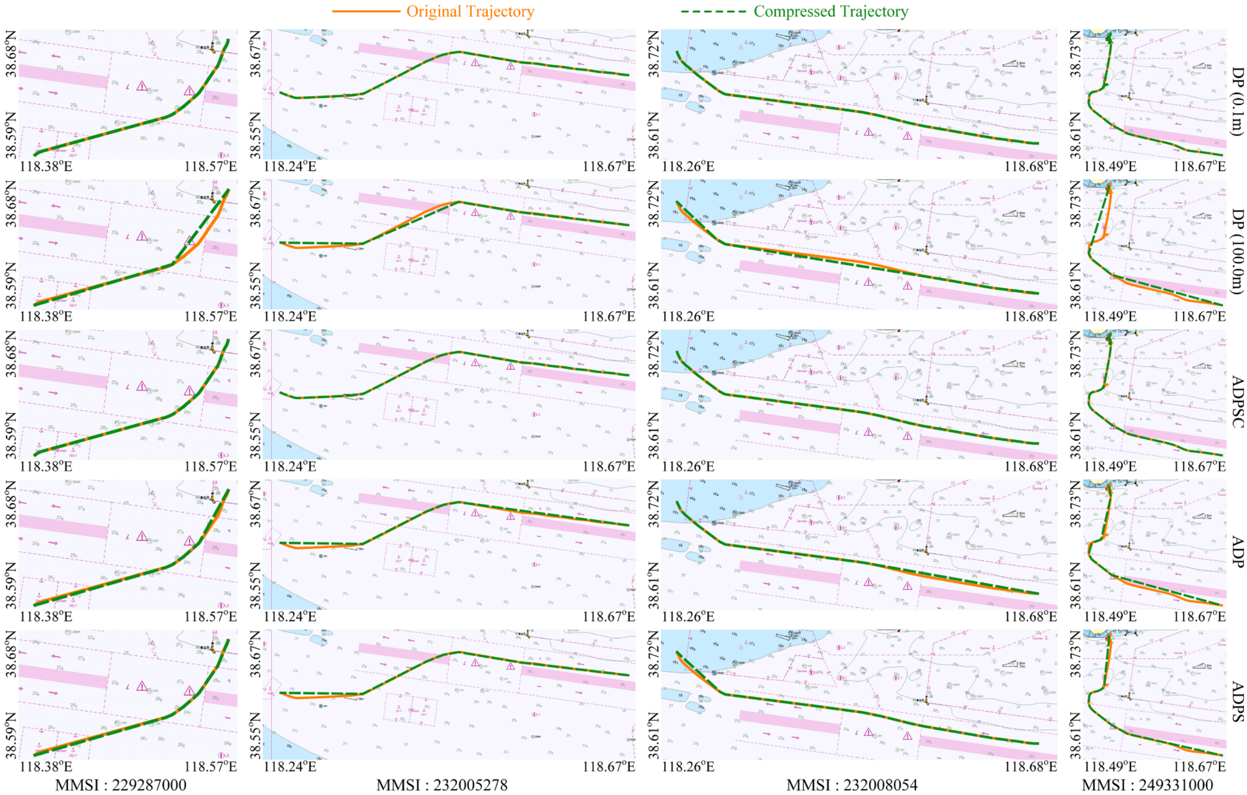


Fig. 21. Visual comparison of compression performance of four representative trajectories based on original DP (i.e., 0.1m and 100m), other adaptive DP (i.e., ADP and ADPS) and the proposed ADPSC algorithms in Caofeidian Port.

Table 12
The comparative evaluation of compression results of four representative trajectories data based on original DP (i.e., 0.1m and 100m), other adaptive DP (i.e., ADP and ADPS), and the proposed ADPSC algorithms in Tianjin Port.

MMSI	Method	Threshold (m)	CR (↑%)	RLL (↓%)	DTW (↓)	TP-DTW (↓)
210274000	DP	0.1	58.0071	0.0054	0.4077	0.0796
		100.0	99.5552	0.6530	59.8126	9.0797
	ADPSC	2.6906	96.6192	0.0448	6.3684	1.1626
	ADP	24.0951	97.1530	0.2713	15.9862	4.6692
	ADPS	19.8884	96.8971	0.1044	9.9219	3.2283
219857000	DP	0.1	55.9546	0.0014	0.2388	0.0337
		100.0	99.2439	0.8604	37.1240	2.8048
	ADPSC	4.1081	93.1947	0.0173	3.5966	0.5122
	ADP	29.6453	94.7069	0.4234	12.2387	1.8768
	ADPS	20.9182	94.2508	0.3065	10.7851	1.2408
229380000	DP	0.1	56.5744	0.0044	0.2317	0.0535
		100.0	99.4810	1.4623	27.2048	4.0459
	ADPSC	3.5157	95.3287	0.0341	2.5681	0.4840
	ADP	33.2375	97.2318	0.7988	13.3371	2.6791
	ADPS	16.2609	96.0207	0.1252	8.0259	1.4536
235011580	DP	0.1	78.5064	0.0017	0.6200	0.1440
		100.0	99.4536	1.7858	28.0332	4.0019
	ADPSC	6.0255	94.8998	0.0183	3.0053	0.7501
	ADP	31.3695	96.3570	0.9607	10.9447	2.1926
	ADPS	20.2482	95.9927	0.1246	7.0329	1.5877

results in Tables 9, 10, and 11, compared to ADPSC, ADP compresses trajectory data from three water areas, increasing CR by 13.7595 %, 6.9819 %, and 7.2871 %, respectively. Meanwhile, the growth rates of indicator RLL in three different water areas are 348.8745 %, 55.5300 %, and 114.2494 %, respectively. The growth rates of average DTW in three different water areas are 372.3583 %, 112.4785 %, and 80.0441 %, respectively. The growth rates of average TP-DTW in three different water areas are 478.5035 %, 102.2880 %, and 63.3042 %, respectively. Additionally, compared with ADPSC, ADPS compresses vessel trajectory data from three water areas,

Table 13
The comparative evaluation of compression results of four representative trajectories based on original DP (i.e., 0.1m and 100m), other adaptive DP (i.e., ADP and ADPS), and the proposed ADPSC algorithms in Chengshan Jiao Promontory.

MMSI	Method	Threshold (m)	CR (↑%)	RLL (↓%)	DTW (↓)	TP-DTW (↓)
210616000	DP	0.1	71.4286	0.0009	0.3089	0.0281
		100.0	99.3031	0.6484	18.5550	1.3611
	ADPSC	5.1938	95.8188	0.0070	2.6893	0.2236
	ADP	31.8505	96.8641	0.3154	8.2064	0.8598
	ADPS	14.8402	96.0818	0.0956	4.9892	0.6235
229980000	DP	0.1	74.0343	0.0008	0.4389	0.0746
		100.0	99.1416	0.4224	16.2348	1.9550
	ADPSC	4.5186	93.9914	0.0137	2.1256	0.3463
	ADP	63.5849	98.2643	0.3958	13.2674	1.4606
	ADPS	28.2039	96.2060	0.1962	8.1285	1.0667
312369000	DP	0.1	58.2103	0.0042	0.3916	0.1784
		100.0	99.3918	2.6520	30.4369	11.9079
	ADPSC	4.2388	94.9609	0.0394	5.1327	2.0871
	ADP	34.2432	96.1772	1.0636	13.4103	5.0986
	ADPS	17.2732	95.0477	0.4021	9.1531	3.9059
312454000	DP	0.1	59.4354	0.0025	0.2511	0.0605
		100.0	99.4056	0.7918	25.9335	4.6652
	ADPSC	7.7530	96.2853	0.0381	5.5157	1.4210
	ADP	35.2794	97.6225	0.5727	11.6736	2.8836
	ADPS	15.2307	96.4338	0.1187	9.5250	2.1216

Table 14
The comparative evaluation of compression results of four representative trajectories based on original DP (i.e., 0.1m and 100m), other adaptive DP (i.e., ADP and ADPS), and the proposed ADPSC algorithms in Caofeidian Port.

MMSI	Method	Threshold (m)	CR (↑%)	RLL (↓%)	DTW (↓)	TP-DTW (↓)
229287000	DP	0.1	67.8010	0.0034	0.1867	0.0571
		100.0	99.2147	1.0223	8.8133	2.3111
	ADPSC	5.9086	95.0262	0.0254	3.7757	0.9178
	ADP	30.1356	97.1204	0.6328	6.0392	1.8809
	ADPS	17.2888	96.8586	0.2077	5.2924	1.3695
232005278	DP	0.1	67.0103	0.0011	0.3815	0.0635
		100.0	99.1753	0.5117	16.6808	2.4856
	ADPSC	5.2300	94.0206	0.0128	3.2106	0.5697
	ADP	26.6493	95.0515	0.3638	9.8157	1.5739
	ADPS	18.6996	94.6391	0.1004	5.2332	1.1697
232008054	DP	0.1	51.6049	0.0014	0.1822	0.0223
		100.0	99.2593	0.3730	25.0586	2.7437
	ADPSC	4.4573	95.0617	0.0193	3.7387	0.3759
	ADP	31.3096	96.5432	0.1566	10.5689	1.7813
	ADPS	15.5434	95.3086	0.0998	6.7399	1.0741
249331000	DP	0.1	24.0132	0.9274	0.0490	0.0261
		100.0	99.3421	6.6100	11.3299	6.2540
	ADPSC	4.6332	90.4605	2.4995	1.0457	0.6063
	ADP	29.9813	95.0657	5.2677	6.8873	4.0763
	ADPS	19.7861	93.2565	4.5958	4.7501	2.3035

increasing CR by 11.7251 %, 5.2574 %, and 6.1618 %, respectively. Regarding the three indicators of evaluating trajectory quality, the growth rates of indicator RLL in three different water areas are 267.2026 %, 37.7880 %, and 101.5267 %, respectively. The growth rates of average DTW in three different water areas are 275.5338 %, 97.5731 %, and 43.4611 %, respectively. The growth rates of average TP-DTW in three different water areas are 365.8212 %, 84.1827 %, and 56.7656 %, respectively.

Based on the comparison of the growth rates of the above four indicators in different compression algorithms, although ADP can compress more trajectory points than ADPS, it deletes more key feature points, thus altering the trajectory structure. This result is because ADP represents the compression threshold by calculating the average distance from each trajectory point to the baseline, which may blur key information, as analyzed in Section 4.2. Furthermore, the ADPSC algorithm proposed in this paper has a slightly lower CR value compared to the ADPS algorithm, but it offers the highest compressed data quality. In practical application scenarios, the compression quality of vessel trajectory data is paramount. Hence, the ADPSC algorithm outperforms other classic adaptive compression algorithms, such as DP, ADP, and ADPS, in vessel trajectory data compression tasks.

Further deepening the exploration, four representative trajectories with multiple feature points from each region are selected. These trajectories undergo compression using the original DP, two other adaptive compression algorithms (i.e., ADP and ADPS), and the optimized ADPSC algorithm. This allows for a direct comparison of their efficiency in handling trajectory compression. The chosen trajectories are specifically selected for their multiple feature points, providing a comprehensive representation of the predominant

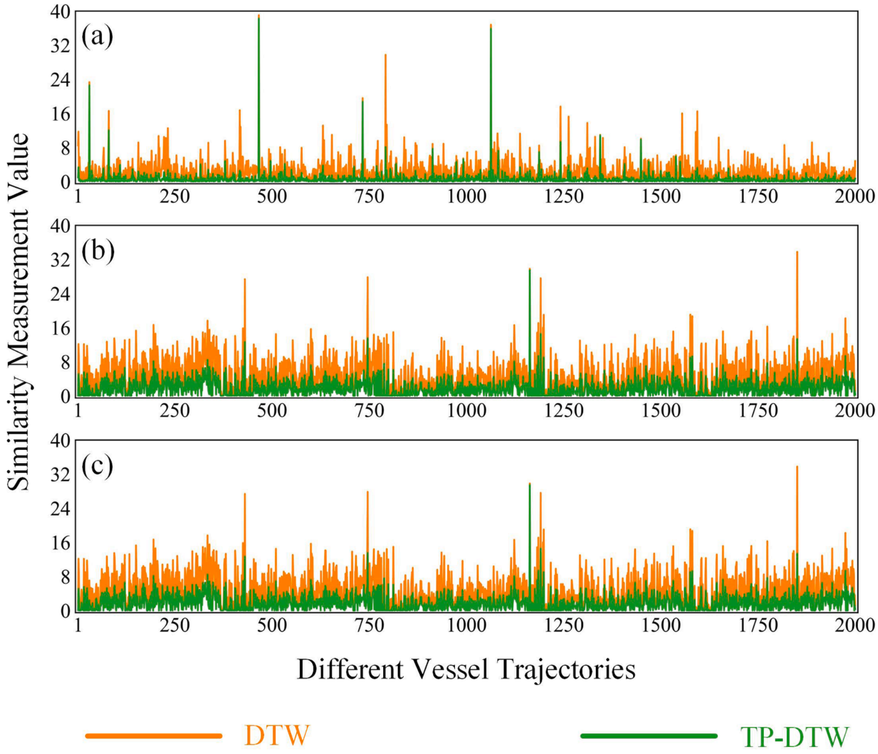


Fig. 22. Comparison of two similarity measurement methods (i.e., DTW and TP-DTW) for evaluating compression performance: (a) Tianjin Port, (b) Chengshan Jiao Promontory, and (c) Caofeidian Port.

Table 15
Comparison of storage space occupied before and after data compression. In particular, the stored data mainly includes timestamps, longitude and latitude coordinates, SOG, and COG.

Water areas	Storage size without compression (MB)	Storage size with compression (MB)
Tianjin Port	938.3889	154.0128
Chengshan Jiao Promontory	1735.2518	175.6007
Caofeidian Port	2673.6407	329.6135

distribution characteristics found in most trajectories. The compression effects of deploying the DP, ADP, ADPS, and proposed ADPSC algorithms on these select trajectories within the three areas are illustrated in Figs. 19, 20, and 21. Unlike the broader overview presented in Fig. 18, these figures offer a more detailed, micro-level assessment of compression performance.

When the four representative trajectories of each research area are compressed using the DP algorithm, thresholds of 0.1 m and 100 m are selected and used for comparative experiments. As depicted in Figs. 19, 20, and 21, a larger threshold set by the DP algorithm for trajectory compression leads to a more distorted trajectory compared to its original version, failing to accurately reflect its inherent distribution characteristics. At a compression threshold of 0.1 m with the DP method, the trajectory retains features almost identical to the original, but as shown in Tables 12, 13, and 14, the CR value is notably low, indicating the presence of redundant data. In summary, developing adaptive compression algorithms to automatically calculate the compression threshold for each trajectory has practical application and research value.

On the contrary, the ADPSC algorithm is designed to automatically determine suitable thresholds based on the unique distribution attributes of each trajectory and its dynamic navigation details. This approach ensures that significant features from the original trajectory are retained while eliminating redundant data. Although ADP and ADPS algorithms can also automatically calculate the compression threshold for each trajectory and compress more trajectory points compared to the ADPSC algorithm, they incur the cost of losing the original structure of the trajectory, which is not advisable in practical applications. The visual representations in Figs 19, 20, and 21 qualitatively indicate that the ADP and ADPS algorithms sometimes fail to accurately preserve the feature points of vessels turning slightly, while the new ADPSC algorithm can achieve this. This implies that the trajectory compressed using the ADPSC algorithm is consistent with the original spatial distribution on the map. Additionally, Tables 12, 13, and 14 show that all the trajectories compressed via the ADPSC method achieve lower RLL and similarity metrics (either DTW or TP-DTW) while maintaining a superior CR value.

Tables 12, 13, and 14 preliminarily reflect the disparities between DTW and TP-DTW in evaluating compression performance. To

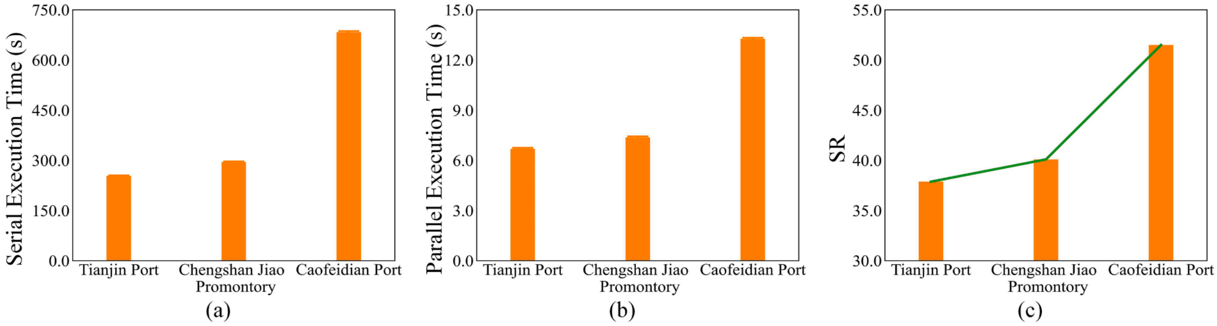


Fig. 23. Accelerated effect of the proposed ADPSC compression algorithm based on GPU parallel computing framework, (a) the CPU (serial) execution times (average value + standard deviation), (b) GPU (parallel) execution times (average value + standard deviation), and (c) the speedup ratio between CPU and GPU. Specifically, the calculation time across ten iterations for each experimental scenario shows minimal variation, resulting in a very low standard deviation.

further explore the differences in assessing the compression performance of each trajectory using these two measurement methods, this experiment randomly selected 2000 trajectories from each study area. At the same time, the DTW and TP-DTW metrics of each trajectory after compression are drawn into a line chart, depicted in Fig. 22.

According to the results displayed in the line chart, it becomes apparent that the novel TP-DTW metric consistently produces lower values than its DTW counterpart when calculating the similarity between the original and compressed trajectories. This is largely because traditional DTW when assessing the similarity between the two trajectories, tends to record enhanced measurements even with minor trajectory alterations (such as a reduction in trajectory points). In contrast, the TP-DTW approach focuses on assessing similarity by evaluating the deviations of points in both trajectories from a baseline. The advantage of the TP-DTW method over DTW is that it measures compression quality by checking the similarity between the distance from each point in the original trajectory and the baseline in the compressed trajectory. Specifically, the baseline represents a line connecting the trajectory's starting and ending points. In summary, if a compression algorithm retains all pivotal points in the compressed trajectory, the value derived from TP-DTW will be considerably lower than that from DTW. For instance, the trajectory depicted in Fig. 16 (a) retains all key points after compression, mirroring the original distribution characteristics. Theoretically, their similarity metric should stand at zero. While TP-DTW aligns with this theoretical estimate, DTW offers a different result. Conclusively, the newly proposed TP-DTW metric emerges as a more fitting tool for assessing compression efficacy.

The above analysis demonstrates the effectiveness of the ADPSC algorithm in compressing vessel trajectory data and its ability to eliminate redundant information in the dataset. To further quantitatively illustrate how these compression algorithms can economize a storage space, this experiment counted the storage size of three regional datasets before and after compression, with the specifics displayed in Table 15. The trajectory data from Tianjin Port, Chengshan Jiao Promontory, and Caofeidian Port, when compressed using the ADPSC algorithm, led to savings of 784.3761 MB, 1559.6511 MB, and 2344.0272 MB, respectively. In summary, the newly proposed ADPSC algorithm in this paper excels in removing redundant data from trajectories, thereby substantially curtailing storage costs.

5.4. Acceleration performance of GPU-based ADPSC method

To verify the enhanced efficiency of the accelerated ADPSC compression algorithm in processing vast vessel trajectory data, this paper tests the ADPSC algorithm on trajectory data from three experimental areas by both CPU serial and GPU parallel computing frameworks. The execution time results are displayed in Figs. 23 (a) and (b), respectively. Moreover, each experimental scenario is executed ten times, and the average is calculated to minimize random variations. Fig. 23 (c) employs the indicator SR to visually demonstrate the significant speed advantage of the ADPSC algorithm under the GPU parallel computing framework compared with traditional CPUs. According to the statistical results in Table 7, Caofeidian Port has 35,967 more vessel trajectories than Chengshan Jiao Promontory, with a discrepancy of 8,431,912 in the count of trajectory points. On the other hand, Tianjin Port records the least data compared to the other two zones, trailing Chengshan Jiao Promontory by 12,018 trajectories and 8,445,629 trajectory points. The SR of the accelerated ADPSC algorithm, when processing data in the Caofeidian Port region, stands at 51.4988. This is 11.4 and 2.2 points higher than the rates for Chengshan Jiao Promontory and Tianjin Port, respectively. Conclusively, the ADPSC algorithm's acceleration under the GPU parallel framework becomes more evident as the scale of the vessel trajectory dataset increases.

6. Conclusion and future perspectives

This paper enhances the conventional DP algorithm by developing an adaptive ADPSC method. This innovative compression approach autonomously determines compression thresholds for individual trajectories, relying on the distribution characteristics of vessel trajectories and navigational data, encompassing time stamps, SOG, and COG. Unlike applying a universal compression threshold to all trajectories, this method ensures optimal compression outcomes. Additionally, a GPU parallel computing framework is proposed to expedite the ADPSC algorithm's data compression processes. This optimization framework divides the algorithm into two segments: parallel threshold calculation and parallel compression. This customized methodology is better suited for managing large-scale datasets than conventional CPU-based systems.

Furthermore, this paper introduces the TP-DTW method, which calculates the similarity between trajectory point distributions on the baseline before and after compression to evaluate the compression effectiveness. Compared with the traditional DTW method, the novel TP-DTW method is based on the different distributions of the trajectory points, enhancing the compression analysis capability of the algorithm. Experimental datasets from three research areas, Tianjin Port, Chengshan Jiao Promontory, and Caofeidian Port, demonstrate that the ADPSC algorithm outperforms the DP counterpart by delivering better compression rates while minimizing data distortion. Simultaneously, leveraging the GPU parallel computing framework accelerates the ADPSC algorithm, especially when handling massive trajectory data. In essence, the proposed framework offers efficient and rapid trajectory compression that eliminates redundant data, making it crucial for intelligent maritime transportation systems. The GPU-accelerated adaptive compression methodology serves a multifaceted function by reducing data size while preserving accuracy and essential information. This enables efficient data storage, faster transmission over networks, cost savings in storage and bandwidth, improved data analysis, and enhanced safety and reliability in applications in maritime operations. Overall, it optimizes data management, enhancing cost-effectiveness and efficiency without compromising critical data integrity.

Nevertheless, future research directions could use multiple GPUs to construct new parallel computing frameworks, enhancing accelerated ADPSC compression algorithms. While the single GPU parallel compression algorithm highlighted in this paper already showcases promising acceleration outcomes, subsequent investigations could leverage multiple GPUs to either shorten execution times even further or process even larger data volumes more efficiently. It remains crucial to determine different thresholds for each vessel trajectory during compression tasks. Yet, trajectories exhibiting significant similarities could potentially be compressed using a shared threshold without compromising compression quality. Hence, an intriguing future research direction lies in accurately clustering vessel trajectories, followed by tailored threshold calculations for different trajectory clusters, which holds the promise of substantial computational savings.

CRediT authorship contribution statement

Yan Li: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Resources, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Huanhuan Li:** Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Project administration, Resources, Software, Supervision, Visualization, Writing – original draft, Writing – review & editing. **Chao Zhang:** Formal analysis, Investigation, Validation, Visualization, Writing – original draft. **Yunfeng Zhao:** Formal analysis, Investigation, Validation, Visualization, Writing – original draft. **Zaili Yang:** Funding acquisition, Methodology, Project administration, Resources, Validation, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgements

This work is supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (Grant Agreement No. 864724) and Royal Society International Exchanges 2021 Cost Share (NSFC) (IEC\NSFC\211211).

Appendix A

Clustering evaluation metrics, including the Silhouette Coefficient (SC), Calinski-Harabasz Score (CHS), and Davies-Bouldin Index (DBI), assess clustering effectiveness by examining point density within clusters and separation between clusters (Li et al., 2022; Li and Yang, 2023). After performing the DBSCAN clustering task, each trajectory obtains SC, CHS, and DBI values. Additionally, the average values of these metrics across all trajectories in the three study areas are calculated. The experimental results showing the average SC, CHS, and DBI values for varying *Eps* and *Minpt* parameters are displayed in Fig. 24. According to the experimental findings, when *Eps* is 0.01 and *Minpt* is 3, the three study areas achieve the maximum SC, CHS, and minimum DBI. Therefore, these values are optimal for the DBSCAN algorithm to identify outliers in all vessel trajectories within these study areas.

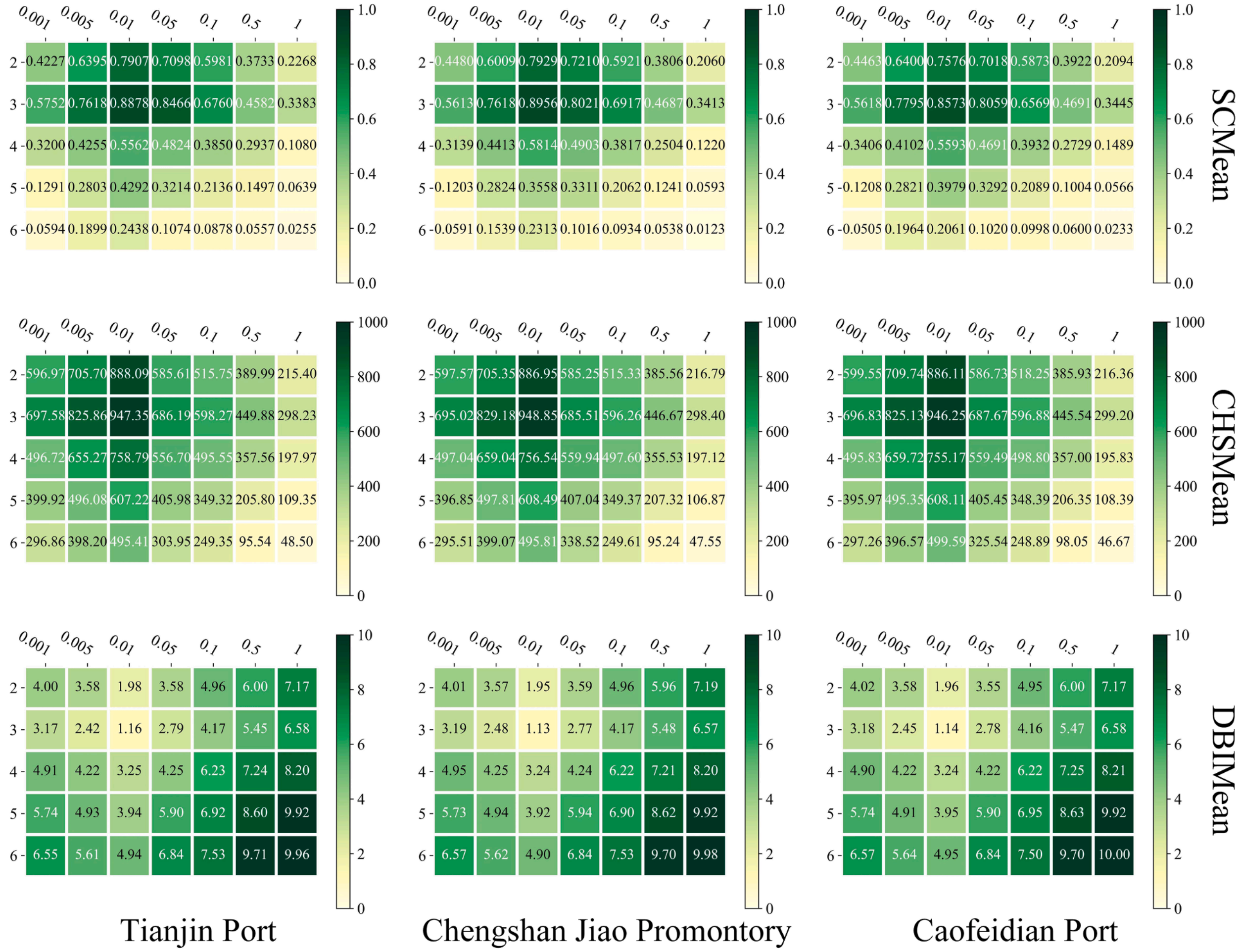


Fig. 24. Visual illustration of the average matrices for SC, CHS, and DBI for different parameter values (i.e., *Eps* and *Minpt*) in three water areas. Specifically, the vertical axis of each matrix represents the value of the parameter *Minpt*, while the horizontal axis represents the value of the parameter *Eps*.

References

Arslan, M., Cruz, C., Roxin, A.-M., Ginhac, D., 2018. Spatio-temporal analysis of trajectories for safer construction sites. Smart Sustain. Built Environ. 7, 80–100. https://doi.org/10.1108/SASBE-10-2017-0047.

Bai, X., Xie, Z., Xu, X., Xiao, Y., 2023. An adaptive threshold fast DBSCAN algorithm with preserved trajectory feature points for vessel trajectory clustering. Ocean Eng. 280, 114930 https://doi.org/10.1016/j.oceaneng.2023.114930.

Basnet, M.B., Anas, M., Rizvi, Z.H., Ali, A.H., Zain, M., Cascante, G., Wuttke, F., 2022. Enhancement of in-plane seismic full waveform inversion with CPU and GPU parallelization. Appl. Sci. 12, 8844. https://doi.org/10.3390/app12178844.

Blu, T., Thevenaz, P., Unser, M., 2004. Linear interpolation revitalized. IEEE Trans. Image Process. 13, 710–719. https://doi.org/10.1109/TIP.2004.826093.

Cagigas-Muñiz, D., Diaz-del-Rio, F., Sevillano-Ramos, J.L., Guisado-Lizar, J.-L., 2022. Efficient simulation execution of cellular automata on GPU. Simul. Model. Pract. Theory 118, 102519. https://doi.org/10.1016/j.simpat.2022.102519.

Chen, C., Li, K., Ouyang, A., Zeng, Z., Li, K., 2018. GFlink: An in-memory computing architecture on heterogeneous CPU-GPU clusters for big data. IEEE Trans. Parallel Distrib. Syst. 29, 1275–1288. https://doi.org/10.1109/TPDS.2018.2794343.

Chen, C., Ding, Y., Wang, Z., Zhao, J., Guo, B., Zhang, D., 2020a. VTracer: when online vehicle trajectory compression meets mobile edge computing. IEEE Syst. J. 14, 1635–1646. https://doi.org/10.1109/JSYST.2019.2935458.

Chen, C., Ding, Y., Xie, X., Zhang, S., Wang, Z., Feng, L., 2020b. TrajCompressor: an online map-matching-based trajectory compression framework leveraging vehicle heading direction and change. IEEE Trans. Intell. Transp. Syst. 21, 2012–2028. https://doi.org/10.1109/TITS.2019.2910591.

Cheng, J.R., Gen, M., 2019. Accelerating genetic algorithms with GPU computing: a selective overview. Comput. Ind. Eng. 128, 514–525. https://doi.org/10.1016/j.cie.2018.12.067.

Dogancay, K., Tu, Z., Ibal, G., 2021. Research into vessel behaviour pattern recognition in the maritime domain: past, present and future. Digital Signal Process. 119, 103191 https://doi.org/10.1016/j.dsp.2021.103191.

Douglas, D.H., Peucker, T.K., 1973. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. Cartographica 10, 112–122. https://doi.org/10.3138/FM57-6770-U75U-7727.

Gao, M., Shi, G.-Y., 2019. Ship spatiotemporal key feature point online extraction based on AIS multi-sensor data using an improved sliding window algorithm. Sensors 19, 2706. https://doi.org/10.3390/s19122706.

Gao, C., Zhao, Y., Wu, R., Yang, Q., Shao, J., 2019. Semantic trajectory compression via multi-resolution synchronization-based clustering. Knowl.-Based Syst. 174, 177–193. https://doi.org/10.1016/j.knosys.2019.03.006.

Gu, Q., Zhen, R., Liu, J., Li, C., 2023. An improved RRT algorithm based on prior AIS information and DP compression for ship path planning. Ocean Eng. 279, 114595 https://doi.org/10.1016/j.oceaneng.2023.114595.

Han, W., Deng, Z., Chu, J., Zhu, J., Gao, P., Shah, T., 2018. A parallel online trajectory compression approach for supporting big data workflow. Computing 100, 3–20. https://doi.org/10.1007/s00607-017-0563-8.

Hansuddhisuntorn, K., Horanont, T., 2019. Improvement of TD-TR Algorithm for Simplifying GPS Trajectory Data, in: 2019 First International Conference on Smart Technology & Urban Development (STUD). Presented at the 2019 First International Conference on Smart Technology & Urban Development (STUD), pp. 1–6. DOI: 10.1109/STUD49732.2019.9018800.

Heywood, P., Maddock, S., Bradley, R., Swain, D., Wright, I., Mawson, M., Fletcher, G., Guichard, R., Himlin, R., Richmond, P., 2019. A data-parallel many-source shortest-path algorithm to accelerate macroscopic transport network assignment. Transport. Res. Part c: Emerg. Technol. 104, 332–347. https://doi.org/10.1016/j.trc.2019.05.020.

Huang, Y., Li, Y., Zhang, Z., Liu, R.W., 2020. GPU-Accelerated compression and visualization of large-scale vessel trajectories in maritime IoT industries. IEEE Internet Things J. 7, 10794–10812. https://doi.org/10.1109/JIOT.2020.2989398.

Jeong, I., Oh, Y., Ro, W.W., Yoon, M.K., 2022. TEA-RC: thread context-aware register cache for GPUs. IEEE Access 10, 82049–82062. https://doi.org/10.1109/ACCESS.2022.3196149.

Jurczuk, K., Czajkowski, M., Kretowski, M., 2021. Multi-GPU approach to global induction of classification trees for large-scale data mining. Appl Intell 51, 5683–5700. https://doi.org/10.1007/s10489-020-01952-5.

Kallioras, N.A., Kepaptsoglou, K., Lagaros, N.D., 2015. Transit stop inspection and maintenance scheduling: A GPU accelerated metaheuristics approach. Transport. Res. Part c: Emerg. Technol. 55, 246–260. https://doi.org/10.1016/j.trc.2015.02.013.

Karataş, G.B., Karagoz, P., Ayran, O., 2021. Trajectory pattern extraction and anomaly detection for maritime vessels. Internet of Things 16, 100436. https://doi.org/10.1016/j.iot.2021.100436.

Keogh, E., Chu, S., Hart, D., Pazzani, M., 2001. An online algorithm for segmenting time series. In: Proceedings 2001 IEEE International Conference on Data Mining. Presented at the Proceedings 2001 IEEE International Conference on Data Mining, pp. 289–296. DOI: 10.1109/ICDM.2001.989531.

Lahreche, A., Boucheham, B., 2021. A fast and accurate similarity measure for long time series classification based on local extrema and dynamic time warping. Expert Syst. Appl. 168, 114374 https://doi.org/10.1016/j.eswa.2020.114374.

Li, C., Bai, L., Liu, W., Yao, L., Travis Waller, S., 2021. Urban mobility analytics: A deep spatial–temporal product neural network for traveler attributes inference. Transport. Res. Part c: Emerg. Technol. 124, 102921 https://doi.org/10.1016/j.trc.2020.102921.

Li, Y., Liu, R.W., Liu, J., Huang, Y., Hu, B., Wang, K., 2016. Trajectory compression-guided visualization of spatio-temporal AIS vessel density, in: 2016 8th International Conference on Wireless Communications & Signal Processing (WCSP). Presented at the 2016 8th International Conference on Wireless Communications & Signal Processing (WCSP), pp. 1–5. DOI: 10.1109/WCSP.2016.7752733.

Li, L., Xia, X., Liu, X., An, Y., 2019. Batched Trajectory Compression Algorithm Based on Hierarchical Grid Coordinates, in: 2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS). Presented at the 2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS), pp. 414–418. DOI: 10.1109/ICSESS47205.2019.9040741.

Li, H., Lam, J.S.L., Yang, Z., Liu, J., Liu, R.W., Liang, M., Li, Y., 2022. Unsupervised hierarchical methodology of maritime traffic pattern extraction for knowledge discovery. Transport. Res. Part C: Emerg. Technol. 143, 103856 https://doi.org/10.1016/j.trc.2022.103856.

Li, Y., Liang, M., Li, H., Yang, Z., Du, L., Chen, Z., 2023. Deep learning-powered vessel traffic flow prediction with spatial-temporal attributes and similarity grouping. Eng. Appl. Artif. Intell. 126, 107012. https://doi.org/10.1016/j.engappai.2023.107012.

Li, H., Liu, J., Yang, Z., Liu, R.W., Wu, K., Wan, Y., 2020. Adaptively constrained dynamic time warping for time series classification and clustering. Inf. Sci. 534, 97–116. https://doi.org/10.1016/j.ins.2020.04.009.

Li, H., Xing, W., Jiao, H., Yang, Z., Li, Y., 2024. Deep bi-directional informationempowered ship trajectory prediction for maritime autonomous surface ships. Transp. Res. Pt. E: Logist. Transp. Rev. 181, 103367. https://doi.org/10.1016/j.

Li, H., Yang, Z., 2023. Incorporation of AIS data-based machine learning into unsupervised route planning for maritime autonomous surface ships. Transport. Res. Part E: Logist. Transportat. Rev. 176, 103171 https://doi.org/10.1016/j.tre.2023.103171.

Li, H., Jiao, H., Yang, Z., 2023a. Ship trajectory prediction based on machine learning and deep learning: a systematic review and methods analysis. Eng. Appl. Artif. Intel. 126, 107062 https://doi.org/10.1016/j.engappai.2023.107062.

Li, H., Jiao, H., Yang, Z., 2023b. AIS data-driven ship trajectory prediction modelling and analysis based on machine learning and deep learning methods. Transport. Res. Part E: Logist. Transport. Rev. 175, 103152 https://doi.org/10.1016/j.tre.2023.103152.

Li, Q., Lam, J.S.L., 2017. Conflict resolution for enhancing shipping safety and improving navigational traffic within a seaport: vessel arrival scheduling. Transport. A: Transp. Sci. 13, 727–741. https://doi.org/10.1080/23249935.2017.1326068.

Liang, M., Liu, R.W., Zhan, Y., Li, H., Zhu, F., Wang, F.-Y., 2022. Fine-grained vessel traffic flow prediction with a spatio-temporal multigraph convolutional network. IEEE Trans. Intell. Transp. Syst. 23, 23694–23707. https://doi.org/10.1109/TITS.2022.3199160.

Lin, S., Liu, J., Young, E.F.Y., Wong, M.D.F., 2023. GAMER: GPU-accelerated maze routing. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 42, 583–593. https://doi.org/10.1109/TCAD.2022.3184281.

Liu, Y., Yang, Z., 2023. Improved Opening Window Trajectory Simplification Algorithm, in: 2023 5th International Conference on Electrical Engineering and Control Technologies (CEECT). Presented at the 2023 5th International Conference on Electrical Engineering and Control Technologies (CEECT), IEEE, Chengdu, China, pp. 230–234. DOI: 10.1109/CEECT59667.2023.10420704.

Liu, J., Zhao, K., Sommer, P., Shang, S., Kusy, B., Jurdak, R., 2015. Bounded Quadrant System: Error-bounded trajectory compression on the go, in: 2015 IEEE 31st International Conference on Data Engineering. Presented at the 2015 IEEE 31st International Conference on Data Engineering, pp. 987–998. DOI: 10.1109/ICDE.2015.7113350.

Liu, J., Zhao, K., Sommer, P., Shang, S., Kusy, B., Lee, J.-G., Jurdak, R., 2016. A novel framework for online amnesic trajectory compression in resource-constrained environments. IEEE Trans. Knowl. Data Eng. 28, 2827–2841. https://doi.org/10.1109/TKDE.2016.2598171.

Liu, J., Li, H., Yang, Z., Wu, K., Liu, Y., Liu, R.W., 2019b. Adaptive douglas-peucker algorithm with automatic thresholding for AIS-based vessel trajectory compression. IEEE Access 7, 150677–150692. https://doi.org/10.1109/ACCESS.2019.2947111.

Liu, C., Mao, Q., Chu, X., Xie, S., 2019a. An improved A-star algorithm considering water current, traffic separation and berthing for vessel path planning. Appl. Sci. 9, 1057. https://doi.org/10.3390/app9061057.

Lv, C., Chen, F., Xu, Y., Song, J., Lv, P., 2015. A trajectory compression algorithm based on non-uniform quantization, in: 2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD). Presented at the 2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), pp. 2469–2474. DOI: 10.1109/FSKD.2015.7382342.

Manduhu, M., Jones, M.W., 2019. A work efficient parallel algorithm for exact euclidean distance transform. IEEE Trans. Image Process. 28, 5322–5335. https://doi.org/10.1109/TIP.2019.2916741.

Meratnia, N., de By, R.A., 2004. Spatiotemporal Compression Techniques for Moving Point Objects, in: Bertino, E., Christodoulakis, S., Plexousakis, D., Christophides, V., Koubarakis, M., Böhm, K., Ferrari, E. (Eds.), Advances in Database Technology - EDBT 2004, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 765–782. DOI: 10.1007/978-3-540-24741-8_44.

Muckell, J., Hwang, J.-H., Patil, V., Lawson, C.T., Ping, F., Ravi, S.S., 2011. SQUISH: an online approach for GPS trajectory compression, in: Proceedings of the 2nd International Conference on Computing for Geospatial Research & Applications, COM.Geo '11. Association for Computing Machinery, New York, NY, USA, pp. 1–8. DOI: 10.1145/1999320.1999333.

Muckell, J., Olsen, P.W., Hwang, J.-H., Lawson, C.T., Ravi, S.S., 2014. Compression of trajectory data: a comprehensive evaluation and new approach. GeoInformatica 18, 435–460. https://doi.org/10.1007/s10707-013-0184-0.

Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A.E., Purcell, T.J., 2007. A survey of general-purpose computation on graphics hardware. Comput. Graphics Forum 26, 80–113. https://doi.org/10.1111/j.1467-8659.2007.01012.x.

Potamias, M., Patroumpas, K., Sellis, T., 2006. Sampling Trajectory Streams with Spatiotemporal Criteria, in: 18th International Conference on Scientific and Statistical Database Management (SSDBM'06). Presented at the 18th International Conference on Scientific and Statistical Database Management (SSDBM'06), pp. 275–284. DOI: 10.1109/SSDBM.2006.45.
Qu, Y., Zhou, X., 2017. Large-scale dynamic transportation network simulation: a space-time-event parallel computing approach. Transport. Res. Part c: Emerg. Technol. 75, 1–16. https://doi.org/10.1016/j.trc.2016.12.003.
Roberge, V., Tarbouchi, M., 2021. Parallel algorithm on GPU for wireless sensor data acquisition using a team of unmanned aerial vehicles. Sensors 21, 6851. https://doi.org/10.3390/s21206851.
Rong, H., Teixeira, A.P., Guedes Soares, C., 2020. Data mining approach to shipping route characterization and anomaly detection based on AIS data. Ocean Eng. 198, 106936 https://doi.org/10.1016/j.oceaneng.2020.106936.
Saalfeld, A., 1999. Topologically consistent line simplification with the douglas-peucker algorithm. Cartogr. Geogr. Inf. Sci. 26, 7–18. https://doi.org/10.1559/152304099782424901.
Shanbhag, A., Yogatama, B.W., Yu, X., Madden, S., 2022. Tile-based Lightweight Integer Compression in GPU, in: Proceedings of the 2022 International Conference on Management of Data, SIGMOD '22. Association for Computing Machinery, New York, NY, USA, pp. 1390–1403. DOI: 10.1145/3514221.3526132.
Sun, S., Chen, Y., Piao, Z., Zhang, J., 2020. Vessel AIS trajectory online compression based on scan-pick-move algorithm added sliding window. IEEE Access 8, 109350–109359. https://doi.org/10.1109/ACCESS.2020.3001934.
Sun, P., Xia, S., Yuan, G., Li, D., 2016. An overview of moving object trajectory compression algorithms. Math. Probl. Eng. 2016, e6587309.
Świrydowicz, K., Darve, E., Jones, W., Maack, J., Regev, S., Saunders, M.A., Thomas, S.J., Peleš, S., 2022. Linear solvers for power grid optimization problems: A review of GPU-accelerated linear solvers. Parallel Comput. 111, 102870 https://doi.org/10.1016/j.parco.2021.102870.
Tagiltseva, J., Vasilenko, M., Kuzina, E., Drozdov, N., Parkhomenko, R., Prokopchuk, V., Skichko, E., Bagiryan, V., 2022. The economic efficiency justification of multimodal container transportation. Transportation Research Procedia, X International Scientific Siberian Transport Forum — TransSiberia 2022 63, 264–270. DOI: 10.1016/j.trpro.2022.06.012.
Tang, C., Chen, M., Zhao, J., Liu, T., Liu, K., Yan, H., Xiao, Y., 2021a. A novel ship trajectory clustering method for Finding Overall and Local Features of Ship Trajectories. Ocean Eng. 241, 110108 https://doi.org/10.1016/j.oceaneng.2021.110108.
Tang, C., Wang, H., Zhao, J., Tang, Y., Yan, H., Xiao, Y., 2021b. A method for compressing AIS trajectory data based on the adaptive-threshold Douglas-Peucker algorithm. Ocean Eng. 232, 109041 https://doi.org/10.1016/j.oceaneng.2021.109041.
Wang, C., Wang, N., Gao, H., Wang, L., Zhao, Y., Fang, M., 2024. Knowledge transfer enabled reinforcement learning for efficient and safe autonomous ship collision avoidance. International Journal of Machine Learning and Cybernetics 1–17. https://doi.org/10.1007/s13042-024-02116-4.
Wang, C., Zhang, X., Gao, H., Bashir, M., Li, H., Yang, Z., 2024. Optimizing Anti-collision Strategy for MASS: A Safe Reinforcement Learning Approach to Improve Maritime Traffic Safety. Ocean and Coastal Management 253, 107161. https://doi.org/10.1016/j.ocecoaman.2024.107161.
Wang, C., Zhang, X., Yang, Z., Bashir, M., Lee, K., 2023. Collision avoidance for autonomous ship using deep reinforcement learning and prior-knowledge-based approximate representation. Frontiers in Marine Science 9, 1084763. https://doi.org/10.3389/fmars.2022.1084763.
Xin, X., Liu, K., Loughney, S., Wang, J., Li, H., Ekere, N., Yang, Z., 2023. Multi-scale collision risk estimation for maritime traffic in complex port waters. Reliability Engineering & System Safety 240, 109554. https://doi.org/10.1016/j.ress.2023.109554.
Xin, X., Liu, K., Loughney, S., Wang, J., Li, H., Yang, Z., 2023. Graph-based ship traffic partitioning for intelligent maritime surveillance in complex port waters. Expert Syst. Appl. 231, 120825. https://doi.org/10.1016/j.eswa.2023.120825.
Yan, Z., Xiao, Y., Cheng, L., He, R., Ruan, X., Zhou, X., Li, M., Bin, R., 2020. Exploring AIS data for intelligent maritime routes extraction. Appl. Ocean Res. 101, 102271 https://doi.org/10.1016/j.apor.2020.102271.
Yang, D., Wu, L., Wang, S., Jia, H., Li, K.X., 2019. How big data enriches maritime research – a critical review of Automatic Identification System (AIS) data applications. Transp. Rev. 39, 755–773. https://doi.org/10.1080/01441647.2019.1649315.
You, L., Jiang, H., Hu, J., Chang, C.H., Chen, L., Cui, X., Zhao, M., 2022. GPU-accelerated Faster Mean Shift with euclidean distance metrics, in: 2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC). Presented at the 2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC), pp. 211–216. DOI: 10.1109/COMPSAC54236.2022.00037.
Zhao, L., Shi, G., 2018. A method for simplifying ship trajectory based on improved Douglas-Peucker algorithm. Ocean Eng. 166, 37–46. https://doi.org/10.1016/j.oceaneng.2018.08.005.
Zheng, K., Zhao, Y., Lian, D., Zheng, B., Liu, G., Zhou, X., 2020. Reference-based framework for spatio-temporal trajectory compression and query processing. IEEE Trans. Knowl. Data Eng. 32, 2227–2240. https://doi.org/10.1109/TKDE.2019.2914449.
Zhou, Z., Zhang, Y., Yuan, X., Wang, H., 2023. Compressing AIS trajectory data based on the multi-objective peak Douglas-Peucker algorithm. IEEE Access 11, 6802–6821. https://doi.org/10.1109/ACCESS.2023.3234121.
Zhu, F., Ma, Z., 2021. Ship trajectory online compression algorithm considering handling patterns. IEEE Access 9, 70182–70191. https://doi.org/10.1109/ACCESS.2021.3078642.