International Journal of Computer Vision (2024) 132:1463–1483
https://doi.org/10.1007/s11263-023-01933-4

FastTrack: A Highly Efficient and Generic GPU-Based Multi-object Tracking Method with Parallel Kalman Filter

Chongwei Liu[1] · **Haojie Li**[2] · **Zhihui Wang**[1]

Received: 27 July 2022 / Accepted: 17 October 2023 / Published online: 21 November 2023

Abstract

The Kalman Filter based on uniform assumption has been a crucial motion estimation module in trackers. However, it has limitations in non-uniform motion modeling and computational efficiency when applied to large-scale object tracking scenarios. To address these issues, we propose a novel ***Parallel Kalman Filter (PKF)***, which simplifies conventional state variables to reduces computational load and enable effective non-uniform modeling. Within PKF, we propose a non-uniform formulation which models non-uniform motion as uniform motion by transforming the time interval Δt from a constant into a variable related to displacement, and incorporate a deceleration strategy into the control-input model of the formulation to tackle the escape problem in Multi-Object Tracking (MOT); an innovative parallel computation method is also proposed, which transposes the computation graph of PKF from the matrix to the quadratic form, significantly reducing the computational load and facilitating parallel computation between distinct tracklets via CUDA, thus making the time consumption of PKF independent of the input tracklet scale, i.e., $O(1)$. Based on PKF, we introduce ***Fast***, *the first fully GPU-based tracker paradigm*, which significantly enhances tracking efficiency in large-scale object tracking scenarios; and ***FastTrack***, the MOT system composed of Fast and a general detector, offering high efficiency and generality. Within FastTrack, Fast only requires bounding boxes with scores and class ids for a single association during one iteration, and introduces innovative GPU-based tracking modules, such as an efficient GPU 2D-array data structure for tracklet management, a novel cost matrix implemented in CUDA for automatic association priority determination, a new association metric called HIoU, and the first implementation of the Auction Algorithm in CUDA for the asymmetric assignment problem. Experiments show that the average time per iteration of PKF on a GTX 1080Ti is only 0.2 ms; Fast can achieve a real-time efficiency of 250FPS on a GTX 1080Ti and 42FPS even on a Jetson AGX Xavier, outperforming conventional CPU-based trackers. Concurrently, FastTrack demonstrates state-of-the-art performance on four public benchmarks, specifically MOT17, MOT20, KITTI, and DanceTrack, and attains the highest speed in large-scale tracking scenarios of MOT20.

Keywords Multi-object tracking · GPU-based tracker · Parallel Kalman filter · Real-time efficiency

1 Introduction

Multi-Object Tracking (MOT) predominantly follows the tracking-by-detection paradigm. An MOT system typically comprises a general detector (Ren et al., 2015; Ge et al., 2021) and a generic[1] motion-based tracker (Zhang et al., 2022; Cao et al., 2022; Bewley et al., 2016).

Although the Kalman Filter (KF) is a crucial motion estimation module for many state-of-the-art trackers (Wang et al., 2020; Zhou et al., 2020; Zhang et al., 2021, 2022), it has

[1] The term "generic" implies that a tracker can easily be combined with any general object detector for object tracking.

Communicated by Svetlana Lazebnik.

✉ Haojie Li
hjli@sdust.edu.cn

Chongwei Liu
lcwdllg@mail.dlut.edu.cn

Zhihui Wang
zhwang@dlut.edu.cn

[1] DUT-RU International School of Information Science and Engineering, Dalian University of Technology, Dalian, Liaoning, China

[2] College of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao, Shandong, China

limitations in motion modeling and computational efficiency when applied to MOT.

Concerning motion modeling, the KF assumes uniform object motion, which is unsuitable for various motion patterns in general tracking scenes[2] as shown in Fig. 1. Although Extended KF (Smith et al., 1962) and Unscented KF (Julier & Uhlmann, 1997) were introduced to handle non-uniform motions of Taylor approximations, they are computationally complex and cannot estimate arbitrary non-uniform motion, such as the highly random motions of dancers in stage scenes. Additionally, the KF is sensitive to noise, leading to the escape problem (Cao et al., 2022). When an object is lost, its bounding box, continuously predicted by KF without observation information supervision, rapidly escapes along the current velocity direction, making it difficult to retrace. For example, in Fig. 1, when an object is lost at a certain circular point, the box predicted by KF rapidly escapes along the velocity direction.

Concerning computational efficiency, each tracklet is represented using a distinct KF, and all KFs are updated in a sequential manner. This process leads to a linear rise in time consumption of all KFs proportional to the count of input tracklets. This computational expense corresponds to a time complexity of $O(n)$, where n denotes the total number of tracklets. Such substantial resource allocation weakens tracking efficiency, especially in scenarios of large-scale object tracking. For example, the CPU-based tracking algorithm OCSORT (Cao et al., 2022), despite improving the KF to attain state-of-the-art performance, falls short in terms of computational efficiency. As a result, OCSORT experiences nearly a 30× increase in time consumption when the number of input tracklets increases from 6 (as in KITTI) to 139 (as in MOT20), as illustrated in Fig. 2. Thus, an optimal tracker must strike a balance between tracking precision and computational efficiency.

To address these issues, we introduce a novel ***Parallel Kalman Filter (PKF)*** that models non-uniform motion while achieving a time complexity of $O(1)$. In modeling, the importance of a suitable set of state variables for the KF cannot be understated. Therefore, we revise the conventional eight-tuple state variables and replace them with a more simplified four-tuple state variable set, which focuses specifically on the 2D coordinates of the object center along with their corresponding velocities. This simplification reduces computational load and aligns more appropriately with the assumption of adjacent frame approximation of objects, thus enabling effective modeling of non-uniform motion. Taking cues from the fundamental motion equation $V = S/T$, we propose a unique state transfer formulation, called the non-uniform formulation, based on the simplified state variables. It models non-uniform motion as uniform motion by transforming the time interval Δt from a constant into a variable related to displacement. Moreover, to address the escape problem, we incorporate a deceleration strategy into the control-input model of our proposed formulation. In computation, reducing the computational load and increasing parallel computation are the two main approaches to improving computational efficiency. By simplifying state variables and adhering to the strict matrix representation of our non-uniform formulation, we introduce an innovative parallel computation method. This method transposes the compu-

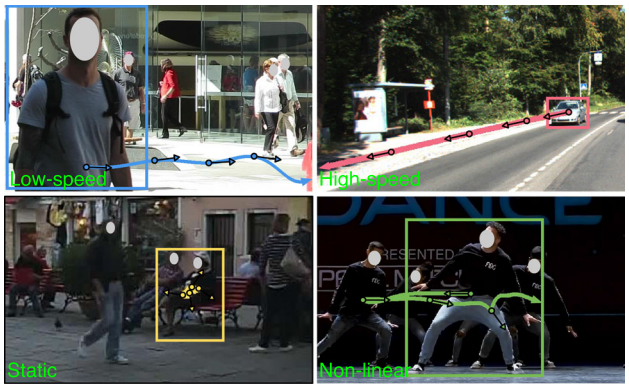

Fig. 1 Illustration of object trajectories in typical tracking scenarios. Four prevalent motion patterns are displayed: Low-speed, High-speed, Static, and Non-linear. The bold colored lines represent the central coordinate trajectories of the objects, while the black arrow lines indicate the direction of the object's velocity at specific circular points

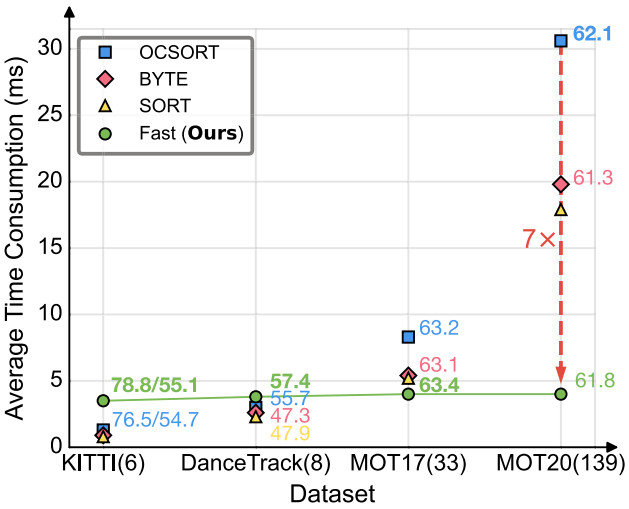

Fig. 2 Efficiency and HOTA comparisons between our method and other CPU&motion-based trackers across various benchmarks. The number in parentheses represents the average number of tracklets per frame for the respective test set. In the case of KITTI, the two scores correspond to HOTA for Car and Pedestrian categories, respectively

[2] The term "general tracking scene" refers to one of the most common cases of MOT, i.e., scenes containing people or vehicles captured by handheld or fixed cameras. For this study's convenience, it is equivalent to the set of MOT17 (Milan et al., 2016), MOT20 (Dendorfer et al., 2020), KITTI (Geiger et al., 2013), and DanceTrack (Sun et al., 2021) in this paper.

tation graph of PKF from the matrix to the quadratic form, significantly reducing the computational load and facilitating parallel computation between distinct tracklets via CUDA. Consequently, the time consumption of the PKF becomes independent of the input tracklet scale, i.e., $O(1)$. Overall, within the scope of PKF, the simplified state variables serve as the cornerstone for both modeling and computation. The design of the non-uniform formulation facilitates parallel computing, and the practical implementation of parallel computing significantly accelerates the non-uniform formulation across diverse tracklets.

Although PKF can achieve high tracking efficiency through CUDA acceleration, the other conventional modules of the tracker remain CPU-based, leading to a bottleneck in large-scale object tracking. To further improve tracking efficiency in large-scale object tracking scenarios, we introduce ***Fast***, *the first fully GPU-based tracker paradigm*, based on PKF; and ***FastTrack***, the MOT system composed of Fast and a general detector, offering high efficiency and generality. Within FastTrack, Fast only requires bounding boxes with scores and class ids to perform a single association during one iteration, allowing for enhanced efficiency and generality.

Within Fast, we propose corresponding GPU-based modules to replace the conventional CPU-based modules. We innovatively introduce a highly efficient GPU 2D-array data structure to manage tracklets instead of instances like most previous works (Zhang et al., 2022; Cao et al., 2022; Bewley et al., 2016), enabling efficient parallel access. Furthermore, we propose a novel cost matrix implemented in CUDA, capable of automatically determining association priorities based on scores within a single association. This novel cost matrix also facilitates multi-object and multi-class tracking by simply shifting all boxes along the x-axis by the distance of class id times the input image width before calculating the Intersection over Union (IoU). Additionally, we propose a new association metric, HIoU, to replace IoU when tracking pedestrian or traffic scenes. Lastly, we implement the Auction Algorithm (Bertsekas, 1992a) for the asymmetric assignment problem using CUDA for the first time, replacing conventional CPU-based linear assignment algorithms such as the Hungarian Algorithm (Kuhn, 1955) or LAPJV (Jonker and Volgenant, 1987).

The conducted experiments demonstrate that the average time per iteration of PKF on GTX 1080Ti is only 0.2 ms and is independent of the input scale. Based on PKF and other proposed modules, Fast can achieve a real-time efficiency of 250FPS on GTX 1080Ti and 42FPS even on the embedded CUDA device Jetson AGX Xavier. The efficiency is **unaffected** by the number of tracklets even on the MOT20 dataset with 139 objects per frame on average, which has never been achieved in conventional CPU-based trackers. As shown in Fig. 2, Fast is 7× faster than the state-of-the-art CPU&motion-based tracker OCSORT in large-scale tracking scenes of MOT20 and obtains the state-of-the-art performance on four benchmarks, i.e., MOT17 (Milan et al., 2016), MOT20 (Dendorfer et al., 2020), KITTI (Geiger et al., 2013), and DanceTrack (Sun et al., 2021).

In summary, our work presents three significant contributions:

- We propose a novel Parallel Kalman Filter (PKF) that models non-uniform motion and achieves a time complexity of $O(1)$. PKF modifies the conventional state variables, proposes a non-uniform formulation, incorporates a deceleration strategy to tackle the escape problem, and leverages a parallel computation method to reduce computational load.
- We introduce the first fully GPU-based tracker paradigm called Fast, which greatly improves tracking efficiency in large-scale object tracking scenarios; and FastTrack, the MOT system consisting of Fast and a general detector, allowing for high efficiency and generality. Within FastTrack, Fast only requires bounding boxes with scores and class ids to perform a single association during one iteration.
- We propose innovative GPU-based modules within Fast to replace conventional CPU-based modules, such as a highly efficient GPU 2D-array data structure for managing tracklets, a novel cost matrix implemented in CUDA for automatic association priority determination, a novel association metric HIoU, and the first implementation of the Auction Algorithm via CUDA for the asymmetric assignment problem. These GPU-based modules contribute to the real-time efficiency and generality of FastTrack in large-scale object tracking scenarios.

2 Related Works

2.1 Tracking-by-Detection

Tracking-by-detection has become the dominant paradigm in the MOT task. This paradigm divides an MOT system into two separate parts, i.e., the detector and the tracker. In the basic case, the detector provides the tracker with detection results (bounding boxes with confidence scores and class ids) for each video frame, and the tracker uses motion estimation to achieve tracking. In recent years, with the rapid development of object detection, more general object detectors (Redmon and Farhadi, 2018; Bochkovskiy et al., 2020) have achieved both high recall and high precision. Consequently, numerous tracking methods (Lu et al., 2020; Peng et al., 2020; Zhou et al., 2020; Wu et al., 2021; Zhang et al., 2022) have started utilizing powerful detectors (e.g., RetinaNet (Lin et al., 2017), CenterNet (Zhou et al., 2019), or

YOLOX (Ge et al., 2021) to obtain superior tracking performance. It has become a trend to combine high-performance detectors with concise and generic motion-based trackers into MOT systems. For instance, SORT (Bewley et al., 2016) first employed Faster R-CNN (Ren et al., 2015) as its detector and the Kalman Filter (Kalman, 1960) as its motion estimation module, achieving state-of-the-art performance in 2016 with a simple and efficient tracker. Building on SORT, ByteTrack (Zhang et al., 2022) achieved state-of-the-art results in MOT17 and MOT20 by using the advanced detector YOLOX (Ge et al., 2021). Before ByteTrack (Zhang et al., 2022), many methods employing RetinaNet or CenterNet opted to directly filter low score boxes (scores below 0.5) to eliminate most False Positive (FP) boxes and guarantee tracking performance due to the low precision of detectors at the time. However, the high recall and precision of YOLOX ensure that even low score boxes are likely to be True Positive (TP) boxes. Therefore, ByteTrack achieves state-of-the-art performance while ensuring simplicity and efficiency by employing YOLOX and cascade association based on score. In our approach, Fast, we take it a step further by exploiting the score, i.e., by fusing tracklet and detection scores into the cost matrix to automatically prioritize matches within a single association.

In addition, the tracker is essentially a computationally intensive task, but current trackers are primarily CPU-based and implemented with object-oriented programming. GPUs have not been well-explored for tracker implementation due to the programming gap between GPU and CPU. In this paper, we propose the first fully GPU-based tracker paradigm, which significantly improves tracking efficiency in large-scale object tracking scenarios.

2.2 Kalman Filter

Introduction The Kalman Filter (Bishop et al., 2001) is a classical motion estimation algorithm consisting of two phases: prediction and update. In the prediction phase, the KF uses the previous state to estimate the current state. The update phase incorporates observations of the current state to provide a more accurate state estimate.

At each iteration, two variables are maintained for each tracked object: the state estimate $\mathbf{x}$ and its posterior estimated error covariance matrix $\mathbf{P}$. The prediction phase of the KF is characterized by the state-transition model $\mathbf{F}$, the control-input model $\mathbf{B}$ with the control vector $\mathbf{u}$, and the covariance of the process noise $\mathbf{Q}$. The update phase is described by the observation model $\mathbf{H}$ and the covariance of the observation noise $\mathbf{R}$.

At each time step t, the KF first predicts the state estimate $\mathbf{x}_{t|t-1}$ and its covariance matrix $\mathbf{P}_{t|t-1}$ using the following equations:

$$\mathbf{x}_{t|t-1} = \mathbf{F}_t \mathbf{x}_{t-1} + \mathbf{B}_t \mathbf{u}_t, \tag{1a}$$

$$\mathbf{P}_{t|t-1} = \mathbf{F}_t \mathbf{P}_{t-1} \mathbf{F}_t^\top + \mathbf{Q}_t, \tag{1b}$$

where Eq. 1a models the object motion with a state transfer formulation.

The KF then updates the state estimate and covariance matrix based on the observation $\mathbf{z}_t$ to obtain more accurate estimates ($\mathbf{x}_t$ and $\mathbf{P}_t$) using the following equations:

$$\mathbf{S}_t = \mathbf{H}_t \mathbf{P}_{t|t-1} \mathbf{H}_t^\top + \mathbf{R}_t, \tag{2a}$$

$$\mathbf{K}_t = \mathbf{P}_{t|t-1} \mathbf{H}_t^\top \mathbf{S}_t^{-1}, \tag{2b}$$

$$\mathbf{x}_t = \mathbf{x}_{t|t-1} + \mathbf{K}_t (\mathbf{z}_t - \mathbf{H}_t \mathbf{x}_{t|t-1}), \tag{2c}$$

$$\mathbf{P}_t = \mathbf{P}_{t|t-1} - \mathbf{K}_t \mathbf{S}_t \mathbf{K}_t^\top, \tag{2d}$$

where the Kalman gain is denoted by matrix $\mathbf{K}$, and the system uncertainty is represented by matrix $\mathbf{S}$, which is the projected $\mathbf{P}$ in the measurement space. Additionally, Eq. 2d can also be expressed as:

$$\mathbf{P}_t = (\mathbf{I} - \mathbf{K}_t \mathbf{H}_t) \mathbf{P}_{t|t-1}, \tag{3}$$

where the identity matrix is denoted by $\mathbf{I}$. The corresponding proof can be found on the following website.[3]

The uniform motion assumption of the KF is restrictive, leading to the development of the Extended KF (EKF) (Smith et al., 1962) and Unscented KF (UKF) (Julier & Uhlmann, 1997) to handle non-uniform motion through first- and third-order Taylor approximations. However, these methods still rely on the Gaussian approximation under the KF assumption and cannot accurately estimate arbitrary non-uniform motions, such as the highly random motion of dancers. Particle filters (Gustafsson et al., 2002) address non-uniform motions through sampling-based a posteriori estimation, but at the cost of exponential computational complexity.

Application In the context of the MOT task, SORT (Bewley et al., 2016) initially applies the KF to model objects with uniform motion, assuming that the inter-frame displacements of each object are approximately equal. DeepSORT (Wojke et al., 2017) builds on SORT by improving the representation of $\mathbf{x}$. Subsequently, most of the related works (Wang et al., 2020; Zhou et al., 2020; Zhang et al., 2021, 2022) directly employ the same KF used in DeepSORT as their motion estimation module.

In DeepSORT, the KF's state estimate $\mathbf{x}$ is an eight-tuple, $\mathbf{x} = [u, v, \gamma, h, \dot{u}, \dot{v}, \dot{\gamma}, \dot{h}]^\top$, where (u, v) represents the 2D coordinates of the object center, γ is the box aspect ratio, and

[3] robotics.stackexchange.com/questions/15393/the-final-step-in-kalman-filter-to-correct-update-the-covariance-matrix.

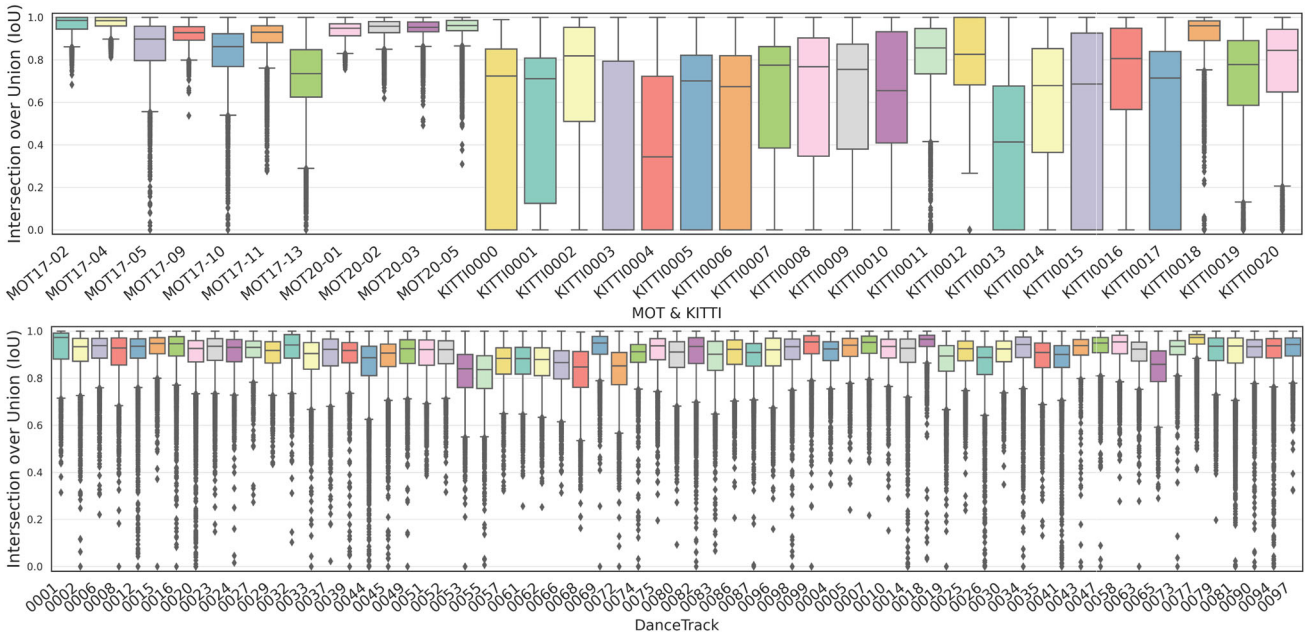

Fig. 3 IoU statistics on general tracking scenes. The top plot displays the videos from the train sets of MOT17, MOT20, and KITTI; the bottom plot exhibits the videos from the train set and the validation set of DanceTrack

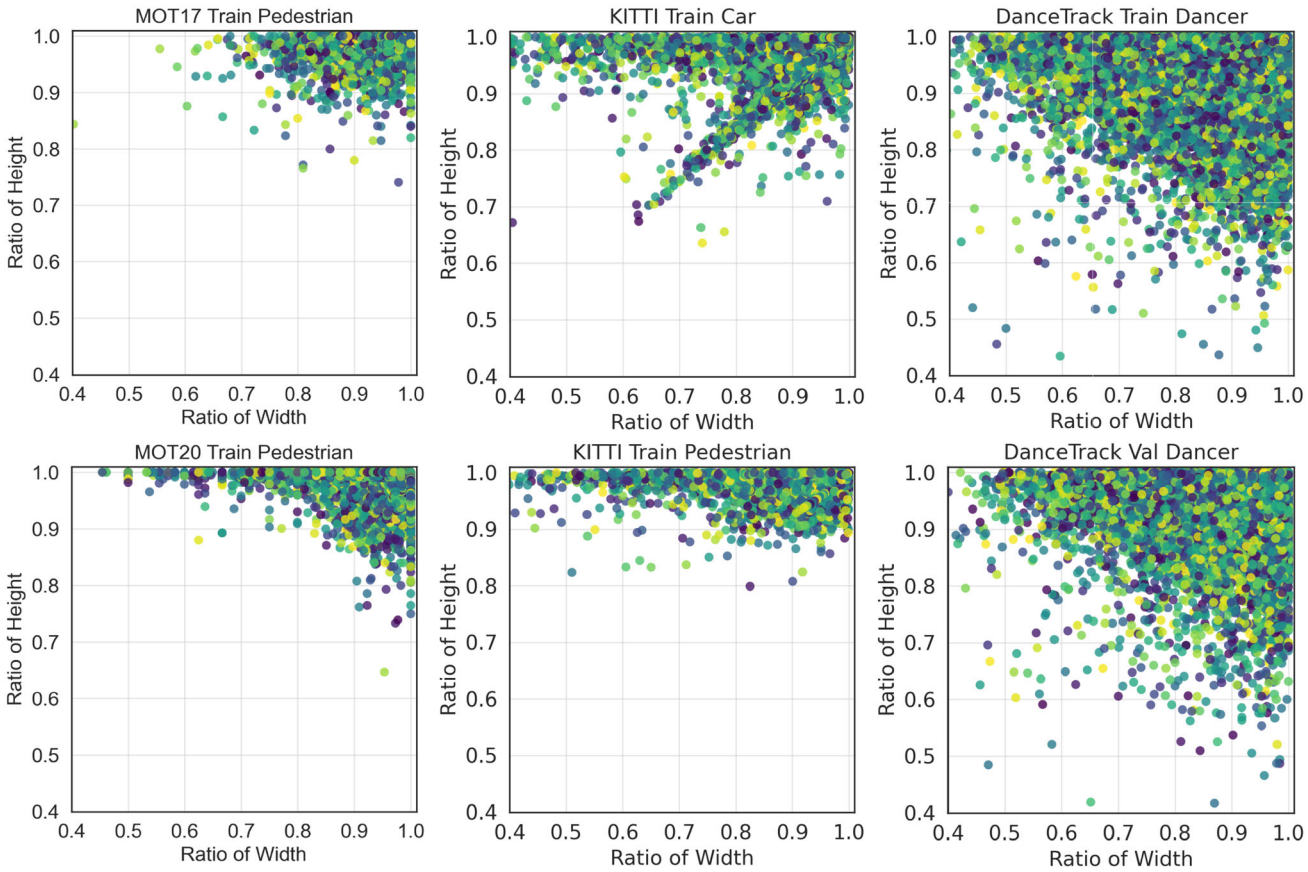

Fig. 4 Height and width ratio statistics on general tracking scenes. The title of each plot indicates the dataset and category

h is the box height. The remaining four variables with dots indicate the corresponding velocities.

DeepSORT and SORT assume uniform motion for all tracked objects. Consequently, $\mathbf{B}$ and $\mathbf{u}$ are discarded in Eq. 1a, and the state-transition model $\mathbf{F}$ becomes:

$$\mathbf{F}_t = \begin{bmatrix} \mathbf{I}_{4\times4} & \Delta t\mathbf{I}_{4\times4} \\ \mathbf{0}_{4\times4} & \mathbf{I}_{4\times4} \end{bmatrix}, \tag{4}$$

where the time difference Δt between two steps is consistent (1 by default) throughout the iterations. The process noise $\mathbf{Q}$ and the observation noise $\mathbf{R}$ are defined as follows:

$$\mathbf{Q}_t = \mathbf{diag}(\phi h^2, \phi h^2, 10^{-4}, \phi h^2, \psi h^2, \psi h^2, 10^{-10}, \psi h^2), \tag{5a}$$

$$\mathbf{R}_t = \mathbf{diag}(\phi h^2, \phi h^2, 10^{-2}, \phi h^2), \tag{5b}$$

where ϕ and ψ represent the position weight and the velocity weight, respectively. The observation model $\mathbf{H}$ is given by:

$$\mathbf{H}_t = \begin{bmatrix} \mathbf{I}_{4\times4} & \mathbf{0}_{4\times4} \end{bmatrix}. \tag{6}$$

Limitations Nevertheless, there are several issues with the aforementioned KF application.

First, the variables γ and h should not be incorporated into the state estimate $\mathbf{x}$. Due to the assumption that displacements of objects in adjacent frames are similar, γ and h should remain constant between adjacent frames and should not exhibit uniform motion. To adhere to this assumption, γ and h need to be excluded from the state estimation.

Second, the matrix $\mathbf{F}$ can only predict objects based on uniform motion, which is unsuitable for most motion patterns, particularly for non-linear objects with significant impact, such as dancers in DanceTrack.

Third, KF is sensitive to noise and thus susceptible to the escape problem when the tracked object is lost. Initially, KF operates as a predict-update loop, where the update phase is employed to supervise the predicted state estimate to correct noise. When the tracked object is lost, KF only executes the prediction phase, leading to continuous amplification of noise (visualized by the rapid escape of the predicted box) and making it challenging to retrace. Recently, OCSORT (Cao et al., 2022) introduced the Observation-centric Online Smoothing strategy to mitigate noise accumulation in KF due to a lack of observations when a lost object is retraced. However, this strategy does not enhance the probability of objects being retraced and is not GPU-friendly.

Fourth, KF is computationally demanding, with a time complexity of $O(n)$, which implies that its efficiency drastically declines as the number of tracked objects increases.

In this paper, we present the Parallel Kalman Filter to address these limitations.

Table 1 Basic information about four benchmarks

Dataset	MOT17	MOT20	DanceTrack	KITTI
Videos	14	8	100	50
Splits	7/7	4/4	40/20/40	21/29
Avg. len. (s)	35.4	66.8	52.9	38.2
Total len. (s)	463	535	5292	1910
FPS	30	25	20	10
Total images	11,235	13,410	105,855	19,103

MOT17, MOT20, and KITTI divide their videos into train/test sets, while DanceTrack separates them into train/validation/test sets. The ground truth for all test sets is not provided

2.3 Association

Association is also a core aspect of the MOT task, which primarily involves calculating a cost matrix between tracklets and detections, and then matching them based on the cost matrix. Among all the cues, position information is the most generic. In contrast to appearance or feature information, it can be directly obtained from a general object detector without the need for additional feature extraction networks or modifications to the original detection network. As a result, numerous methods (Bewley et al., 2016; Wojke et al., 2017; Zhang et al., 2022) utilize IoU to compute the cost matrix. Following the cost matrix calculation, tracklets and detections are matched using an assignment strategy. This can be achieved through classical linear assignment problem solutions such as the Hungarian Algorithm (Kuhn, 1955) or LAPJV (Jonker and Volgenant, 1987). For instance, SORT employs the Hungarian Algorithm for single association; DeepSORT (Wojke et al., 2017) uses the Hungarian Algorithm for cascade association; ByteTrack utilizes LAPJV for cascade association. However, both the Hungarian Algorithm and LAPJV are CPU-based implementations. To implement a fully GPU-based tracker, we introduce another classical solution to the linear assignment problem called the Auction Algorithm (Bertsekas, 1992a), which can be implemented on a GPU. In this paper, we successfully leverage CUDA to implement the Auction Algorithm and utilize it as the assignment strategy for our tracker paradigm.

3 Numerical Statistics

To address the modeling issues of KF, we conduct statistical analyses of general tracking scenarios to elucidate the characteristics of object motion and summarize priors.

For the sake of convenience, four benchmarks are selected for our study, namely, MOT17 (Milan et al., 2016), MOT20 (Dendorfer et al., 2020), KITTI (Geiger et al., 2013), and DanceTrack (Sun et al., 2021), with the corresponding infor-

mation presented in Table 1. MOT17 and MOT20 comprise pedestrian scenes. MOT17 contains a relatively small number of videos and scenes compared to the other datasets, while MOT20 increases the density of pedestrians and emphasizes occlusions between them. The pedestrian movements in MOT17 and MOT20 are quite regular (low-speed and nearly uniform), and they maintain distinguishable appearances. DanceTrack includes a large number of stage scenes. The similar-looking dancers in these stages move chaotically, and their movements differ significantly from frame to frame, posing considerable challenges for modeling non-uniform motion. KITTI is among the first large-scale MOT datasets for traffic scenes, focusing on tracking high-speed cars and pedestrians. In this section, we perform statistical analyses on the ground truths of the above four benchmark datasets, including the train sets of MOT17, MOT20, KITTI, and DanceTrack, as well as the validation set of DanceTrack.

3.1 Trajectory Overlap

As illustrated in Fig. 3, we count the IoU of the same trajectory in adjacent frames. We find that for low-speed objects (mostly in MOT17/20 and DanceTrack), the majority of IoU distributions lie between 0.7 and 1, indicating that low-speed objects can be easily associated between frames even without estimation. For high-speed objects (cars or pedestrians in KITTI), the uniform motion modeling of KF is crucial for their tracking because they predominantly exhibit near-linear motion, as demonstrated by the upper right car in Fig. 1.

Modeling non-uniform motion while maintaining the simplicity of KF's state transfer formulation is challenging because we cannot predict the object's motion pattern in advance or express the object's motion pattern in a mathematical formulation. However, based on the aforementioned IoU statistics, we can suppress KF's predicted displacement by determining whether the object is moving at low speed, thereby achieving non-uniform motion modeling.

Hence, the prior derived from **Trajectory Overlap** is that, compared to high-speed objects, low-speed objects exhibit denser IoU distributions with high scores in adjacent frames. This demonstrates that when tracking low-speed objects, KF does not need to perform aggressive state estimation because the probability of high overlap between the current state and the observed state is high. Conversely, KF needs to perform aggressive state estimation to increase the overlap probability with the observation for tracking high-speed objects.

3.2 Height and Width Ratio

As depicted in Fig. 4, we count the ratio of height and width of the same trajectory in adjacent frames. The calculation

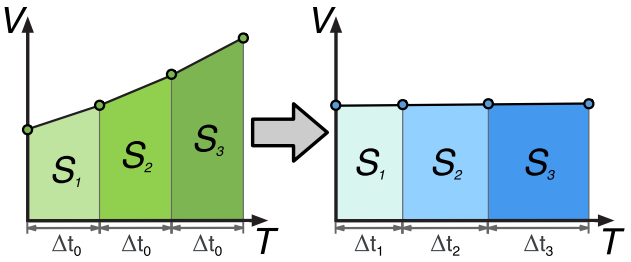

Fig. 5 Illustration of the concept of adaptive Δt. The relative stability of the velocity V is achieved by combining Δt with the displacement S, i.e., $T = S/V$

formula is as follows:

$$R = \frac{min(x_{cur}, x_{next})}{max(x_{cur}, x_{next})}, \tag{7}$$

where R denotes the height/width ratio and x_{cur}/x_{next} represent the height/width of the same trajectory in the current and next frames. It is evident that, compared to dancers in DanceTrack, pedestrians and cars in MOT17/20 and KITTI exhibit stronger height invariance (ratio distribution between 0.7 and 1) than width invariance (ratio distribution between 1 and 0.4). This implies that we can enhance the accuracy of the tracker by introducing a height invariance prior in the tracker design.

Therefore, the prior from **Height & Width Ratio** is that pedestrians and cars between adjacent frames exhibit strong height invariance, demonstrating that when the height ratio of two cars or pedestrians is small (e.g., less than 0.7), there is a high probability that these two objects do not belong to the same trajectory.

4 Proposed Methods

4.1 Parallel Kalman Filter

Modeling As discussed in Sect. 2.2 **Limitations**, we remove the variables γ and h thus the PKF defines the state $\mathbf{x}$ as a four-tuple, i.e., $\mathbf{x} = [u, v, \dot{u}, \dot{v}]^\top$, where (u, v) is the 2D coordinates of the object center and the other two variables $\dot{u}$ and $\dot{v}$ are the corresponding velocities. Therefore, the process noise $\mathbf{Q}$, the observation noise $\mathbf{R}$, and the observation model $\mathbf{H}$ are reformed as

$$\mathbf{Q}_t = \mathbf{diag}(\phi h^2, \phi h^2, \psi h^2, \psi h^2), \tag{8a}$$

$$\mathbf{R}_t = \mathbf{diag}(\phi h^2, \phi h^2), \tag{8b}$$

$$\mathbf{H}_t = \left[\mathbf{I}_{2\times 2}\ \mathbf{0}_{2\times 2}\right], \tag{8c}$$

where weights ϕ and ψ are $(\frac{1}{20})^2$ and $(\frac{1}{80})^2$, respectively. The simplified state variables allow for more effective modeling

(a) (b)

Fig. 6 Illustration of the escape problem. **a**, **b** Display the predicted trajectories of the same bounding box by KF and PKF, respectively, after the tracked person gets lost. The red dashed box represents the box at the beginning of the lost trajectory; the red solid box represents the box after 15 consecutive prediction phases of KF or PKF. The box predicted by the KF escapes in the direction of the velocity with noise, while the PKF suppresses the noise and keeps the box around the dashed box so that the person can be retraced (Color figure online)

of non-uniform motion and reduces computational complexity.

As discussed in Sect. 3.1, to model non-uniform motion, as shown in Fig. 5, we start from the basic motion equation $V = S/T$ and model non-uniform motion as uniform motion by transforming Δt in Eq. 1a from a constant to an adaptive variable related to the displacement s as Eq. 9 shows. Based on the prior of **Trajectory Overlap**, we introduce the Δt threshold factor ξ into Δt to distinguish high-speed objects from low-speed objects, with the aim of suppressing the estimated displacement of low-speed objects only. The variable Δt for $\dot{u}$ and $\dot{v}$ is defined as

$$\begin{cases} \Delta t_{\dot{u}_{t-1}} = min(\xi, \frac{1}{|\dot{u}_{t-1}|}) s_{u_{t-1}}, \\ \Delta t_{\dot{v}_{t-1}} = min(\xi, \frac{1}{|\dot{v}_{t-1}|}) s_{v_{t-1}}, \end{cases} \tag{9}$$

where the object is considered to be moving at a low speed when the absolute value of the velocity is less than $\frac{1}{\xi}$ and the displacement s is expressed as follows:

$$\begin{cases} s_{u_{t-1}} = |u_{t-1} - u_{t-2}|, \\ s_{v_{t-1}} = |v_{t-1} - v_{t-2}|. \end{cases} \tag{10}$$

In practice, s is usually set to smooth to reduce noise through a linear smooth weight ω:

$$\begin{cases} s_{u_{t-1}} = \omega |u_{t-1} - u_{t-2}| + (1-\omega) s_{u_{t-2}}, \\ s_{v_{t-1}} = \omega |v_{t-1} - v_{t-2}| + (1-\omega) s_{u_{t-2}}. \end{cases} \tag{11}$$

To solve the escape problem, we recover the control model in Eq. 1a to implement the deceleration strategy. As discussed in Sect. 2.2, when the tracked object is lost, KF only performs the prediction phase, resulting in the noise being continuously amplified as shown in Fig. 6. Therefore we gradually reduce the current velocity by the degree of loss to suppress the accumulated noise. The deceleration ratio r is defined as

$$r = \frac{f_{id} - f_{end} - 1}{\tau}, \tag{12}$$

where f_{id} is the current frame number and f_{end} is an attribute of the tracklet called EndFrame (see Sect. 4.2 **Storage**); τ is the max lost threshold for removing tracklets with too much lost time and is set to 30 by default. r is 0 when the tracklet is tracked and decelerates the velocity once the tracklet is lost. When an object is lost (i.e., $f_{id} - f_{end} - 1 > 0$ in Eq. 12), the deceleration strategy can stop the object at an early stage of loss to increase the probability of retraced. Therefore, the four variables in the non-uniform formulation (Eq. 1a) is expressed as follows:

$$\begin{cases} u_{t|t-1} = u_{t-1} + \dot{u}_{t-1} \Delta t_{\dot{u}_{t-1}} (1 - \frac{r}{2}), \\ v_{t|t-1} = v_{t-1} + \dot{v}_{t-1} \Delta t_{\dot{v}_{t-1}} (1 - \frac{r}{2}), \\ \dot{u}_{t|t-1} = \dot{u}_{t-1}(1-r), \\ \dot{v}_{t|t-1} = \dot{v}_{t-1}(1-r). \end{cases} \tag{13}$$

Algorithm 1 Pseudo-code of PKF Prediction.

Input: state vector $\hat{\boldsymbol{x}}$; covariance vector $\mathbf{p}$; property vector $\mathbf{o}$; max lost threshold τ; Δt threshold factor ξ; frame id f_{id}.

/* $\mathbf{x} = \mathbf{F}\mathbf{x} + \mathbf{B}\mathbf{u}$ */

$r = (f_{id} - f_{end} - 1)/\tau$;
$s_u = \omega|u - u'| + (1-\omega)s'_u$; $s_v = \omega|v - v'| + (1-\omega)s'_v$;
$\Delta t_{\dot{u}} = min(\xi, \frac{1}{|\dot{u}|}) s_u$; $\Delta t_{\dot{v}} = min(\xi, \frac{1}{|\dot{v}|}) s_v$;
$u \mathrel{+}= \dot{u}\Delta t_{\dot{u}}(1 - \frac{r}{2})$; $v \mathrel{+}= \dot{v}\Delta t_{\dot{v}}(1 - \frac{r}{2})$;
$\dot{u} \mathrel{\times}= 1 - r$; $\dot{v} \mathrel{\times}= 1 - r$;
$u' = u$; $v' = v$;
$s'_u = s_u$; $s'_v = s_v$;

/* $\mathbf{P} = \mathbf{F}\mathbf{P}\mathbf{F}^\top + \mathbf{Q}$ */

$p_1 \mathrel{+}= p_5 \Delta t_{\dot{u}}$; $p_2 \mathrel{+}= p_6 \Delta t_{\dot{v}}$;
$p_7 \mathrel{+}= p_3 \Delta t_{\dot{u}}$; $p_8 \mathrel{+}= p_4 \Delta t_{\dot{v}}$;
$p_1 \mathrel{+}= p_7 \Delta t_{\dot{u}}$; $p_2 \mathrel{+}= p_8 \Delta t_{\dot{v}}$;
$p_5 \mathrel{+}= p_3 \Delta t_{\dot{u}}$; $p_6 \mathrel{+}= p_4 \Delta t_{\dot{v}}$;
$p_1 \mathrel{+}= \phi h^2$; $p_2 \mathrel{+}= \phi h^2$;
$p_3 \mathrel{+}= \psi h^2$; $p_4 \mathrel{+}= \psi h^2$;

Output: $\hat{\boldsymbol{x}}$; $\mathbf{p}$; $\mathbf{o}$.

Under the matrix representation, $\mathbf{F}$ and $\mathbf{B}\mathbf{u}$ can be represented as

$$\mathbf{F}_t = \begin{bmatrix} \mathbf{I}_{2\times 2} & \begin{bmatrix} \Delta t_{\dot{u}_{t-1}} & 0 \\ 0 & \Delta t_{\dot{v}_{t-1}} \end{bmatrix} \\ \mathbf{0}_{2\times 2} & \mathbf{I}_{2\times 2} \end{bmatrix}, \tag{14}$$

$$\mathbf{B}_t \mathbf{u}_t = \begin{bmatrix} -\dot{u}_{t-1} \Delta t_{\dot{u}_{t-1}} \frac{r}{2} \\ -\dot{v}_{t-1} \Delta t_{\dot{v}_{t-1}} \frac{r}{2} \\ -\dot{u}_{t-1} r \\ -\dot{v}_{t-1} r \end{bmatrix}. \tag{15}$$

Algorithm 2 Pseudo-code of PKF Update.

Input: state vector $\hat{\boldsymbol{x}}$; covariance vector $\mathbf{p}$; observation vector $\hat{z}$; property vector $\mathbf{o}$; max lost threshold τ; Δt threshold factor ξ; frame id f_{id}.

/*$\mathbf{S} = \mathbf{HPH}^\top + \mathbf{R}$*/

$s_1 = p_1;\ s_2 = p_2;$

$s_1 \mathrel{+}= \phi h^2;\ s_2 \mathrel{+}= \phi h^2;$

/*$\mathbf{K} = \mathbf{PH}^\top \mathbf{S}^{-1}$*/

$k_1 = p_1/s_1;\ k_2 = p_2/s_2;$

$k_3 = p_5/s_1;\ k_4 = p_6/s_2;$

/*$\mathbf{x} = \mathbf{x} + \mathbf{K}(\mathbf{z} - \mathbf{Hx})$*/

$r_u = \bar{u} - u;\ r_v = \bar{v} - v;$

$r_w = \bar{w} - w;\ r_h = \bar{h} - h;$

$u \mathrel{+}= k_1 r_u;\ v \mathrel{+}= k_2 r_v;$

$w \mathrel{+}= k_1 r_w;\ h \mathrel{+}= k_2 r_h;$

$\dot{u} \mathrel{+}= k_3 r_u;\ \dot{v} \mathrel{+}= k_4 r_v;$

/*$\mathbf{P} = \mathbf{P} - \mathbf{KSK}^\top$*/

$p_1 \mathrel{-}= k_1 k_1 s_1;\ p_2 \mathrel{-}= k_2 k_2 s_2;$

$p_3 \mathrel{-}= k_3 k_3 s_1;\ p_4 \mathrel{-}= k_4 k_4 s_2;$

$p_5 \mathrel{-}= k_1 k_3 s_1;\ p_6 \mathrel{-}= k_2 k_4 s_2;$

$p_7 \mathrel{-}= k_1 k_3 s_1;\ p_8 \mathrel{-}= k_2 k_4 s_2;$

Output: $\hat{\boldsymbol{x}}$; $\mathbf{p}$; $\mathbf{o}$.

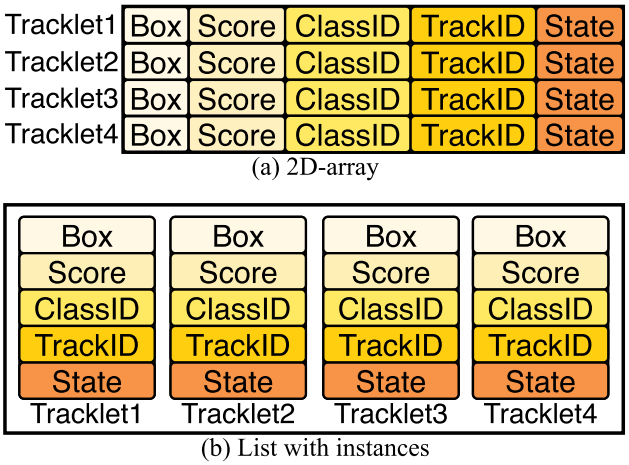

Fig. 7 Illustration of different storage structures for trackers. **a** 2D-array. **b** List with instances. The illustration shows only a portion of the components (e.g., Box or Score) of tracklets

Computation To attain an $O(1)$ time complexity, we exploit our simplified state variables and the strict matrix representation of our non-uniform formulation. In this context, we introduce an innovative parallel computation method. This method transposes the computation graph of the PKF from a matrix to a quadratic form, capitalizing on the attributes of sparse matrices within the PKF. Consequently, we facilitate parallel computation across distinct objects employing CUDA.

Specifically, the conventional KF uses matrices to represent the variables (e.g., $\mathbf{F}$, $\mathbf{P}$, and $\mathbf{H}$), and calculations are achieved by matrix multiplication. However, all the matrices involved in KF are sparse matrices with fixed positions, which means that a large amount of computation in matrix multiplication is used to compute meaningless **zeros**. Therefore, reducing the computation by separating out these 0-related computations is particularly critical to improve the efficiency. Meanwhile, the separated computation is equivalent to finite quadratic operations, which can be easily implemented in parallel between different tracklets via CUDA. For example, the covariance matrix $\mathbf{P}$ of PKF is represented as follows

$$\mathbf{P} = \begin{bmatrix} p_1 & & p_7 & \\ & p_2 & & p_8 \\ p_5 & & p_3 & \\ & p_6 & & p_4 \end{bmatrix}, \tag{16}$$

where the positions without variables has a value of 0 at any time step, thus we can reduce the computation load by eliminating these 0-value positions and associated computations both in storage and computation. Therefore, in storage, we can define the covariance vector $\mathbf{p}$ to equivalently represent matrix $\mathbf{P}$, i.e., $\mathbf{p} = [p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8]$. We also define the other vectors that need to be involved in the PKF calculation, and they are the state vector $\hat{\boldsymbol{x}} = [u, v, \dot{u}, \dot{v}]$, the observation vector $\hat{\boldsymbol{z}} = [\bar{u}, \bar{v}, \bar{w}, \bar{h}]$, and the property vector $\mathbf{o} = [w, h, f_{end}, s'_u, s'_v, u', v']$ where f_{end} is the attribute of the tracklet called EndFrame; s'_u, s'_v, u', and v' are the previous displacements and coordinates, respectively. Algorithms 1 and 2 indicate the calculation process of the prediction phase and the the update phase of PKF, respectively. In Algorithm 2, s_1 and s_2 denote the elements on the main diagonal of the matrix $\mathbf{S}$, i.e., $\mathbf{S} = \mathbf{diag}(s_1, s_2)$. k_1, k_2, k_3, and k_4 denote the valuable numbers in kalman gain, i.e.,

$$\mathbf{K} = \begin{bmatrix} k_1 & & \\ & k_2 & \\ k_3 & & \\ & k_4 & \end{bmatrix}. \tag{17}$$

r_u, r_v, r_w, r_h are the residual of u, v, w, h between observation and estimation. For initialization, u, v in $\hat{\boldsymbol{x}}$ and w, h in $\mathbf{o}$ are set to the corresponding values in the input detection result; $\dot{u}$, $\dot{v}$ in $\hat{\boldsymbol{x}}$ and s'_u, s'_v in $\mathbf{o}$ are set to 0; u', v' in $\mathbf{o}$ are set to the values of u and v, respectively; f_{end} is set to the current frame id f_{id}. The vector $\mathbf{p}$ is initialized to $[4\phi h^2, 4\phi h^2, 100\psi h^2, 100\psi h^2, 0, 0, 0, 0]$ following DeepSORT.

In the CUDA implementation, we employ two threads for u- and v-related computations, respectively. One block owns 32 threads and the number of blocks is set to $\lceil \frac{n}{16} \rceil$ where n is the number of tracked objects.

4.2 Fast

Storage In previous works (Lu et al., 2020; Peng et al., 2020; Zhou et al., 2020; Wu et al., 2021; Zhang et al., 2022), tracklets are usually represented as instances and stored through a list, as shown in Fig. 7b. Since the program does not support parallel accession to these instances, the updating or accession operation to each tracklet is executed under the time complexity $O(n)$. As shown in Fig. 7a, guided by Occam's Razor, we abandon the instance and creatively propose a GPU 2D-array to storage tracklets. In functionality, the two storages can perform exactly the same functions such as modifying certain information or adding/removing certain tracklets; in efficiency, due to the CUDA acceleration, efficient parallel updating or accession operation can be performed on all tracklets with a time complexity of $O(1)$. In Fast, each tracklet has the attributes Box (u, v, w, h), Score, ClassID, TrackID, State (one of $Track$, $Lost$, or New), EndFrame, and the PKF-related variables ($\hat{\boldsymbol{x}}$, $\mathbf{p}$, and $\mathbf{o}$).

Algorithm 3 Pseudo-code of Fast.

Input: A video $\mathcal{V}$; a detector $Det(\cdot)$; max lost threshold τ; tracking score threshold ϵ.

$\mathcal{T} \Leftarrow \emptyset$; $f_{id} \Leftarrow 1$;
for f_t **in** $\mathcal{V}$ **do**
  /* generate detections */
  $\mathcal{D}_t \Leftarrow Det(f_t)$;
  /* motion estimation */
  $\mathcal{T} \Leftarrow Predict(\mathcal{T})$;
  /* associate */
  $\mathcal{D}_m \Leftarrow$ matched detections from$\mathcal{D}_t$;
  $\mathcal{T}_m \Leftarrow$ matched tracklets from$\mathcal{T}$;
  $\mathcal{D}_u \Leftarrow$ unmatched detections from$\mathcal{D}_t$;
  $\mathcal{T}_u \Leftarrow$ unmatched tracklets from$\mathcal{T}$;
  /* update tracklets */
  for $\tilde{t}, \tilde{d}$ **in** $\mathcal{T}_m, \mathcal{D}_m$ **do**
    $\tilde{t} \Leftarrow Update(\tilde{t}, \tilde{d})$;
  end for
  /* remove tracklets */
  for $\tilde{t}$ **in** $\mathcal{T}_u$ **do**
    if f_{id} - $\tilde{t}$.EndFrame $\geq \tau$ **or** $\tilde{t}$.State == New **then**
      $\mathcal{T}_u \Leftarrow \mathcal{T}_u \setminus \{\tilde{t}\}$;
    else
      $\tilde{t}$.State $\Leftarrow Lost$;
    end if
  end for
  $\mathcal{T} \Leftarrow \mathcal{T}_m \cup \mathcal{T}_u$;
  /* initialize new tracklets */
  for $\tilde{d}$ **in** $\mathcal{D}_u$ **do**
    if $\tilde{d}$.Score $\geq \epsilon$ **then**
      $\mathcal{T} \Leftarrow \mathcal{T} \cup \{Init(\tilde{d})\}$;
    end if
  end for
  $f_{id} \Leftarrow f_{id} + 1$;
end for

Output: $\mathcal{T}$.

Cost Matrix Since Fast maintains the cleanest processing flow and only performs the association operation once during one iteration, the quality of the cost matrix has a crucial impact on the tracking results. We take into account the quality (score) of boxes into the cost matrix to determine the priority of association automatically. Specifically, for a list of tracklets with N elements and a list of detections with M elements where each element owns a box B with the corresponding score $\mathcal{S}$, the $N \times M$ cost matrix $\mathbf{C}$ is calculated as follows:

$$\mathbf{C}_{ij} = IoU(B_i, B_j) \times \mathcal{S}_i \times \mathcal{S}_j, \tag{18}$$

where i/j denotes the i-th/j-th element in the input tracklets/detections. Through this calculation, tracklets and detections with high scores will be assigned first; with low scores will be assigned last; the remaining tracklets and detections are of medium priority. From another perspective, we achieve the effect of the cascade association by score (like BYTE (Zhang et al., 2022) does) within one association. In addition, the novel cost matrix can also achieve multi-object&class tracking by simply shifting all boxes along the x-axis by the distance of class id times input image width before calculating the IoU. In implementation, we assign each element of the cost matrix to a CUDA kernel to improve computational efficiency. Besides, under the prior obtained in the **Height & Width Ratio**, we propose the HIoU metric to replace the original IoU in our cost matrix, which is calculated as follows:

$$HIoU(B_{\mathcal{T}}, B_{\mathcal{D}}) = \begin{cases} 0, & \frac{min(h_{\mathcal{T}}, h_{\mathcal{D}})}{max(h_{\mathcal{T}}, h_{\mathcal{D}})} < \lambda, \\ IoU(B_{\mathcal{T}}, B_{\mathcal{D}}), & else. \end{cases} \tag{19}$$

HIoU receives a detection box $B_{\mathcal{D}}$ and a tracklet box $B_{\mathcal{T}}$ and calculates IoU between them if the ratio of their heights ($h_{\mathcal{D}}$&$h_{\mathcal{T}}$) is greater than or equal to the height ratio threshold λ. Otherwise, the result of HIoU is 0. HIoU can be used in pedestrian or traffic scenes to further improve the performance.

Assignment In order for the tracker to be fully accelerated by GPU, we first implement the Auction Algorithm (Bertsekas, 1992b) for the asymmetric assignment problem. The naive Auction Algorithm models the auction to solve the classical symmetric assignment problem where there are n persons and n objects that we have to match on a one-to-one basis. There is a benefit a_{ij} (belonging to a $n \times n$ cost matrix) for matching person i with object j and we want to assign persons to objects so as to maximize the total benefit. The "Appendix A" describes in detail about the calculation and implementation of the naive Auction Algorithm. While the shape of the cost matrix is an arbitrary $(n \times m)$ at most time, the Reverse Auction Algorithm is required and we implement

it via CUDA successfully. Specifically, when n is less than or equal to m, we apply the naive Auction Algorithm to match the tracklets and detections; when n is greater than m, we first transpose the cost matrix (from $n \times m$ to $m \times n$) and then apply the naive Auction Algorithm, while the results need to be mapped back to the original cost matrix.

Architecture Along the concise and generic tracker design concept, Fast only requires boxes with scores and class ids to do only one association during one iteration, thus allowing for high generality and efficiency. The input of Fast is a video sequence $\mathcal{V}$ and a detector $Det(\cdot)$. The max lost threshold τ and tracking score threshold ϵ (0.7 by default) are also set. The output of Fast is the tracklets $\mathcal{T}$ of the video and each tracklet has attributes such as Box, Score, ClassID, TrackID, State (one of $Track$, $Lost$, or New), and EndFrame. As the pseudo-code of Fast shown in Algorithm 3, for each frame f_t in the video $\mathcal{V}$, Fast first obtains the detections $\mathcal{D}_t$ including Box, Score, and ClassID via the detector $Det(\cdot)$. Then the motion estimation (the prediction phase of PKF) of tracklets $\mathcal{T}$ will be calculated (line 4). After associating the detections $\mathcal{D}_t$ and tracklets $\mathcal{T}$ via the cost matrix generated by them, Fast obtains four groups: the matched detections $\mathcal{D}_m$, the matched tracklets $\mathcal{T}_m$, the unmatched detections $\mathcal{D}_u$, and the unmatched tracklets $\mathcal{T}_u$ (line 5 to 8). The detections in $\mathcal{D}_m$ will be updated into the corresponding tracklets in $\mathcal{T}_m$ (line 9 to 10) where the update phase of PKF will be performed, the State of tracklets will be set to $Track$, and the EndFrame will be set to the current frame id f_{id}. Then the tracklet in $\mathcal{T}_u$ will be removed if the difference between its EndFrame and f_{id} is greater than or equal to the τ or its State is New following BYTE. Otherwise, it will be remained and its State will be set to $Lost$ (line 12 to 18). Next, the detections in $\mathcal{D}_u$ will be added into the $\mathcal{T}$ after initialization when its score is greater than or equal to the ϵ (line 20 to 24). $Init(\cdot)$ includes initializing the State to New, the EndFrame to f_{id}, and the initialization of PKF. The output of each frame is the new tracklets $\mathcal{T}$, where the tracklets with $Lost$ state are not allowed to be output.

5 Experiment

5.1 Settings

Datasets We evaluate Fast on the aforementioned MOT benchmarks, which include general tracking scenes such as MOT17 (Milan et al., 2016), MOT20 (Dendorfer et al., 2020), KITTI (Geiger et al., 2013), and DanceTrack (Sun et al., 2021). MOT17 (Milan et al., 2016) and MOT20 (Dendorfer et al., 2020) focus on pedestrian tracking, with mostly low-speed movements. However, MOT20 scenes are more crowded than those in MOT17 (139 vs. 33 on average per frame). KITTI (Geiger et al., 2013) targets traffic tracking with high-speed cars and pedestrians. DanceTrack (Sun et al., 2021) is a recently proposed benchmark for tracking dancers in stage scenes. In DanceTrack, dancers exhibit highly non-linear movements and share similar appearances, while severe occlusions and frequent crossovers occur. For ablation datasets, we follow previous works (Cao et al., 2022; Zhang et al., 2022) and use MOT17-val (see "Appendix B" for details) and DanceTrack validation set (DanceTrack-val) to verify the effectiveness of our proposed modules, such as the non-uniform formulation or the cost matrix.

Metrics We employ HOTA (Luiten et al., 2021), IDF1, and MOTA (Milan et al., 2016) as the main metrics to evaluate various aspects of tracking performance. HOTA is a recently proposed metric that explicitly balances the effects of accurate detection, association, and localization. IDF1 evaluates identity preservation ability and focuses more on association performance. MOTA is computed based on FP, FN, and IDs. Given that the number of FP and FN is larger than IDs, MOTA focuses more on detection performance. Additionally, we report other metrics, such as DetA or AssA, and raw statistics like ID switch (IDs) or Fragments (Frag).

Implementations All experiments are conducted on a PC equipped with a GTX 1080Ti GPU and an i5-9500 CPU. Fast is implemented using CuPy (Okuta et al., 2017) in Python. CuPy is a powerful CUDA-based library for fast matrix operations on NVIDIA GPUs, offering nearly identical APIs to NumPy and supporting the loading of raw CUDA kernel functions written in C/C++. Our storage method uses the 2D array provided by CuPy; cost matrix, PKF, and Auction Algorithm are implemented in CUDA C/C++ and called via CuPy. Following previous works (Zhang et al., 2022; Cao et al., 2022) for a fair comparison, we directly employ the same detector in FastTrack. For MOT17, MOT20, and DanceTrack, we use the publicly available YOLOX (Ge et al., 2021) detector weights from ByteTrack (Zhang et al., 2022); for KITTI (Geiger et al., 2013), we use the detections from PermaTrack (Tokmakov et al., 2021) publicly available in the official release following OCSORT (Cao et al., 2022). Methods using the same detector are placed at the bottom of each benchmark table, and linear interpolation (Zhang et al., 2022) is also employed in the benchmark comparisons.

5.2 Ablation Study

All the ablation studies are conducted on MOT17-val and DanceTrack-val, with four metrics being used to evaluate the performance: HOTA (H↑), MOTA (M↑), IDF1 (I↑), and IDs ($I\downarrow$).

5.2.1 Association

Cost Matrix Table 2 shows the experimental results of comparing different cost matrix strategy in Fast. "Baseline"

Table 4 Ablation study on the accuracy of different assignment algorithms

Assignment	MOT17-val				DanceTrack-val			
	H↑	M↑	I↑	$I\downarrow$	H↑	M↑	I↑	$I\downarrow$
LAPJV (Jonker and Volgenant, 1987)	72.392	82.206	85.257	116	55.521	90.173	55.484	1932
Auction (Bertsekas, 1992a)	72.338	82.197	85.193	115	55.671	90.130	55.745	1934

Table 2 Ablation study on our proposed cost matrix

Strategy	MOT17-val				DanceTrack-val			
	H↑	M↑	I↑	$I\downarrow$	H↑	M↑	I↑	$I\downarrow$
Baseline	71.3	81.8	83.1	131	54.5	90.1	55.4	2002
IoU	70.6	80.8	81.6	197	50.5	87.2	49.7	3287
$\text{IoU}_{\mathcal{D}}$	71.6	82.1	83.7	133	52.4	89.6	52.8	2383
$\text{IoU}_{\mathcal{DT}}$	**71.8**	**82.2**	**84.5**	**124**	**55.7**	**90.1**	**55.7**	**1934**

"Baseline" means the cascade strategy used in BYTE. $\mathcal{D}$ and $\mathcal{T}$ in the footnote indicate the corresponding scores of detections or tracklets multiplied into IoU
Bold indicates the best performance at a given metric

Table 3 Ablation study on $\text{HIoU}_{\mathcal{DT}}$ with different height ratio threshold λ on MOT17-val and DanceTrack-val

$\text{HIoU}_{\mathcal{DT}}$	MOT17-val				DanceTrack-val			
	H↑	M↑	I↑	$I\downarrow$	H↑	M↑	I↑	$I\downarrow$
0	71.8	82.2	84.5	124	**55.7**	**90.1**	**55.7**	**1934**
0.2	71.8	82.2	84.5	124	55.7	90.1	55.7	1946
0.4	71.8	82.2	84.5	124	55.7	90.1	55.7	1950
0.6	72.2	82.2	85.0	120	55.2	90.1	55.0	2006
0.8	**72.3**	**82.2**	**85.2**	**115**	51.8	89.4	48.3	3182

Bold indicates the best performance at a given metric

indicates the results of employing the original BYTE's cascade association based on score into Fast. The effects of IoU, $\text{IoU}_{\mathcal{D}}$, and $\text{IoU}_{\mathcal{DT}}$ are incremental over the six metrics of the two datasets. Compared to "Baseline", $\text{IoU}_{\mathcal{DT}}$ outperforms in all metrics with the concise assignment strategy.

HIoU As discussed in Sect. 3.2, compared with the pedestrians in MOT17, the dancers in DanceTrack have a greater range of motion and thus do not have the height invariance prior. The experimental results of $\text{HIoU}_{\mathcal{DT}}$ with different height ratio threshold λ in Table 3 reflect this as well. HIoU with 0.8 threshold in MOT17-val can increase the HOTA and IDF1 because it excludes potential error assignment; DanceTrack-val does not meet the prior thus the application of HIoU leads to a drop in all metrics. In later ablation studies, $\text{HIoU}_{\mathcal{DT}}$ with 0.8 threshold is employed in MOT17-val; $\text{IoU}_{\mathcal{DT}}$ is employed in DanceTrack-val.

Table 5 Ablation study on different states of KF

State	MOT17-val				DanceTrack-val			
	H↑	M↑	I↑	$I\downarrow$	H↑	M↑	I↑	$I\downarrow$
$uv\gamma h$	71.0	81.6	83.4	142	48.8	88.1	53.3	2003
uv	**71.8**	**82.1**	**84.0**	**126**	**54.2**	**89.8**	**53.9**	**1978**

The motion estimation module used in Fast is the original KF with different states. $uv\gamma h$ means the original eight-tuple state; uv means our proposed four-tuple state
Bold indicates the best performance at a given metric

Assignment Table 4 presents the accuracy of the Auction and LAPJV[4] algorithms. The metric values for both algorithms are nearly identical, with the Auction algorithm slightly outperforming LAPJV by 0.15% and 0.26% for the HOTA and IDF1 metrics on the DanceTrack-val dataset, respectively. In conclusion, the accuracies of the two algorithms in the context of MOT can be considered equal. Therefore, it is functionally feasible for the Auction algorithm to replace the conventional CPU-based linear assignment algorithm LAPJV in Fast.

5.2.2 Parallel Kalman Filter

State Estimate Table 5 shows the experimental results of different states of KF. Here we employ the original KF with different states as the motion estimation module of our Fast. It can be found that Fast has better performance on all metrics when γ and h are removed, which is also consistent with the previous analysis in Sect. 2.2, i.e., uniform modeling of variables γ and h is unreasonable and they should be removed from the state estimate of KF.

Δt **Threshold Factor** ξ. The hyperparameter ξ is used to distinguish low-speed objects from high-speed objects, and as shown in Table 6, Fast with the best ξ performs better in HOTA and IDF1 in both datasets compared to the "Baseline". Due to the diversity of image resolutions, object sizes and frame rates for different types of scenes, the value of ξ needs

[4] LAPJV has replaced the Hungarian Algorithm as the default assignment strategy in all current trackers. This is because scikit-learn (Pedregosa et al., 2011), the only library that supported the Hungarian Algorithm, has deprecated it. Now, all trackers perform assignment using LAPJV, which is implemented in the third-party library lap (https://github.com/gatagat/lap) or scipy (Virtanen et al., 2020).

Table 6 Ablation study on Δt threshold factor ξ

ξ	MOT17-val				DanceTrack-val			
	H↑	M↑	I↑	$I\downarrow$	H↑	M↑	I↑	$I\downarrow$
Baseline	72.1	82.2	84.6	140	55.1	90.1	54.6	1983
0.01	70.8	81.2	82.9	406	53.0	90.1	52.7	2007
0.03	71.8	82.0	84.3	132	53.6	90.1	53.7	1964
0.05	**72.3**	**82.2**	**85.2**	**115**	54.7	90.1	54.3	1968
0.07	72.3	82.2	85.1	122	**55.7**	**90.1**	**55.7**	**1934**
0.09	72.3	82.3	85.1	119	55.6	90.1	55.5	1945

"Baseline" means Δt is the constant value 1 following DeepSORT. In MOT17-val, the linear smooth weight ω is 0.85; in DanceTrack-val, the smooth weight is 0.7
Bold indicates the best performance at a given metric

Table 7 Ablation study on linear smooth weight ω

ω	MOT17-val				DanceTrack-val			
	H↑	M↑	I↑	$I\downarrow$	H↑	M↑	I↑	$I\downarrow$
Baseline	72.2	82.1	85.0	125	54.7	90.1	54.8	1975
0.85	**72.3**	**82.2**	**85.2**	**115**	55.5	90.1	55.5	1971
0.70	72.2	82.2	84.9	122	**55.7**	**90.1**	**55.7**	**1934**
0.55	72.1	82.1	84.9	123	54.9	90.1	55.0	1967

"Baseline" means $\omega = 1$, i.e., the smooth strategy is not leveraged. In MOT17-val, ξ is 0.05; in DanceTrack-val, ξ is 0.07
Bold indicates the best performance at a given metric

to be set specifically, and as can be seen from the experiments in Table 6, the value of ξ takes a range roughly between 0.05 and 0.09. Empirically, we suggest that the default value of ξ is 0.05, which can be appropriately increased to 0.07 or 0.08 for better tracking results when the overall motion of objects in the tracking scene is faster or the frame rate is lower.

Linear Smooth Weight ω The introduction of an appropriate smoothing hyperparameter is crucial to reduce noise generated in the object motion, thus in Table 7, Fast with the best ω performs better in the HOTA and IDF1 in both datasets compared to the "Baseline". Since the noise differs in different types of scenes, the value of ω also needs to be set specifically, and as can be seen from the experiments in Table 7, the value of ω takes a range roughly between 0.85 and 0.70. Empirically, we suggest that the default value of ω is 0.85, which can be appropriately reduced to 0.70 when the overall motion of objects in the tracking scene (such as stage scenes) is highly non-linear to obtain better tracking results.

Deceleration Strategy Table 8 shows the experimental results of our deceleration strategy. For MOT17-val, there is little improvement compared with the "Baseline" in the four metrics due to the small sample size and the fact that the occlusion problem in it is not severe. For DanceTrack-val, the sufficient data sample combined with the severe occlusion problem makes our strategy improve another 1% on HOTA and IDF1 compared with the "Baseline". Besides, stopping the lost object immediately does not have a significant advantage over "Baseline" in both datasets.

Table 8 Ablation study on the deceleration strategy

Strategy	MOT17-val				DanceTrack-val			
	H↑	M↑	I↑	$I\downarrow$	H↑	M↑	I↑	$I\downarrow$
Baseline	72.2	82.2	85.1	121	54.9	90.1	54.5	1948
Stop	71.8	82.0	84.3	159	55.1	90.1	54.9	1971
Decelerate	**72.3**	**82.2**	**85.2**	**115**	**55.7**	**90.1**	**55.7**	**1934**

"Baseline" means leveraging the original strategy of KF, i.e., canceling the deceleration strategy; "Stop" means stopping the box immediately once the tracklet is lost; "Decelerate" means employing our proposed deceleration strategy. In MOT17-val, ξ is 0.05 and ω is 0.85; in DanceTrack-val, ξ is 0.07 and ω is 0.7
Bold indicates the best performance at a given metric

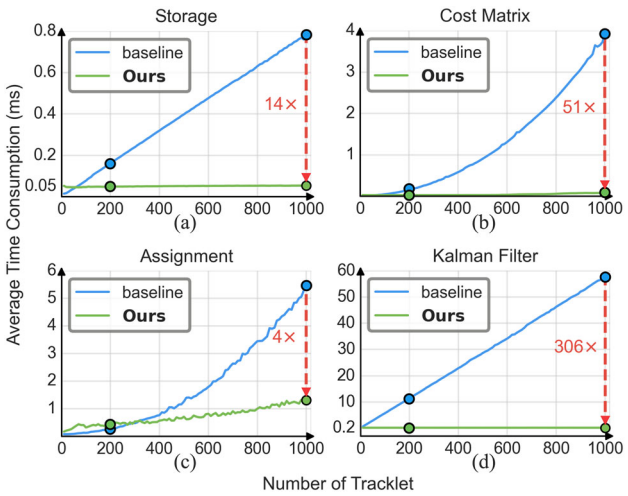

Fig. 8 Computional efficiency comparison between different modules and the corresponding baselines. The x-axis indicates the number of tracklets processed simultaneously; the y-axis indicates the average consumption time (in ms)

5.2.3 Comparison with Other Trackers

Table 9 presents the experimental results of different motion-based trackers. Fast outperforms the second-best tracker by 1–2% in HOTA, MOTA, and IDF1 metrics on both datasets. Thanks to PKF's ability to model non-uniform motion, Fast achieves a 3.4% higher HOTA score and a 3.8% higher IDF1 score in the DanceTrack-val dataset compared to the second-ranked OCSORT, while maintaining a nearly identical number of ID switches.

5.2.4 Module Efficiency

As illustrated in Fig. 8, we compare the time consumption of our proposed modules with their corresponding CPU base-

Table 9 Comparison with other state-of-the-art motion-based trackers

Tracker	MOT17-val				DanceTrack-val			
	H↑	M↑	I↑	$I\downarrow$	H↑	M↑	I↑	$I\downarrow$
SORT (Bewley et al., 2016)	70.0	80.2	81.4	186	49.7	86.3	48.8	2180
BYTE (Zhang et al., 2022)	71.3	81.8	83.4	133	47.1	88.3	51.9	1936
OCSORT (Cao et al., 2022)	70.3	79.2	82.2	**112**	52.3	87.3	51.9	**1926**
Ours	**72.3**	**82.2**	**85.2**	115	**55.7**	**90.1**	**55.7**	1934

The detectors (YOLOX) for each tracker are all the same in both two datasets
Bold indicates the best performance at a given metric

lines. To investigate the maximum computational efficiency of each module, we define the *extreme case* by setting the tracklet count at 1000.[5] Conversely, we denote the *typical case* by setting the tracklet count at 200,[6] serving as a representative of computational efficiency in realistic large-scale object tracking situations.

Storage As depicted in Fig. 8a, we compare the computational efficiency of our GPU 2D-array storage method to the baseline, which uses a list with instances. The average time consumption refers to the time taken to change the attribute State of all n tracklets from $Lost$ to $Track$. Our method and the baseline method exhibit time complexities of $O(1)$ and $O(n)$, respectively. Under the typical case, our method performs 3.6× faster, while under the extreme case, it performs 14× faster than the baseline method.

Cost Matrix Figure 8b compares the computational efficiency of our proposed cost matrix method (which includes HIoU) to the baseline (cython_box[7]). The average time consumption denotes the time required to create an $n \times n$ cost matrix. Our method has a time complexity of $O(1)$, whereas the baseline method has a time complexity of $O(n^2)$. Our method outperforms the baseline by a factor of 6.2× in the typical case and by 51× in the extreme case.

Assignment Figure 8c showcases the computational efficiency of our implemented Auction Algorithm, contrasted with the baseline (LAPJV). The average time consumption is the time needed to solve an $n \times n$ random matrix with all values between 0 and 1. The theoretical time complexity of LAPJV is $O(n^3)$, while our method's complexity is $O(n)$ with a low slope. Architectural differences between CPUs and GPUs mean that initiating kernel functions on GPUs require additional time. This extra time cost overshadows the computational advantage of our GPU method when the number of tracklets is less than 300. However, as the data scale increases, the parallel computing capabilities of the GPU render this time cost increasingly insignificant. Therefore, our GPU method's processing speed surpasses that of the CPU method when the number of tracklets exceeds 300. Under the typical case, our method is nearly identical to the baseline, but it is 4× faster in the extreme case.

Kalman Filter Figure 8d compares the computational efficiency of our PKF with the baseline KF. The average time consumption is the time taken to complete a predict-update loop for all n tracklets. The time complexities of our method and the baseline method are $O(1)$ and $O(n)$, respectively. Under the typical case, our method is 60× faster than the baseline, and under the extreme case, it is 306× faster. Furthermore, our PKF can process 10 million objects simultaneously with an average time of 0.2 ms per iteration when fully utilizing the GTX 1080Ti 12 G video memory, whereas the original KF takes about 600 s per iteration to process the same number of objects.

Discussion From the aforementioned analyses, it becomes clear that the time consumption of each crucial module, particularly the Kalman Filter, in conventional CPU-based trackers escalates with increasing input scale. This escalation negatively impacts the computational efficiency of large-scale object tracking and the stability of the MOT system, inducing processing speed to fluctuate with the input scale. In contrast, our proposed modules successfully disentangle the computational efficiency from the input scale. This separation leads to a novel GPU-based tracker paradigm that can efficiently manage large-scale object tracking while maintaining the stability of the MOT system.

5.3 Benchmark Results

We report Fast's performance on the four benchmarks separately, and the visualized results of Fast on each benchmark are shown in Fig. 9.

[5] Due to the number 1000 is a typical upper limit of the number of objects detected by the detector in a single image (Chen et al., 2019), we choose it to align the upper limits of the tracker and the detector.

[6] Considering the MOT20 test set, averaging 139 objects per frame with a peak of 300 objects in a single frame, owns the densest large-scale object tracking scenarios in reality, we pragmatically consider 200 objects as a representative case, striking a balance between the average and peak object count.

[7] A CPU-based cost matrix calculation method (https://github.com/samson-wang/cython_bbox) used in BYTE.

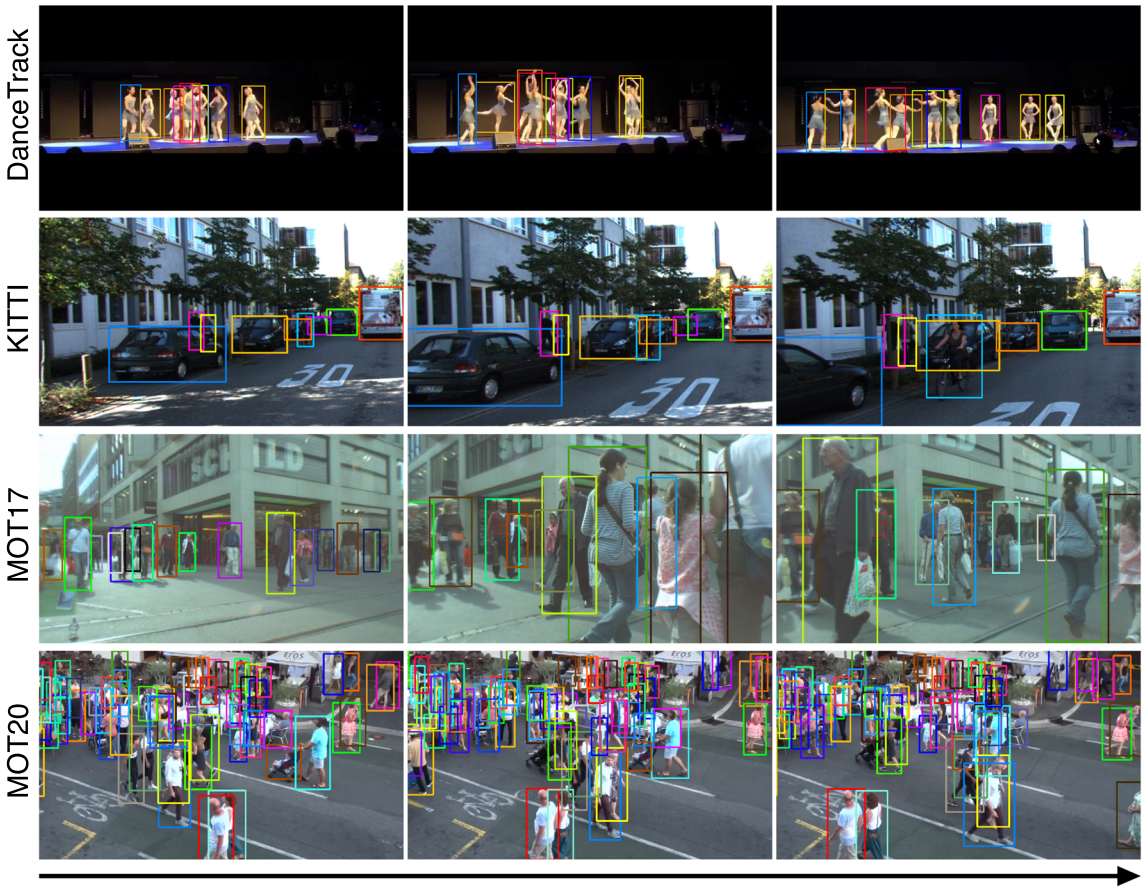

Fig. 9 Visualizations of Fast's tracking results on each benchmark. Boxes of the same color represent the same object (Color figure online)

5.3.1 DanceTrack

We report the DanceTrack test set results in Table 10 to evaluate Fast's performance under challenging non-uniform motion. The setting of the hyperparameters during testing is the same as in DanceTrack-val, i.e., ξ is 0.07 and ω is 0.7 and HIoU is not employed in the cost matrix. Our proposed non-uniform formulation and deceleration strategy successfully model the diverse non-uniform motions of the dancers in the stage scenes, and thus Fast achieves a new state-of-the-art on HOTA, MOTA, and IDF1. With the same detector (YOLOX), Fast is 1.7% higher than OCSORT and 10.1% higher than BYTE on HOTA.

5.3.2 KITTI

Table 11 shows the results of the comparison on the KITTI test set. Compared to the pedestrian scene in MOT17-val, the traffic scene in KITTI tracks both cars and pedestrians at high-speed linear motion from the perspective of trackers due to the low frame rate (10 vs. 30). Therefore we set ξ appropriately higher at 0.08 to handle the high-speed motion and ω still follows the default setting of 0.85, and HIoU is employed where λ is set to 0.5 from the statistical prior in Fig. 4 KITTI Train Car. Following OCSORT, we utilize the detection results of PermaTr (Tokmakov et al., 2021) for tracking both cars and pedestrians. Notably, Fast can track both pedestrians and cars at the same time by our proposed cost matrix. Fast surpasses PermaTr and OCSORT on HOTA and MOTA of both Car and Pedestrian and improves the pedestrian tracking performance (HOTA) to a new state-of-the-art level, i.e., 55.10%.

5.3.3 MOT Challenge

We report in Tables 12 and 13 the performance of Fast on MOT17 and MOT20 test sets using the same detector YOLOX as well as ByteTrack and OCSORT. Following the MOT17-val, ξ is 0.05 by default and ω is 0.85 in both MOT17 and MOT20. HIoU is employed on both MOT17 and MOT20 and λ is set to 0.8. Although receiving the same detection results, OCSORT discards the low-scoring detection results first before starting tracking following SORT, while Fast utilizes the score prior to automatically determine association priorities in our proposed cost matrix, which allows Fast to track more objects with low scores and thus achieves

Table 10 Comparison in DanceTrack test set

Tracker	HOTA↑	DetA↑	AssA↑	MOTA↑	IDF1↑
CenterTrack (Zhou et al., 2020)	41.8	78.1	22.6	86.8	35.7
FairMOT (Zhang et al., 2021)	39.7	66.7	23.8	82.2	40.8
QDTrack (Pang et al., 2021)	45.7	72.1	29.2	83.0	44.8
TransTrk (Sun et al., 2020)	45.5	75.9	27.5	88.4	45.2
TraDes (Wu et al., 2021)	43.3	74.5	25.4	86.2	41.2
MOTR (Zeng et al., 2021)	48.4	71.8	32.7	79.2	46.1
SORT (Bewley et al., 2016)	47.9	72.0	31.2	91.8	50.8
DeepSORT (Wojke et al., 2017)	45.6	71.0	29.7	87.8	47.9
ByteTrack (Zhang et al., 2022)	47.3	71.6	31.4	89.5	52.5
OCSORT (Cao et al., 2022)	55.7	**81.7**	38.3	92.0	54.6
FastTrack (ours)	**57.4**	81.1	**40.7**	**92.8**	**58.2**

Methods in the bottom block use the same detector
Bold indicates the best performance at a given metric

Table 11 Comparison in KITTI test set

Tracker	Car					Pedestrian				
	HOTA↑	MOTA↑	AssA↑	IDs↓	Frag↓	HOTA↑	MOTA↑	AssA↑	IDs↓	Frag↓
IMMDP Xiang et al. (2015)	68.66	82.75	69.76	211	201	–	–	–	–	–
AB3D Weng et al. (2020)	69.99	83.61	69.33	**113**	206	37.81	38.13	44.33	**181**	879
SMAT Gonzalez et al. (2020)	71.88	83.64	72.13	198	294	–	–	–	–	–
TrackMPNN Rangesh et al.(2021)	72.30	87.33	70.63	481	237	39.40	52.10	35.45	626	669
MPNTrack Brasó & Leal-Taixé (2020)	–	–	–	–	–	45.26	46.23	47.28	397	1,078
CenterTrack Zhou et al. (2020)	73.02	88.83	71.20	254	227	40.35	53.84	36.93	425	618
QD-3DT Hu et al. (2022)	72.77	85.94	72.19	206	525	41.08	51.77	38.82	717	1,194
QDTrack Pang et al. (2021)	68.45	84.93	65.49	313	567	41.12	55.55	38.10	487	951
LGM Wang et al. (2021a)	73.14	87.60	72.31	448	164	–	–	–	–	–
Eager Kim et al. (2021)	74.39	87.82	74.16	239	390	39.38	49.82	38.72	496	1,410
TuSimple Choi. (2015)	71.55	86.31	71.11	292	220	45.88	57.61	47.62	246	651
PermaTr Tokmakov et al. (2021)	78.03	91.33	78.41	258	250	48.63	65.98	45.61	403	646
OCSORT Cao et al. (2022)	76.54	90.28	76.39	250	280	54.69	65.14	**59.08**	204	609
FastTrack (**Ours**)	**78.78**	**92.06**	**80.66**	264	**104**	**55.10**	**67.92**	57.88	305	**487**

Methods in the bottom block use the same detector
Bold indicates the best performance at a given metric

higher MOTA than OCSORT by 2.5% and 1.8% on MOT17 and MOT20, respectively. Meanwhile, Fast also achieves the the highest HOTA and IDF1 on MOT17 under the same detector (YOLOX) and competitive HOTA on MOT20. It is important to note that on MOT20, Fast gains significant efficiency improvements with its revolutionary GPU paradigm as shown in Fig. 2, i.e., 7× faster than OCSORT.

5.3.4 Efficiency

Figure 2 illustrates the average processing time of different motion-based trackers on the four benchmarks. The comparison is conducted on a single PC equipped with an NVIDIA GTX 1080Ti and Intel Core i5-9500. The average number of objects per frame processed by the trackers on the four test sets are 6 (KITTI), 8 (DanceTrack), 33 (MOT17), and 139 (MOT20), respectively. As discussed in Sect. 5.2.4, the computational efficiency of our new paradigm, Fast, is independent of the input scale, with an average time consumption of 4 ms across all four benchmarks.

When the input scale is small (e.g., 6 on KITTI), the extra startup time of kernel functions offsets the computational advantage of Fast, resulting in Fast consuming more time than other CPU-based trackers (4 ms vs. 1 ms). As the input scale increases, the time consumption of the CPU-based trackers rapidly rises, while Fast's time consumption remains constant at 4 ms. For instance, on MOT20, Fast stays at 4 ms per frame, while OCSORT and BYTE take 30 ms and 20 ms,

Table 12 Comparison under the "private detector" protocol in MOT17 test set

Tracker	HOTA↑	MOTA↑	IDF1↑	FP (10^4)↓	FN (10^4)↓	IDs↓	Frag↓	AssA↑	AssR↑
FairMOT (Zhang et al., 2021)	59.3	73.7	72.3	2.75	11.7	3303	8073	58.0	63.6
TransCt (Xu et al., 2021)	54.5	73.2	62.2	2.31	12.4	4614	9519	49.7	54.2
TransTrk (Sun et al., 2020)	54.1	75.2	63.5	5.02	8.64	3603	4872	47.9	57.1
Semi-TCL (Li et al., 2021)	59.8	73.3	73.2	2.29	12.5	2790	8010	59.4	64.7
CSTrack (Zhou et al., 2020)	59.3	74.9	72.6	2.38	11.4	3567	7668	57.9	63.2
GRTU (Wang et al., 2021b)	62.0	74.9	75.0	3.20	10.8	1812	**1824**	62.1	65.8
QDTrack (Pang et al., 2021)	53.9	68.7	66.3	2.66	14.66	3378	8091	52.7	57.2
MAA (Stadler and Beyerer, 2022)	62.0	79.4	75.9	3.73	**7.77**	**1452**	2202	60.2	67.3
MOTR (Zeng et al., 2021)	57.2	71.9	68.4	2.11	13.6	2115	3897	55.8	59.2
ReMOT (Yang et al., 2021)	59.7	77.0	72.0	3.32	9.36	2853	5304	57.1	61.7
PermaTr (Tokmakov et al., 2021)	55.5	73.8	68.9	2.90	11.5	3699	6132	53.1	59.8
TransMOT (Chu et al., 2021)	61.7	76.7	75.1	3.62	9.32	2346	7719	59.9	66.5
ByteTrack (Zhang et al., 2022)	63.1	80.3	77.3	2.55	8.37	2196	2277	62.0	**68.2**
OCSORT (Cao et al., 2022)	63.2	78.0	77.5	**1.51**	10.8	1950	2040	**63.2**	67.5
FastTrack (ours)	**63.4**	**80.5**	**77.6**	2.47	8.36	2013	2337	62.3	67.8

Methods in the bottom block use the same detector
Bold indicates the best performance at a given metric

Table 13 Comparison under the "private detector" protocol in MOT20 test set

Tracker	HOTA↑	MOTA↑	IDF1↑	FP(10^4)↓	FN(10^4)↓	IDs↓	Frag↓	AssA↑	AssR↑
FairMOT (Zhang et al., 2021)	54.6	61.8	67.3	10.3	8.89	5243	7874	54.7	60.7
TransCt (Xu et al., 2021)	43.5	58.5	49.6	6.42	14.6	4695	9581	37.0	45.1
TransTrk (Sun et al., 2020)	48.5	65.0	59.4	2.72	15.0	3608	11,352	45.2	51.9
Semi-TCL (Li et al., 2021)	55.3	65.2	70.1	6.12	11.5	4139	8508	56.3	60.9
CSTrack (Zhou et al., 2020)	54.0	66.6	68.6	2.54	14.4	3196	7632	54.0	57.6
GSDT (Wang et al., 2021c)	53.6	67.1	67.5	3.19	13.5	3131	9875	52.7	58.5
RelationT (Yu et al., 2021)	56.5	67.2	70.5	6.11	10.5	4243	8236	55.8	66.1
MAA (Stadler and Beyerer, 2022)	57.3	73.9	71.2	2.49	10.9	1331	1450	55.1	61.1
ReMOT (Yang et al., 2021)	61.2	77.4	73.1	2.83	8.67	1789	2121	58.7	63.1
TransMOT (Chu et al., 2021)	61.9	77.5	75.2	3.42	**8.08**	1615	2421	60.1	66.3
ByteTrack (Zhang et al., 2022)	61.3	**77.8**	75.2	2.62	8.76	1223	1460	59.6	66.2
OCSORT (Cao et al., 2022)	**62.1**	75.5	**75.9**	**1.80**	10.8	**913**	**1198**	**62.0**	**67.5**
FastTrack (ours)	61.8	77.3	74.7	2.53	9.06	1434	1337	60.2	66.9

Methods in the bottom block use the same detector
Bold indicates the best performance at a given metric

respectively. Even on the embedded CUDA device Jetson AGX Xavier, Fast's average time consumption on MOT20 is only 24 ms, while OCSORT and BYTE take 231 ms and 67 ms, respectively.

To further showcase Fast's efficiency in the extreme case, we compare the time consumption of the four methods on MOT20 at different scales, as demonstrated in Table 14. We evaluate the trackers' efficiency in large-scale object tracking scenarios by replicating each frame's detection results N times on the MOT20 test set. Specifically, we shift the x-coordinate of the ith replicated result by $i \times$W pixels to the right (W being the frame width), while keeping the y-coordinate unchanged, thereby increasing the number of objects in each frame by N times. This simulation is reasonable because, while a single video stream may not contain 1000 tracklets per frame, it is typical to merge and track multiple streams simultaneously in real-world scenarios.

Table 14 demonstrates that Fast holds a significant advantage over other trackers in large-scale object tracking scenarios. For instance, when N equals 7, Fast processes 29 ms per frame with an average of 973 objects per frame, whereas OCSORT and BYTE take 241 ms and 161 ms,

Table 14 Average time consumption (ms) of different trackers on MOT20 at different scales

MOT20	OCSORT	BYTE	SORT	Fast (ours)
×3(417)	84	58	50	**8**
×5(695)	154	102	90	**16**
×7(973)	241	161	136	**29**

The term "×N(139·N)" signifies that each frame's detection results on the MOT20 dataset are replicated N times, leading to an average of 139·N tracklets per frame
Bold indicates the best performance at a given metric

respectively. Although the time consumption of each module in Fast remains independent of the input scale, the creation and destruction overhead of intermediate data variables (e.g., the cost matrix) in Fast increases as the input scale expands, resulting in greater time consumption for large-scale input. Additionally, the GTX 1080Ti also constrains Fast's efficiency on large-scale input. When using the RTX 4090, Fast only requires 10 ms to process 973 objects per frame, making it 24× faster than OCSORT and 16× faster than BYTE.

The above comparison results highlight the remarkable efficiency of our new GPU-based tracker paradigm.

6 Conclusion

In this paper, we propose Parallel Kalman Filter to model non-uniform motion via the proposed non-uniform formulation and achieve a time complexity of $O(1)$ via the proposed parallel computation method in large-scale object tracking scenarios. Further, based on PKF, we propose Fast, the first fully GPU-based tracker paradigm, to greatly improve tracking efficiency in large-scale object tracking; and FastTrack, the MOT system consisting of Fast and a general detector, allowing for high efficiency and generality. Within Fast, we introduce innovative GPU-based tracking modules, such as an efficient GPU 2D-array data structure, a novel cost matrix, a new association metric called HIoU, and the Auction Algorithm. The conducted experiments demonstrate that PKF owns the highly computational efficiency and is independent of the input scale, while other modules in Fast are also highly efficient compared to the CPU-based baselines. FastTrack demonstrates state-of-the-art performance on four public benchmarks, and attains the highest speed in large-scale tracking scenarios of MOT20.

Acknowledgements We thank Yifu Zhang and Jinkun Cao for providing codes of ByteTrack and OCSORT. This work is supported in part by the National Natural Science Foundation of China (NSFC) under Grants (No.61976038 and No.61932020), and the Taishan Scholar Program of Shandong Province (tstp20221128).

Data Availability Statement The above four datasets supporting the findings of this study are available in github.com/DanceTrack/DanceTrack for DanceTrack, cvlibs.net/datasets/kitti/eval_tracking.php for KITTI, motchallenge.net for MOT17 and MOT20.

Appendix A Naive Auction

Algorithm Algorithm 4 indicates the naive Auction Algorithm. For a cost matrix $\mathbf{C}$ generated by n persons and m items where m must be greater than or equal to n, we define the following variables to execute the algorithm: a positive scaler ϵ equal to $\frac{1}{n}$; a bids table $\mathbf{b}$ initialized to a 2D-array with 0 values and shape $(n+1, m)$ where $\mathbf{b}[n, :]$ denotes the flag vector; a person2item mapping **p2i** initialized to a 1D-array with -1 values and length n; an item2person mapping **i2p** initialized to a 1D-array with -1 values and length m; a price vector $\mathbf{p}$ initialized to a 1D-array with 0 values and length m. The algorithm is based on a bidding-assignment loop to solve the best match, and the loop terminates when there is no value -1 in **p2i**.

Algorithm 4 Pseudo-code of Naive Auction.

Input: A cost matrix $\mathbf{C}$; the number of persons n; the number of items m ($m \geq n$).
1: positive scalar $\epsilon \Leftarrow \frac{1}{n}$;
2: bids table $\mathbf{b} \Leftarrow [0] \times (n+1)m$;
3: person2item mapping **p2i** $\Leftarrow [-1] \times n$;
4: item2person mapping **i2p** $\Leftarrow [-1] \times m$;
5: price vector $\mathbf{p} \Leftarrow [0] \times m$;
6: **while** (**p2i** includes any -1) **do**
/* bidding phase */
7: **for** i **in** range(n) **do**
8: **if p2i**$[i] == -1$ **then**
9: $\mathbf{t} \Leftarrow \mathbf{C}[i, :] - \mathbf{p}$;
10: $x_1, x_2, x_{idx} \Leftarrow bid(\mathbf{t})$;
11: $\mathbf{b}[i, x_{idx}] \Leftarrow x_1 - x_2 + \epsilon$;
12: $\mathbf{b}[n, x_{idx}] \Leftarrow 1$;
13: **end if**
14: **end for**
/* assignment phase */
15: **for** j **in** range(m) **do**
16: **if b**$[n, j] == 1$ **then**
17: $x_1, x_{idx} \Leftarrow assign(\mathbf{b}[:n, j])$;
18: $\mathbf{p}[j] \mathrel{+}= x_1$;
19: **if i2p**$[j] > -1$ **then**
20: **p2i**[**i2p**$[j]] \Leftarrow -1$;
21: **end if**
22: **p2i**$[x_{idx}] \Leftarrow j$;
23: **i2p**$[j] \Leftarrow x_{idx}$;
24: **end if**
25: **end for**
/* reset bids table */
26: $\mathbf{b} \Leftarrow [0] \times (n+1)m$;
27: **end while**
Output: **p2i**; **i2p**.

In the bidding phase (line 7–14), a temporary vector $\mathbf{t}$ with length m is first calculated via $\mathbf{C}[i, :] - \mathbf{p}$ where **p2i**$[i] == -1$. For each $\mathbf{t}$, $bid(\cdot)$ is employed to obtain the max value

x_1 with its index x_{idx} and the second max value x_2. Then the biding increment is computed and set into $\mathbf{b}[i, x_{idx}]$ (line 11); the flag in $\mathbf{b}[n, x_{idx}]$ is set to 1 (line 12).

In the assignment phase (line 15–25), for each vector $\mathbf{b}[:n, j]$ where $\mathbf{b}[n, j] == 1$, $assign(\cdot)$ is employed to obtain the max value x_1 with its index x_{idx}. Then x_1 is added into $\mathbf{p}[j]$ (line 18) and a new mapping is established between the x_{idx} person and the j item (line 19–23).

Next, all positions in $\mathbf{b}$ are reset to 0 values for the next iteration (line 26).

Computation In the CUDA implementation, the bidding phase and the assignment phase are established into two separate kernel functions.

In the bidding kernel function, the grid includes n blocks to obtain the $x_{1/2/idx}$ of different rows in $\mathbf{C}$ simultaneously. Each block contains $\lceil \frac{m}{2} \rceil$ threads and within each thread, two different values are read from $\mathbf{t}$ and the CUDA built-in function `atomicMax()` is employed to obtain the $x_{1/2/idx}$.

In the assignment kernel function, the grid includes m blocks to obtain the $x_{1/idx}$ of different columns in $\mathbf{b}$ simultaneously. Each block contains $\lceil \frac{n}{2} \rceil$ threads and within each thread, two different values are read from $\mathbf{b}[:n, j]$ and `atomicMax()` is also employed to obtain the $x_{1/idx}$.

Appendix B MOT17-val

MOT17 (Milan et al., 2016) contains a train set with the ground truth and a test set without the ground truth. For the ablation study, we define the first half of each video in the original MOT17 train set as the train set of the ablation dataset and the last half as the validation set of the ablation dataset called MOT17-val following (Zhang et al., 2022; Zhou et al., 2020). Table 15 shows the basic information about the ablation dataset. There are seven videos with 5316 images and the detector YOLOX performs both high recall and precision on the seven videos except MOT17-02. As illustrated in Fig. 10, the validation video of MOT17-02 in MOT17-val has a total of 300 images (numbered from Frame299 to Frame469) and the pedestrians in the green dashed boxes are obscured most of the time from Frame299 to Frame469 (a total of 170 frames) while they are stilled labeled as the ground truth even if they are completely invisible. This is the reason why the validation video of MOT17-02 has the lowest recall among all the seven videos. MOT17 has a tendency to label pedestrians that are not visible at all as the ground truth. This is reflected in all seven videos, with MOT17-02 being the most severe, which explains why all recalls in Table 15 are lower than precisions, i.e., the detector cannot detect fully obscured pedestrians who have all ready been labeled as the ground truth. Fortunately, this situation is not serious in the other six validation videos. For this reason, the metric number obtained on video MOT17-02 in the ablation studies would be considered less meaningful and we only use the remaining six videos as MOT17-val in our ablation studies.

Fig. 10 Illustration of MOT17-02 in Val Half. FrameXXX denotes the frame number in the video (start from 299 to 600). The solid boxes with same color in different images represent the ground truth of the same trajectory; the small pedestrians in the dashed boxes are also marked as the ground truth even if they are completely invisible (Color figure online)

Table 15 Basic information about the ablation dataset

Name	Train Half	Val Half	Rcll.%	Prcn.%
MOT17-02	301	299	64.2	86.0
MOT17-04	526	524	93.4	94.6
MOT17-05	419	418	81.7	95.0
MOT17-09	263	262	83.9	99.5
MOT17-10	328	326	75.6	93.7
MOT17-11	451	449	81.4	87.7
MOT17-13	376	374	80.3	97.3
Overall	2664	2652	83.1	93.0

Train Half and Val Half mean the number of images within each half video. Rcll. and Prcn. are the recall and precision of the MOT17-val of the ByteTrack

References

Bertsekas, D. P. (1992a). Auction algorithms for network flow problems: A tutorial introduction. *Computational Optimization & Applications, 1*(1), 7–66.

Bertsekas, D. P. (1992b). Auction algorithms for network flow problems: A tutorial introduction. *Computational Optimization and Applications, 1*(1), 7–66.

Bewley, A., Ge, Z., & Ott, L., et al. (2016). Simple online and realtime tracking. In: *ICIP* (pp. 3464–3468). IEEE.

Bishop, G., Welch, G., et al. (2001). An introduction to the kalman filter. *Proc of SIGGRAPH, Course, 8*(27599–23175), 41.

Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020) Yolov4: Optimal speed and accuracy of object detection. arXiv:2004.10934.

Brasó, G., & Leal-Taixé, L. (2020) Learning a neural solver for multiple object tracking. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 6247–6257).

Cao, J., Weng, X., & Khirodkar, R., et al. (2022). Observation-centric sort: Rethinking sort for robust multi-object tracking. https://doi.org/10.48550/ARXIV.2203.14360.

Chen, K., Wang, J., & Pang, J., et al. (2019). MMDetection: Open mmlab detection toolbox and benchmark. arXiv:1906.07155.

Choi, W. (2015). Near-online multi-target tracking with aggregated local flow descriptor. In *Proceedings of the IEEE international conference on computer vision* (pp. 3029–3037).

Chu, P., Wang, J., & You, Q., et al. (2021) Transmot: Spatial-temporal graph transformer for multiple object tracking. arXiv:2104.00194.

Dendorfer, P., Rezatofighi, H., & Milan, A., et al. (2020). Mot20: A benchmark for multi object tracking in crowded scenes. arXiv:2003.09003.

Ge, Z., Liu, S., & Wang, F., et al. (2021). Yolox: Exceeding yolo series in 2021. arXiv:2107.08430.

Geiger, A., Lenz, P., Stiller, C., et al. (2013). Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research, 32*(11), 1231–1237.

Gonzalez, N. F., Ospina, A., & Calvez, P. (2020). Smat: Smart multiple affinity metrics for multiple object tracking. In *International conference on image analysis and recognition* (pp. 48–62). Springer.

Gustafsson, F., Gunnarsson, F., Bergman, N., et al. (2002). Particle filters for positioning, navigation, and tracking. *IEEE Transactions on Signal Processing, 50*(2), 425–437.

Hu, H. N., Yang, Y. H., Fischer, T., et al. (2022). Monocular quasi-dense 3d object tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 45*(2), 1992–2008.

Jonker, R., & Volgenant, A. (1987). A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing, 38*(4), 325–340.

Julier, S. J., & Uhlmann, J. K. (1997). New extension of the kalman filter to nonlinear systems. In *Signal processing, sensor fusion, and target recognition VI* (pp. 182–193). Spie.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Journal of Fluids Engineering, 82*(1), 35–45.

Kim, A., Ošep, A., & Leal-Taixé, L. (2021). Eagermot: 3d multi-object tracking via sensor fusion. In *2021 IEEE international conference on robotics and automation (ICRA)* (pp. 11315–11321). IEEE.

Kuhn, H. W. (1955). The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly, 2*(1–2), 83–97.

Li, W., Xiong, Y., & Yang, S., et al. (2021). Semi-tcl: Semi-supervised track contrastive representation learning. arXiv:2107.02396.

Lin, T. Y., Goyal, P., & Girshick, R., et al. (2017). Focal loss for dense object detection. In *ICCV* (pp. 2980–2988).

Lu, Z., Rathod, V., & Votel, R., et al. (2020). Retinatrack: Online single stage joint detection and tracking. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 14668–14678).

Luiten, J., Osep, A., Dendorfer, P., et al. (2021). Hota: A higher order metric for evaluating multi-object tracking. *International Journal of Computer Vision, 129*(2), 548–578.

Milan, A., Leal-Taixé, L., & Reid, I., et al. (2016). Mot16: A benchmark for multi-object tracking. arXiv:1603.00831.

Okuta, R., Unno, Y., & Nishino, D., et al. (2017). Cupy: A numpy-compatible library for nvidia gpu calculations. In *Proceedings of workshop on machine learning systems (LearningSys) in the thirty-first annual conference on neural information processing systems (NIPS)*. http://learningsys.org/nips17/assets/papers/paper_16.pdf

Pang, J., Qiu, L., & Li, X., et al. (2021). Quasi-dense similarity learning for multiple object tracking. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 164–173).

Pedregosa, F., Varoquaux, G., Gramfort, A., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*, 2825–2830.

Peng, J., Wang, C., & Wan, F., et al. (2020). Chained-tracker: Chaining paired attentive regression results for end-to-end joint multiple-object detection and tracking. In *European conference on computer vision* (pp. 145–161). Springer.

Rangesh, A., Maheshwari, P., & Gebre, M., et al. (2021). Trackmpnn: A message passing graph neural architecture for multi-object tracking. arXiv:2101.04206.

Redmon, J., & Farhadi, A. (2018) Yolov3: An incremental improvement. arXiv:1804.02767.

Ren, S., He, K., Girshick, R., et al. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in Neural Information Processing Systems, 28*, 91–99.

Smith, G. L., Schmidt, S. F., & McGee, L. A. (1962). *Application of statistical filter theory to the optimal estimation of position and velocity on board a circumlunar vehicle*. National Aeronautics and Space Administration.

Stadler, D., & Beyerer, J. (2022). Modelling ambiguous assignments for multi-person tracking in crowds. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision* (pp. 133–142).

Sun, P., Cao, J., & Jiang, Y., et al. (2020). Transtrack: Multiple object tracking with transformer. arXiv:2012.15460.

Sun, P., Cao, J., & Jiang, Y., et al. (2021). Dancetrack: Multi-object tracking in uniform appearance and diverse motion. arXiv:2111.14690.

Tokmakov, P., Li, J., & Burgard, W., et al. (2021). Learning to track with object permanence. arXiv:2103.14258.

Virtanen, P., Gommers, R., Oliphant, T. E., et al. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in python. *Nature Methods, 17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2

Wang, G., Gu, R., & Liu, Z., et al. (2021a). Track without appearance: Learn box and tracklet embedding with local and global motion patterns for vehicle tracking. In *Proceedings of the IEEE/CVF international conference on computer vision* (pp. 9876–9886).

Wang, S., Sheng, H., & Zhang, Y., et al. (2021b). A general recurrent tracking framework without real data. In *Proceedings of the IEEE/CVF international conference on computer vision* (pp. 13219–13228).

Wang, Y., Kitani, K., & Weng, X. (2021c). Joint object detection and multi-object tracking with graph neural networks. In *2021 IEEE international conference on robotics and automation (ICRA)* (pp. 13708–13715). IEEE.

Wang, Z., Zheng, L., & Liu, Y., et al. (2020). Towards real-time multi-object tracking. In *Computer vision–ECCV 2020: 16th European conference, Glasgow, UK, August 23–28, 2020, proceedings, Part XI 16* (pp. 107–122). Springer.

Weng, X., Wang, J., & Held, D., et al. (2020). 3d multi-object tracking: A baseline and new evaluation metrics. In *2020 IEEE/RSJ international conference on intelligent robots and systems (IROS)* (pp. 10359–10366). IEEE.

Wojke, N., Bewley, A., & Paulus, D. (2017). Simple online and realtime tracking with a deep association metric. In *2017 IEEE international conference on image processing (ICIP)* (pp. 3645–3649). IEEE.

Wu, J., Cao, J., & Song, L., et al. (2021). Track to detect and segment: An online multi-object tracker. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 12352–12361).

Xiang, Y., Alahi, A., & Savarese, S. (2015). Learning to track: Online multi-object tracking by decision making. In *ICCV* (pp. 4705–4713).

Xu, Y., Ban, Y., & Delorme, G., et al. (2021). Transcenter: Transformers with dense queries for multiple-object tracking. arXiv:2103.15145.

Yang, F., Chang, X., Sakti, S., et al. (2021). Remot: A model-agnostic refinement for multiple object tracking. *Image and Vision Computing, 106*(104), 091.

Yu, E., Li, Z., & Han, S., et al. (2021). Relationtrack: Relation-aware multiple object tracking with decoupled representation. arXiv:2105.04322.

Zeng, F., Dong, B., & Wang, T., et al. (2021). Motr: End-to-end multiple-object tracking with transformer. arXiv:2105.03247.

Zhang, Y., Sun, P., & Jiang, Y., et al. (2022). Bytetrack: Multi-object tracking by associating every detection box. In *ECCV 2022*.

Zhang, Y., Wang, C., Wang, X., et al. (2021). Fairmot: On the fairness of detection and re-identification in multiple object tracking. *International Journal of Computer Vision, 129*(11), 3069–3087.

Zhou, X., Koltun, V., Krähenbühl, P. (2020). Tracking objects as points. In *ECCV* (pp. 474–490). Springer.

Zhou, X., Wang, D., & Krähenbühl, P. (2019). Objects as points. arXiv:1904.07850.