2024 Design, Automation & Test in Europe Conference (DATE 2024)
Special Initiative "Autonomous Systems Design"

Driving Autonomy with Event-based Cameras: Algorithm and Hardware Perspectives

Nael Mizanur Rahman
Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia
naelmr@gatech.edu

Uday Kamal
Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia
uday.kamal@gatech.edu

Manish Nagaraj
Electrical and Computer Engineering
Purdue University
West Lafayette, Indiana
mnagara@purdue.edu

Shaunak Roy
West Lafayette Junior/Senior High School
West Lafayette, Indiana

Saibal Mukhopadhyay
Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia
saibal.mukhopadhyay@ece.gatech.edu

***Abstract*—In high-speed robotics and autonomous vehicles, rapid environmental adaptation is necessary. Traditional cameras often face issues with motion blur and limited dynamic range. Event-based cameras address these by tracking pixel changes continuously and asynchronously, offering higher temporal resolution with minimal blur. In this work, we highlight our recent efforts in solving the challenge of processing event-camera data efficiently from both algorithm and hardware perspective. Specifically, we present how brain-inspired algorithms such as spiking neural networks (SNNs) can efficiently detect and track object motion from event-camera data. Next, we discuss how we can leverage associative memory structures for efficient event-based representation learning. And finally, we show how our developed Application Specific Integrated Circuit (ASIC) architecture for low-latency, energy-efficient processing outperforms typical GPU/CPU solutions, thus enabling real-time event-based processing. With a 100× reduction in latency and a 1000× lower energy per event compared to state-of-the-art GPU/CPU setups, this enhances the front-end camera systems capability in autonomous vehicles to handle higher rates of event generation, improving control.**

***Index Terms*—event-based camera, spiking neural network, multi-object tracking, associative memory, hardware**

I. Introduction

Event-based cameras are poised to significantly advance real-time machine vision applications, particularly in robotics and autonomous driving [1, 2], thanks to their low power consumption, extensive dynamic range, high temporal resolution, and minimal latency. While existing methods, primarily based on convolutional and recurrent neural networks and initially designed for traditional frame-based cameras, have shown impressive perceptual accuracy when adapted to event cameras [3], they tend to convert sparse event data into dense, frame-like representations. This process, unfortunately, overlooks the natural sparsity of event data and leads to considerable computational overhead. Recent studies have explored various event-based processing methods, especially for tasks like object recognition, to leverage the inherent sparsity of event-based camera data and reduce computational complexity. However, these approaches have not yet reached the performance levels of their dense representation-based counterparts, underlining the need for developing computationally efficient algorithms that effectively utilize sparsity while achieving high accuracy.

In this work, we briefly discuss the recent developments for efficient event-based perception from both algorithm and hardware perspective. From algorithm aspects, we first outline the novel formulation event-based multi object tracking through SNN [4]. Obstacles pose and their motion estimation in the scene being important aspect of autonomous navigation, external motion-capture mechanisms [5, 6], have been mostly utilized to estimate the pose and motion of obstacles while maneuvering. However, such systems can not be deployed in real-world applications where new environments are constantly explored and external motion-capture mechanisms are not easy to install. Perception-based algorithms eliminate this dependency on external motion-capture mechanisms in such systems. While event-based cameras have the potential to provide an energy-efficient solution with low latency, a recently developed algorithm, DOTIE [4], fully utilizes these advantages by leveraging a lightweight SNN that can isolate events based on the speed of movement of their corresponding objects.

SNNs are efficient in processing event-camera data but fall short in accuracy compared to modern dense architectures like Convolutional Neural Networks (CNNs) and Graph Networks [1, 2]. This is mostly due to limited differentiability in spiking operations and complex hyperparameter tuning. Alternate architectures, such as CNNs, often require temporal aggregation of events, reducing throughput and leading to synchronous operation [3]. Recent sparse alternatives, including graph neural networks [7, 8] and point-cloud-based methods [9, 10], reduce computational demands but still process past events redundantly. In contrast, human brains efficiently integrate sensory inputs with stored memory patterns, a process studied in cognitive research [11, 12]. Inspired by this, EventFormer as proposed in [13], is an event-based perception framework

N. Rahman, U. kamal, and S. Mukhopadhyay would like to acknowledge the project EventFormer supported by the Defense Advanced Research Projects Agency (DARPA) under Grant Numbers GR00019153

979-8-3503-4859-0/DATE24/© 2024 EDAA

that leverages associative memory mechanism to store and retrieve compressed latent features in a highly compute efficient manner. More specifically, it processes incoming event streams by encoding positional coordinates into a higher-dimensional space, computes interactions using self-attention, and retrieves past representations from memory. A recurrent module combines past and present states to produce refined representations, which are stored for future reference. A task-specific head operates directly on this memory representation for the target task (classification).

While the EventFormer algorithm reduces computational complexity in event-based perception, it faces high latency in Software/GPU implementations due to heterogeneous computing, memory access delays, and synchronization issues. Additionally, transferring sensor data to remote GPU servers can cause bandwidth and latency problems. Implementing EventFormer using on-site dedicated processors can address these issues by allowing local event data processing, greatly reducing latency and bandwidth needs, crucial for quick data analysis and decision-making. Dedicated hardware accelerators, specifically an ASIC design with associative memory architecture, can further reduce memory access latency, increase computing throughput, and enable faster event processing. Therefore, we present a high-level ASIC architecture designed for the EventFormer algorithm, capable of handling 100 simultaneous events with an impressive latency of only $0.48\mu s$ per event. Compared to traditional GPU setups, this architecture offers a significant $100\times$ reduction in latency and a $1000\times$ increase in energy efficiency, making it highly advantageous for event-based perception applications.

The rest of the paper is structured as follows: Section II discusses the event-based solution for multi-object motion tracking. Section III discusses the potential application of associated memory structure for event-based representation learning. Finally, we discuss potential custom hardware solution for real-time event-based perception in section IV and conclude the study with a brief discussion in section V.

II. Event-based High Speed Multi Object Tracking

Detecting and tracking moving objects in the surrounding environment is essential to avoid collision in autonomous navigation systems. Consequently, there has been an extensive amount of research focusing on these tasks. For traditional frame-based camera outputs, the abundance of rich photometric features makes it simpler to detect and track objects. Events on the other hand, while temporally rich, lack such photometric features. To make more optimal use of the events and reduce the compute and latency overheads caused by standard Artificial Neural Networks (ANNs), recently proposed DOTIE[4] uses a shallow spiking architecture to separate events based on the speed of the corresponding object. This study explores the use of spiking neurons, specifically the leaky-integrate-and-fire (LIF) model [14], to capture temporal information in events. LIF neurons have a membrane potential and a user-set threshold. Inputs at each time step increase the membrane potential, which also decays at a fixed rate (the leak factor). If the membrane potential surpasses the threshold, the neuron fires an output spike and resets. Frequent input spikes can overcome the decay, triggering an output, whereas slower inputs result in decay before reaching the threshold. This mechanism is illustrated in Fig. 1.

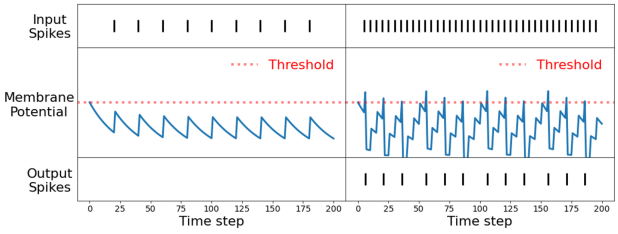


Fig. 1. Temporal sensitivity of a spiking neuron. A neuron with faster inputs (on the right) emits output spikes, while the one with slower inputs (on the left) does not. Figure borrowed from [4].

Objects moving fast produce events at a faster rate, LIF neurons receiving event streams corresponding to these objects as inputs hence produce an output spike. By fine-tuning the leak factor and threshold values of the spiking architecture, DOTIE, a single layer of LIF neurons, can selectively isolate objects moving faster than a particular speed. Multiple (parallelly run) branches of DOTIE fine-tuned for different speeds can be used to isolate objects moving within a particular range of speed (as shown in Fig. 2).

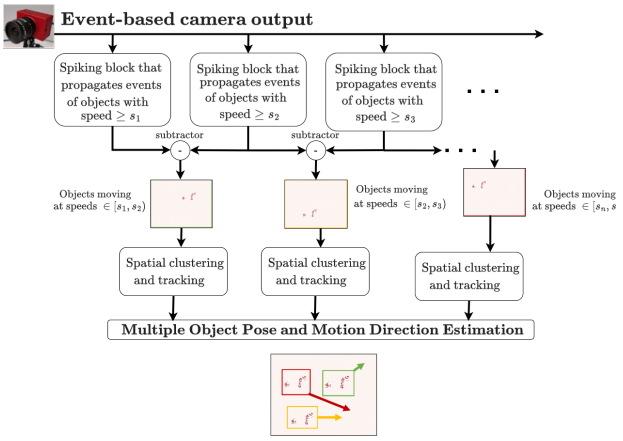


Fig. 2. Multiple branches of DOTIE fine-tuned for specific speeds can be used along with spatial clustering and tracking to estimate the pose and motion direction of multiple objects. Figure adapted from architecture proposed in [4].

The clusters of events isolated by the spiking architecture can then be clustered (based on spatial proximity) to detect object boundaries. Clustering techniques independent of the number of clusters and the size of clusters (such as DBSCAN [15]) are deployed as the number and size of objects are unknown. By computing the center of these clusters and comparing the position of the center between two timestamps, it is possible to track the object and estimate the direction of motion (as shown in Fig. 2.) The DOTIE algorithm also shows an improvement in performance both in accuracy of detection as well as computational efficiency. We show that while most existing object detection works on event-based cameras can only operate in idealistic scenarios without any background events or camera noise, DOTIE can be deployed in realistic scenarios. DOTIE

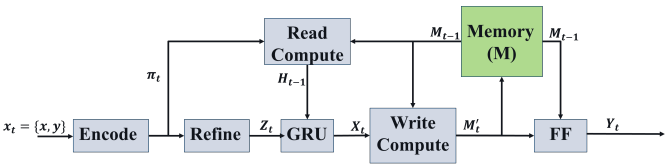


Fig. 3. Overall compute flow diagram of EventFormer architecture

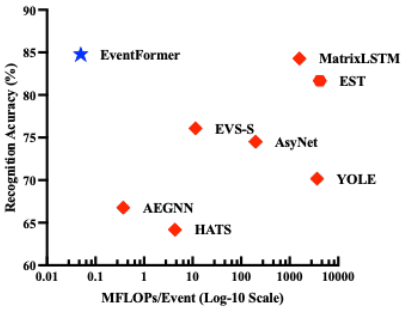


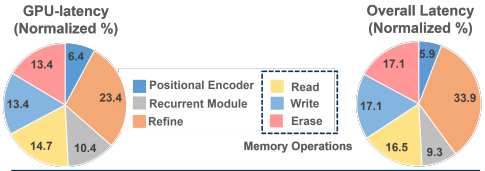
Fig. 4. EventFormer performance comparison with existing SoTA methods on event-based NCaltech-101 dataset.

showed a mean Intersection-over-Union (IoU) of 0.8593 on the MVSEC dataset [16], a publicly available dataset that contains data collected by deploying an event camera in a driving scenario. It outperformed all other existing works, which could only produce mean IoUs ≤ 0.4. Further, owing to the sparsity of event-based inputs and the energy efficiency of spiking neurons, DOTIE consumed an average energy of $11.03nJ$ when deployed on the MVSEC dataset. This was a huge reduction in energy consumption as compared to an average energy of $44.06mJ$ that a standard ANN such as YOLOv3 [17] uses during inference on the same dataset.

III. Memory Augmented Event-based Perception

EventFormer Overview: Eventformer [13] is a recently proposed associative memory augmented representation learning framework for event-based perception. Given a set of n-events coordinates $x_t \in \mathbb{R}^{2\times n}$ at time t, it first encodes and maps these positional information to a higher d-dimensional feature space, $\pi_t \in \mathbb{R}^{d\times n}$ by using a positional encoder [18]. This is followed by a pairwise interaction module (Refine) that calculates spatial relationships among the events in the embedding space. A recurrent module takes the current latent state $\mathcal{Z}_t$ and generates a spatiotemporal representation $\mathcal{X}_t$ by accessing past hidden states $\mathcal{H}_{t-1}$ stored in an associative memory called $\mathcal{M}$. The output of this recurrent module updates the memory representation $\mathcal{M}_t$, which is then passed to an MLP (FF) for the recognition task, as illustrated in Fig. 3.

Detailed Operation: EventFormer introduces three unique operators: $Read$, which retrieves past representations; $Write$, responsible for updating the memory with new information; and $Refine$, computes spatial correlations among the events. All of these operators employ a multi-head residual attention (MRA) mechanism [19] as a foundational building block. $Read$ operation extracts the past states at the current event location by using the MRA block to perform query-key-based associations



EventFormer Operation	Overall latency (us)	GPU Kernel compute latency (us)	CPU-GPU I/O and synchronization latency (us)
Positional Encoder	164	24	140
Recurrent Module	259	39	220
Refine Operator	942	89	853
Memory (Read + Write + Erase)	1446	221	1225
Total	2811	373	2438

Fig. 5. EventFormer sub-module level latency breakdown to process one event.

in the memory. To be more specific, we query the past memory representation $\mathcal{M}_{t-1}$ using the positional embedding of the current event locations, π_t. The complete retrieval of the past hidden representation, $\mathcal{H}_{t-1}$ involves the following operations:

$$H_{t-1} = Read(\pi_t, \mathcal{M}_{t-1}) := LayerNorm(O_r + FF_r^o(O_r)) \quad (1)$$

$$O_r = LayerNorm(\pi_t + MRA(Q_r, K_r, V_r; w, a)) \quad (2)$$

Here, Q_r represents the query vector calculated from π_t, and K_r, V_r represents the key and value vectors from the $\mathcal{M}_{t-1}$. Similar to the $Read$ operator, we adopt a separate MRA block for $Write$ operator to calculate the new memory representation, $\mathcal{M}'_t$. However, this time we compute the query vectors from $\mathcal{M}_{t-1}$ and *key-value* pair from the refined spatiotemporal representation, $\mathcal{X}_t$. The idea here is that we want to query the location of the memory that needs to be updated while the contents to be updated are provided by the new representation.

$$\mathcal{M}'_t = Write(\mathcal{M}_{t-1}, \mathcal{X}_t) := LayerNorm(O_w + FF_w^o(O_w)) \quad (3)$$

$$O_w = LayerNorm(\mathcal{M}_{t-1} + MRA(Q_w, K_w, V_w; w, a)) \quad (4)$$

Here, Q_w represents the query vector calculated from $\mathcal{M}_{t-1}$, and K_w, V_w represents the key and value vectors computed from the $\mathcal{X}_t$. We also introduce $Erase$ operator (follows the same operations of $Write$ operator) that calculates a set of element-wise scaling factors, $\alpha_t \in \{\mathbb{R}^{m\times d} \mid 0 \leq \alpha_t \leq 1\}$ to control the strength of update:

$$\alpha_t = sigmoid(Erase(\mathcal{M}_{t-1}, \mathcal{X}_t)) \quad (5)$$

The final memory update is computed as below:

$$\mathcal{M}_t = \alpha_t \mathcal{M}_{t-1} + (1 - \alpha_t)\mathcal{M}'_t \quad (6)$$

Algorithmic and Compute Performance:

EventFormer exhibits impressive performance on the NCaltech101 dataset, outperforming both conventional frame-based and event-based methods in accuracy and computational efficiency as shown in Fig. 4. On a system with an NVIDIA 3090 GPU Accelerator, EventFormer shows an overall latency of 2.8 ms for processing a single event. Fig. 5 details the latency breakdown for EventFormer's components, revealing that over 85% of this latency is due to CPU-GPU data transfer and synchronization. The GPU kernel latency itself is 373

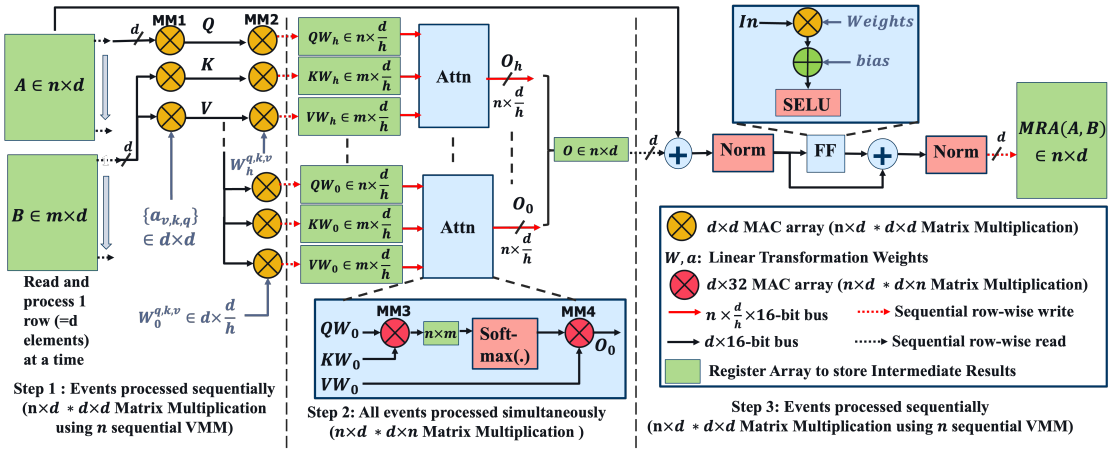


Fig. 6. Multi head-Residual Attention Block. MM1 and MM2 are $n \times d * n \times d$ matrix multiplication and can be performed sequentially with respect to n; MM3 is an $n \times d * d \times n$ matrix multiplication and MM4 is an $n \times m * m \times d$ matrix multiplication and require the full input matrices for processing

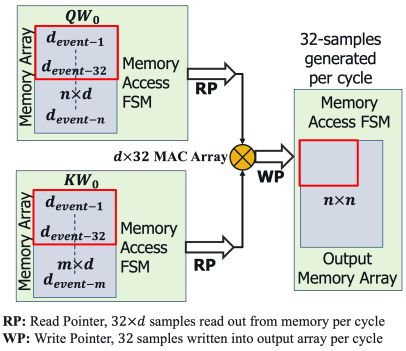


Fig. 7. Example FSM to carry out $n \times d * d \times m$ matrix multiplication for MM3 and MM4

µs, which is high compared to the event generation rate of 100KHz or 10µs. This highlights the need for specialized hardware to address these limitations. An ASIC accelerator specifically designed for EventFormer, with integrated on-chip associative memory, can reduce both memory access latency and synchronization delays common in GPU/CPU systems. This development is crucial for low-latency event processing, essential in applications requiring rapid decision-making.

IV. Hardware Solution for Real-Time Application

Figure 3 outlines the overall eventformer architecture. The algorithm introduces three unique operators: Refine, Read Compute, and Write Compute. Each of these operators is based on the multihead residual attention operation.

Multi-head Residual Attention Block: The MRA's hardware architecture, outlined in 6, commences with the generation of the Query (Q), Value (V), and Key (K) matrices. This process involves MM (denoted by MM1 in Fig. 6) of the input matrices $A \in \mathbb{R}^{n \times d}$ and $B \in \mathbb{R}^{m \times d}$ with weight matrices $a_{v,k,q} \in \mathbb{R}^{d \times d}$. Here, n defines the number of events, m is the number of latent spaces in the memory representation and d is the dimensions of each latent space. Therefore, generating Q through MM1 requires $n \times d \times d$ Multiply-Accumulate (MAC) operations, while generating K, V require $m \times d \times d$ MAC operations. Next, multi-headed transformations of Q (and K, and V) are performed. This involves another MM process (MM2 in Fig. 6) with a weight matrix $W_h \in \mathbb{R}^{d \times d/h}$, again requiring $n \times d \times d$ MAC operations ($m \times d \times d$ for K, V) for all heads combined. In both MM1 and MM2, the dimension of weight multiplicands (W_h and a) are independent of n or m, while the input multipliers (A, B, Q, K, and V) vary with n or m. This allows for the serialization of computation with respect to the number of events (n). Specifically, in each pass, the corresponding row of A is processed through MM1 and MM2, sequentially producing the rows of the output matrix QW_h. This requires n passes in total and is facilitated by a $d \times d$ MAC array, reutilized across each pass. Similarly, generation of KW_h and VW_h requires a total of m serial passes based on m rows of the input B. This serialized operation ensures efficient hardware area utilization. For $n > m$ as is usually the case, the total latency of MM1 and MM2 is therefore $O(n)$. Once all events have been sequentially processed and stored in buffers, the matrices QW_h, KW_h, and VW_h advance to the Attention block for further processing.

The attention block (step 2 in Fig. 6) first involves the MM of $QW_h \in \mathbb{R}^{n \times \frac{d}{h}}$ and $KW_h^T \in \mathbb{R}^{d \times m}$ matrices to yields an intermediate $n \times m$ matrix (denoted by MM3). Unlike the previous processing stages where serialization was possible for n, MM3 requires the entire input matrices to be stored beforehand, due to both the multiplier and multiplicand being dependent on n and m. Therefore, QW_h and KW_h matrices are initially stored in separate memory units and a Finite State Machine (FSM) is employed to sequentially access the relevant rows from these matrices and compute individual elements of the output matrix. the total number of MAC operations required for MM3 is $n \times m \times d$ and implies a computational latency of $O(nm)$. To balance hardware resource utilization and computational latency, MM3 in the attention block uses a $32 \times d$ parallel MAC array. This design choice enables parallel computation of 32 elements of the resultant $n \times m$ output matrix simultaneously. The same parallel MAC array structure is also implemented for the subsequent matrix multiplication step (denoted by MM4) involving the $n \times m$ intermediate product and the $VW_0 \in \mathbb{R}^{m \times \frac{d}{h}}$, ultimately yielding the final result (O_h) for each head of the attention operation. the parallel

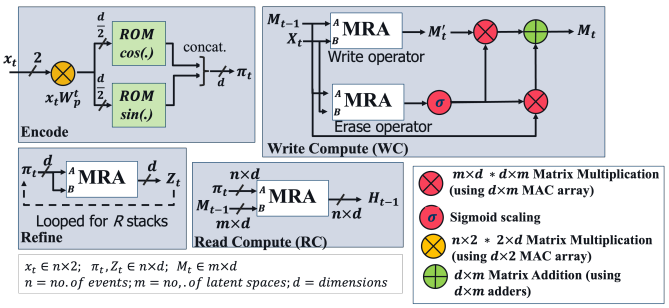


Fig. 8. Details of unique Eventformer operators: Refine, Memory operations and Encode, as a function of Multihead Residual Attention Operation

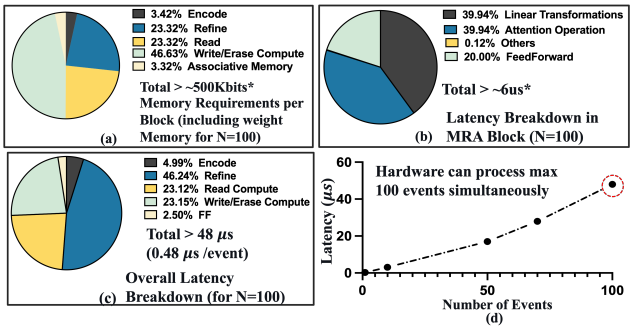


Fig. 9. (a) Memory storage requirement for EventFormer Hardware (b) Latency breakdown within MRA block (c) Overall latency breakdown of $0.48\mu s$ per event for 100 events (d) Near linear increase in latency with increasing number of events processed simultaneously

outputs of multiple heads are concatenated to generate the $n \times d$ result, O, which is used in the final step of the MRA operation. This parallel computation approach reduces the overall latency while maintaining efficient hardware use.

All the subsequent operations in the MRA block after the Attention operation (Step 3 in Fig. 6) (normalize, Feed Forward etc.) can be serialized for the number of events, similar to step 1, and therefore yields a latency of $O(n)$.

EventFormer Operators: Fig. 3 shows the overall flow of the EventFormer algorithm and Fig. 8 demonstrates the hardware implementation of each of the unique EventFormer operators as a function of the MRA block described previously. The input to the encode operation is a list of events ($n \times 2$ matrix) and undergoes an MM operation with $n \times d$ weight matrix, W_p^t. This is implemented using d MAC engines, serially operating on each row of X_t. This is followed by a non-linear sinusoid computation, which is implemented using a Read Only Memory (ROM) to generate the result, $\pi_t \in \mathbb{R}^{n \times d}$. The Refine block operates on π_t and uses the MRA block sequentially (output of MRA is fed back into input) and this is repeated R times, where R is a model hyperparameter. It is to be noted that for the MRA operation in the refine block, both the inputs have the same dimension (i.e $n = m$). Both the read and write compute blocks are also functions of the MRA block. WC requires two parallel MRA blocks for write and erase operation, which are then proceeded by another sigmoid scaling followed by two MM operations (MM5 and MM6). MM5 and MM6 are both a multiplication of two $m \times d$ matrices, implemented using the FSM shown in Fig. 7.

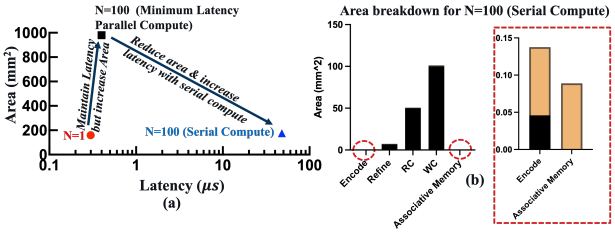


Fig. 10. (a) Area/latency tradeoff between three hardware implementations of EventFormer (b) Register vs Compute area breakdown for different components

	ASIC Accelerator	RTX 3090 GPU
No. of Events (n)	100	100
Technology	28nm	8nm
Latency	48μs	4.78ms
Area	176mm^2	628mm^2 *
Power	42W	700W*
Energy/event	20μJ / event	33460μJ/ event

*100% of chip area and power may not be utilized for inference task

Fig. 11. Comparison of Eventformer ASIC implementation with GPU

Hardware Cost Analysis: The attention operation in Eventformer requires data storage for all events before MM3 and MM4 operations can commence, as shown in Fig. 7. This requirement sets a practical limit on the number of events that the hardware can process simultaneously. We assess the hardware designed for processing up to 100 events at once, with hyper-parameters set at $m = d = 32$, $R = 2$, and $h = 4$, and implemented in 28nm CMOS technology at a 1 GHz synthesis frequency. All required memory is implemented using register files, facilitating parallel access and minimizing latency. This analysis does not delve into the implementation of nonlinear functions like softmax and activation, hence the presented area, power, and latency estimates represent a lower bound.

Fig. 9(a) illustrates the memory distribution within the hardware, showing associative memory as a minor component of the overall memory requirement. The latency breakdown of a single MRA block, shown in Fig. 9(b), highlights that linear transformation and attention operations (MM1 to MM4) are the most time-intensive. The total latency for concurrently processing 100 events is approximately $48\mu s$, averaging around $0.48\mu s$ per event. The Refine block consumes the majority of this latency due to its need for multiple MRA iterations, as shown by $R = 2$. Fig. 9(d) demonstrates a nearly linear relationship between the number of events (n) and latency. Fig. 10 (a) investigates the area and latency trade-offs in the hardware implementation. An initial design capable of processing a single event at a time yields an area of $150mm^2$ and a latency of $0.3\mu s$. Expanding this to handle 100 events simultaneously, with full parallel MAC engines without serializing computations ((for ex: $n \times d \times d$ MAC engines for MM1 etc.)), results in a substantial area increase to $980mm^2$ but only slightly elevates latency to $0.4\mu s$ ($0.04\mu s$ per event). To balance area constraints with latency, the serialized compute architecture discussed above thus consumes an area of $176mm^2$ and a total latency of $48\mu s$ for 100 events ($0.48\mu s$ per event). Fig. 10(b) details the area breakdown between compute and

register components across all designs, showing that compute dominates area usage in critical blocks. Thus, replacing register arrays with SRAM may not offer significant area benefits and could restrict parallel data access capabilities.

Figure 11 compares the performance of the ASIC Eventformer architecture with a GPU, for processing 100 events. Processing in the GPU has a higher latency and significantly higher energy of computation per event, as compared to the ASIC implementation, showing the benefits of dedicated ASIC hardware implementation of the EventFormer algorithm.

V. Conclusion

This work briefly overviews our recent efforts in developing algorithm and hardware solution efficient event-based perception with an application towards autonomous driving. We discussed how it is possible to develop low-weight neural architectures capable of performing complex tasks (i.e., multi-object tracking) with minimal overheads by efficiently utilizing the temporal processing capabilities of spiking neurons. Subsequently, we outlined how an Associative Memory-augmented architecture can efficiently transform sparse event streams into a compact memory space, achieving tremendous reduction in computation compared to conventional dense and other event-based methods. Additionally, we examined the hardware cost of an ASIC accelerator for such event-based representation, integrated with on-chip associative memory, capable of handling up to 100 simultaneous events with significant reduction in latency and energy consumption per event, vastly outperforming standard GPU/CPU systems. These advancements can significantly improve the capability of front-end camera/sensor systems in managing higher rates of event generation, greatly enhancing control in autonomous systems.

References

[1] Davide Falanga, Kevin Kleber, and Davide Scaramuzza. Dynamic obstacle avoidance for quadrotors with event cameras. *Science Robotics*, 5(40):eaaz9712, 2020.

[2] Sihao Sun, Giovanni Cioffi, Coen De Visser, and Davide Scaramuzza. Autonomous quadrotor flight despite rotor failure with onboard vision sensors: Frames vs. events. *IEEE Robotics and Automation Letters*, 2021.

[3] Daniel Gehrig, Antonio Loquercio, Konstantinos G Derpanis, and Davide Scaramuzza. End-to-end learning of representations for asynchronous event-based data. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5633–5643, 2019.

[4] Manish Nagaraj, Chamika Mihiranga Liyanagedera, and Kaushik Roy. Dotie-detecting objects through temporal isolation of events using a spiking architecture. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4858–4864. IEEE, 2023.

[5] Daniel Mellinger, Nathan Michael, and Vijay Kumar. Trajectory generation and control for precise aggressive maneuvers with quadrotors. *The International Journal of Robotics Research*, 31(5):664–674, 2012.

[6] Sergei Lupashin, Angela Schöllig, Michael Sherback, and Raffaello D'Andrea. A simple learning strategy for high-speed quadrocopter multi-flips. In *2010 IEEE international conference on robotics and automation*, pages 1642–1648. IEEE, 2010.

[7] Simon Schaefer, Daniel Gehrig, and Davide Scaramuzza. Aegnn: Asynchronous event-based graph neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

[8] Yijin Li, Han Zhou, Bangbang Yang, Ye Zhang, Zhaopeng Cui, Hujun Bao, and Guofeng Zhang. Graph-based asynchronous event processing for rapid object recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 934–943, 2021.

[9] Sai Vemprala, Sami Mian, and Ashish Kapoor. Representation learning for event-based visuomotor policies. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 4712–4724. Curran Associates, Inc., 2021.

[10] Qinyi Wang, Yexin Zhang, Junsong Yuan, and Yilong Lu. Space-time event clouds for gesture recognition: From rgb cameras to event cameras. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1826–1835. IEEE, 2019.

[11] Thomas D Albright. On the perception of probable things: neural substrates of associative memory, imagery, and perception. *Neuron*, 74(2):227–245, 2012.

[12] Jin-Hui Wang et al. Associative memory cells and their working principle in the brain. *F1000Research*, 7, 2018.

[13] Uday Kamal, Saurabh Dash, and Saibal Mukhopadhyay. Associative memory augmented asynchronous spatiotemporal representation learning for event-based perception. In *ICLR*, 2023.

[14] Arnaud Delorme, Jacques Gautrais, Rufin Van Rullen, and Simon Thorpe. Spikenet: A simulator for modeling large networks of integrate and fire neurons. *Neurocomputing*, 26:989–996, 1999.

[15] Lian Duan, Lida Xu, Feng Guo, Jun Lee, and Baopin Yan. A local-density based spatial clustering algorithm with noise. *Information systems*, 32(7):978–986, 2007.

[16] Alex Zihao Zhu, Dinesh Thakur, Tolga Ozaslan, Bernd Pfrommer, Vijay Kumar, and Kostas Daniilidis. The multi vehicle stereo event camera dataset: An event camera dataset for 3d perception. pages arXiv–1801, 2018.

[17] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[18] Yang Li, Si Si, Gang Li, Cho-Jui Hsieh, and Samy Bengio. Learnable fourier features for multi-dimensional spatial positional encoding. *Advances in Neural Information Processing Systems*, 34:15816–15829, 2021.

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.