Eventor: An Efficient Event-Based Monocular Multi-View Stereo Accelerator on FPGA Platform

Mingjun Li¹, Jianlei Yang¹, Yingjie Qi¹, Meng Dong², Yuhao Yang¹, Runze Liu³, Weitao Pan², Bei Yu⁴, Weisheng Zhao¹

¹ Beihang University, Beijing, China
² Xidian University, Xi’an, Shaanxi, China
³ Beijing Real Imaging Medical Technology Co., Ltd.
⁴ The Chinese University of Hong Kong, Hong Kong

July 12, 2022
Outline

• Research Background and Motivation
• Eventor
  • Algorithm Framework
  • Software Optimizations
  • Hardware Architecture
• Evaluation
• Conclusions
Outline

• **Research Background and Motivation**
• **Eventor**
  • Algorithm Framework
  • Software Optimizations
  • Hardware Architecture
• **Evaluation**
• **Conclusions**
Multi View Stereo (MVS)

- **Multi View Stereo (MVS)**
  - **Input**: a set of **photographs** of an object or a scene
  - **Target**: estimate the most likely **3D shape** that explains those photographs
  - **Assumption**: known viewpoints
Event Camera

- **Event Camera**
  - Bio-inspired vision sensor (DVS)
  - Asynchronous output: event stream
  - \( e = < x, y, t, p > \) pixel coordinates, timestamp, polarity of brightness changes

- **Advantages**
  - Low latency (~1 micro-second)
  - High dynamic range (120 dB instead 60 dB)
  - Low data rate, low storage capacity (KB vs. MB)
  - Low power consumption (~20 mW)

[Scaramuzza D. Tutorial on Event-based Vision for High-Speed Robotics. URL: http://rpg.ifi.uzh.ch, 2015.]
Event-based Multi View Stereo (EMVS)

- Monocular EMVS
  - Estimate semi-dense 3D structure from an event camera with known trajectory
  - Critical task in the mapping part of monocular event-based SLAM

[Rebecq H, Gallego G, Mueggler E, et al. EMVS: Event-based multi-view stereo - 3D reconstruction with an event camera in real-time. IJCV’18.]
EMVS Application Scenarios

Event-based SLAM

Drones

Robots

Self-driving Cars

AR/VR

3D Map Modeling
Existing Works on Monocular EMVS

- [Kim et al., ECCV’16]
  - Three filters running in parallel to jointly estimate the motion of the event camera and 3D map
  - Only runs on GPUs for real-time performance and cannot process high event rate input (up to 1M events/s)

- [Gallego et al., CVPR’18]
  - A unified event processing framework for motion estimation, depth estimation and optical flow estimation
  - Only evaluated on a desktop CPU and no quantitative results are provided

- [Rebecq et al., IJCV’18]
  - Event-based space-sweep method
  - Runs in real-time on a desktop CPU (1.2 M events/s with a single core)

Existing Monocular EMVS implementations only run on desktop processors, with inadequate performance!
New Paradigm: EMVS vs MVS

- Question: Can we directly use existing MVS accelerators on EMVS?

|       | MVS                              | EMVS                                       |
|-------|----------------------------------|--------------------------------------------|
| Input Data | Frame-based images              | Asynchronous event stream                  |
| Algorithm | Traditional multi-view stereo algorithm | Novel event-based multi-view stereo algorithm |
| Output | Dense/Sparse 3D reconstruction | Semi-dense 3D reconstruction               |

- Different data structure and algorithm pipelines!

Previous accelerators for frame-based MVS can not be directly applied to EMVS!
Challenges & Motivation

• Real-time Demand
  • EMVS: computational intensive
  • Utilize low-latency advantage: high computation speed required
  • Expected event processing rate: over 1.8 Million events per second

• Limited Platform Resources
  • Implement EMVS on embedded platforms: high energy efficiency processors required
  • Desktop processors (CPU or GPU): not practical for resources-limited and power-limited platforms

• New Computation Paradigm
  • Current EMVS algorithms: lack hardware-oriented optimization
  • Previous MVS accelerators: incompatible with EMVS

Accelerate monocular EMVS via algorithm-hardware co-optimization!
Outline

• Research Background and Motivation
• Eventor
  • Algorithm Framework
  • Software Optimizations
  • Hardware Architecture
• Evaluation
• Conclusions
EMVS Algorithm Framework

- Basic Framework
  - Monocular EMVS using event-based space-sweep method [Rebecq et al., IJCV’18]
  - Relatively high parallelism
  - Relatively low data dependency
  - Relatively low computational redundancy
  - Suitable for customized hardware (e.g. FPGA) acceleration
EMVS Algorithm Framework

- **Event Aggregation**
- Divide the event stream to event frames (i.e. event packets) which will be processed together.
EMVS Algorithm Framework

- Key Frame Selection
- Select key reference view and construct local discretized space volume (i.e. Disparity Space Image, DSI)

Event Stream → Event Aggregation → Event Frame

New Key Frame? (True)
- Scene Structure Detection ($D$)

False
- Reset DSI
- Point Cloud Conversion
- Map Updating
- Semi-Dense 3D Map

Event Back-Projection ($P$)

Volumetric Ray-Counting ($R$)
EMVS Algorithm Framework

- Event Back-Projection ($P$)
- Back-project events from the input event frame to the reference viewing space

- Event Stream
- Event Aggregation
- Event Frame
- New Key Frame?
  - True: Scene Structure Detection ($D$)
  - False: Reset DSI
- Volumetric Ray-Counting ($R$)
- Event Back-Projection ($P$)
- Map Updating
- Semi-Dense 3D Map

- Back-project events from the input event frame to the reference viewing space
EMVS Algorithm Framework

- Volumetric Ray-Counting ($\mathcal{R}$)
- Count the number of back-projection rays that pass through each DSI voxel
EMVS Algorithm Framework

- **Scene Structure Detection (D)**
- Determine 3D points by finding local maximum of the ray density
EMVS Workload Profiling

EMVS Runtime Profiling %

- Scene Structure Detection ($D$): 7.23%
- Volumetric Ray-Counting ($R$): 6.92%
- Others: 6.92%
- $P + R$: 85.85%

Total: 100%

FPGA Acceleration

- Event Stream
- Event Aggregation
- Event Frame
- New Key Frame? Yes/No
  - Yes: Scene Structure Detection
  - No: Reset DSI
- Event Back-Projection ($P$)
- Volumetric Ray-Counting ($R$)
- Point Cloud Conversion
- Map Updating
- Semi-Dense 3D Map

Evaluated on the DAVIS event-camera dataset and simulator
Outline

• Research Background and Motivation
• Eventor
  • Algorithm Framework
  • Software Optimizations
  • Hardware Architecture
• Evaluation
• Conclusions
Critical Tasks Breakdown

- Two-step back-projection in $P$
- Canonical Event Back-Projection ($CP$): current event frame $\rightarrow$ canonical homography plane
- Proportional Event Back-Projection ($PP$): canonical plane $\rightarrow$ the whole viewing space (DSI)

Most computational intensive tasks: $CP$, $PP$, $R$
Hardware-Friendly Reformulation

- Partially Reschedule
  - Improve memory access efficiency
  - Reduce data transfer between FPGA and external memory
  - Compact computational intensive stages, efficiently accelerate them in a fully pipelined manner
Approximate Computing

**Depth estimation error (AbsRel) comparison between different voting strategies**

- **Bilinear Voting**
  - 1 projection updates 4 voxels

- **Nearest Voting**
  - 1 projection updates 1 voxel

- **Nearest voting**
  - Lower computation complexity
  - More hardware-friendly memory access pattern
  - Slightly higher reconstruction error

*Adopt nearest voting strategy in volumetric ray-counting (R)*

@ DAVIS Dataset: simulation_3planes, simulation_3walls, slider_close, slider_far
Hybrid Data Quantization

Table: data quantization strategies for $\mathcal{P}$ and $\mathcal{R}$.

| Quantized Data | Total #bit | #bit of Integer | #bit of Decimal |
|----------------|------------|-----------------|-----------------|
| $(x_e, y_e)$    | 16         | 9               | 7               |
| $\{x_e(\mathcal{CP}), y_e(\mathcal{CP})\}$ | 16 | 9 | 7 |
| $\{x_e(\mathcal{PP}), y_e(\mathcal{PP})\}$ | 8  | 8  | 0  |
| $\mathcal{H}$  | 32         | 11              | 21              |
| $\phi$         | 32         | 11              | 21              |
| DSI Scores     | 16         | 16              | 0               |

- $(x_e, y_e)$: input event coordinates
- $\{x_e(\mathcal{CP}), y_e(\mathcal{CP})\}$: back-projected event coordinates after $\mathcal{CP}$
- $\{x_e(\mathcal{PP}), y_e(\mathcal{PP})\}$: back-projected event coordinates after $\mathcal{PP}$
- $\mathcal{H}$: homography matrix used in $\mathcal{CP}$
- $\phi$: parameters used in $\mathcal{PP}$
- **DSI Scores**: the number of back-projected viewing rays passing through each DSI voxel

- **Floating-point $\rightarrow$ Fixed-point** (linear quantization)
- Save up to **50% memory requirement** and **data transfer bandwidth**
- Simplify computational logic
Hybrid Data Quantization

Table: data quantization strategies for $\mathcal{P}$ and $\mathcal{R}$.

| Quantized Data               | Total #bit | #bit of Integer | #bit of Decimal |
|------------------------------|------------|-----------------|-----------------|
| $(x_e, y_e)$                 | 16         | 9               | 7               |
| $\{x_e(\mathcal{CP}), y_e(\mathcal{CP})\}$ | 16         | 9               | 7               |
| $\{x_e(\mathcal{PP}), y_e(\mathcal{PP})\}$ | 8          | 8               | 0               |
| $\mathcal{H}$                | 32         | 11              | 21              |
| $\phi$                       | 32         | 11              | 21              |
| DSI Scores                   | 16         | 16              | 0               |

- Maximum depth estimation error difference: **1.01%**

Accuracy of the quantized framework is acceptable.
Outline

• Research Background and Motivation

• Eventor
  • Algorithm Framework
  • Software Optimizations
  • Hardware Architecture

• Evaluation

• Conclusions
Computation Parallelism Analysis

**Operator-Level Parallelism**
- Multiple arithmetic logic units (ALUs) can be deployed for fine-grained parallelism

**Event-Level Parallelism**
- Different events can be processed in parallel and the computation stages can be fully pipelined

**DSI-Level Parallelism**
- Event back-projections and voting for different DSI voxels can be executed in parallel
Eventor Overall Architecture

- ARM-FPGA Heterogeneous Acceleration
- ARM configures DMA to transfer input data
- ARM fires up the FPGA acceleration modules
- FPGA Acceleration modules receive input event frames and update DSI data stored in DRAM
Eventor Overall Architecture

- **Canonical Projection Module**: executes $CP$
- **Proportional Projection Module**: executes $PP, R$
Canonical Projection Module

- **Buffer**: double-buffering structure
  - **Buf_H, Buf_E, Buf_P**: input buffers
  - **Buf_I**: intermediate buffer
- **PE_Z0**: executes $CP$, fully pipelined
  - **MV MAC Units** (matrix-vector multiply-accumulate units)
  - **Normalization Function Unit**
- **Canonical Projection Controller**
  - finite-state machine (FSM)
 Canonical Projection Module

- **Multiple ALUs** are deployed in PE_Z0 to accelerate matrix and vector calculation
- Input **events** are processed in a **fully-pipelined** scheme without data dependency
  - Exploit parallelism
    - Operator-Level
    - Event-Level
Proportional Projection Module

- **Data Allocator**: fetches and allocates input data
- **PE_Zi**: execute $\mathcal{PP}$ and part of $\mathcal{R}$
  - Scalar MAC Units
  - Nearest Voxel Finder
  - Vote Address Generator
- **Buf_V**: double-buffering structure, output buffer
- **Vote Execute Unit**: votes DSI voxels (updates DSI scores), completes $\mathcal{R}$
- **Proportional Projection Controller**
Proportional Projection Module

- **Multiple ALUs** are deployed in PE_Zi to accelerate matrix and vector calculation
- Input events are processed in a **fully-pipelined** scheme without data dependency
- **Multiple PE_Zi** simultaneously back-project an event to **multiple DSI voxels**
  - Exploit parallelism
    - Operator-Level
    - Event-Level
    - DSI-Level
Pipelined Workflow

\[ CP : \text{Canonical Event Back-Projection} \]
\[ PP : \text{Proportional Event Back-Projection} \]
\[ R : \text{Volumetric Ray-Counting} \]
Pipelined Workflow

- For **normal event frames**, two modules work simultaneously in a **pipelined** manner.
- The execution time of **CP** is overlapped.

**Exploit parallelism: Event-Level**

\[ \mathcal{C}P : \text{Canonical Event Back-Projection} \]
\[ \mathcal{P}P : \text{Proportional Event Back-Projection} \]
\[ \mathcal{R} : \text{Volumetric Ray-Counting} \]
Outline

• Research Background and Motivation
• Eventor
  • Algorithm Framework
  • Software Optimizations
  • Hardware Architecture
• Evaluation
• Conclusions
Experimental Setup

• Hardware Implementation
  • Xilinx Zynq XC7Z020 SoC
  • Eventor clock 130 MHz, DDR clock 533 MHz

• Dataset
  • DAVIS Dataset: [Mueggler et al., The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and SLAM. IJRR’17.]
  • Camera resolution: 240 × 180
  • Simulated sequences: simulation_3planes, simulation_3walls
  • Real scene sequences: slider_close, slider_far

• Baseline
  • Original EMVS implementation on Intel i5-7300HQ CPU

Table: The resources utilization of Eventor

|          | Utilization  |
|----------|--------------|
| # LUT    | 17538(32.97%)|
| # FF     | 22830(21.46%)|
| BRAM     | 64KB(11.43%) |
Accuracy Analysis

The depth estimation error (AbsREL) of our reformulated hardware-friendly EMVS compared with original EMVS.

@ DAVIS Dataset

The accuracy of our reformulated framework is comparable to original EMVS!
### Accelerator Performance Evaluation

Table: Performance comparison between Eventor and original EMVS runs on Intel i5 CPU

|                                | Intel i5 CPU | Eventor |
|--------------------------------|--------------|---------|
| Runtime per Event Frame (μs / task) |              |         |
| CPU                            | 22.40        | 8.24    |
| PP & R                         | 559.55       | 551.58  |
| Runtime per Event Frame (μs / frame) |          |         |
| Normal Frame                   | 581.95       | 551.58  |
| Key Frame                      | 581.95       | 559.82  |
| Event Processing Rate (10^6 events / second) |       |         |
| Normal Frame                   | 1.76         | 1.86    |
| Key Frame                      | 1.76         | 1.83    |
| Power (W)                      | 45           | 1.86    |

*Each event frame consists of 1024 events

Eventor can achieve $24 \times$ improvement in energy efficiency compared with Intel i5 CPU!
Outline

• Research Background and Motivation
• Eventor
  • Algorithm Framework
  • Software Optimizations
  • Hardware Architecture
• Evaluation
• Conclusions
Conclusions

- An efficient EMVS accelerator, **Eventor**, is proposed for real-time applications and evaluated on Zynq FPGA platform.

- **Algorithm-hardware co-optimization** strategies are utilized to improve the system performance.

- **Eventor** could achieve \(24 \times\) improvement in **energy efficiency** compared with Intel i5 CPU.

- The overall performance could satisfy the requirements of **real-time reconstruction** on power-limited embedded platforms.
Thank You!

Q & A