RESEARCH-ARTICLE

A Reconfigurable Architecture for Real-time Event-based Multi-Object Tracking

YIZHAO GAO, The University of Hong Kong, Hong Kong, Hong Kong

SONG WANG, The University of Hong Kong, Hong Kong, Hong Kong

HAYDEN KWOK HAY SO, The University of Hong Kong, Hong Kong, Hong Kong

Published: 01 September 2023
Online AM: 21 April 2023
Accepted: 04 April 2023
Revised: 03 February 2023
Received: 14 September 2022

A Reconfigurable Architecture for Real-time Event-based Multi-Object Tracking

YIZHAO GAO, SONG WANG, and HAYDEN KWOK-HAY SO, University of Hong Kong, Hong Kong

Although advances in event-based machine vision algorithms have demonstrated unparalleled capabilities in performing some of the most demanding tasks, their implementations under stringent real-time and power constraints in edge systems remain a major challenge. In this work, a reconfigurable hardware-software architecture called REMOT, which performs real-time event-based multi-object tracking on FPGAs, is presented. REMOT performs vision tasks by defining a set of actions over attention units (AUs). These actions allow AUs to track an object candidate autonomously by adjusting its region of attention and allow information gathered by each AU to be used for making algorithmic-level decisions. Taking advantage of this modular structure, algorithm-architecture codesign can be performed by implementing different parts of the algorithm in either hardware or software for different tradeoffs. Results show that REMOT can process 0.43–2.91 million events per second at 1.75–5.45 W. Compared with the software baseline, our implementation achieves up to 44 times higher throughput and 35.4 times higher power efficiency. Migrating the Merge operation to hardware further reduces the worst-case latency to be 95 times shorter than the software baseline. By varying the AU configuration and operation, a reduction of 0.59–0.77 mW per AU on the programmable logic has also been demonstrated.

CCS Concepts: • **Computer systems organization → Real-time system architecture**; **Reconfigurable computing**; • **Computing methodologies → Tracking;**

Additional Key Words and Phrases: REMOT, Dynamic Vision Sensors, multi-object tracking, event sensors, event camera, hardware/software co-design, attention unit, FPGA, HOTA

ACM Reference format:
Yizhao Gao, Song Wang, and Hayden Kwok-Hay So. 2023. A Reconfigurable Architecture for Real-time Event-based Multi-Object Tracking. *ACM Trans. Reconfig. Technol. Syst.* 16, 4, Article 58 (September 2023), 26 pages.
https://doi.org/10.1145/3593587

1 INTRODUCTION

Event cameras are neuromorphic imaging devices that have been receiving renewed interest in recent years due to their unique capabilities, including high temporal resolution, high dynamic range, low lighting imaging, and energy efficiency [15]. At the heart of an event camera is a **dynamic**

This work was supported in part by the Research Grants Council (RGC) of Hong Kong under the Research Impact Fund project R7003-21. This work was also supported by AI Chip Center for Emerging Smart Systems (ACCESS), sponsored by InnoHK funding, Hong Kong SAR..
Authors' address: Y. Gao, S. Wang, and H. K.-H. So, Department of Electrical and Electronic Engineering, University of Hong Kong, Chow Yei Ching Building, Pok Fu Lam Rd, Lung Fu Shan, Hong Kong; emails: {yzgao, wangsong, hso}@eee.hku.hk.

1936-7406/2023/09-ART58 $15.00
https://doi.org/10.1145/3593587

vision sensor (DVS), sometimes simply referred to as an event sensor, which detects *changes* in exposed light intensity and reports them asynchronously as spiking events localized to the pixels involved. This contrasts with a conventional image sensor that reports the light intensity of every pixel synchronously at a regular frame rate regardless of the activity level of the scene. With their unique imaging capabilities, event cameras are driving a new generation of event-based vision applications, from internet-of-thing sensing to high-speed autonomous vehicle guidance.

From a data processing point of view, the outputs of event sensors are sparse and asynchronous, while that of traditional image sensors are dense and synchronous. As a result, depending on the activity level of the scene, an event sensor output can switch from an idle state where no event is produced to an active state that produces millions of events per second within microseconds. To cope with such highly varied data processing requirements, it is desirable to develop a flexible architecture that can operate at low power mode efficiently during low activity periods, while being able to transition into a high-performance mode to process bursts of events during high activity periods. Further adding to this processing challenge is the fact that the output event sensors are fragmented with limited visual information when compared to the images produced by a conventional frame-based sensor. Although techniques such as time-surface construction can serve as an intermediate representation on which sophisticated learning-based algorithms may be built upon [22, 38], performing such complex algorithms in edge devices with limited processing capabilities in real time remains an open challenge.

In this work, we present REMOT, a reconfigurable event-based multi-object tracking hardware-software system, which performs the complex high-level vision task of **multi-object tracking (MOT)** in real-time on FPGA with an event camera input. The REMOT tracking algorithm is co-designed with the hardware architecture to facilitate high-performance processing of the event input using a modular framework surrounding the concept of an **attention unit (AU)**. In a REMOT system, an AU is an autonomous entity that tracks the subset of the events falling under its **region of attention (ROA)**. The overall tracking algorithms are then constructed by defining the way these AUs interact with one another, either in hardware or software, based on their aggregated information about the events in its ROA.

To demonstrate the flexibility of the REMOT architecture, a family of event-based multi-object tracking algorithms has been implemented with a wide range of hardware and software configurations. In our baseline architecture, a layer of AUs is implemented in hardware to provide high throughput distributed processing of vision events as they are produced. On top of that, software running on the embedded processor queries the status of each hardware AU and makes decisions to merge or split these AUs to enhance tracking accuracy. In addition, an enhanced architecture with hardware-accelerated AU Merge is presented, demonstrating the benefit of REMOT's modular design. Finally, two low-level hardware techniques have been explored to enhance the power efficiency of the system. In the first case, the maximum number of AUs is configured on the programmable logic of the FPGA, but their usage is limited dynamically during runtime. In the second case, the number of AUs configured is adjusted through the reconfiguration of FPGAs.

Our results show that the proposed architecture is scalable and is capable of processing 0.43 to 2.91 **million events per second (Meps)** while consuming 1.75 to 5.45 W of system power. With regard to MOT accuracy, we show that our proposed attention-guided MOT algorithms can achieve 43.1 % to 73.2 % in terms of the **Higher Order Tracking Accuracy (HOTA)** metric across a range of datasets. By utilizing novel hardware-software codesign strategies, REMOT is able to implement high-level event-based computer vision tasks efficiently on FPGAs. To this end, we consider the main contributions of this work as follows:

- We proposed a reconfigurable event-based multi-object tracking hardware-software system that can effectively support real-time operations in FPGA;

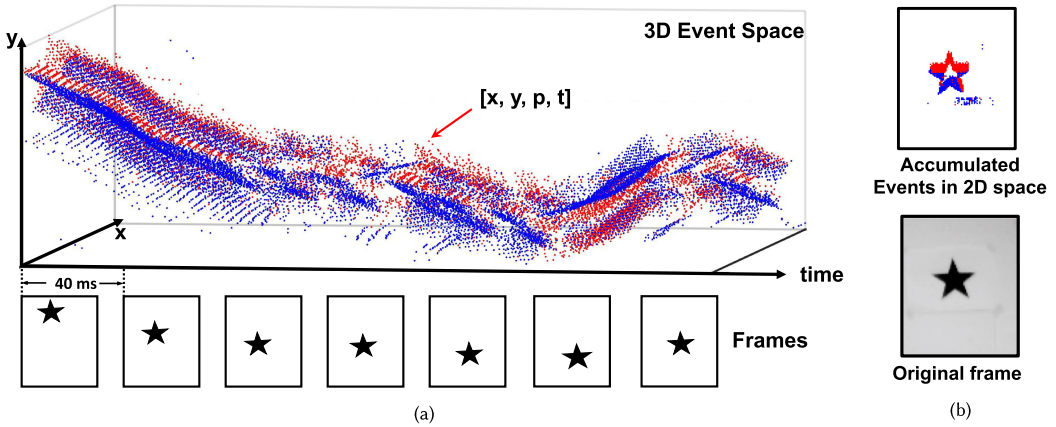


Fig. 1. Comparing working principles of event camera and conventional frame-based cameras. (a) Event is encoded with its pixel location (x, y), polarity p (shown as red and blue dots), and is typically timestamped at 1-μs intervals, t. As the star shape moves, events are produced near the edges of the star where light intensity changes. (b) (Top) Accumulating events forms a two-dimensional representation in the original x-y coordinate. (Bottom) Original frame captured by the APS sensor of DAVIS.

- We demonstrated the flexibility, scalability, power efficiency, and real-time performance of the proposed architecture by performing design space exploration for implementations on two FPGA-based edge platforms;
- We proposed a family of real-time event-based attention-guided multi-object tracking algorithms that run on our proposed architecture and demonstrated their efficacy using a set of real-world traffic monitoring data.

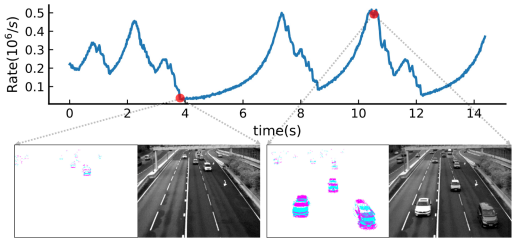
An earlier version of this work appeared in Reference [16]. We extend the original work by introducing additional dynamic power-saving strategies that further improve the system's power efficiency and developing new hardware AU Merge module that greatly enhanced the performance of the original implementation. In the next section, background and related works on event-based vision algorithms will first be discussed. The REMOT hardware-software architecture and algorithm will be discussed in Section 3. An extensive evaluation of our proposed system will be shown in Section 4. Limitations of our current system will be discussed in Section 5, and we will conclude in Section 6.

2 BACKGROUND AND RELATED WORK

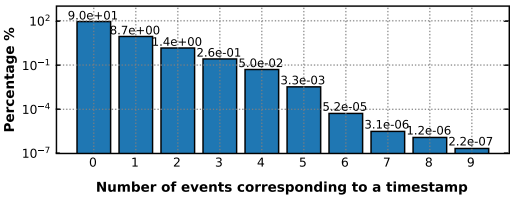
2.1 Dynamic Vision Sensors

Dynamic Vision Sensors only report local brightness changes asynchronously for each pixel. Whenever the change in log intensity of a pixel is higher than a predefined threshold, it emits an event, or spike, which is usually encapsulated in an **address event representation (AER)** format for downstream processing [15]. A typical event in AER can be written as $[x, y, p, t]$, where x, y is the location of the event, p is the polarity of brightness change in ± 1, and t is the timestamp generated by the sensor.

Figure 1 illustrates the working principle of a DVS by showing the outputs of an event sensor alongside a conventional frame-based sensor in a segment of *shapes_6dof* dataset. The figure shows the period when the *star* shape moves relative to the camera. As shown at the top half of Figure 1(a), a DVS reports changes in light intensity asynchronously as spiking events shown as red and blue dots. These events are produced around the edges of the star shape, where the



(a) The number of events produced by a DVS camera depends on the amount of dynamic activity in the scene.



(b) Event statistics of inbound traffic dataset.

Fig. 2. Dynamic properties of a DVS camera.

dark color of the shape interacts with the light background. The events are produced in an almost-continuous fashion as long as there is relative movement between the camera and the star with a temporal resolution of about 1 µs. Conversely, events are rarely produced when there is no change in light intensity, such as near the background and in the middle segment of the star trajectory. When the events in a small time window are accumulated, they form a projection of the events in the original imaging two-dimensional (2D) x-y coordinate as illustrated at the top of Figure 1(b). The bottom of the figure shows the corresponding frame captured during this time.

However, the lower half of Figure 1(a) illustrates the output of a conventional frame-based camera. With a conventional camera, frames are captured at regular intervals regardless of the activity of the scene. The frame capturing rate determines the temporal resolution of the camera. Each frame is a dense representation of the entire field of view regardless of any object movement. As a result, redundant information, such as identical background, is captured between frames, while information during the time between two consecutive is lost.

Due to its asynchronous behavior, a variable data rate can be expected from DVS depending on the activity level of the scene. Figure 2(a) shows the rate of events produced over time in a traffic scene. As the cars move toward the camera, the relative speed in the view increases, which results in higher event rates. Common industrial DVS timestamps events in 1-μs resolution. However, in some active dynamic scenes, more than one event might share the same timestamp value. Figure 2(b) shows the statistics in our traffic dataset. The special column labeled as "0" corresponds to the percentage of time when no event is generated, which captures its sparsity in the time dimension. In this particular example, the DVS was idle 90% of the time. Furthermore, among all the timestamps with events, 83% contains only 1 event. On average, the data rate of our current dataset ranges from 0.22 to 0.3 Meps. This rate ultimately determines the minimum average processing throughput our proposed hardware-software system must achieve to avoid dropping events.

In this work, we employed an advanced **Dynamic and Active Pixel Vision Sensor (DAVIS)** camera [7] that implements both a dynamic vision sensor and a conventional frame-based **active pixel sensor (APS)** on the same pixel array. Since the two sensors are integrated at the pixel level, no image registration is needed between the events and frame output. We took advantage of this feature to produce ground-truth bounding boxes in our dataset.

2.2 Hardware Processing of DVS Output

The asynchronous and sparse nature of DVS output brings both opportunities and challenges to processing them efficiently in hardware. Table 1 shows a list of previous works that demonstrated efficient hardware processing of DVS output. Depending on the operating principle of a work's main algorithm, two different performance measurements have typically been used in the literature regarding DVS processing in hardware. Designs with their main algorithms operating on raw

Table 1. Previous Hardware Deployments of Event-based Vision Tasks Using Dynamic Vision Sensors

	Hardware Platform	Task	Main Algorithm	Per-Event Processing	Performance	Chip/System Power (W)
[36]	Loihi	Object Tracking	SNN	✓	—	—/—
[40]	FPGA+TrueNorth	Object Tracking, Classification	Event-based Tracker+SNN	✗	15 fps	0.55/-
[37]	Neuromorphic Chips	Object Recognition, Tracking	SNN	✓	3 Meps	0.4/—
[24, 25]	FPGA	Object Tracking	Center of Mass Calculation	✓	120–140 ns	—/10
[39]	FPGA	Gesture Recognition	Hierarchy of Time Surface	✓	0.16–2 Meps	0.077/1.6
[32]	FPGA	Pedestrain Detection	BNN	✗	130 fps	—/—
[35]	FPGA	Object Detection	PCA, kd-tree, SVM	✓	550 ns	0.37/3
[26]	FPGA	Object Classification	CNN	✗	160 fps	0.27/1
[33]	GPU	Object Detection	CNN	✗	25 fps	—/—
REMOT	FPGA	Object Tracking	Attention-Guided MOT	✓	0.43–2.91 Meps	—/(1.75–5.45)

event input from the DVS usually emphasize their event processing throughput as measured in Meps. These works are marked with a checkmark in the table under the "Per-event processing" column. However, a group of event-based hardware algorithms performed computer vision tasks using framelike intermediate representations such as by aggregating events over a time window. In these cases, the literature typically reports performance in terms of frames per second (fps).

Owing to the neuromorphic nature of DVS, a number of works have explored the use of **spiking neural networks (SNN)** [19] to perform dynamic vision tasks including object classification [10, 27] and object tracking [36, 37]. Subsequently, from a hardware implementation perspective, both dedicated SNN chip [37] or general-purpose SNN accelerators including Intel Loihi [11] and IBM TrueNorth [29] have been used to accelerate the corresponding SNN inference task, resulting in highly energy-efficient processing in general.

At the same time, another school of work approached the challenge of performing event-based vision tasks by developing custom architecture and algorithms that operate on the DVS events natively. For instance, Reference [25] developed a real-time object-tracking system based on center-of-mass computation using FPGA for object tracking. In Reference [39], a hand-gesture recognition system with real-time FPGA implementation by using a hierarchy of time-surface was proposed. In Reference[35], object classification and detection was performed by mapping and categorizing the input events using PCA-RECT transform on FPGA.

Recently, leveraging their extraordinary success in processing conventional frame-based images, deep learning methods that utilize **convolutional neural networks (CNNs)** have also been exploited to process DVS output for various dynamic vision tasks [26, 32, 33]. Unfortunately, typical CNN accelerators are designed to operate with dense tensors and thus cannot fully take advantage of the sparseness of DVS output to improve power efficiency. To address that, some progress has been made in accelerating CNN with sparse feature maps, making them suitable for inference on DVS histogram output [2].

REMOT follows the line of work of operating directly on the event output from a DVS by introducing a customized HW/SW architecture to perform multi-object tracking. However, we further expand its flexibility by allowing it to be reconfigurable and programmable through dedicated codesign strategies.

2.3 Multi-object Tracking Using DVS

Multi-object tracking [34] is a challenging computer vision task that tracks multiple objects in a dynamic scene. Apart from detecting an object in a scene, it also requires an algorithm to assign a unique index to each independent object and track its trajectory. There has been a wide variety of MOT algorithms proposed for the frame-based camera, with "tracking by detection" being the mainstream [3, 4, 6]. It is mainly composed of two steps: (i) apply a detector to detect objects in each frame and (ii) perform association on the detected objects across frames.

Following a similar approach, a number of event-based object-tracking algorithms have been proposed. For instance, in Reference [23], the authors demonstrated effective tracking using a correlation filter on top of a CNN structure. In the work of EBBIOT [1], a vehicle tracking system based on event sensors was demonstrated. Using an adaptive time surface formulation of events, Chen et al. [8] have demonstrated multiple object tracking in a controlled environment. Leveraging recent advent in machine learning, an offline-online learning approach was proposed in Reference [21] to perform event-based object tracking with comparable performance to frame-based algorithms. In a recent work of EKLT [17], the use of simultaneous event and frame-based input to perform feature tracking was proposed.

While the above works have achieved good object-tracking performance, their complex designs were not optimized for real-time implementations. For that, the authors of E-MS [5] demonstrated an effective real-time event-based multi-object tracking by performing mean-shift clustering on the incoming events as they were produced. Subsequently, in Reference [25], the authors demonstrated a low-latency event-based object tracker by performing inline center-of-mass computation using FPGA.

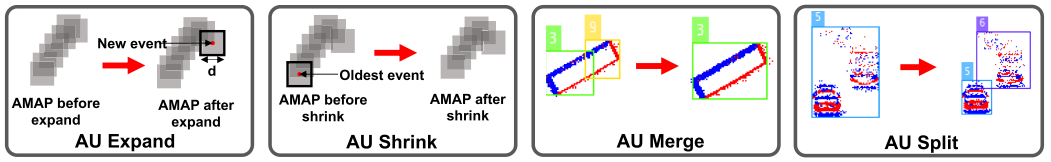
Our proposed REMOT framework similarly performs multi-object tracking directly on the dynamic vision events as they are produced. Taking advantage of the high temporal resolution and spatial sparsity of DVS, REMOT identifies and tracks multiple objects simultaneously using a layer of AUs. Each AU is an independent tracker that only pays attention to a small region that is updated in a per-event manner (Section 3). As an object moves, the attention region follows its motion based on the corresponding events that are produced, thereby tracking the moving object. During the lifetime of an AU, it will be assigned a unique global index as the tracking ID. Therefore, no object association in the traditional MOT sense is needed. Instead, supervisory functions are needed to ensure that each AU is indeed tracking useful objects. These high-level decisions are made based on the status of the AUs and may operate in millisecond scale comparable to a frame speed. In this way, a hierarchy of vision system is established with different processing rates on different levels.

3 REMOT ARCHITECTURE AND ALGORITHMS

REMOT defines a hardware-software architecture and its associated operations that allow real-time event-based multi-object tracking algorithms to be developed. The design of REMOT is based on the notion of an AU. An AU is an autonomous entity that observes events from a DVS as they are produced. An AU maintains an ROA that defines the area in the imaging field where this AU is currently paying attention to. An implementation of REMOT will typically include a large set of independent AUs, which collectively attend to different parts of the imaging field where there are interesting events. Furthermore, a set of high-level algorithms oversee the operations of the entire set of AUs and make group decisions based on the state of each individual AU. It is by carefully manipulating the actions of the AUs that a family of attention-guided multi-object algorithms can be defined. The basic abstractions of AU actions are as follows.

Expand. An AU may choose to expand its ROA when it observes an event that falls within its ROA. In other words, this event is considered as *interested* by the AU. In that case, the AU captures the event and adjusts its ROA centered around the new events. All the events that fall outside the current ROA will be ignored by the AU. Consequently, the ROA remains unchanged. The expanded region is designed to be a $d \times d$ square area center at the new events as shown in Figure 3(a) our current implementation. The captured events will be pushed into an Active Event FIFO.

Shrink. An AU may choose to shrink its ROA as events age and no longer require attention. In that case, the AU may *forget* the event according to criteria set up by the algorithm. In REMOT, the order to shrink events is identical to the order as it is captured, which follows a *First-in, First-out* manner. Thus, we use an Active Event FIFO to temporarily store events considered as interested.



(a) (b) (c) (d)

Fig. 3. Primitives of AU actions. (a) Expand: An AU expands its ROA when a new event falls within its ROA. (b) Shrink: An AU shrinks its ROA when an old event is discarded. (c) Merge: Two AUs aggregate and become a new AU. (d) Split: An AU splits into multiple AUs.

As shown in Figure 3(b), the event popped out from the Active Event FIFO will shrink the attention center around its location similar to the Expand operation.

Merge. An algorithm may choose to merge two AUs during runtime as more information about each AU is aggregated. When two AU merges, a new AU is formed with a combined ROA that is the union of the two original ROAs. We propose two Merge algorithms, the Distance-based algorithm and the Ratio-based algorithm. If the Hausdorff distance [20] between two sets of active events from two AUs is less than a threshold value, or the ratio of **Interaction over Minimum (IoM)** is larger than a threshold value, then the neighboring AUs will be merged.

Split. An algorithm may decide that an AU should split into multiple AUs depending on algorithm-specific criteria. When an AU splits, two new AUs are formed with each of them inheriting a subset of the original region of attention. In REMOT, we leverage different cluster algorithms to determine whether the internal events develop into different separated groups. We employ two clustering algorithms to partition AUs: density-based Split algorithm and hierarchy-based Split algorithm. Each attention region of the separated AU is reconstructed using the new clustered events.

Spawn. An AU is spawned when no existing AU is interested in a new coming event and the ROA of the new AU will center around the new event. Besides, Merge/Split algorithms can also decide to spawn a new AU with aggregated/separated events. In software implementations, an unlimited number of AUs can be spawned. However, in hardware implementations, the number of physical AUs is fixed and limits the maximum number of AU that can be spawned.

Delete. An AU is deleted when its information is already aggregated by the Merge operation. An AU can also be deleted when its ROA remains unchanged for a long period of time. In hardware implementations, the deleted AU will be idle and wait to be spawned by new events or Merge/Split.

3.1 A Reconfigurable Architecture for REMOT

Figure 4 shows the overall system architecture of REMOT. It typically targets an embedded system with both microprocessor and FPGA fabric, e.g., a Zynq MPSoC device with **Programmable Logic (PL)** and **Processing System (PS)** units.

In its basic form, the Expand and Shrink operations are implemented on hardware to process each event while a software-based controller running on the microprocessor with Merge and Split algorithm will oversee all AUs on hardware and perform group decisions. The proposed hardware architecture takes the event stream as input. Each event will be broadcast to all hardware AUs to check whether the event lies within their attention region. The AU that considers the new event interested will push the event into its Active Events FIFO and updates its ROA. If none of the active AU is interested in the new event, then a new hardware AU will be spawned if there is still an idle AU. When an old event is popped out by the AU from its Active Event FIFO, the corresponding attention region will then shrink. The hardware also provides interfaces that allow the controller

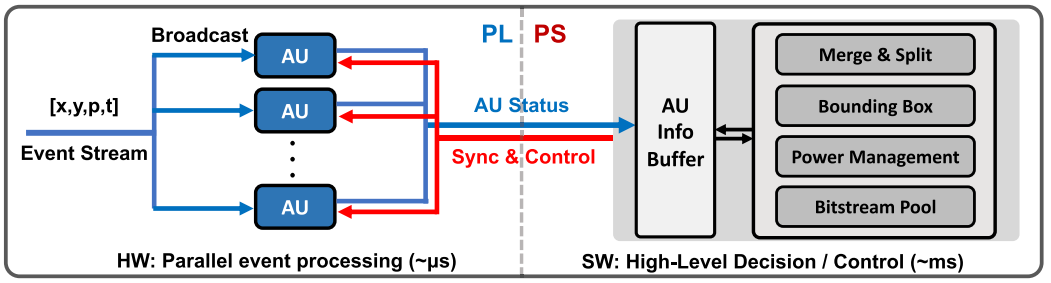


Fig. 4. HW/SW architecture.

ALGORITHM 1: Expand Algorithm with AMAP

Input: $event\ [x, y, p, t]$

```
if amap[x, y] > 0 then                                   ▷ Check if interested in the new event
    efifo.push(event)
    for i = −(d − 1)/2 + x : (d − 1)/2 + x do
        for j = −(d − 1)/2 + y : (d − 1)/2 + y do
            if i ≥ 0 AND i < W AND j ≥ 0 AND j < H then   ▷ Check boundary
                amap[i, j] += 1                          ▷ Expand attention
            end if
        end for
    end for
end if
```

on PS to read and manipulate the status of AUs, e.g., the attention map and Active Event FIFO, to enable Merge and Split operations.

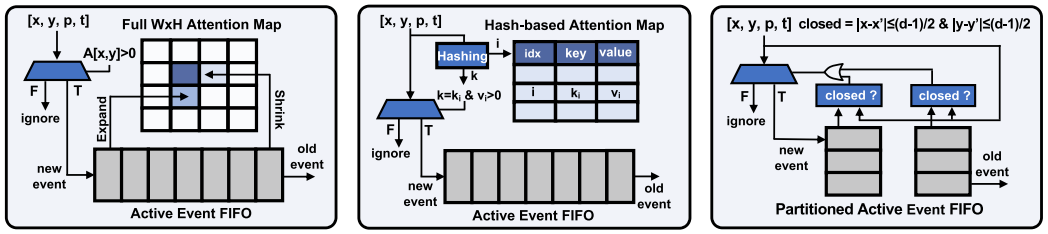
However, the hardware/software architecture can be constructed differently by reconfiguring the system with different implementations of the AU functions. For example, a hardware Merge unit can also be implemented on PL to facilitate the decision-making and attention aggregation of the Merge algorithm. Besides, the hardware can also be reconfigured with different maximum numbers of AUs or different micro-architectures of AUs to achieve flexible tradeoffs in power efficiency, throughputs, and so on. In the next few sections, we will further discuss the detailed design of the reconfigurable architecture for different functional units.

3.2 Implementation of AU with Expand/Shrink Actions

3.2.1 Software Implementation. In this section, we first discuss the software implementation of AU with Expand and Shrink actions. The software-only implementation is considered the baseline and is also used to study the MOT algorithms. Then we introduce three different hardware implementations of AU with Expand/Shrink operations that can achieve high-throughput and real-time processing of events *in situ* as they are produced by the event camera.

The software-based Expand and Shrink are centered around the use of an **attention map (AMAP)** in each AU that records its current ROA. An AMAP can be realized as a 2D matrix with the size of the camera that records the magnitude of attention in each pixel location. Each AU also contains an Active Event FIFO to store its recently interested events. When a new event arrives within the ROA of an AU (i.e., AMAP$[x, y] > 0$), the $d \times d$ area of the AMAP centered at $[x, y]$ is incremented by 1 and the event is pushed into the FIFO.

Algorithm 1 shows the pseudo code of the described Expand algorithm. However, the Active Event FIFO is set to a fixed depth. As a result, the oldest event in the FIFO will be popped out if the



(a) Full-amap (b) Hash-amap (c) Fifo-only

Fig. 5. Three different hardware implementations of an AU. (a) Full-amap implementation uses a full $W \times H$ size attention map to record attention. (b) Hash-amap implementation reduces the size of the buffer by using a hash table. (c) Fifo-only implementation relies merely on Active Event FIFO by comparing the Chebyshev distances between new events and the captured events. The "closed" function returns True if the distance is smaller than the Expand/Shrink ROA radius $(d-1)/2$.

FIFO is full and a new event is pushed in, which works similarly to a circular buffer. When an event is popped out from the FIFO, the corresponding $d \times d$ area will be decremented by 1, shrinking the AMAP accordingly.

3.2.2 Hardware Implementation. In terms of the hardware implementation of AU, we introduce three different designs: Full-amap, Hash-amap, and Fifo-only with different tradeoffs between accuracy, throughput, resource, and power consumption. The hardware diagram of the three implementations are shown in Figure 5.

Full-amap Implementation of AU. The Full-amap is a straightforward implementation similar to the software implementation without additional hardware optimization. Each AU is composed of a $W \times H$ AMAP and an Active Event FIFO, where W and H are the width and height of the camera. The hardware Expand and Shrink behaviors follow the same procedures as the software implementation described above. **Block RAM (BRAM)** is used to store AMAP on FPGA, which takes around $d \times d$ cycles to update for every Expand or Shrink operation. The Full-amap is considered as a costly hardware baseline without taking advantage of the inherent spatial sparsity of the DVS. The BRAM usage for an AU in Full-amap can be estimated as

$$B_{\text{Full-amap}} = \lceil (W \times H \times 16 + D_{FIFO} \times 64)/16\text{Kb} \rceil, \tag{1}$$

where D_{FIFO} is the depth of the 64 bits Active Event FIFO and the attention map uses 16 bits precision. In addition, the throughput of the Full-amap implementation is mainly determined by Expand and Shrink size d, which can be estimated using

$$T_{\text{Full-amap}} = freq/(2(d^2 + C)), \tag{2}$$

where $freq$ is the clock frequency of the programmable logic and C is a constant overhead including the pipeline latency.

Hash-amap Implementation of AU. As mentioned above, the inherent spatial sparsity in DVS brings great opportunities for hardware optimization. Thus, hash table becomes a good candidate to implement a sparse attention map in AU. By only storing non-zero attention regions in a hash table, large on-chip memory is saved for each AU. In this way, more AUs can be deployed under the same resource constraints. Figure 6 shows the diagram of the hashing scheme. For each location $[x, y]$, the hash key can be generated using

$$key = f(x, y) = y \times W + x + 1 \tag{3}$$

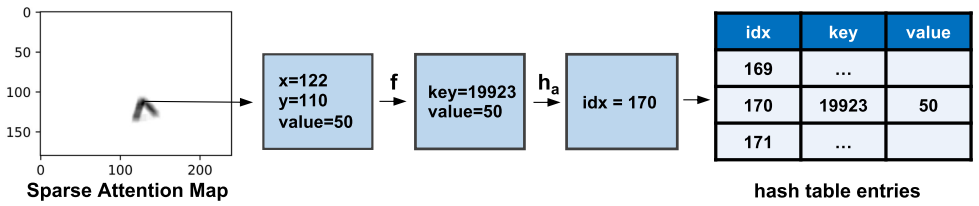


Fig. 6. The hashing diagram of the sparse attention map in Hash-amap implementation. Function f and h_a refer to Equations (3) and (4).

ALGORITHM 2: Check attention with Hash-amap implementation

Input: $event\ [x, y, p, t]$
Output: If an AU is interested in an event
1: $key = f(x, y)$
2: $i = h_a(key)$
3: **for** all s in S **do** ▷ S number of slots in each hash table entry
4: $k = key[i][s]$
5: $v = value[i][s]$
6: **if** $key == k$ AND $v > 0$ **then**
7: return True
8: **end if**
9: **end for**
10: return False

where W is the width of the camera screen. This key generation function calculates the flattened 1D array index of $[x, y]$ in the original 2D $W \times H$ space. It ensures that a unique hash key can be assigned to each location in the $W \times H$ space. Other mapping functions that can guarantee this uniqueness are also acceptable. In addition, we choose the binary multiplicative hashing function [13] to obtain the index of the hash table entry to store a key–value pair. For a hash table with 2^l entries and w bits hash key, the hashing function can be described as

$$idx = h_a(key) = (a \times key)[w - 1 : w - l], \tag{4}$$

where a is a w bits constant number and $[w - 1 : w - l]$ refers to $w - l$ to $w - 1$ bits of the $2w$ bits multiplication result. In this way, we obtain an l bits index to store a key–value pair in the hash table. For a sparse attention map in Figure 6, the hash table only stores the non-zero locations, which also corresponds to the ROA of the AU. When a new event comes, it will query the hash table using Equations (3) and (4). If finding a matched key and its associated value is positive, then this event will be considered interested, shown in Algorithm 2. Similarly to Full-amap implementation, the newly captured event will be pushed into the Active Event FIFO. The AU will also iterate through the $d \times d$ area center at the newly captured event and increment the attention values in the hash table.

However, using a hash table to store the sparse attention map also brings in the problem of hash collision. In other words, two different non-zero locations might be mapped into the same hash table entry. To simplify the design and achieve higher throughput, we use chaining as the collision handling strategy by allocating multiple slots for each hash table entry. If a collision happens, then the new key–value pair goes to the next available slot. Otherwise, this key–value pair will simply be dropped. The throughput of the Hash-amap implementation is similar to the Full-amap in Equation (2). However, the BRAM consumption can be largely reduced by using hash table, which

can be written as

$$B_{\text{Hash-amap}} = \lceil((w + 16) \times 2^l \times S + D_{FIFO} \times 64)/16\text{Kb}\rceil, \tag{5}$$

where S is the number of slots in one hash table entry. Similarly, the precisions of the attention map and FIFO are 16 and 64 bits. The hash key precision w is set to 18 in our implementation.

Fifo-only Implementation of AU. Both Full-amap and Hash-amap implementation keep a record of the attention map to determine whether the new coming event is considered interested. They achieve constant query time but spend around d^2 cycles to update the ROA. The Fifo-only implementation reformulates the attention-based algorithm differently and leads to a different design tradeoff. It compares the location of a new coming event with all the existing events in the Active Event FIFO. If its location lies within the expanding region of any active event, then the new event will be considered interested and be pushed into the Active Event FIFO. The algorithm can be formulated as

$$interested = \exists\ e \in F, s.t.\ |e.x - x| \leq (d-1)/2\ \&\ |e.y - y| \leq (d-1)/2, \tag{6}$$

where F is the Active Event FIFO. The above formulation is also equivalent to finding whether there exists an event in the FIFO with Chebyshev distance less than $(d-1)/2$ from the new event.

Different from the attention-map-based methods, the complexity of query is O(n), and the complexity of updating FIFO is O(1) for Fifo-only, where n refers to the depth of the Active Event FIFO. Therefore, the throughput of Fifo-only is bounded by how fast it can traverse the entire FIFO. This generally means that the conventional hardware implementation of a FIFO with one push/pop operation per cycle is incapable of this design. For example, if the FIFO depth is 1,024, then it takes at least 1,024 cycles to determine whether a new event is interested or not. Assuming the PL fabric runs at 100 MHz, the throughput will be less than 0.1 Meps, which is slower than the real-time requirement. Thus, as depicted in Figure 5(c), we can achieve parallel access to the FIFO by partitioning. Given a target throughput T, we can estimate the partition factor of the FIFO and the overall BRAM consumption using

$$\begin{aligned} P &= \lceil D_{FIFO}/(T/freq)\rceil \\ B_{\text{Fifo-only}} &= P \times \lceil 64 \times D_{FIFO}/(P \times 16\text{Kb})\rceil, \end{aligned} \tag{7}$$

which leads to a different throughput-resources tradeoff compared to the attention-map-based implementations.

3.2.3 Applicability of Different AU Implementations. Previous sections have introduced the software implementation and three hardware implementations of AU. In general, their functionalities are the same but can tradeoff between performances and resources to satisfy the actual application constraints. Specifically, one can choose the Fifo-only implementation if a specific throughput requirement should be satisfied. This can be done by using different FIFO partition factors (shown in Equation (7)) to tradeoff between BRAM consumption and performances. While the Full-amap and Hash-amap implementations are able to provide opportunities to be integrated with other frame-based algorithms like CNN by using the attention map. Last, the software version of AU can apply to cases in a CPU-only system with fewer AU requirements, but the performances hardly scale with a large number of AUs. More detailed results of different tradeoffs in AU implementations will be discussed in Section 4.2.3.

ALGORITHM 3: Merge Algorithm

```
for i = 0 : Number of AUs − 1 do
    for j = i + 1 : Number of AUs do
        if IOM(AUs[i], AUs[j]) > α then
            events = Unique(AU[i].efifo, AU[j].efifo)      ▷ Combine the events and remove identical ones
            events = Sort(events.t)                        ▷ Sort events using timestamps in descending order
            events = events[0 : FIFO_DEPTH]                ▷ Shrink old events to fit maximum FIFO size
            amap = [W, H]                                  ▷ Create new empty attention map
            for all e in events do                         ▷ Reconstruct attention map
                Expand(amap, e)
            end for
            delete AUs[i], AUs[j]
            au = AU(events, amap)                          ▷ Instantiate new AU
            AUs.append(au)
        end if
    end for
end for
```

3.3 Implementation of Merge Algorithms

3.3.1 Software Implementation. We propose two Merge decision algorithms for software implementation running on the processor to oversee hardware AUs, the Distance-based and the Ratio-based algorithms.

Distance-based Merge Algorithm. This method uses the Hausdorff distance to determine whether two neighboring AUs should be merged into a single AU. If the Hausdorff distance between two sets of active events from two AUs is less than a threshold value, then a *Merge* operation will be carried out. The Hausdorff distance is effective in evaluating the distance between two sets of points, defined as

$$\begin{aligned} H(A,B) &= \max(h(A,B), h(B,A)) \\ h(A,B) &= \max_{a\in A} \min_{b\in B} \|a-b\|_2, \end{aligned} \tag{8}$$

where $\|\cdot\|$ denotes the L2 norm and A and B are two set of points.

Ratio-based Merge Algorithm. This Merge algorithm decides whether to merge two neighboring AUs based on the ratio of IoM. If the IoM ratio is larger than a threshold value, then the neighboring AUs will be combined. The IoM ratio is defined as the overlapping area over the minimum area of ith and jth AUs, which can be written as

$$\text{IoM} = (area(\text{i}) \cap area(\text{j}))/min(area(\text{i}), area(\text{j})), \tag{9}$$

where $area(\cdot)$ returns the bounding box area of an AU.

When two AUs are decided to merge by the algorithms, the events in the Active Event FIFO from both AUs will be joined and sorted based on the timestamp. Since old events have a larger timestamp, sorting the aggregated events maintains the *First-In, First Out* order of events in the new AU, and old events can also shrink properly. When two AUs are merged, it is very likely that they have paid attention to similar events for a period of time. As a result, identical events captured by two AUs should also be reduced. After reducing duplicated events, the new AU can operate as if it captures these events by itself throughout the tracking history. Algorithm 3 shows the example pseudo code of the Ratio-based Merge algorithm.

3.3.2 Hardware Implementaion. Besides software-based Merge, we also implement a hardware-based Merge with the Ratio-based Merge decision algorithm. As Merge can happen on any

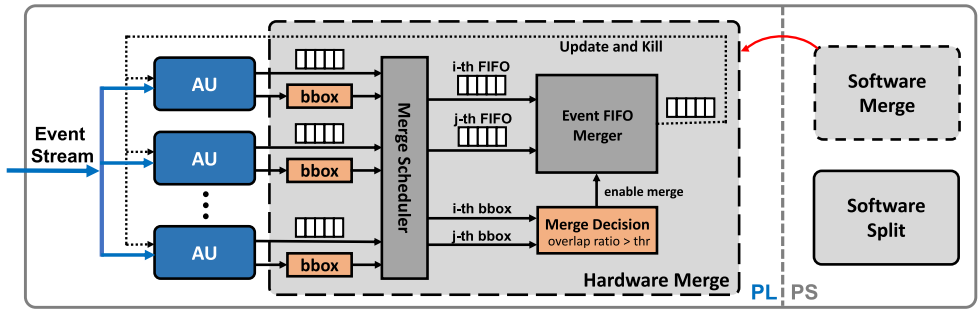


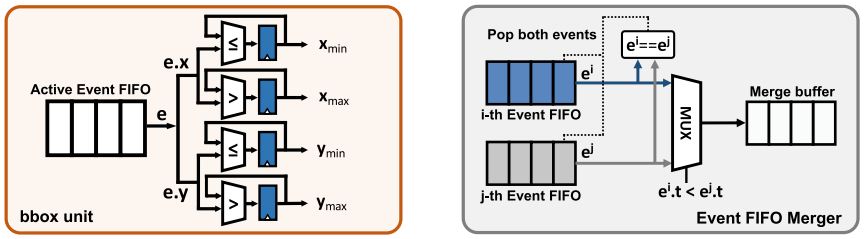
Fig. 7. Overall system diagram with hardware implementation of Merge operation. The hardware Merge unit is mainly composed of a Merge Scheduler, a Merge Decision Unit, and an Event FIFO Merger. The Merge Scheduler iterates through all possible combinations of AUs. In each iteration, the Merge Scheduler selects two AUs (marked as ith, jth) for Merge Decision Unit to decide whether to be merged, also used in Algorithm 3. If the Merge condition is satisfied, then the captured events will be merged by the Event FIFO Merger.

combination of two AUs (as shown in Algorithm 3), the complexity is $O(n^2)$, where n is the total number of AUs. When the number of AU increases, the time spent on the Merge operation grows quadratic and can possibly be the bottleneck running on PS. In that case, a hardware-based Merge unit can help to eliminate this bottleneck. In addition, implementing a hardware Merge unit not only parallelizes some of the computation but also saves the frequent IO between PS and PL to synchronize the AU status during Merge. Figure 7 shows the overall system diagram that incorporates a hardware Merge unit on top of the Fifo-only implementation. By using the Fifo-only implementation, we only need to aggregate the Active Event FIFOs of two AUs. The hardware Merge unit is mainly composed of three different parts: (1) Merge Scheduler, (2) Merge Decision Unit, and (3) Event FIFO Merger.

Merge Scheduler. The Merge Scheduler will schedule all the possible combinations of active AUs one by one to the Merge decision unit to decide whether to perform a Merge. For example, if all the AUs are active, then there will be in total $C(n, 2) = n(n-1)/2$ possible combinations. After a valid Merge is carried out, the aggregated events will be placed in one AU while the other will be deleted.

Merge Decision Unit. We implement the Ratio-based Merge algorithm on hardware as mentioned above. This decision algorithm uses the bounding boxes of two AUs to calculate their IoM ratio. If the IoM is larger than a given threshold, then two AUs will be merged. Computing the bounding box of an AU is realized as finding the minimum and maximum locations of the x- and y-axes of all the events in an Active Event FIFO. Figure 8(a) shows the hardware implementation of the bounding box unit. It iterates all the events one by one and compares each x, y value with current min/max values in the registers. Each AU is equipped with one bounding box unit, and they can operate in parallel. When the Merge Scheduler selects ith and jth AUs in a round, the Merge Decision Unit will compute the IoM value using Equation (9). If the IoM is larger than the threshold, then these two AUs will be merged by the Event FIFO Merger.

Event FIFO Merger. If the decision unit decides to merge the ith and jth AUs, then the Event FIFO Merger will merge the corresponding Active Event FIFOs. Since all the events are sorted in the time dimension in the Active Event FIFO, the Event FIFO Merger can compare the timestamps of two events from 2 AU one by one and output the one with the smaller timestamp first to maintain the sorted pattern. This procedure is similar to the merge phase of merge-sort algorithms on the time



(a) (b)

Fig. 8. (a) Hardware bounding box unit. (b) Event FIFO Merger that merges two Active Event FIFOs.

ALGORITHM 4: Split Algorithm

```
for i = 0 : Number of AUs do
    clusters = DBSCAN(AUs[i].efifo)
    if Number of clusters ≥ 2 then
        for all event_cluster in clusters do
            amap = [W, H]
            for all e in event_cluster do
                Expand(amap, e)
            end for
            au = new AU(event_cluster, amap)
            AUs.append(au)
        end for
        Delete AUs[i]
    end if
end for
```

dimension. Similarly to the software implementation, the Event FIFO Merger should also need to reduce the duplicated events from two AUs in the pipeline. If the events from the two AU FIFO are the same, then both events will be popped out from the FIFO and only one copy of it will be written out. Figure 8(b) shows the diagram of the hardware Event FIFO Merger.

3.3.3 Applicability of Different Merge Implementations. In its basic form, the hardware AUs on the PL process raw events from the sensors, and the CPU on the PS side carry out high-level Merge/Split decisions to manipulate the hardware AUs status. In some demanding cases where having a large number of AUs, the hardware implementation of the Merge algorithm can offload the computation to PL and further improve the system performance. Results show that the hardware Merge unit can achieve 95× faster than the software version in worst-case latency with 11 AUs configuration. More results will be discussed in Section 4.4.2.

3.4 Implementation of Split Algorithms

In REMOT, we employ two Split algorithms to decide whether to partition an AU: a density-based Split algorithm and a hierarchy-based Split algorithm. Each attention region of the separated AU is reconstructed using the new clustered events. In the current design, we only have the software-based implementation of the Split algorithms.

Density-based Spilt Algorithm. It uses the **density-based spatial clustering of applications with noise (DBSCAN)** algorithm [14]. DBSCAN describes the spatial density of a location by the number of points in a given neighborhood radius. The points in high-density regions are clustered

together. If the active events of one AU have at least two clusters, then the AU will be split, shown in Algorithm 4.

Hierarchy-based Spilt Algorithm. This method utilizes the **Hierarchical Agglomerative Clustering (HAC)** algorithm [31], which builds a hierarchy of clusters to split events. HAC aggregates events from clusters, starting from one event per cluster. Two nearest clusters are merged into one at each iteration, until all the events are amalgamated into one cluster, forming a hierarchy of clusters. If the last two clusters are too far apart to merge, then the events will be split.

3.5 Dynamic Power Saving

Energy efficiency is one of the unique advantages of DVS as it only produces events when there are dynamic activities in the scene. In real-world applications, the system may want to save its energy by running in power saving mode when there is no outstanding activity happening for a period of time. For example, in a traffic monitoring system, there can be few cars late at night. A traditional frame-based system still needs to process frame by frame regardless of whether the traffic is heavy or not. However, in an event-based system, both sensing and processing stages can conserve a lot of energy during the low-activities period thanks to the working principle of event cameras. Thus, in this work, we propose two different mechanisms that allow the system to dynamically adjust the number of active AUs to achieve a better tradeoff between energy efficiency and tracking abilities.

3.5.1 Adjust by Inactivating AUs. One of the strategies is to dynamically inactivate AUs without changing the configuration of PL. This is done by applying a mask on the hardware AUs that blocks some of the AUs from processing new events. When an AU is flagged as inactive by the mask, it will no longer read new events and carry out Expand/Shrink operations anymore. This can save a certain amount of dynamic power consumption. Figure 9(a) illustrates the procedures of this method. The controller first reads the current status of all the AUs and chooses some idle AUs to inactivate. Then it deletes the chosen AUs in the buffer and sends a mask to PL. Finally, the hardware AUs that are flagged as inactive stop processing new events.

The advantage of this approach is that the time spent on inactivating/activating hardware AUs can be very short, because the PS only needs to send a mask signal through the control interface. However, it cannot achieve 100 % of power saving, since the inactivated hardware AUs can still consume a certain amount of static power.

3.5.2 Adjust by Reconfiguring PL. At the other extreme, we also propose another method that can fully save the power on hardware AU by reconfiguring the PL with fewer AUs. This is at a cost of spending a longer time in switching the configuration of hardware. Figure 9(b) shows the procedures of this method. This requires users to synthesize all the bitstreams with different numbers of hardware AUs beforehand. During runtime, the controller running on the PS first stashes all the status and information of hardware AUs in the memory, e.g., Active Event FIFOs and AMAPs. Then it reprograms the PL by downloading the bitstream corresponding to the targeted number of hardware AU instances. Finally, all the buffered data are written back to PL, and the system continues to run. In this way, more power can be saved on the PL side, but it also takes longer to reconfigure. These two methods together lead to a tradeoff between power efficiency and configuration overhead.

3.5.3 Applicability of Different Power Saving Strategies. The two dynamic power saving strategies of hardware AUs together form a tradeoff between configuring latency and power saving rate. In general, Inactivating AUs can respond faster while reconfiguring PL saving more power per AU. More discussion on the results and tradeoff will be in Section 4.5. In real-world applications, the power saving decision should cooperate with the other algorithms that can predict the maximum

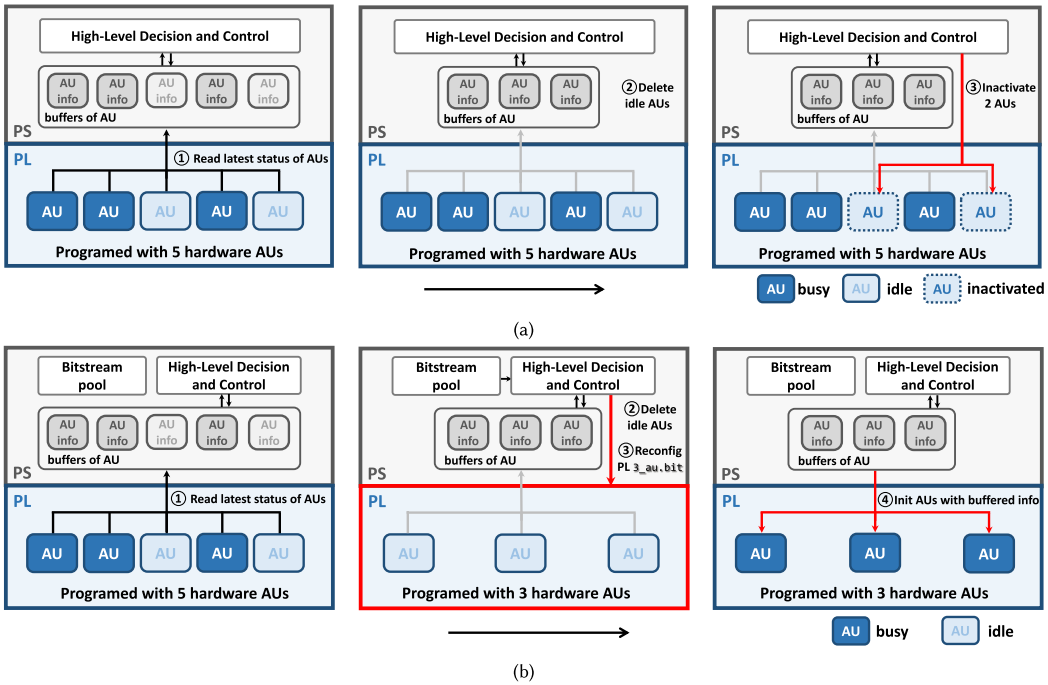


Fig. 9. (a) Power saving by deactivating idle AUs. (b) Power saving by reconfiguring PL with fewer AUs.

number of objects throughout different time periods and decide the frequency/strategy to perform power saving.

4 RESULTS

4.1 Datasets and Evaluation Metrics

We evaluated the proposed algorithms on three event camera datasets. The first one is a 10-s segment from the open source data *shapes_6dof* [30], and the other two datasets, *inbound traffic*, and *outbound traffic*, are captured by ourselves that shows the inbound and outbound traffic respectively. The data were captured using a DAVIS 346 camera [7] that produced simultaneous events and image frames. The ground-truth bounding boxes for object tracking were manually labeled using frame-based images.

To evaluate tracking performance, we use HOTA [28], which is the default metric for multi-object tracking in many frame-based object tracking benchmarks including MOTChallenge MOT20 [12] and KITTI MOTS [18]. HOTA is a unified metric that evaluates both detection and association accuracy of an algorithm as follows.

Detection accuracy measures the alignment between the predicted bounding boxes and the ground-truth bounding boxes:

$$\text{DetA} = \int_{0<\alpha\leq 1} \text{DetA}_\alpha = \int_{0<\alpha\leq 1} \frac{|\text{TP}_\alpha|}{|\text{TP}_\alpha| + |\text{FP}_\alpha| + |\text{FN}_\alpha|}, \tag{10}$$

where $|\text{TP}_\alpha|$, $|\text{FP}_\alpha|$, and $|\text{FN}_\alpha|$ refer to the numbers of true positives, false positives, and false negatives, with **Intersection over Union (IoU)** between predicted and ground-truth bounding boxes larger than the minimum match threshold α.

Association accuracy measures the alignment between the predicted track and the ground-truth track:

$$\text{AssA} = \int_{0<\alpha\leq 1} \text{AssA}_\alpha = \frac{1}{|\text{TP}_\alpha|} \sum_{c\in\text{TP}} \frac{|\text{TPA}_\alpha^c|}{|\text{TPA}_\alpha^c| + |\text{FPA}_\alpha^c| + |\text{FNA}_\alpha^c|}, \tag{11}$$

where c is a given true positive, α is minimum IoU, and $|\text{TPA}_\alpha^c|$, $|\text{FNA}_\alpha^c|$, and $|\text{FPA}_\alpha^c|$ correspond to the size of true-positive association, false-negative association, and false-positive association.

HOTA unites detection accuracy and association accuracy:

$$\text{HOTA} = \int_{0<\alpha\leq 1} \text{HOTA}_\alpha = \int_{0<\alpha\leq 1} \sqrt{\text{DetA}_\alpha \cdot \text{AssA}_\alpha}. \tag{12}$$

4.2 Hardware Implementation Results

4.2.1 Hardware Experiment Setting. In this section, we present the results of the hardware implementation of REMOT. A series of HW/SW experiments were carried out to demonstrate the multiple design tradeoffs in accuracy, throughput, resources, and power. To demonstrate the flexibility and scalability of REMOT, we implemented multiple REMOT configurations on two embedded FPGA platforms: PYNQ-Z2 (Zynq 7Z020) and Ultra96 (Zynq UltraScale+ MPSoC ZU3EG). In addition, a software-only baseline of the proposed algorithm was implemented on the processor of Ultra96 (Arm CORTEX-A53) for performance comparison.

The power consumption was measured using a power source connected to the development board, which reflects the total power consumption for the system. Specifically, we set the voltage of the power source and observed the stable current value of the power source in a continuous input test. In Section 4.5, we also measured the fine-grained power consumption on the PL side only through the PMBus rails on Ultra96.

4.2.2 Performance Comparison between FPGA and CPU. Figure 10(b) shows a general picture of the different performance models of CPU and FPGA for low-level event processing (Expand and Shrink). Both FPGA and CPU results were measured on Ultra96. The Fifo-only implementation was used for FPGA results, and the throughput was measured on the development board after synthesis, place, and route with different numbers of AUs deployed. According to Figure 10(b), the parallel hardware AUs on FPGA achieve a comparable performance across different AU numbers, while the processing throughput of the CPU decreases dramatically as the number of AU increases. Even though the throughput of FPGA does drop slightly owing to lower PL clock frequencies as the resource utilization increases for more AUs, the speed-up continues to grow and achieves up to 44×. The results demonstrate our hardware architecture in REMOT can lead to a scalable performance in low-level parallel event processing. In terms of power, the CPU implementation has a relatively static power consumption of 4.64W while the FPGA implementation with 13 AUs runs at 5.45W, resulting in 35.4× improvement in terms of power efficiency (Meps/W).

At the same time, more AUs means more capabilities to potentially track objects at the same time. Figure 10(a) shows the tracking results on different datasets versus the maximum AU allowed. It points out the fact that if the CPU-only implementation wants to achieve higher tracking accuracy by using more AUs, then it cannot meet the real-time throughput requirement (>0.3 Meps) for event processing.

Generally, for a given dataset, the accuracy will saturate at some points depending on how many objects would appear at the same time. In our case, 10 AUs would be sufficient for all three datasets. However, this result might not genuinely reflect the situation for other scenarios, e.g., a heavier traffic scene. The purpose of Figure 10(a) is to illustrate the important accuracy-resource tradeoff affected by the maximum number of AUs allowed. If more AU can be deployed under

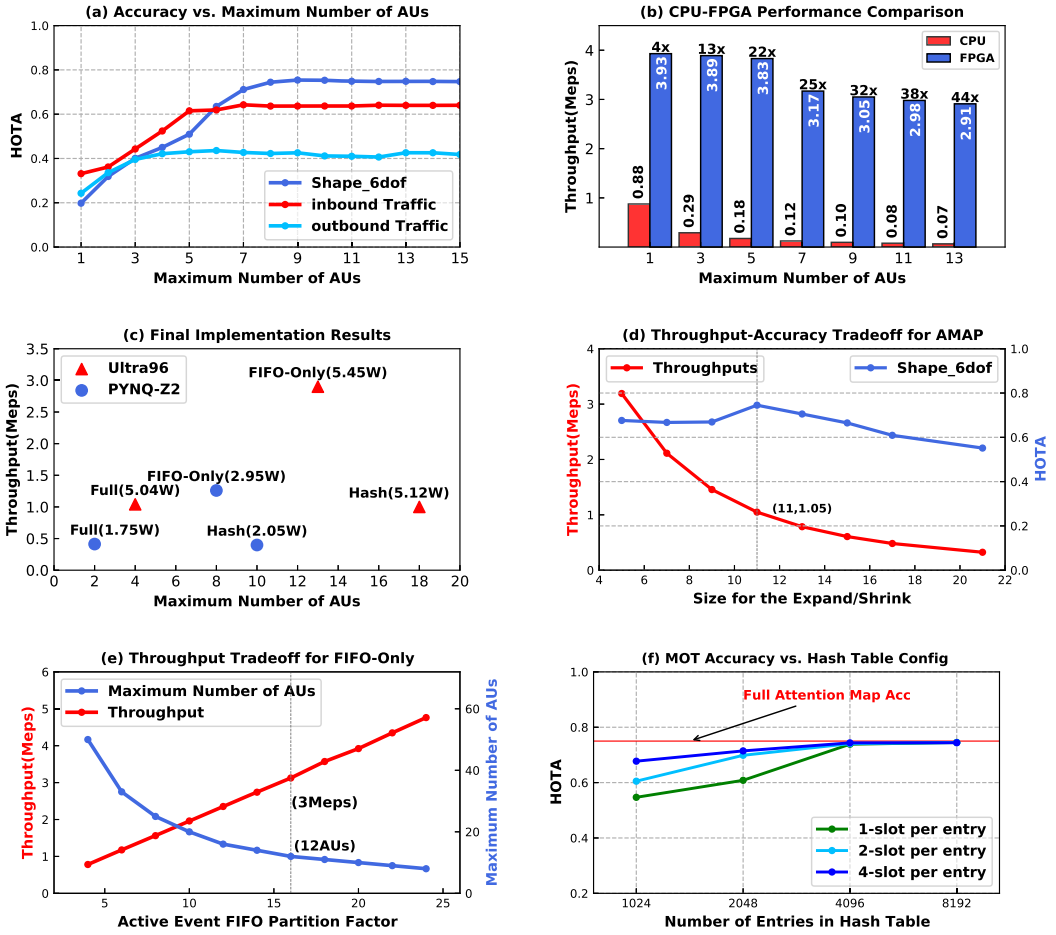


Fig. 10. Hardware Implementation. Panel (a) shows MOT accuracy with a different maximum number of AUs on three datasets; panel (b) compares performances between CPU and FPGA; panel (c) presents final performances for the three implementations on Ultra96 and PYNQ-Z2; panel (d) captures throughput-accuracy tradeoff for Hash-amap and Full-amap implementations; panel (e) shows throughput-resource tradeoff for Fifo-only; and (f) illustrates accuracy degradation for different hash-table configurations.

given resource constraints, then a higher multi-object tracking accuracy can be expected to some extent.

4.2.3 Tradeoffs in Different Hardware Implementations of AU. In this section, we present some detailed discussions on implementation results with different design tradeoffs.

Tradeoffs in Throughput. For the attention-map-based implementations (Full-amap and Hash-amap), throughput is bounded by sequentially updating the attention region. Figure 10(d) demonstrates this accuracy–throughput tradeoff with different Expand and Shrink size d shown in Figure 4. The results were obtained based on the *shapes_6dof* dataset, while the throughput is calculated using Equation (2) assuming a 300-MHz clock frequency. As shown in Figure 10(d), the optimal value of d is 11 for the *shapes_6dof* dataset, leading to 1.04 Meps throughput.

The Fifo-only implementation has a different throughput tradeoff compared with attention-map-based design, which is determined by how fast it can traverse through the entire FIFO as

Table 2. Resource Utilization When Maximum Number of Hardware AUs Are Instantiated

	DSP	LUTs	BRAM	FF	Freq (MHz)
PYNQ-Z2	220	53200	280	106400	
Full-amap	1%	20%	77%	14%	125
Hash-amap	5%	28%	80%	20%	125
Fifo-only	0%	20%	96%	13%	125
Ultra96	360	70560	432	141120	
Full-amap	1%	14%	96%	9%	300
Hash-amap	5%	26%	87%	5%	300
Fifo-only	0%	22%	99%	14%	250

described in Equation (7). The throughput grows linearly with the FIFO partition factor while the BRAM usage for the FIFO also increases. Figure 10(e) shows the theoretical throughput and the maximum number of AUs that can be deployed on Ultra96 with respect to different FIFO partition factors. The depth of the Active Event FIFO is set to 1,024 in the experiments. As marked by the dashed line in Figure 10(e), if the partition factor is 16, then we can embed around 12 AUs with 3 Meps throughput, which is close to the final implementation result shown in Figure 10(c).

Tradeoff in Hash-amap Design. Another accuracy–resource tradeoff exists in Hash-amap implementation. As discussed in Section 3, the Hash-amap leverages the intrinsic sparsity in attention-map to save on-chip memory consumption. However, it also brings in a new problem of hash collision that can potentially flaw the attention map. Figure 10(f) shows the accuracy degradation for different hash table configurations on *shapes_6dof*. The accuracy would hardly drop when the number of entries exceeds 4,096 compared to a full attention map. The results also show that using both more slots and hash table entries can benefit the accuracy. However, since the total hash table size is the number of entries times the number of slots per entry, increasing the entries number seems to bring more marginal benefits as shown in Figure 10(f). However, this is an empirical conclusion that only reflects the overall effects of our hashing function, data accessing pattern, and collision handling strategy. In the final implementations with a conservative configuration (8,192 $\times$ 1), the Hash-amap enables around 4$\times$ more AUs to be deployed compared to the Full-amap as shown in Figure 10(c). In this way, more potential objects can be tracked with more hardware AUs deployed.

4.2.4 Scalability and Power Consumption. In the final deployment, we devise three different implementations on both PYNQ-Z2 and Ultra96 and measure their throughput and power consumption. The internal AU configurations are identical for Ultra96 and PYNQ-Z2, leaving the available hardware resources to determine the maximum number of AUs. The final hardware AU quantities as well as the corresponding throughput and power consumption are shown in Figure 10(c). Table 2 lists the detailed resource utilization of different implementations. In general, our design shows high scalability. The Ultra96 development board has 54% more on-chip BRAM compared to PYNQ-Z2, resulting in around 60% increase in the maximum number of AUs deployed. The throughput on Ultra96 is also higher than the corresponding version on PYNQ-Z2 for a higher clock frequency after place and route.

4.3 MOT Performance

Table 3 summarizes the tracking accuracy of REMOT as measured by the HOTA metrics while Figure 11 shows the visualization of tracking results. Specifically, the results were produced under

Table 3. Tracking Results and Data Rates on Different Datasets

	shapes_6dof	inbound traffic	outbound traffic
Detection Accuracy (%)	70.4	50.9	39.0
Association Accuracy (%)	76.3	58.0	47.8
HOTA (%)	73.2	54.3	43.1
Average Event Rate (Meps)	0.30	0.26	0.22
Peak Event Rate (Meps)	2.16	1.03	0.84

(a) *shapes_6dof* (b) *inbound traffic* (c) *outbound traffic*

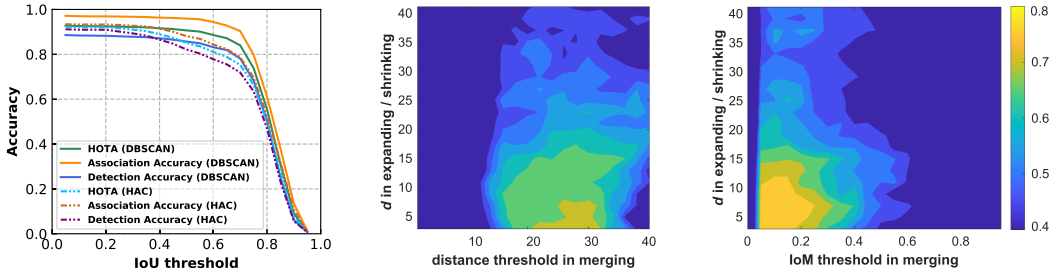
Fig. 11. Visualization of tracking results. The left-hand side of each figure shows the events accumulated from the 40 ms prior to the corresponding image frame and tracking results from REMOT. The right-hand side shows the corresponding image frame with the ground-truth bounding boxes.

the distance-based Merge algorithm and DBSCAN Split algorithm. Overall, REMOT performed best with the relatively simple *shapes_6dof* benchmark followed by the more complex real-world benchmark of *inbound traffic* and *outbound traffic*. Real-world challenges such as the presence of shadows, which the AUs regard as part of a vehicle but the human-produced ground-truth labels did not, caused mismatched bounding box calculations. Furthermore, vehicles movements, such as when individual cars begin to merge in outbound traffic or when cars emerged from afar in inbound traffic, challenge our current simplistic AU Merge, Split, and Expand/Shrink actions.

We further compared the performance of REMOT against three related works that addressed similar event-based MOT challenges and have reported results using *shapes_6dof*, as shown in Table 4. E-MS [5] tracks objects by cluster events, while ETD [9] and RMRNet [8] accumulate events and reconstruct 2D representations that allow other frame-based algorithms to be applied. Typically, RMRNet uses a Convolution Neural Network and LSTM to perform end-to-end object motion regression, and both ETD and RMRNet are implemented on GPU. Without access to the source code of ETD and RMRNet, we instead evaluated REMOT using the specialized metrics employed in these two papers, namely, **average precision (AP)**, (also known as average overlap rate in Reference [8]) and **average robustness (AR)** with our segment from *shapes_6dof*. The HOTA value for E-MS was produced by evaluating their released source code using our segment from *shapes_6dof*. Minimum enclosing bounding boxes were created based on the segmented clusters

Table 4. Comparison Results with Other Methods on the *shapes_6dof* Dataset

	Metrics	AP (%)	AR (%)	HOTA (%)
[5]	E-MS	61.2	66.8	40.2
[9]	ETD	80.9	99.8	N.A.
[8]	RMRNet	86.6	98.0	N.A.
	REMOT	76.5	94.3	73.2



(a) (b) (c)

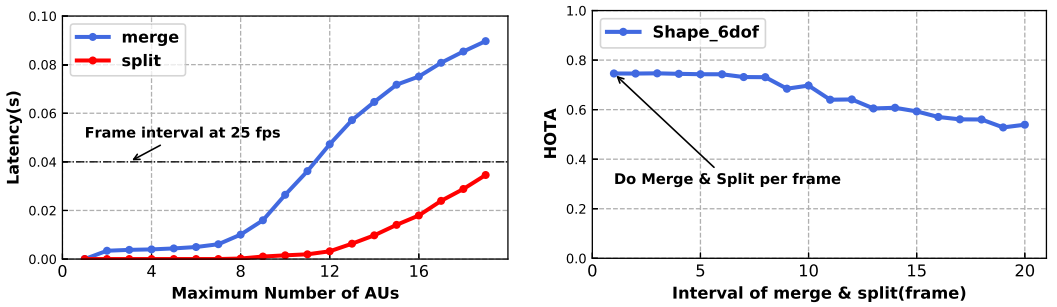
Fig. 12. MOT results on different Merge/Split algorithms. (a) Decomposed accuracies using two Split algorithms of DBSCAN and HAC, with the changes of minimum IoU between tracked and ground-truth bounding boxes. ((b) and (c)) Accuracy varies as the changes of Expand/Shrink parameter (vertical) and Merge parameter (horizontal). Both y-axes are the size d of Expand and Shrink. The x-axis of (a) is the maximum distance of Merge, and x-axis of (b) is the minimum IoM ratio of Merge.

of events per frame and were labeled with the corresponding segment color for association. The AR and AP values for E-MS were reproduced from Reference [8]. Results show that REMOT performs better than E-MS across all three metrics but is shy of achieving the same performance as ETD and RMRNet in the AP and AR metrics. Nevertheless, REMOT was able to achieve such accuracy in real time while consuming only a fraction of power (≤5.5 W) using low-end FPGA-based HW/SW systems. Both ETD and RMRNet use NVIDIA GTX 1080 GPU and require 21.97 and 38.75 ms to track one object per frame.

4.4 Flexibility of the High-level Merge/Split Algorithms

As described in Section 3, REMOT decouples the low-level event processing from the high-level vision decision in a hierarchical way. The high-level Merge/Split algorithms can be highly flexible and modular in two aspects. On the one hand, Merge/Split can be highly modular by assembling and combining with different decision algorithms. On the other hand, our reconfigurable architecture allows some of the algorithms like Merge to be implemented either on PS or PL to achieve different performance tradeoffs.

4.4.1 Performance of Merge/Split on PS. In its basic form, when implementing the Merge/Split algorithms on the processing system, they can be highly modular. For example, Figure 12(a) shows the decomposed accuracies of *shapes_6dof* dataset under two different Split algorithms. The solid lines are the accuracies using the DBSCAN Split algorithm, and the dash lines correspond to the HAC Split algorithm, both combined with the Distance-based Merge algorithm. Results show that the software under different configurations can exhibit different performances in detection and association.



(a) (b)

Fig. 13. Latencies of Merge and Split get longer as the maximum number of AUs increases (a) Accuracy drops slightly as the time interval of Merge and Split operations is prolonged (b).

Figure 12(b) and (c) also demonstrates the performance of the two Merge algorithms, Distance based and ratio based, assuming the same DBSCAN Split algorithm. The colors of Figure 12(b) and (c) indicate the HOTA value. The y-axes of the figures are the Expand and Shrink size d. The x-axis of Figure 12(b) is the maximum distance to merge neighboring AUs, and the x-axis of the Figure 12(c) is the minimum IoM threshold to merge AUs. Since the size d is a hardened hardware parameter, Figure 12 brings in additional variability on the software side to explore a better overall tracking performance.

When implementing the high-level vision algorithms on the PS, it can process at a lower rate than conventional frame-based vision algorithms. Normally, frame-based tracking algorithm would process each frame one after another. The real-time requirement for these algorithms is that the processing frame rate should be higher than the frame rate of the camera, e.g., 25 fps. However, in REMOT, since the low-level hardware AU is continuously tracking the dynamic scene in real time, the high-level Merge/Split software does not necessarily need to operate at a fixed frame rate similar to a frame-based camera. Figure 13(a) shows the average latencies of Merge and Split operations, measured on Ultra96 using the *shapes_6dof* dataset. As the maximum number of AUs grows, it takes longer to perform a Merge/Split operation for all AUs. If using 25 fps as requirement, then the maximum number of AUs cannot exceed 11. However, Figure 13(b) shows that the tracking accuracy would hardly dropped even when the Merge/Split happens every 6 frames (240 ms). This brings much room for tolerance to a low-end processor on edge platforms.

4.4.2 Performance of Hardware Merge. As described above, the system has a certain degree of tolerance in latency of Merge and Split depending on the actual application scenario. However, since the complexity of software Merge is $O(n^2)$, it can still potentially be slow, especially with a large number of AUs. Fast Merge operations are needed to ensure AUs can be merged efficiently in high-activity scenarios. For that, we explore the benefit of accelerating Merge operation with hardware.

Table 5 summarizes the hardware resource consumption of the hardware Merge unit. It is based on the Fifo-only implementation on Ultra96 and has an identical implementation configuration used in Section 4.2. Since hardware-based Merge requires additional hardware resources, the maximum number of AU that can be deployed on Ultra96 decreases from 13 to 11. Compared to the 11-AUs baseline, there is 41%, 4%, and 67% additional resource consumption on LUTs, BRAM, and FF introduced by the Merge unit.

To evaluate the performances of the hardware Merge unit, we compared the *worst-case* latency between the software and hardware Merge. This is under the consideration that at the application

Table 5. Performance of Hardware Merge Using Fifo-only Implementation on Ultra96

	Number of AUs	DSP	LUTs	BRAM	FF	Event Processing Thoughput	Worst Case Latency of Merge	PL Power
w.o. HW Merge	11	0	14,374	364	17,886	2.98 Meps	28.5 ms	1.28 W
w. HW Merge	11	0	20,363	378	29,799	2.90 Meps	0.3 ms	1.41 W

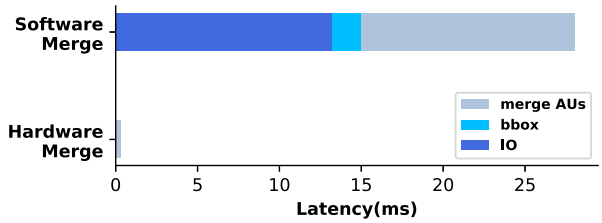


Fig. 14. Breakdown latency of software/hardware Merge.

level, e.g., running a *shapes_6dof* dataset, the number of valid Merge and the number of events in the Active Event FIFOs can be completely random. As a result, it is difficult to quantitatively analyze the performances. Thus, to ensure a controllable comparison, we used the worst-case scenario with artificial data for both hardware and software Merge and measured their execution time. The worst case corresponds to the situation when all the AUs need to merge together and all the Active Event FIFOs are completely occupied. The hardware execution time is then measured by writing artificial data to hardware Active Event FIFOs and directly performing Merge without any event input. In general, the hardware Merge unit can achieve 95× faster compared to the software-based Merge as shown in Table 5.

Figure 14 shows the breakdown latency of Merge in the worst-case scenario. Since software Merge requires reading and writing hardware Active Event FIFO before and after Merge, it also introduces a significant portion of time on IO between PS and PL. The IO portion in Figure 14 also includes the time to encode/decode data during PS-PL communication. When the Merge operation is moved to hardware, these IO can be saved. The hardware Merge unit can also accelerate decisions by using distributed bounding box units and also leverage a dedicated pipeline to perform FIFO merging. In terms of dynamic power consumption, the hardware Merge unit only results in 10% additional power on PL.

4.5 Tradeoffs in Power Saving Strategies

To improve the power efficiency of the system when there are few dynamic activities in the scene, we provide two additional power-saving methods on the hardware by inactivating AUs or reconfiguring PL. Figure 15 shows power consumption versus the number of active AUs using these two methods. The PL power reflects peak power consumption measured by PMBus rails. Results show that reconfiguring the PL with different numbers of AUs can achieve the best power efficiency with an estimated 77-mW power reduction per AU. At the same time, it can also benefit from the higher throughput processing rate with fewer maximum number of AUs on PL as shown in Figure 10(b). However, this is at a cost of a longer adjustment time to reconfigure the entire PL, which takes around 640 ms as shown in Table 6.

However, inactivating AUs can provide timely responses to the adjustment request within around 1 ms without reconfiguring the PL while there is some additional power consumption on the inactivated hardware AU instances. Results show that an estimated of 77-mW power can be saved by inactivating one AU. Besides, the event processing throughput of this method is also fixed at 2.82 Meps.

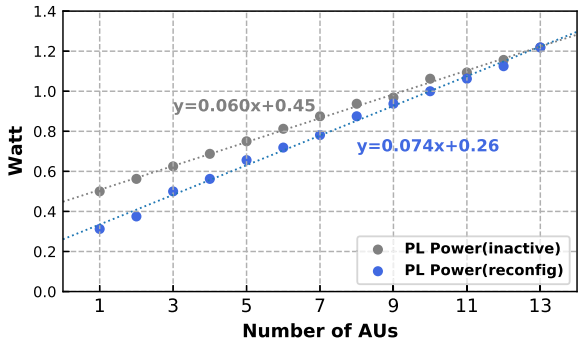


Fig. 15. Dynamic power vs. number of active AUs on two different power saving strategies. The two lines are fitted using linear regression.

Table 6. Comparing Two Power-saving Strategies

	Latency of adjusting AUs	Estimated power saved per AU	Event Processing Throughput
Inactivate	1.1 ms	59 mW	2.82 Meps
Reconfigure	640 ms	77 mW	2.91–3.93 Meps

5 LIMITATIONS AND FUTURE WORK

While our current design of REMOT works well with our target dataset, it has several limitations we plan to address in the future. First, the current REMOT algorithm can only track objects against a clean background that does not produce excessive events. Also, our MOT algorithms currently cannot handle occlusion well when the tracks of two objects cross. Finally, our proposed algorithms include many heuristic parameters that are sensitive to specific scenarios. For instance, the Merge/Split decision thresholds, the Expand/Shrink radius, and so on. In the future, we intend to develop learning-based algorithms that take advantages of the partial information collected by each AU to guide their Merge and Split operations so they can be aware of occlusions and adapt to changing scenarios.

6 CONCLUSIONS

In this article, we have presented REMOT, a reconfigurable hardware-software architecture and a family of multi-object tracking algorithms that run on this system. By partitioning the MOT task to operate in both hardware and software, we demonstrated that real-time performance can be achieved even on modest edge FPGA platforms when tested on real-world DVS datasets. The modular design of REMOT allows different parts of the algorithm to be implemented in either hardware or software for the tradeoff among power, performance, and accuracy. The superior power efficiency of REMOT has been demonstrated with multiple implementations on FPGAs, allowing REMOT to be deployed in real-world edge scenarios with stringent power and performance constraints.

REFERENCES

[1] Jyotibdha Acharya, Andres Ussa Caycedo, Vandana Reddy Padala, Rishi Raj Singh Sidhu, Garrick Orchard, Bharath Ramesh, and Arindam Basu. 2019. EBBIOT: A low-complexity tracking algorithm for surveillance in IoVT using stationary neuromorphic vision sensors. In *Proceedings of the 32nd IEEE International System-on-Chip Conference (SOCC'19)*. 318–323.

[2] Alessandro Aimar, Hesham Mostafa, Enrico Calabrese, Antonio Rios-Navarro, Ricardo Tapiador-Morales, Iulia-Alexandra Lungu, Moritz B. Milde, Federico Corradi, Alejandro Linares-Barranco, Shih-Chii Liu, and Tobi Delbruck. 2019. NullHop: A flexible convolutional neural network accelerator based on sparse representations of feature maps. *IEEE Trans. Neural Netw. Learn. Syst.* 30, 3 (2019), 644–656.

[3] Boris Babenko, Ming-Hsuan Yang, and Serge Belongie. 2009. Visual tracking with online multiple instance learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 983–990.

[4] Boris Babenko, Ming-Hsuan Yang, and Serge Belongie. 2010. Robust object tracking with online multiple instance learning. *IEEE Trans. Pattern Anal. Mach. Intell.* 33, 8 (2010), 1619–1632.

[5] Francisco Barranco, Cornelia Fermuller, and Eduardo Ros. 2018. Real-time clustering and multi-target tracking using event-based sensors. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'18)*, 5764–5769.

[6] Erik Bochinski, Volker Eiselein, and Thomas Sikora. 2017. High-speed tracking-by-detection without using image information. In *Proceedings of the 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS'17)*, 1–6.

[7] Christian Brandli, Raphael Berner, Minhao Yang, Shih-Chii Liu, and Tobi Delbruck. 2014. A 240× 180 130 db 3 μs latency global shutter spatiotemporal vision sensor. *IEEE J. Solid-State Circ.* 49, 10 (2014), 2333–2341.

[8] Haosheng Chen, David Suter, Qiangqiang Wu, and Hanzi Wang. 2020. End-to-end learning of object motion estimation from retinal events for event-based object tracking. *Proc. AAAI Conf. Artif. Intell.* 34, 07 (2020), 10534–10541.

[9] Haosheng Chen, Qiangqiang Wu, Yanjie Liang, Xinbo Gao, and Hanzi Wang. 2019. Asynchronous tracking-by-detection on adaptive time surfaces for event-based object tracking. In *Proceedings of the 27th ACM International Conference on Multimedia*, 473–481.

[10] Gregory K. Cohen, Garrick Orchard, Sio-Hoi Leng, Jonathan Tapson, Ryad B. Benosman, and André Van Schaik. 2016. Skimming digits: Neuromorphic classification of spike-encoded images. *Front. Neurosci.* 10 (2016), 184.

[11] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. 2018. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 1 (2018), 82–99.

[12] P. Dendorfer, H. Rezatofighi, A. Milan, J. Shi, D. Cremers, I. Reid, S. Roth, K. Schindler, and L. Leal-Taixé. 2020. MOT20: A benchmark for multi object tracking in crowded scenes. arXiv: 2003.09003. Retrieved from http://arxiv.org/abs/1906.04567.

[13] Martin Dietzfelbinger, Torben Hagerup, Jyrki Katajainen, and Martti Penttonen. 1997. A reliable randomized algorithm for the closest-pair problem. *J. Algor.* 25, 1 (1997), 19–51.

[14] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD'96)*, 226–231.

[15] Guillermo Gallego, Tobi Delbruck, Garrick Michael Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew Davison, Jorg Conradt, Kostas Daniilidis, and Davide Scaramuzza. 2020. Event-based vision: A survey. *IEEE Trans. Pattern Anal. Mach. Intell.* (2020), 1–26.

[16] Yizhao Gao, Song Wang, and Hayden Kwok-Hay So. 2022. REMOT: A hardware-software architecture for attention-guided multi-object tracking with dynamic vision sensors on FPGAs. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'22)*. Association for Computing Machinery, New York, NY, 158–168. https://doi.org/10.1145/3490422.3502365

[17] Daniel Gehrig, Henri Rebecq, Guillermo Gallego, and Davide Scaramuzza. 2020. EKLT: Asynchronous photometric feature tracking using events and frames. *Int. J. Comput. Vis.* 128, 3 (2020), 601–618.

[18] Andreas Geiger, Philip Lenz, and Raquel Urtasun. 2012. Are we ready for autonomous driving? The kitti vision benchmark suite. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 3354–3361.

[19] Wulfram Gerstner and Werner M. Kistler. 2002. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press.

[20] D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge. 1993. Comparing images using the Hausdorff distance. *IEEE Trans. Pattern Anal. Mach. Intell.* 15, 9 (1993), 850–863. https://doi.org/10.1109/34.232073

[21] Rui Jiang, Xiaozheng Mou, Shunshun Shi, Yueyin Zhou, Qinyi Wang, Meng Dong, and Shoushun Chen. 2020. Object tracking on event cameras with offline–online learning. *CAAI Trans. Intell. Technol.* 5, 3 (2020), 165–171.

[22] Xavier Lagorce, Garrick Orchard, Francesco Galluppi, Bertram E. Shi, and Ryad B. Benosman. 2017. HOTS: A hierarchy of event-based time-surfaces for pattern recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* 39, 7 (2017), 1346–1359.

[23] Hongmin Li and Luping Shi. 2019. Robust event-based object tracking combining correlation filter and CNN representation representation. *Front. Neurorobot.* 13 (2019), 82.

[24] A. Linares-Barranco, F. Gómez-Rodríguez, V. Villanueva, L. Longinotti, and T. Delbrück. 2015. A USB3.0 FPGA event-based filtering and tracking framework for dynamic vision sensors. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'15)*, 2417–2420.

[25] Alejandro Linares-Barranco, Fernando Perez-Peña, Diederik Paul Moeys, Francisco Gomez-Rodriguez, Gabriel Jimenez-Moreno, Shih-Chii Liu, and Tobi Delbruck. 2019. Low latency event-based filtering and feature extraction for dynamic vision sensors in real-time FPGA applications. *IEEE Access* 7 (2019), 134926–134942.

[26] Alejandro Linares-Barranco, Antonio Rios-Navarro, Salvador Canas-Moreno, Enrique Piñero-Fuentes, Ricardo Tapiador-Morales, and Tobi Delbruck. 2021. Dynamic vision sensor integration on FPGA-based CNN accelerators for high-speed visual classification. In *Proceedings of the International Conference on Neuromorphic Systems*, 1–7.

[27] Qianhui Liu, Haibo Ruan, Dong Xing, Huajin Tang, and Gang Pan. 2020. Effective AER object classification using segmented probability-maximization learning in spiking neural networks. *Proc. AAAI Conf. Artif. Intell.* 34, 02 (2020), 1308–1315.

[28] Jonathon Luiten, Aljoša Ošep, Patrick Dendorfer, Philip Torr, Andreas Geiger, Laura Leal-Taixé, and Bastian Leibe. 2021. HOTA: A higher order metric for evaluating multi-object tracking. *Int. J. Comput. Vis.* 129, 2 (2021), 548–578.

[29] Paul A. Merolla, John V. Arthur, Rodrigo Alvarez-Icaza, Andrew S. Cassidy, Jun Sawada, Filipp Akopyan, Bryan L. Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. 2014. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 6197 (2014), 668–673.

[30] Elias Mueggler, Henri Rebecq, Guillermo Gallego, Tobi Delbruck, and Davide Scaramuzza. 2017. The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and SLAM. *Int. J. Robot. Res.* 36, 2 (2017), 142–149.

[31] Daniel Müllner. 2011. Modern hierarchical, agglomerative clustering algorithms. arXiv:1109.2378. Retrieved from https://arxiv.org/abs/1109.2378.

[32] Fernando Cladera Ojeda, Anthony Bisulco, Daniel Kepple, Volkan Isler, and Daniel D. Lee. 2020. On-device event filtering with binary neural networks for pedestrian detection using neuromorphic vision sensors. In *Proceedings of the IEEE International Conference on Image Processing (ICIP'20)*, 3084–3088.

[33] Etienne Perot, Pierre de Tournemire, Davide Nitti, Jonathan Masci, and Amos Sironi. 2020. Learning to detect objects with a 1 megapixel event camera. *Adv. Neural Inf. Process. Syst.* 33 (2020), 16639–16652.

[34] Zenon W. Pylyshyn and Ron W. Storm. 1988. Tracking multiple independent targets: Evidence for a parallel tracking mechanism. *Spat. Vis.* 3, 3 (1988), 179–197.

[35] Bharath Ramesh, Andrés Ussa, Luca Della Vedova, Hong Yang, and Garrick Orchard. 2018. PCA-RECT: An energy-efficient object detection approach for event cameras. In *Proceedings of the Asian Conference on Computer Vision*, 434–449.

[36] Alpha Renner, Matthew Evanusa, and Yulia Sandamirskaya. 2019. Event-based attention and tracking on neuromorphic hardware. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW'19)*. 1709–1716.

[37] Rafael Serrano-Gotarredona, Matthias Oster, Patrick Lichtsteiner, Alejandro Linares-Barranco, Rafael Paz-Vicente, Francisco Gómez-Rodríguez, Luis Camuñas-Mesa, Raphael Berner, Manuel Rivas-Pérez, Tobi Delbruck, et al. 2009. CAVIAR: A 45k neuron, 5M synapse, 12G connects/s AER hardware sensory–processing–learning–actuating system for high-speed visual object recognition and tracking. *IEEE Trans. Neural Netw.* 20, 9 (2009), 1417–1438.

[38] Amos Sironi, Manuele Brambilla, Nicolas Bourdis, Xavier Lagorce, and Ryad Benosman. 2018. HATS: Histograms of averaged time surfaces for robust event-based object classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'18)*.

[39] Ricardo Tapiador-Morales, Jean-Matthieu Maro, Angel Jimenez-Fernandez, Gabriel Jimenez-Moreno, Ryad Benosman, and Alejandro Linares-Barranco. 2020. Event-based gesture recognition through a hierarchy of time-surfaces for FPGA. *Sensors* 20, 12 (2020), 3404.

[40] Andrés Ussa, Chockalingam Senthil Rajen, Deepak Singla, Jyotibdha Acharya, Gideon Fu Chuanrong, Arindam Basu, and Bharath Ramesh. 2020. A hybrid neuromorphic object tracking and classification framework for real-time systems. arXiv:2007.11404. Retrieved from https://arxiv.org/abs/2007.11404.

Received 14 September 2022; revised 3 February 2023; accepted 4 April 2023