ASV: Accelerated Stereo Vision System

Yu Feng
yfeng28@ur.rochester.edu
University of Rochester

Paul Whatmough
paul.whatmough@arm.com
Arm Research

Yuhao Zhu
yzhu@rochester.edu
University of Rochester

http://horizon-lab.org

Abstract

Estimating depth from stereo vision cameras, i.e., "depth from stereo", is critical to emerging intelligent applications deployed in energy- and performance-constrained devices, such as augmented reality headsets and mobile autonomous robots. While existing stereo vision systems make trade-offs between accuracy, performance and energy-efficiency, we describe ASV, an accelerated stereo vision system that simultaneously improves both performance and energy-efficiency while achieving high accuracy.

The key to ASV is to exploit unique characteristics inherent to stereo vision, and apply stereo-specific optimizations, both algorithmically and computationally. We make two contributions. Firstly, we propose a new stereo algorithm, *invariant-based stereo matching* (ISM), that achieves significant speedup while retaining high accuracy. The algorithm combines classic "hand-crafted" stereo algorithms with recent developments in Deep Neural Networks (DNNs), by leveraging the *correspondence invariant* unique to stereo vision systems. Secondly, we observe that the bottleneck of the ISM algorithm is the DNN inference, and in particular the *deconvolution* operations that introduce massive compute-inefficiencies. We propose a set of software optimizations that mitigate these inefficiencies. We show that with less than 0.5% hardware area overhead, these algorithmic and computational optimizations can be effectively integrated within a conventional DNN accelerator. Overall, ASV achieves 5× speedup and 85% energy saving with 0.02% accuracy loss compared to today's DNN-based stereo vision systems.

CCS Concepts

• **Human-centered computing** → **Mobile computing**; *Mobile devices*; • **Hardware** → **Hardware accelerators**; • **Computing methodologies** → *Computer vision tasks.*

Keywords

Stereo vision, Depth from stereo, Mobile computing, DNN accelerator, data-flow, tiling, constrained-optimization

MICRO-52, October 12–16, 2019, Columbus, OH, USA

ACM ISBN 978-1-4503-6938-1/19/10…$15.00
https://doi.org/10.1145/3352460.3358253

Artifacts

ISM Algorithm: https://github.com/horizon-research/ism-algorithm
Systolic-array Data-flow Optimizer: https://github.com/horizon-research/systolic-array-dataflow-optimizer

ACM Reference Format:
Yu Feng, Paul Whatmough, and Yuhao Zhu. 2019. ASV: Accelerated Stereo Vision System. In *The 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-52), October 12–16, 2019, Columbus, OH, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3352460.3358253

1 Introduction

The demand for intelligent applications running on a diverse range of mobile and embedded platforms, such as micro-robots, augmented reality headsets, and smart-city sensor nodes, shows no sign of slowing down. A key primitive in these applications is estimating *depth* information from the environment, which in turn serves as the building block for extracting higher-level semantics. For instance, depth information enables a mobile robot to detect and manipulate objects that are in close proximity.

Among numerous depth sensing techniques, we focus on *stereo* camera systems, which estimate depth from a pair of horizontally displaced cameras that capture two different views of the scene, mimicking the human binocular vision. Compared to other depth sensing techniques such as LiDAR and structured-light sensors [65, 67], stereo vision systems are much cheaper, consume less power, and are physically more compact [3]. In response to the rising significance of stereo vision, recent mobile vision platforms integrate specialized stereo vision accelerators, such as the Stereo Depth Block in the Movidius Enhanced Vision Accelerator Suite [5] and the Stereo & Optical Flow Engine (SOFE) in the Nvidia Xavier mobile Systems-on-a-chip (SoC) [9].

Stereo vision algorithms presented to date broadly define a frontier in the accuracy-efficiency design space. Fig. 1 compares the frame rate and accuracy for four well-known classic stereo algorithms that use "hand-crafted" features, including GCSF [15], SGBN [33], HH [33], and ELAS [26], as well as four state-of-the-art DNNs solutions [16, 37, 48, 59]. The DNN data is characterized on both a Pascal mobile GPU [8] ("-GPU" suffix), as well as on a DNN accelerator [57] ("-Acc" suffix). In using low-dimensional "hand-crafted" features, classic algorithms lead to high error rates (x-axis), but are compute efficient, mostly operating at close to real-time (e.g., 30 FPS, y-axis). In contrast, DNNs models achieve very low error rates, but require 2–5 orders of magnitude more arithmetic operations, resulting in much lower frame rates.

This paper presents ASV, an accelerated stereo vision system that operates in real-time while achieving DNN comparable accuracy.

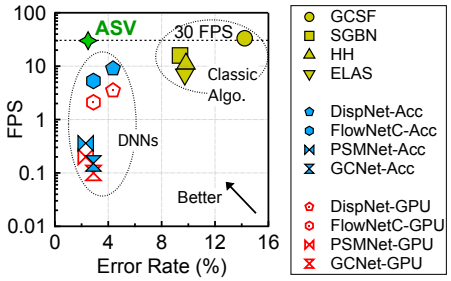


Fig. 1: ASV demonstrates both real-time (30 FPS) frame rates and DNN-like accuracy for stereo vision.

While today's vision accelerators are primarily built for monocular vision tasks, ASV exploits unique characteristics of stereo vision, and applies stereo-specific optimizations, both algorithmic and computational. Critically, we show that with careful algorithmic choices, these stereo-specific optimizations can largely be implemented on the same computer architecture as conventional DNN accelerators with a set of basic, yet principled, hardware extensions, which widens the applicability of this work.

At the core of ASV is a new low-latency, high-accuracy stereo vision algorithm. We exploit the temporal invariance introduced by stereo cameras: a single physical point projects to a unique pair of pixels on the left and right image planes; although the pixel locations move over time, their corresponding geometric relationship is fixed. Our algorithm, ISM, uses compute-intensive DNNs to extract pixel correspondences from a small set of key frames. The correspondences are then propagated as initial estimates for subsequent non-key frames, where we make use of cheaper, classic algorithms. By combining learnt features from DNNs, and classic algorithms that explicitly model the physical world, ISM achieves high accuracy while reducing the compute cost.

While our ISM algorithm reduces the compute overhead, DNNs remain critical, as they generate the initial estimate of the correspondence information. We observe that stereo DNNs make heavy use of the *deconvolution* operation[1] that exposes specific kernel sparsity, making conventional DNN accelerators inefficient. While prior work proposed specialized hardware to exploit deconvolution sparsity [60, 76], we demonstrate that static software optimizations achieve better results without unnecessary hardware modifications.

Our approach is to transform an inherently sparse deconvolution layer into a sequence of dense convolutions, which can then be executed by canonical DNN accelerators. More importantly, this transformation uniquely exposes a new data reuse opportunity: *inter-layer activation reuse* (ILAR), which does not exist in conventional DNNs. While exhaustive search has been previously used to optimize data reuse patterns, it does not scale to optimizing deconvolution, because the transformation increases the layer count by up to 8×, and ILAR adds another search dimension. Instead, we propose a constrained-optimization formulation, and demonstrate an efficient solver using dynamic programming.

We implement a software/hardware co-designed prototype of ASV. The hardware builds on top of a conventional systolic DNN accelerator [36] implemented in 16nm technology. The ASV hardware minimally extends the baseline accelerator with less than 0.5% area overhead. The software integrates the ISM algorithm and the deconvolution optimizations.

We evaluate ASV on a set of standard stereo vision benchmarks. Compared to the DNN baseline, ASV achieves 5× speedup and 85% energy saving with 0.02% accuracy loss. We also demonstrate the general applicability of software deconvolution, by applying it to Generative Adversarial Networks (GANs), which also make heavy use of deconvolutions. Under the same compute and memory resource constraints, we achieve 1.4 × speedup over a purpose-built deconvolution accelerator, due to the unique ILAR that we exploit.

To our best knowledge, this is the first paper that demonstrates a cost-effective stereo vision system. Using a software-hardware co-design approach, we show that carefully designed software optimizations achieve significant performance and energy improvements with simple, principled changes to existing DNN accelerators, which widens the applicability of our work. More specifically:

- We propose the first stereo vision algorithm, ISM, that exploits *temporal invariance* in stereo imaging to improve the performance with minimal accuracy loss;
- We propose the first *static* optimization framework for deconvolution, a key operation in stereo DNNs, which eliminates the sparsity-induced compute inefficiencies in deconvolution layers without hardware changes;
- We are the first to identify *inter-layer activation reuse* in deconvolution, *a unique data reuse opportunity* exposed by our transformation framework, and which we exploit using an efficient constrained optimizer.
- We co-design the hardware with the proposed software optimizations to achieve fast, low-power stereo vision *with minimal changes to existing DNN accelerators.*

The remainder of the paper is organized as follows. Sec. 2 gives an overview of necessary background. Sec. 3 introduces our invariant-based stereo matching algorithm. Sec. 4 describes the software optimizations for efficient implementation of the deconvolution operation. Sec. 5 presents the design of ASV, including both software and hardware considerations. Sec. 6 and Sec. 7 are experimental methodology and results, respectively. Sec. 8 positions ASV in the context of related work, and Sec. 9 concludes the paper.

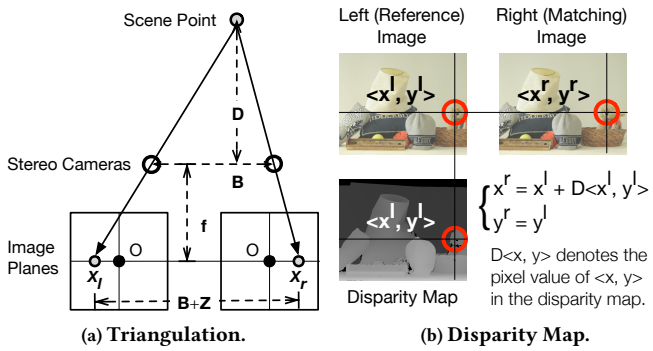
2 Background

We first describe the scope of our work: vision-based systems that extract 3D information from 2D stereo images (Sec. 2.1). We then introduce the necessary background of stereo vision algorithms, including both classic hand-crafted algorithms and contemporary stereo DNNs (Sec. 2.2).

2.1 Depth Sensing

There are two essential methods to extract depth information: passive sensing and active sensing. Passive sensing techniques observe the environment, primarily through cameras, and infer depth using computer vision algorithms. In contrast, active sensing techniques transmit signals and analyze the response to calculate depth; examples include structured light [65] and LiDAR [67].

[1] Deconvolution in deep learning is an incredibly unfortunate misnomer that should really be called "transposed convolution."



(a) Triangulation. (b) Disparity Map.

Fig. 2: "Depth from stereo" illustration: given an image pair, *stereo matching* algorithms first generate the disparity map (b), from which depth is then calculated through *triangulation* (a). Triangulation is computationally trivial; this paper focuses on optimizing stereo matching algorithms.

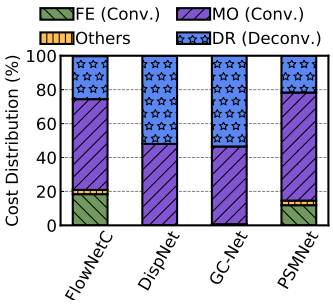


Fig. 3: The arithmetic operation distribution of stereo matching DNNs.

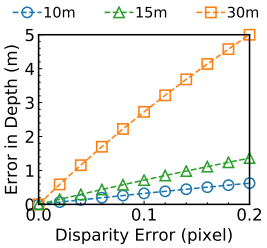


Fig. 4: Depth estimation accuracy is sensitive to stereo matching accuracy.

This paper focuses on camera-based passive sensing. Compared to alternatives such as LiDAR, cameras are much cheaper and less bulky [3]. As a result, camera-based depth sensing is widely adopted in systems such as autonomous vehicles and AR headsets. According to Allied Market Research, the adoption of stereo cameras is expected to grow 60.4% by 2020 [1]. The recent industry trend of integrating dedicated stereo vision accelerators into mobile SoCs (e.g., Movidius [5] and Nvidia [9]) further underlines the significance of stereo vision for depth sensing.

2.2 Depth From Stereo

Triangulation The key idea behind stereo depth estimation is that a single physical scene point projects to a unique pair of pixels, via two observing cameras; the horizontal displacement between the two pixels captured on the left and right image planes is inversely proportional to the distance of the point from the observer (i.e., the depth). Fig. 2a illustrates this process, where the scene point is captured at position x^l and x^r on the left and right image planes, respectively. Using similar triangles, the depth D is calculated by:

$$D = Bf/Z, \tag{1}$$

where f is the focal length of the cameras, B is the distance between the two camera lenses, and Z is the *disparity* $x^r - x^l$, i.e., the horizontal displacement between the two corresponding pixels in the left and right images. This process is widely known as *triangulation* [30, 61].

Stereo Matching and Disparity Map Since both B and f are camera intrinsic parameters, the key to triangulation is to calculate the disparity Z. Given the left (reference) image and the right (matching) image, we must find the pixels in each image that are the projections of the same physical point, a process also known as *stereo matching*. In the end, stereo matching generates a "disparity map", whose $<x, y>$ coordinates are taken to be coincident with the pixel coordinates of the reference image. Fig. 2b shows one such example, in which the correspondence between a pixel $<x^l, y^l>$ in the left image and a pixel $<x^r, y^r>$ in the right image is given by:

$$x^r = x^l + D^{<x^l, y^l>}, \; y^r = y^l, \tag{2}$$

where $D<x^l, y^l>$ denotes the pixel value at $<x^l, y^l>$ in the disparity map. Note that the compute cost of triangulation is trivial (Equ. 1), and thus we focus on stereo matching.

Stereo matching algorithms consist of three stages [13, 58]: Feature Extraction (FE), Matching Optimization (MO), and Disparity Refinement (DR). Both conventional algorithms [13, 33, 58, 63] and DNN-based algorithms [16, 37, 48, 59, 63] follow this processing pipeline, but differ in their implementations. Conventional methods extract hand-crafted features (e.g., SIFT [43], HOG [20], or plain pixel values) in the FE stage, search the feature space of both images to find the matching pixels in the MO stage, and improve the disparity resolution in the DR stage using techniques such as iterative gradient descent [62]. DNNs, in contrast, implement each stage using a set of learnt parameters.

Deconvolution in Stereo DNNs Stereo matching DNNs implement FE and MO stages as convolution layers. The DR stage fundamentally requires *deconvolution* (a.k.a. transposed convolution) layers [53]. Deconvolution layers generate large activation maps from small input feature maps, essentially up-sampling the input. The up-sampling in DR is critical to compensate the down-sampling in FE and MO that scale down the input images to extract high-level features.

To illustrate the importance of deconvolution in stereo vision, Fig. 3 shows the time distribution of four state-of-the-art stereo matching DNNs across the three stages. The convolution and deconvolution layers combined account for over 99% of the execution time, from which 38.2% is attributed to the deconvolution layers.

High Accuracy Stereo Matching Stereo matching is critical because it generates the disparity, from which depth is estimated (Equ. 1). Using the industry-standard Bumblebee2 stereo camera [2] as an example (B is 120 mm, f is 2.5 mm, and pixel size is 7.4 µm), Fig. 4 shows how the depth estimation error (y-axis) varies with the disparity error in pixels (x-axis). Different curves correspond to objects at different distances. We find that even two tenths of a pixel error in stereo matching can result in a depth estimation error of 0.5m–5m, which could be catastrophic at the application level.

While existing stereo matching systems achieve high accuracy at the expense of high compute cost, ASV achieves DNN-level accuracy with significantly less compute.

3 Invariant-based Stereo Matching

This section introduces our new *invariant-based stereo matching algorithm* (ISM). The key idea of ISM is to exploit the *correspondence invariant* between the stereo images over time. After introducing the high-level concept (Sec. 3.1), we then describe the detailed algorithm (Sec. 3.2), and discuss important algorithmic design decisions (Sec. 3.3). We make the implementation of ISM available at: https://github.com/horizon-research/ism-algorithm.

3.1 Overview

Stereo matching produces a disparity map (Fig. 2b), from which depth information is easily obtained through triangulation (Fig. 2a). Classic stereo matching algorithms generate the disparity map by matching pixels/features in the left (reference) frame with pixels/features in the right (matching) frame, typically by searching in a finite window. However, the accuracy of search-based algorithms is sensitive to the heuristics used in the search, such as feature selection, search window size, matching criterion, etc. In contrast, DNN approaches largely avoid heuristics and instead directly learn the matching pairs. Unfortunately, DNNs come at the cost of a massive increase in compute requirement.

Instead of the binary choice between DNNs and conventional search-based algorithms, we use DNNs to *guide* the search process of classic methods. The key observation is that two matched pixels, one from the left image and the other from the right image, correspond to the same point in the physical world. While the locations of the two pixels move from frame to frame, they are always projections of the same scene point, and therefore are always a matched pair in any frame. In other words, the geometric correspondence relationship between two matched pixels is invariant.

Our new stereo matching algorithm, ISM, exploits this *correspondence invariant* by operating in two modes. It obtains stereo correspondences on "key frames" through accurate but compute-intensive DNNs. The correspondences are then propagated to subsequent non-key frames as good initial guesses to guide the cheaper search-based methods. By combining learnt correspondences with search-based methods that explicitly model the physical world, ISM reduces the total compute cost while retaining DNN-like accuracy.

3.2 Algorithm

We illustrate ISM in Fig. 5. ISM consists of four main components. ISM runs DNN inferences (❶) on key frames to obtain pixel correspondences, which are used to guide feature matching on non-key frames (❷, ❸, and ❹).

❶ **DNN Inference** Assuming the left and right frames at timestep t are regarded as key frames, ISM performs DNN inference to generate a disparity map for the left image, in which each pixel value represents the disparity (i.e., Z in Fig. 2a) of each pixel in the left frame. In conventional DNN approaches, this disparity map is used only for triangulation (not shown in the figure) to estimate depth, and is discarded after the depth map is generated.

❷ **Reconstruct Correspondences** Instead of discarding the disparity map, ISM uses it to identify the correspondences in the left and right frames. As per the definition of disparity (Equ. 2), every $<x_t, y_t>$ pixel in the disparity map with the value $D_t^{<x,y>}$ indicates that the $<x_t, y_t>$ pixel in the left frame (P_t^L) and the $<x_t +$

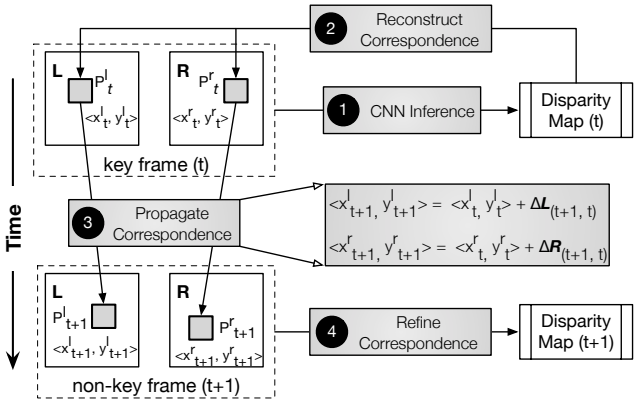


Fig. 5: The ISM algorithm obtains correspondences in key frames using DNNs, and propagates the correspondences to non-key frames to guide the cheap correspondence search. Time progresses from top to bottom in the figure.

$D_t^{<x,y>}, y_t>$ pixel in the right frame (P_t^R) form a correspondence pair. By iterating through all the pixels in the disparity map, ISM identifies all the correspondence pairs in the left and right frames at timestep t.

❸ **Propagate Correspondences** A new pair of frames arrives at the next timestep $(t+1)$. ISM exploits a well-known observation that pixels in consecutive video frames are highly-correlated in time. For instance, P_t^L has moved to P_{t+1}^L, and P_t^R has moved to P_{t+1}^R. Critically, since P_t^L and P_t^R are a correspondence pair projected from a scene point, P_{t+1}^L and P_{t+1}^R must correspond to the same point, and hence highly likely to also be a correspondence pair at timestep $(t+1)$.

The exact coordinates of P_{t+1}^L and P_{t+1}^R can be obtained through a *motion estimation* (ME) algorithm. For each pixel in the left (right) frame, the ME algorithm generates a motion vector $\Delta P_{(t+1,t)}^L$ ($\Delta P_{(t+1,t)}^R$), representing the displacement between the pixel in frame t and frame $(t+1)$. Thus:

$$P_{t+1}^L = P_t^L + \Delta P_{(t+1,t)}^L$$

$$P_{t+1}^R = P_{t+1}^R + \Delta P_{(t+1,t)}^R$$

❹ **Refine Correspondences** Given the correspondence pairs (e.g., P_{t+1}^L and P_{t+1}^R) at timestep $(t+1)$, ISM then calculates the disparity map at $(t+1)$. If the motion estimation from t to $(t+1)$ is precise, the propagated correspondence pairs at $(t+1)$ are also precise. Accordingly, the disparity map could be simply obtained by calculating the horizontal offsets between all the correspondence pairs. For instance, given the correspondence pair P_{t+1}^L and P_{t+1}^R, the disparity at $<x_{t+1}^l, y_{t+1}^l>$ in the disparity map would be $x_{t+1}^r - x_{t+1}^l$.

In reality, motion estimation is imperfect due to various visual artifacts such as occlusion and fast motion [42]. Thus, the correspondences propagated from t are a noisy estimate of the true correspondences at $(t+1)$. To further refine the estimate of $(t+1)$ in ISM, we use classic *correspondence search*, and initializes the search window with the propagated correspondences. This allows ISM to avoid compute-intensive DNNs on non-key frames without sacrificing accuracy.

3.3 Algorithmic Design Decisions

Computing non-key frames requires reconstructing, propagating, and refining correspondences. Reconstructing correspondences has little overhead. The cost of propagating correspondences is dominated by motion estimation, and the cost of refining correspondences is dominated by the correspondence search. Thus, we must carefully choose the motion estimation and correspondence search algorithms such that the compute cost is much lower than DNNs with little accuracy loss. We discuss algorithmic choices below.

Motion Estimation The literature is rich with motion estimation algorithms, which differ in the coverage and densities of estimated motion. The disparity map in stereo matching should ideally be calculated on a per-pixel basis across the frame, so as to enable fine-grained depth estimation. This requirement rules out many classic motion estimation algorithms such as block matching (BM) [35], and sparse optical flow [34, 45]. BM estimates motion at the granularity of a block of pixels, and thus does not provide the pixel-level motion that stereo vision requires. Sparse optical flow algorithms such as Lucas-Kanade [45] and Horn-Schunck [34] only provide pixel-level motion for feature points such as corners, and do not cover all the frame pixels.

Instead, we use a *dense optical flow* algorithm, specifically the Farneback algorithm [21, 22], for motion estimation. Farneback generates per-pixel motion for all the pixels, and is computationally efficient. 99% of the compute in Farneback is due to three operations: Gaussian blur, "Compute Flow", and "Matrix Update". Gaussian blur is inherently a convolution operation that convolves a Guassian kernel (2D matrix) with the image. The latter two are point-wise operations that resemble the activation function in DNNs. Thus, motion estimation in the ISM algorithm can be computed using a DNN accelerator to simplify the hardware design.

Correspondence Search ISM performs correspondence search to refine the initial correspondence estimation propagated through motion. Correspondence search algorithms have been well-studied in the classic computer vision literature [13, 58], and generally fall into two categories: local methods and global methods. At the cost of higher compute demand, global methods provide higher accuracy by minimizing the pixel motion inconsistencies across the entire image. However, with the initial correspondences propagated through key-frames, we find that local methods suffice.

In particular, we leverage the block matching algorithm [35] for local correspondence search. For each pixel in the left image (e.g., P^L_{t+1} in Fig. 5), ISM uses the block of pixels surrounding it to search in a 1D window in the right image in order to find the closest match. The search window is centered around the initial correspondence estimation (e.g., P^R_{t+1} in Fig. 5). We use the sum of absolute differences (SAD) cost function. The horizontal offset between the two matched blocks is the disparity for P^L_{t+1}.

Similar to optical flow, the block matching algorithm has a "convolution-like" structure [55]; the block in the left image is equivalent to a kernel, and the search window in the right image is equivalent to the input image. The only difference is that block matching computes the SAD between the input feature map and the kernel ($\sum_{i=1}^{N} |a_i - b_i|$) as opposed to the dot product in canonical convolution ($\sum_{i=1}^{N} a_i b_i$). Thus, the correspondence search can share the same architecture as DNNs and optical flow.

Compute Cost Due to our algorithmic choices, computation on non-key frames is much cheaper than key-frames. For instance, for a typical qHD frame (960 × 540), computating a non-key frame requires about 87 million operations while stereo DNN inference (key frame) requires about $10^2\times$–$10^4\times$ more arithmetic operations. Thus, ISM leads to significant performance and energy improvements by avoiding DNN inference altogether in non-key frames.

4 Deconvolution Optimizations

While the ISM algorithm removes DNN inference in non-key frames, DNNs remain critical for generating initial key frame correspondences. This section describes optimizations for stereo DNNs, in particular the dominant deconvolution layers. We propose novel software-only optimizations that mitigate the compute overheads in deconvolution (Sec. 4.1), while capturing unique data reuse opportunities (Sec. 4.2).

We make our optimization framework publicly available at: https://github.com/horizon-research/systolic-array-dataflow-optimizer. It targets the systolic-array accelerator architecture, supports the deconvolution optimization described here, and applies tiling optimizations to minimize the inference latency and/or DRAM traffic.

4.1 Deconvolution Transformation

Deconvolution layers on average contribute to 38.2% (50% max) of the total MACs in stereo DNNs (Fig. 3). Due to the inherent sparsity of deconvolution, a naive mapping to hardware results in over 75% of redundant computations due to one or more zero operands. Deconvolution is also used in Generative Adversarial Networks (GANs), and recent studies have proposed specialized hardware specifically for deconvolution [60, 76]. In contrast to previous studies, we propose a *purely algorithmic transformation* that eliminates inefficiencies due to sparsity. We show that an inherently sparse deconvolution layer can be translated to a series of dense convolutions, which then effectively map on to existing DNN accelerators. We next explain the inefficiencies in deconvolution, and then describe our algorithmic transformations.

Standard Deconvolution The standard process (Fig. 6) deconvolves a 3x3 input feature map (*ifmap*) with a 3x3 kernel. The *ifmap* is first upsampled with zero padding, before being convolved with the 3x3 kernel to generate an output feature map (*ofmap*). Note that the upsampling step essentially performs disparity refinement, which is fundamental to general stereo DNNs, rather than being specific to a particular network (Sec. 2.2). The zeros in the upsampled *ifmap* leads to redundant computation and memory traffic.

A key characteristic of deconvolution is that different elements in the *ofmap* are calculated in different "patterns." Consider the first 2×2 outputs in the *ofmap*: (1, 1), (1, 2), (2, 1), and (2, 2). Each of the four outputs is generated using a different set of elements from the kernel. For instance, (1, 1) requires only e while (1, 2) requires d and f. Critically, there are only four different patterns, which are repeated across the *ofmap*. Pixels (4, 4) and (2, 2) are calculated using the same elements from the *kernel*, as are (1, 1) and (5, 5), (1, 2) and (5, 4), as well as (2, 1) and (4, 5). Due to the various patterns needed to generate different output elements, deconvolution is clearly an "irregular" operation. Prior work [76] exploits the four

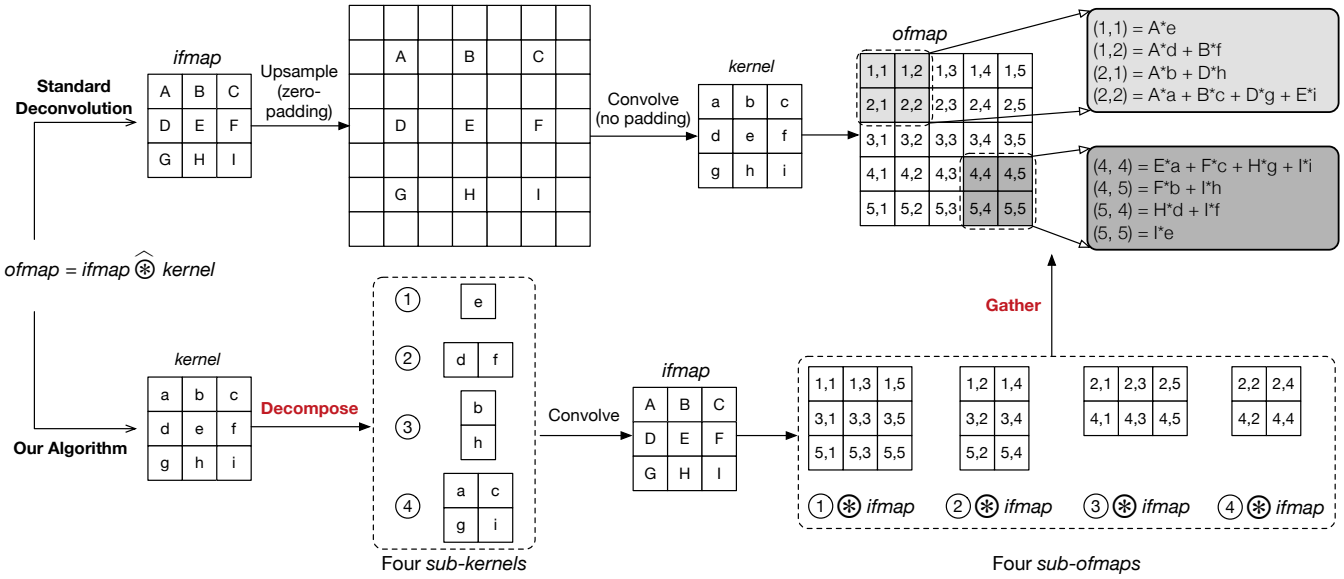


Fig. 6: Translating deconvolution into multiple convolutions. Standard deconvolution first upsamples the *ifmap* before convolving with the kernel. Note that this example assumes the upsampled *ifmap* is not further padded before the convolution, i.e., a 7×7 *ifmap* results in a 5×5 *ofmap*. Our translation algorithm holds regardless of padding.

unique computation patterns by augmenting a conventional DNN accelerator with custom hardware units.

Deconvolution Transformation In contrast, we find that existing DNN accelerators already provide the necessary architectural substrate to efficiently execute the four different patterns. The key is to recognize that the four computation patterns are essentially four different convolutions, each convolving the original *ifmap* with a distinct kernel that is part of the original kernel. For instance, (2, 2), (2, 4), (4, 2), and (4, 4) are generated by convolving $\left[\begin{smallmatrix} a & c \\ g & i \end{smallmatrix}\right]$ with *ifmap*. More generally, the deconvolution in Fig. 6 is calculated as:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \hat{\circledast} I = \mathcal{G}(\begin{bmatrix} e \end{bmatrix} \circledast I, \begin{bmatrix} d & f \end{bmatrix} \circledast I, \begin{bmatrix} b \\ h \end{bmatrix} \circledast I, \begin{bmatrix} a & c \\ g & i \end{bmatrix} \circledast I)$$

where $\hat{\circledast}$ denotes the deconvolution operation, $\circledast$ denotes the standard convolution operation, I is the *ifmap*, and $\mathcal{G}$ is a gather operation to assemble the *ofmap* from the results of the four convolutions. $\mathcal{G}$ is simply implemented as a set of load operations to the on-chip buffer. Essentially, our algorithm decomposes the original 3×3 kernel into four sub-kernels, each requiring a smaller dense convolution with the original *ifmap*, which can be executed efficiently on a conventional DNN accelerator.

This transformation generalizes to kernel shapes other than 3×3. Formally, a 2D kernel K with a dimension $K_H \times K_W$ will be decomposed into four sub-kernels (S_0, S_1, S_2, S_3):

$$S_0^{(i,j)} = K^{(2i,2j)} \; i \in [0, \lceil K_H/2 \rceil), j \in [0, \lceil K_W/2 \rceil)$$
$$S_1^{(i,j)} = K^{(2i+1,2j)} \; i \in [0, \lfloor K_H/2 \rfloor), j \in [0, \lceil K_W/2 \rceil)$$
$$S_2^{(i,j)} = K^{(2i,2j+1)} \; i \in [0, \lceil K_H/2 \rceil), j \in [0, \lfloor K_W/2 \rfloor)$$
$$S_3^{(i,j)} = K^{(2i+1,2j+1)} \; i \in [0, \lfloor K_H/2 \rfloor), j \in [0, \lfloor K_W/2 \rfloor)$$

where $S_*^{(i,j)}$ is the element (i, j) in a particular sub-kernel, and $K^{(*,*)}$ is an element in the original kernel K. For instance, $S_0^{(i,j)} = K^{(2i,2j)}$ means that element (i, j) in the first sub-kernel comes from element $(2i, 2j)$ in the original kernel. The boundary condition of each case denotes the dimension of the corresponding sub-kernel (notice the different floor and ceiling functions in each). Hence, decomposing a 3×3 kernel results in four sub-kernels of shapes 2×2, 1×2, 2×1, and 1×1, confirming the specific example above. The general formulation of the deconvolution transformation with an arbitrary N-dimensional kernel is described in Appendix A.

4.2 Exploiting Inter-Layer Activation Reuse

A beneficial trait of our transformation is that each sub-convolution reads the same *ifmap*, which in modern DNNs does not fit in on-chip buffers and must spill to main memory. In contrast, our transformation can uniquely exploit *inter-layer activation reuse* because each sub-convolution layer shares the same *ifmap*. The challenge is to systematically maximize the reuse exploited across the entire network while minimizing the inference latency.

We primarily consider *loop tiling*, which is known to be critical to exploiting data reuse in DNNs [52, 74]. Prior work in DNN tiling predominately searches for the tiling strategy in a brute-force manner [32, 46]. However, brute-force search does not scale to stereo DNNs for two reasons. First, our translation scheme significantly increases the number of layers, each of which must be individually searched. For instance in the example of Fig. 6, the number of layers quadruples; a 3D kernel could increase layers by 8×. Second, exploiting the inter-layer *ifmap* reuse adds another scheduling dimension, further increasing the search space.

Instead of a search, we formulate the reuse optimization as a constrained optimization problem, minimizing layer latency while

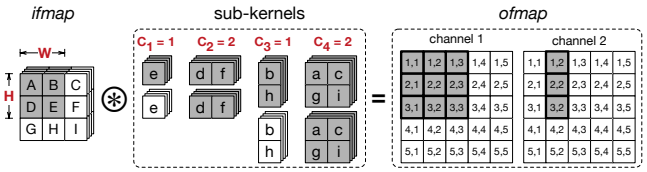


Fig. 7: Tiling in a translated deconvolution with a 3×3 kernel split into four sub-kernels. With a tiling strategy $W = 2, H = 2, C_1 = 1, C_2 = 2, C_3 = 1, C_4 = 1$, only the shaded elements are loaded into the buffer. The *ofmap* elements generated in this round (shaded) are also stored in the buffer.

satisfying hardware resource constraints. Our optimization can be efficiently solved using a greedy algorithm.

Architectural Assumptions We first describe the underlying architecture that the optimization formulation assumes. Overall, we make standard assumptions that generally hold across the vast majority of current DNN accelerators. Sec. 5.2 describes the hardware architecture in detail.

We assume a systolic array accelerator. Each Processing Element (PE) performs one MAC operation per cycle [36, 57]. Systolic arrays use a very efficient neighbor-to-neighbor communication mechanism, particularly well suited to convolution. Alternatively, our formulation could also be extended to support spatial arrays [18], which offer more flexible control at higher hardware cost.

We assume that the accelerator has a unified on-chip buffer (scratchpad) for the *ifmap*, kernels, and *ofmap*. This buffer is generally too small to hold all the data for a whole layer. Therefore, the *ofmap* is computed in multiple rounds. Only part of the *ifmap* and the kernels are stored in the buffer each round. The optimal scheduling of partial *ifmap* and kernels in the buffer for each round is critical to maximizing reuse.

The buffer is evenly split into working and filling sections for double-buffering. While the PE array is computing the current round using data in the working buffer, the data for the next round is pre-fetched to the filling buffer. The next round starts only when the filling buffer is full. This design choice guarantees that any data access by the PEs will hit in the buffer without stalling the PE array.

Optimization Formulation We follow a layer-wise execution model, in which a layer only starts after the previous layer finishes. Therefore, minimizing the total latency is equivalent to minimizing the latency of each individual layer. We describe how the latency of a deconvolution layer is formulated and optimized. Our formulation can be easily extended to support a convolution layer, which can be regarded as a special case of deconvolution without `ILAR`.

The optimization objective is to minimize the deconvolution layer's latency given hardware resource constraints. Note that since a deconvolution is translated to a set of convolutions, it is the cumulative latency of these sub-convolutions that is of interest. The optimization problem is formulated as follows:

$$\min \; L(\Theta, \phi) \tag{3}$$

$$s.t. \; R(\Theta) \leq R^* \tag{4}$$

where Θ denotes a particular hardware configuration, and $R(\cdot)$ is the configuration's hardware resources, which must not exceed the specified resource budget R^*. We consider three main types of hardware resources: 1) PE array size, 2) on-chip buffer size, and 3) off-chip memory bandwidth.

Latency $L(\cdot)$ is affected by both the hardware configuration (Θ) and the tiling schedule (ϕ). The optimal tiling is determined by the following variables: 1) the dimension of the *ifmap* tile to be loaded into the buffer (W and H), and 2) the number of filters in each **sub-kernel** k to be loaded into the buffer (C_k). Critically, C_k can be different for each sub-kernel. Fig. 7 illustrates these optimization variables, with an example where part of the *ifmap* is convolved with certain filters of the four sub-kernels to generate a partial *ofmap*. The vector $\overrightarrow{C}$ denotes the collection of all C_k.

With double buffering, a layer L's latency is the cumulative latency across all N rounds. The latency of each round (l^i) is determined by the maximum value between the memory access time (l^i_m) and the compute time (l^i_c) of the round:

$$L(\Theta, \phi) = \sum_{i=1}^{N} l^i(\Theta, \phi), \; l^i(\Theta, \phi) = \max(l^i_c, l^i_m) \tag{5}$$

With double-buffering, l^i_c is determined by two sets of parameters: 1) W^i, H^i, and $\overrightarrow{C^i}$, which decide the total compute demand, and 2) the PE array size, A^*, which decides the compute capability. l^i_c is the cumulative latency of processing each individual sub-kernel:

$$l^i_c = \sum_{k=1}^{|\overrightarrow{C^i}|} \left\lceil \frac{W^i_k \times H^i_k \times I \times C^i_k \times H^i \times W^i}{A^*} \right\rceil \tag{6}$$

where $|\overrightarrow{C^i}|$ denotes the total number of sub-kernels in round i, W^i and H^i are the dimensions of the *ifmap* tile loaded into the buffer in round i, W^i_k and H^i_k are the dimensions of sub-kernel k in round i[2], C^i_k denotes the number of filters in sub-kernel k loaded into the buffer in round i, and I is the number of input channels. The ceil operator indicates that the next sub-kernel can not start until the previous sub-kernel is finished even if the PE array is underutilized. This is because only one sub-kernel can be calculated on the systolic array at a time as sub-kernels vary in their shapes.

The memory access time, l^i_m, is determined by the available memory bandwidth, B^*, and the amount of data that needs to be transferred to/from DRAM each round, which in turn depends on the reuse order: whether the *ifmap* tile or the sub-kernels remain in the buffer across consecutive rounds. A binary variable β denotes this reuse order, and l^i_m becomes:

$$l^i_m = \beta \times l^i_{m:W} + (1 - \beta) \times l^i_{m:In}, \; \beta \in \{0, 1\} \tag{7}$$

where $l^i_{m:In}$ is the memory access latency if the *ifmap* remains in the buffer, and $l^i_{m:W}$ denotes the memory latency if the sub-kernels remain in the buffer. Specifically:

$$l^i_{m:W} = (\Delta IF^i + \sum_{k=1}^{|\overrightarrow{C^i}|} \Delta OF^i_k) \times \frac{1}{B^*} \tag{8}$$

$$l^i_{m:In} = \sum_{k=1}^{|\overrightarrow{C^i}|} (\Delta W^i_k + \Delta OF^i_k) \times \frac{1}{B^*} \tag{9}$$

[2]The sub-kernels' dimensions do not change across rounds. Given a k, W^i_k and H^i_k are constants for any i. For the consistency of the notations, we still use W^i_k and H^i_k.

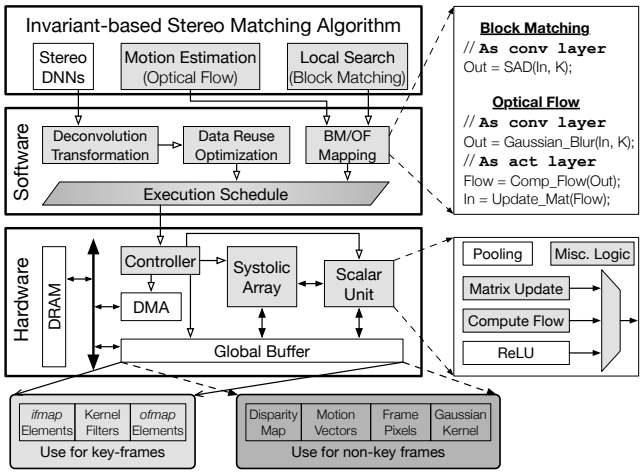


Fig. 8: The ASV overview with augmentations shaded.

where the terms with prefix Δ denote the amount of data that needs to be loaded from DRAM. Depending on the reuse order, either the *ifmap* elements (ΔIF^i), or the sub-kernels (ΔW_k^i) are loaded. The newly computed *ofmap* elements (ΔOF_k^i) are always stored back to DRAM. Note that ΔW_k^i, ΔIF^i, and ΔOF_k^i are all deterministic functions of W_k^i, H_k^i, W^i, H^i, and $|\overrightarrow{C^i}|$. We omit them here for brevity, and describe their exact expressions in Appendix B.

The on-chip buffer capacity (Buf^*) imposes the constraint:

$$\Delta IF^i + \sum_{k=0}^{|\overrightarrow{C^i}|} (\Delta OF_k^i + \Delta W_k^i) \leq Buf^* \tag{10}$$

Finally, C_k^i and N must satisfy:

$$\forall k \in \{1, 2, ..., |\overrightarrow{C}|\}, \mathbb{C} = \sum_{i=1}^{N} C_k^i \tag{11}$$

where $\mathbb{C}$ denotes the number of output channels of a layer, which is a constant invariant to k and i.

Overall, this formulation minimizes the latency L with respect to W^i, H^i, and C_k^i ($i \in \{1, 2, ..., N\}$, $k \in \{1, 2, ..., |\overrightarrow{C}|\}$), under the hardware resource constraints A^*, B^*, and Buf^*.

Efficient Solver The above constrained-optimization problem has non-convex objective and constraints, and thus has no closed-form solutions. To derive a solution efficiently, we convert this problem to a Knapsack-like structure, where each filter in each sub-kernel is an *item*, the size of each filter is the *weight*, and the number of MAC operations associated with each filter is the *value*.

To solve the Knapsack problem, we use a simple greedy heuristic that prioritizes filters from large sub-kernels with standard dynamic programming. In contrast to the classic 0/1 Knapsack problem, our problem formulation requires us to consume all the items, since all the filters in each sub-kernel are required to finish a convolution. We therefore iteratively apply the greedy solver until all the items are used. The solver is executed offline, and finishes within one second on an Intel Core i5-7500 CPU.

5 The ASV System

Building on top of the ISM algorithm and the deconvolution optimizations, this section presents the software and hardware system of ASV. Fig. 8 gives a high-level system overview of ASV. We first present the software system (Sec. 5.1), and then discuss the architecture design decisions (Sec. 5.2).

5.1 Software System

The goal of the software system in ASV is to map the ISM algorithm to the underlying hardware. The static mapping is done offline. There are three components in the ISM algorithm to map: stereo matching DNN, motion estimation, and local correspondence search. We rely on the user to supply a particular stereo DNN depending on their accuracy needs. Motion estimation and correspondence search are implemented using optical flow (OF) and block matching (BM), respectively, as described in Sec. 3.3. We now describe how each component is processed by the software.

Mapping Stereo Matching DNN For the deconvolution layer, the ASV software performs the deconvolution transformation, as well as the data reuse optimization. For convolution layers, while the deconvolution transformation does not apply, we apply the data reuse optimization *without* ILAR. In the end, we obtain a transformed stereo DNN along with an execution schedule, which are both consumed by the hardware at runtime. The schedule includes the tiling strategy and buffer partitioning strategy for each layer.

Mapping OF/BM The software maps the OF and BM algorithms in ISM to a set of convolution and/or activation operations that are directly interfaced with conventional DNN accelerators. The software translates the BM operation to a convolution layer (Sec. 3.3), but calculating SAD instead of dot product at each window.

The OF computations include Gaussian blur, "Compute Flow" and "Matrix Update" operations (Sec. 3.3), shown in the top-right box in Fig. 8. Gaussian blur is naturally expressed as a convolution layer with one output channel. "Compute Flow" and "Matrix Update" are point-wise operations expressed as special activation functions.

5.2 Hardware Architecture

Leveraging the software pass, the hardware requires only minimal, structured augmentations on top of a conventional DNN accelerator. We start from a baseline DNN accelerator and describe how the compute, memory, and control logic is augmented with ASV-specific architectural extensions.

Compute Our baseline DNN accelerator consists of a TPU-like systolic PE array for convolution and a scalar unit for non-convolution operations, e.g., activation [36]. Each PE consists of two 16-bit input registers, a 16-bit fixed-point MAC unit with a 32-bit accumulator register, and simple trivial control logic. This is identical to the PE in the TPU [36].

We use the systolic array as the baseline due to its efficiency in handling convolutions. However, our software optimizations do not depend on a particular baseline DNN architecture. Alternatives such as more flexible spatial architectures [17, 18] are also suitable, albeit requiring different constrained-optimization formulations to those presented in Sec. 4.2. We will later demonstrate the effectiveness of our deconvolution optimizations on Eyeriss [18].

ASV augments both the systolic array and the scalar unit in the baseline architecture to support the ISM algorithm. First, each PE is extended with the capability to accumulate absolute differences (i.e., $a \leftarrow a + |b - c|$) in addition to MAC in order to support BM. Second, we extend the scalar unit to support two additional point-wise operations: "Compute Flow" and "Matrix Update"; both are required by OF (as illustrated in the bottom-right box in Fig. 8).

Finally, the hardware includes a very small amount of additional logic to support the remaining operations in the ISM algorithm that are inefficient to map to either the systolic array or the point-wise scalar unit. These operations are comparisons and control-flow, and are orders of magnitude less costly in area and power compared to the systolic array and the scalar unit. For instance, BM requires comparing the SAD values across different matched blocks, and OF requires checking the value boundaries during "Matrix Update".

Memory ASV uses the familiar three-level memory hierarchy [41]. Each PE has a small register file to exploit intra/inter-PE data reuse. A DMA engine coordinates data transfer between the on-chip global buffer and off-chip memory. The global buffer is temporally-shared between key frames and non-key frames. When processing key frames, the global buffer holds the ifmap, kernels, and ofmaps. The exact buffer partitioning is dictated by the ASV software.

When processing non-key frames, the global buffer holds four pieces of data: the pixels of the current and key frames, the Gaussian kernel, the motion vectors, and the disparity maps. The frame pixels dominate the storage requirement, but could be tiled because they are used in Gaussian blur and BM, both of which are convolution operations. The rest of the data cannot be tiled, and thus imposes a minimum buffer size. Assuming qHD resolution (960×540), we enforce a minimum buffer size of about 512 KB.

Control A micro-sequencer is used to coordinate the computation and memory accesses. In ASV, the sequencer also chooses key frames. Although complex adaptive schemes are feasible [14, 78], we found that a simple strategy to statically set the key-frame window suffices (Sec. 7.2).

6 Evaluation Methodology

This section introduces the basic hardware and software setup (Sec. 6.1), and outlines the evaluation plan (Sec. 6.2).

6.1 Basic Setup

Hardware Implementation We develop validated RTL implementations for the ASV hardware. The hardware is based on a systolic array architecture, consisting of 24×24 PEs clocked at 1 GHz. Each PE is capable of performing both the MAC and absolute difference operations. The hardware also has a scalar unit clocked at 250 MHz, which consists of 8 parallel lanes, each capable of performing the ReLU activation function as well as the point-wise matrix update and compute flow operations required by OF. The on-chip buffer (SRAM) is 1.5 MB in size and is banked at a 128 KB granularity. While we primarily evaluate ASV using this configuration, we will later show sensitivity of ASV performance to different hardware resource configurations.

The RTL is implemented using Synposys synthesis and Cadence layout tools in TSMC 16nm FinFET technology, with SRAMs generated by an ARM compiler. Power is simulated using Synopsys PrimeTimePX, with full annotated switching activity. The off-chip DRAM is modeled after four Micron 16 Gb LPDDR3-1600 channels [7]. Overall, the accelerator layout has a total area of 3.0 mm^2, and produces a raw throughput of 1.152 Tera operations per second.

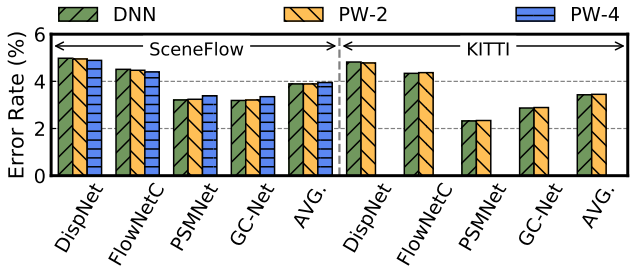


Fig. 9: Error rate comparison between the ISM algorithm in ASV and the DNN baselines.

Stereo DNNs The ISM algorithm can use an arbitrary stereo DNN. We evaluate four state-of-the-art DNNs: FlowNetC [23], DispNet [48], GC-Net [59], and PSMNet [16], with varying accuracy–performance trade-offs (Fig. 1).

Dataset We evaluate ASV on two widely-used datasets: SceneFlow [48] and KITTI [50]. SceneFlow contains 26 pairs of synthetic stereo videos to mimic various scenarios with different depth ranges. KITTI contains 200 pairs of stereo frames captured from real street views that cover varying driving scenarios and conditions.

We use the standard "three-pixel-error" accuracy metric [6, 50], which considers a pixel's depth to be correct if its disparity error is less than 3 pixels compared to ground truth. We then report the percentage of correct pixels, following the convention in the vision and robotics literature [16, 37, 48, 59].

6.2 Evaluation Plan

Our goal is to demonstrate the effectiveness of ASV over generic CNN accelerators that are not optimized for stereo vision workloads. We separate the efficiency gains of the new ISM algorithm from that of the deconvolution optimizations.

Baselines Our baseline is a generic systolic array CNN accelerator, which executes stereo DNNs without any ASV optimizations. Today's CNN accelerators mostly statically partition the on-chip buffer across *ifmap*, weights, and *ofmap*. To obtain a strong baseline, we determine the partitioning strategy by exhaustively searching all the partitions offline and use the one that achieves the lowest latency for the entire DNN. Note that the same partition is used for all the layers whereas our data reuse optimization generates different partitions for different layers.

We also compare against Eyeriss [18], a DNN accelerator based on a more flexible spatial architecture. Eyeriss performance and energy are obtained using the public simulator [4, 24]. For a fair comparison, we configure Eyeriss to have the same PE counts, on-chip memory capacity, and memory bandwidth as ASV. Finally, to establish a baseline, we also show the results of the Pascal mobile GPU found in the 16 nm Nvidia Parker SoC hosted on the Jetson TX2 development board [8]. We use the built-in power sensing circuity to obtain the energy consumption.

ASV Variants We present an ablation study on ASV to separate the gains from different optimizations:

- ISM: ISM algorithm without deconv. optimizations.
- DCO: Deconv. optimizations without ISM algorithm.
- ISM+DCO: Both ISM and deconv. optimizations.

7 Evaluation

We first show that ASV adds negligible overhead to the baseline DNN accelerator (Sec. 7.1) and introduces negligible accuracy loss (Sec. 7.2). We then show the performance and energy improvements of ASV (Sec. 7.3), which are robust against the underlying hardware configuration (Sec. 7.4). ASV also out-performs Eyeriss and GPUs (Sec. 7.5). Finally, we demonstrate the general applicability of our deconvolution optimizations by showing that they even improve runtime of GANs without hardware modifications (Sec. 7.6).

7.1 Hardware Overhead

Owing to the software transformations, ASV only minimally augments existing DNN accelerators. Relative to the baseline accelerator, ASV extends each PE to support accumulating absolute difference. This adds 6.3% area (15.3 μm^2) and 2.3% power (0.02 mW) overhead *per PE*. ASV also extends the scalar unit to support new point-wise operations, with an area and power overhead of 2 mm^2 and 2.2 mW, respectively. The overall area and power overhead introduced by ASV are both below 0.5%.

7.2 Accuracy Results

ASV matches or even outperforms DNN accuracy. Fig. 9 shows the accuracy of applying the ISM algorithm to stereo matching DNNs. We use *Propagation Window* (PW) to denote how far in time the correspondence invariant is propagated, which in turn decides how often key frames are selected. With PW-2, every other frame is selected as a key frame, and for PW-4, every fourth frame is a key frame. Note that the KITTI dataset contains at most two consecutive frames, and thus we evaluate only PW-2.

On both datasets, PW-2 retains the same accuracy as the stereo DNNs. On SceneFlow, PW-4 results in only 0.02% accuracy loss. In some cases, ISM combined with the DNNs can outperform the DNNs alone. For instance, applying the ISM algorithm with FlowNetC reduces error by 0.11% at PW-4. Overall, our experiments shows that by leveraging the correspondence invariant over time, ISM is able to preserve the DNN-like accuracy with cheap, classic stereo matching algorithms. We will now show that ASV achieves high accuracy while greatly improving the performance and energy-efficiency of stereo vision.

7.3 Speedup and Energy Reduction

ASV significantly improves performance and energy consumption of stereo vision. To understand the contributions of the ISM algorithm and the deconvolution optimizations, Fig. 10 shows the speedup and energy reduction of the three ASV variants (Sec. 6.2) over the baseline, when applied to different stereo DNNs. We choose PW-4 for the ISM algorithm. On average, combining ISM and deconvolution optimizations (DCO) ASV achieves 4.9× speedup and 85% energy reduction. Specifically, ISM achieves, on average, 3.3× speedup and 75% energy reduction, while DCO achieves 57% performance improvement and 38% energy reduction. ISM contributes

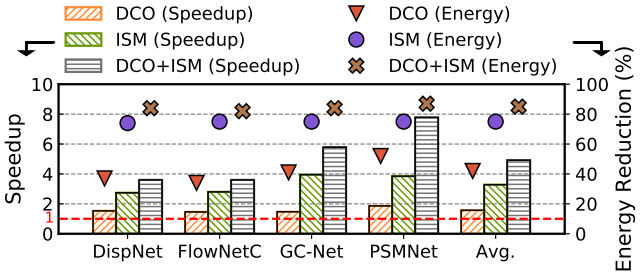
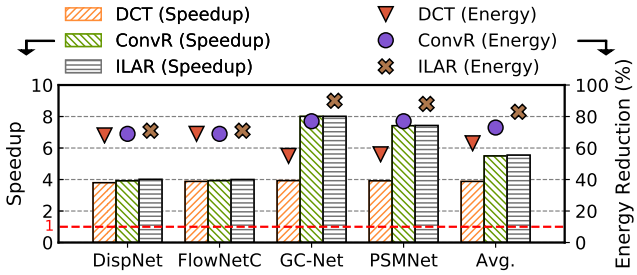
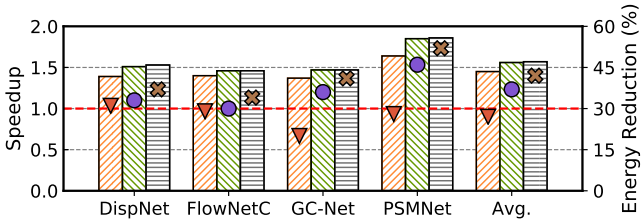


Fig. 10: Speedup and energy reduction of the three ASV variants over the baseline.



(a) Speedup and energy reduction on deconvolution layers only.



(b) Speedup and energy reduction on the entire network.

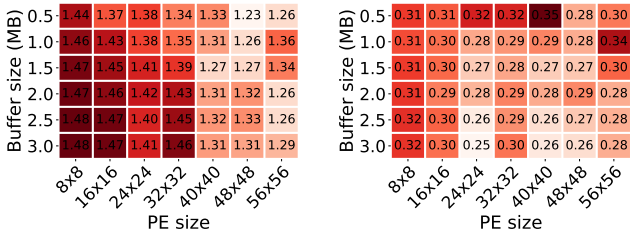
Fig. 11: The speedup and energy reduction of various deconvolution optimizations. Higher is better.

more than DCO because ISM avoids DNNs in non-key frames altogether by using the much cheaper BM and OF algorithms (Sec. 3.3).

Next, we dissect different optimization components within DCO to further understand the effect of each optimization.

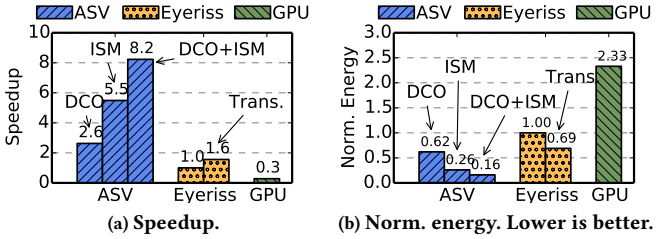
Deconvolution Optimizations Deconvolution optimizations consist of two components: the deconvolution to convolution transformation (`DCT` - Sec. 4.1) and the data-reuse optimization (Sec. 4.2). In particular, our data-reuse formulation unifies the exploitation of two kinds of reuse: the conventional data reuse in convolution layers and the inter-layer activation reuse in deconvolutions that is uniquely exposed by `DCT`. To clearly tease apart our contributions, we show the results of both the conventional reuse optimization (`ConvR`), which is obtained by applying our reuse optimizer (Sec. 4.2) *without* exploiting inter-layer activation reuse, and the additional effect of exploiting inter-layer activation reuse (`ILAR`).

Fig. 11 shows the speedup and energy reduction of `DCT`, `ConvR`, and `ILAR`. Fig. 11a shows the improvements of deconvolution layers only, and Fig. 11b shows the improvements of the entire network. The majority of speedup is from deconvolution transformation, which yields an average 3.9× speedup on deconvolution layers alone



(a) Speedup. (b) Energy reduction.

Fig. 12: Sensitivity analysis of DCO speedup and energy reduction with buffer size and PE array size on FlowNetC. Speedup is normalized to the corresponding configurations, not to a single, common baseline.



(a) Speedup. (b) Norm. energy. Lower is better.

Fig. 13: Comparison of speedup and energy reduction for ASV, Eyeriss, and GPU. The results are normalized to Eyeriss. We show three variants of ASV. We also apply the deconvolution transformation to Eyeriss to obtain a stronger baseline, hence the DCT bar of Eyeriss.

and 1.4× speedup on the entire network. On top of DCT, ConvR and ILAR further increase speedup to 5.6× and 1.6× on deconvolution layers alone and the entire networks, respectively.

Across different stereo DNNs, we find that 3D DNNs (GC-Net and PSMNet) have a speedup of 7.7× on deconvolution layers, higher than the 3.9× speedup of 2D DNNs (DispNet and FlowNetC). The reason is twofold. First, 3D DNNs have the higher percentage of zero-padding than 2D DNNs (8× vs. 4×), which are effectively eliminated by our deconvolution transformation. Second, after the deconvolution transformation the 3D DNNs have many small kernels (e.g., 1×1×1), which leads to low data-reuse. Thus, reuse optimizations become more critical to these networks. In contrast, most 2D stereo DNNs inherently have better data reuse with larger kernels (e.g., 5×5). We also observe that ConvR and ILAR have similar performance. This is because both optimize the data reuse to the extent that the layer processing becomes limited by the PE size.

While ILAR is similar in speedup compared to ConvR, ILAR is much more effective in reducing energy than ConvR. To demonstrate this, Fig. 11 overlays the energy reductions of different DCO variants on the right *y*-axis. DCO achieves 83% energy reduction on deconvolution alone and 38% on the entire network. Specifically, DCT reduces the deconvolution energy by 62%; ConvR and ILAR further improve the energy reduction to 73% and 83%, respectively.

The energy saving of DCT comes from eliminating redundant movement of padded zeros in the upsampled *ifmap*. ILAR achieves additional energy reduction over ConvR by exploiting inter-layer activation reuse, a unique reuse behavior in our transformed deconvolution layers. 3D DNNs benefit much more from ILAR than 2D DNNs, as is evident by examining the additional energy reductions of ILAR over ConvR across networks. This is because 3D stereo DNNs have low *ifmap* data-reuse; ILAR uniquely exploits inter-layer *ifmap* reuse, and thus reduces more memory traffics.

7.4 Sensitivity Analysis

While the speedup and energy reduction studied so far are based on one representative baseline accelerator configuration (Sec. 6.1), we find that our deconvolution optimization generally achieves similar improvements on other hardware configurations with resource provisions. In particular, we focus on two key types of hardware resource: PE size and on-chip buffer size. For brevity we only report results on FlowNetC [23], but the trends generally hold.

Fig. 12a and Fig. 12b show how DCO's average speedup and energy reduction of the entire network vary with different PE size and buffer size combinations, respectively. Note that the results are normalized to their corresponding hardware configurations rather than to the baseline described in Sec. 6.1. For instance, on the hardware with an 8×8 PE array and a 0.5 MB on-chip buffer, DCO achieves an 1.44× speedup.

DCO achieves speedups of 1.2× – 1.5× and energy reductions of 25% – 35% across different hardware capabilities, demonstrating broad applicability. In general, the performance improvement of DCO is more pronounced with small PE arrays, where the performance is compute-bound. As the PE size increases, the performance becomes memory bound, such that memory bandwidth limitations mask the benefit of data reuse. In addition, as the buffer size increases, the reuse opportunities inherently exposed by the buffer is higher, and thus data reuse optimizations become less critical, hence the lower energy savings.

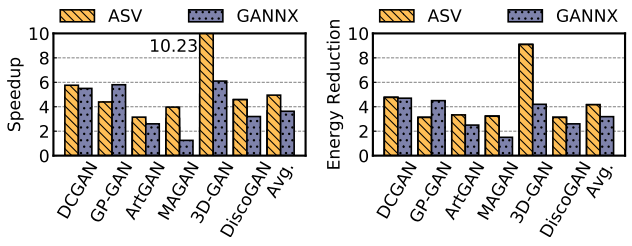
7.5 Eyeriss and GPU Comparisons

For completeness, we also compare ASV with Eyeriss [18] and a mobile Pascal GPU [8]. Fig. 13 shows the speedup and energy reductions of the three ASV variants, Eyeriss and GPU. Data is normalized to Eyeriss. To obtain a stronger Eyeriss baseline, we extended the Eyeriss simulator [4, 24] to support our deconvolution optimization. Our ILAR optimization does not apply because Eyeriss's spatial architecture requires a different reuse formulation from the one presented here, which targets a systolic array.

On average, when combing DCO and ISM, ASV achieves 8.2× speedup against Eyeriss while consuming only 16% of the energy. DCO and ISM contribute to 38% and 74% on energy saving, respectively. Critically, Eyeriss can also benefit from the deconvolution transformation (DCT), which achieves a 1.6× speedup and 31% energy saving compared to the baseline Eyeriss. Finally, ASV is 27× faster while consuming 15× lower energy than the GPU.

7.6 GANNX Comparison

Our deconvolution optimizations are not limited to stereo DNNs, but also apply broadly to deconvolution. To demonstrate general applicability, we apply the deconvolution optimizations to Generative Adversarial Networks (GANs), a class of DNNs that also make heavy use of deconvolution layers [27]. We compare against



(a) Speedup. (b) Energy reduction.

Fig. 14: Speedups and energy reductions on GANs between ASV and GANNX. Results are normalized to Eyeriss.

GANNX [76], a dedicated DNN accelerator for accelerating deconvolution in GANs. We configure both ASV and GANNX to have the same PE and buffer sizes. We normalize the ASV results to Eyeriss, consistent with what GANNX reports.

Fig. 14 shows speedup and energy comparisons across the six GANs used by GANNX. On average, ASV achieves 5.0× speedup and 4.2× energy reduction, higher than the improvements from GANNX (3.6× speedup and 3.2× energy reduction). The higher gains from ASV are mainly attributed to the inter-layer activation reuse, which is uniquely exposed by our deconvolution transformation and is unavailable in GANNX. Critically, our deconvolution optimizations are purely software optimizations without requiring the specialized hardware support of GANNX.

8 Related Work

Stereo Vision Accelerators Recently, commercial mobile vision systems [77] have started integrating dedicated stereo accelerators, such as the Stereo Depth Block in the Movidius Enhanced Vision Accelerator Suite [5], and the Stereo & Optical Flow Engine (SOFE) in the Nvidia Xavier mobile SoC [9]. From publicly available details, these are fixed-functioned accelerators targeting classic stereo algorithms, similar to previous stereo vision accelerators [28, 49, 64, 66, 73]. In contrast, ASV combines the efficiency of classic stereo algorithms with the accuracy of stereo DNNs.

Motion-based Algorithms Our ISM algorithm shares a similar key observation as some recent motion-based vision algorithms such as EVA^2 [14] and Euphrates [78], in that correlation across frames in a video stream can be used to simplify continuous vision tasks. Euphrates [78] focuses on computing regions-of-interest (ROIs) in object detection and tracking tasks. In contrast, stereo vision is concerned with the depth of the whole frame rather than discrete ROIs. EVA^2 [14] is not limited to ROIs. However, it relies on estimating the motion of an intermediate activation's receptive field. In the stereo task, the receptive field of an intermediate activation necessarily spans both the left and right images. Thus, the motion of the receptive field would be difficult, if not impossible, to calculate.

Fundamentally, motion-based relaxations fall within the realm of incremental computing, a general technique used in program analysis and optimization [51, 54] and applies beyond the temporal and/or vision domain. Diffy [47] exploits the spatial similarity across pixels in the same frame to improve DNN efficiency. Riera et al. [56] exploit repeated *ifmap* elements in speech recognition.

Deconvolution Sparsity Many prior studies optimize hardware to exploit sparsity in DNNs [10, 11, 29, 31, 38, 40, 68, 69, 72]. Stereo vision DNNs make use of deconvolution layers, which expose structured sparsity patterns. Recent work has prosed specialized hardware specifically for exploiting sparsity in deconvolution layers [60, 76]. Our observation, however, is that mitigating sparsity-induced efficiencies in deconvolution does not necessarily require hardware support. We propose novel software optimizations to eliminate the compute inefficiencies without hardware changes.

Data-Reuse Optimization Exploiting data-reuse (through tiling) is critical to DNN efficiency [12, 18, 19, 24, 25, 32, 39, 44, 46, 52, 70, 71, 74, 75]. Orthogonal to generic data-reuse, we identify a new reuse dimension, inter-layer activation reuse (ILAR), that is uniquely enabled by our deconvolution transformation.

Previous DNN mapping frameworks mostly rely on exhaustive search [32, 74, 75], which does not scale to exploiting ILAR (Sec. 4.2). Instead, ASV uses a constrained-optimization that can solved efficiently using dynamic programming. TETRIS [24] also uses a constrained-optimization for DNN scheduling, albeit with certain problem-specific simplifications. However, it does not exploit ILAR. Our formulation directly optimizes for latency rather than memory traffic [24, 75] or resource utilization [46].

9 Conclusion

ASV simultaneously improves performance and energy-efficiency of "depth from stereo", while maintaining high accuracy. ASV combines algorithmic and computational optimizations that leverage characteristics unique to stereo vision. We demonstrate careful design choices that let these optimizations be integrated with existing DNN accelerators with minor hardware extensions. As intelligent machine perception increasingly relies on depth sensing, ASV provides a promising first step towards comprehensive system support.

Appendices

A General Deconvolution Transformation with an N-dimensional Kernel

Here we show a general formulation for decomposing a deconvolution kernel. A N-dimension kernel is decomposed into 2^N sub-kernels, each sub-kernel S_k can be calculated as follows:

$$S_k^{(i_0,i_1,\ldots,i_{N-1})} = K^{(2i_0+\delta_0,2i_1+\delta_1,\ldots,2i_{N-1}+\delta_{N-1})},\ \forall k \in [0, 2^N-1]$$

$$\delta_j = (k \gg j\ \&1), i_j \in [0, \lfloor(|K^{(j)}| - \delta_j)/2\rfloor),\ \forall j \in [0, N-1]$$

where N is the number of dimensions in the original kernel K, $S_k^{(i_0,i_1,\ldots,i_{N-1})}$ is the element $(i_0, i_1, \ldots, i_{N-1})$ in k^{th} sub-kernel, and $K^{(*,\ldots,*)}$ denotes an element in the original kernel K.

Our formulation essentially shows that the element $(i_0, i_1, \ldots, i_{N-1})$ in the k^{th} sub-kernel comes from the element $(2i_0 + \delta_0, 2i_1 + \delta_1, \ldots, 2i_{N-1} + \delta_{N-1})$ in the original kernel. Each δ_* is a binary value calculated by: $\delta_i = (k \gg j\&1)$, where $\gg$ is the right shift operator and & is the bitwise AND operator. The dimension of the each sub-kernel is determined by: $i_j \in [0, \lfloor(|K^{(j)}| - \delta_j)/2\rfloor)$, where $|K^{(j)}|$ is the size of jth dimension of the original kernel.

B The Expressions of ΔW_k^i, ΔIF^i, and ΔOF_k^i

The expression of ΔW_k^i, ΔIF^i, and ΔOF_k^i are determined by W_k^i, H_k^i, and $|\overrightarrow{C^i}|$. Their exact expressions are shown below:

$$\Delta W_k^i = W_k^i \times H_k^i \times C_k^i$$

where C_k^i denotes the total number of sub-kernel k in round i; W_k^i and H_k^i are the dimensions of sub-kernel k in round i.

$$\Delta IF^i = W^i \times H^i \times I$$

where W^i and H^i are the weight and height of an *ifmap* tile to be loaded in round i, and I is the number of input channels.

$$\Delta OF_k^i = \frac{W^i \times H^i \times C_k^i}{s^2}$$

where s denotes the stride of this layer.

References

[1] [n. d.]. 3D Camera Market is Expected to Reach $7.6 Billion, Globally, by 2020. https://www.alliedmarketresearch.com/press-release/global-3D-camera-market-is-expected-to-reach-7-6-billion-by-2020-allied-market-research.html

[2] [n. d.]. Bumblebee2 Stereo Vision Camera Systems. https://www.ptgrey.com/bumblebee2-firewire-stereo-vision-camera-systems

[3] [n. d.]. Choose the Right 3D Vision Camera For Your IoT Device. https://medium.com/iotforall/choose-the-right-3d-vision-camera-for-your-iot-device-962d95c581cb

[4] [n. d.]. Eyeriss Simulator. https://github.com/stanford-mast/nn_dataflow

[5] [n. d.]. Intel Movidius Myriad X VPU. https://www.movidius.com/myriadx

[6] [n. d.]. KITTI Stereo Benchmark. http://www.cvlibs.net/datasets/kitti/eval_scene_flow.php?benchmark=stereo

[7] [n. d.]. Micron 178-Ball, Single-Channel Mobile LPDDR3 SDRAM Features. https://www.micron.com/-/media/client/global/documents/products/data-sheet/dram/mobile-dram/low-power-dram/lpddr3/178b_8-16gb_2c0f_mobile_lpddr3.pdf

[8] [n. d.]. NVIDIA Jetson TX2 Module. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules/

[9] [n. d.]. NVIDIA Reveals Xavier SOC Details. https://www.forbes.com/sites/moorinsights/2018/08/24/nvidia-reveals-xavier-soc-details/amp/

[10] Angshuman Parashar and Minsoo Rhu and Anurag Mukkara and Antonio Puglielli and Rangharajan Venkatesan and Brucek Khailany and Joel Emer and Stephen Keckler and William Dally. 2017. SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks. In *Proceedings of the 44th Annual International Symposium on Computer Architecture.*

[11] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos. 2016. Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing. In *Proceedings of the 43rd ACM/IEEE Annual International Symposium on Computer Architecture.*

[12] Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. 2016. Fused-layer CNN Accelerators. In *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture.*

[13] M. Z. Brown, D. Burschka, and G. D. Hager. 2003. Advances in computational stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence.*

[14] Mark Buckler, Philip Bedoukian, Suren Jayasuriya, and Adrian Sampson. 2018. EVA^2: Exploiting Temporal Redundancy in Live Computer Vision. In *Proceedings of the 45th ACM/IEEE Annual International Symposium on Computer Architecture.*

[15] Jan Cech, Jordi Sanchez-Riera, and Radu P. Horaud. 2011. Scene Flow Estimation by Growing Correspondence Seeds. In *Proceedings of the 21st IEEE Conference on Computer Vision and Pattern Recognition.*

[16] Jia-Ren Chang and Yong-Sheng Chen. 2018. Pyramid Stereo Matching Network. In *Proceedings of 28th IEEE/CVF Conference on Computer Vision and Pattern Recognition.*

[17] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems.*

[18] Y. Chen, J. Emer, and V. Sze. 2016. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In *Proceedings of the 43rd ACM/IEEE Annual International Symposium on Computer Architecture.*

[19] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam. 2014. DaDianNao: A Machine-Learning Supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture.*

[20] Navneet Dalal and Bill Triggs. 2005. Histograms of Oriented Gradients for Human Detection. In *Proceedings of the 15th IEEE Computer Society Conference on Computer Vision and Pattern Recognition.*

[21] Gunnar Farnebäck. 2002. *Polynomial expansion for orientation and motion estimation.* Ph.D. Dissertation. Linköping University Electronic Press.

[22] Gunnar Farnebäck. 2003. Two-frame motion estimation based on polynomial expansion. In *Proceedings of the 13th Scandinavian Conference on Image Analysis.*

[23] Philipp Fischer, Alexey Dosovitskiy, Eddy Ilg, Philip Häusser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. 2015. FlowNet: Learning Optical Flow with Convolutional Networks. In *Proceedings of the 15th IEEE International Conference on Computer Vision.*

[24] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. 2017. Tetris: Scalable and efficient neural network acceleration with 3d memory. In *Proceedings of the 22nd ACM International Conference on Architectural Support for Programming Languages and Operating Systems.*

[25] Mingyu Gao, Xuan Yang, Jing Pu, Mark Horowitz, and Christos Kozyrakis. 2019. Tangram: Optimized Coarse-Grained Dataflow for Scalable NN Accelerators. In *Proceedings of the 24th ACM International Conference on Architectural Support for Programming Languages and Operating System.*

[26] Andreas Geiger, Martin Roser, and Raquel Urtasun. 2010. Efficient Large-Scale Stereo Matching. In *Proceedings of the 10th Asian Conference on Computer Vision.*

[27] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Proceeding of the 28th Conference on Neural Information Processing Systems.*

[28] Eduardo Gudis, Pullan Lu, David Berends, Kevin Kaighn, Gooitzen Wal, Gregory Buchanan, Sek Chai, and Michael Piacentino. 2013. An embedded vision services framework for heterogeneous accelerators. In *Proceedings of the 23rd IEEE conference on computer vision and pattern recognition workshops.*

[29] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. 2016. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In *Proceedings of the 43rd ACM/IEEE Annual International Symposium on Computer Architecture.*

[30] Richard Hartley and Andrew Zisserman. 2003. *Multiple view geometry in computer vision.* Cambridge university press.

[31] Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel Pruning for Accelerating Very Deep Neural Networks. In *Proceedings of the 16th IEEE International Conference on Computer Vision.*

[32] Kartik Hegde, Rohit Agrawal, Yulun Yao, and Christopher W Fletcher. 2018. Morph: Flexible Acceleration for 3D CNN-Based Video Understanding. In *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture.*

[33] H. Hirschmuller. 2005. Accurate and efficient stereo processing by semi-global matching and mutual information. In *Proceedings of the 15th IEEE Conference on Computer Vision and Pattern Recognition.*

[34] Berthold KP Horn and Brian G Schunck. 1981. Determining optical flow. *Artificial intelligence.*

[35] M Jakubowski and G Pastuszak. 2013. Block-based Motion Estimation Algorithms–A Survey. *Opto-Electronics Review.*

[36] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual ACM/IEEE International Symposium on Computer Architecture.*

[37] Alex Kendall, Hayk Martirosyan, Saumitro Dasgupta, Peter Henry, Ryan Kennedy, Abraham Bachrach, and Adam Bry. 2017. End-to-End Learning of Geometry and Context for Deep Stereo Regression. In *Proceedings of 27th IEEE/CVF Conference on Computer Vision and Pattern Recognition.*

[38] H.T. Kung, Bradley McDanel, and Sai Qian Zhang. 2019. Packing Sparse Convolutional Neural Networks for Efficient Systolic Array Implementations: Column Combining Under Joint Optimization. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems.*

[39] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. 2018. Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects. In *Proceedings of the 23rd ACM International Conference on Architectural Support for Programming Languages and Operating Systems.*

[40] S. K. Lee, P. N. Whatmough, D. Brooks, and G. Wei. 2019. A 16-nm Always-On DNN Processor With Adaptive Clocking and Multi-Cycle Banked SRAMs. *IEEE Journal of Solid-State Circuits.*

[41] Haitong Li, Mudit Bhargava, Paul N. Whatmough, and H-S Philip Wong. 2019. On-Chip Memory Technology Design Space Explorations for Mobile Deep Neural Network Accelerators. In *Proceedings of the 56th Annual Design Automation Conference.*

[42] Hongche Liu, Tsai-Hong Hong, Martin Herman, Ted Camus, and Rama Chellappa. 1998. Accuracy vs efficiency trade-offs in optical flow algorithms. *Computer vision and image understanding*.

[43] David G Lowe. 2004. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*.

[44] Wenyan Lu, Guihai Yan, Jiajun Li, Shijun Gong, Yinhe Han, and Xiaowei Li. 2017. Flexflow: A flexible dataflow accelerator architecture for convolutional neural networks. In *Proceedings of the 23rd IEEE International Symposium on High Performance Computer Architecture*.

[45] Bruce D Lucas and Takeo Kanade. 1981. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*.

[46] Yufei Ma, Yu Cao, Sarma Vrudhula, and Jae-sun Seo. 2017. Optimizing Loop Operation and Dataflow in FPGA Acceleration of Deep Convolutional Neural Networks. In *Proceedings of the 25th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*.

[47] Mostafa Mahmoud, Kevin Siu, and Andreas Moshovos. 2018. Diffy: a Déjà vu-Free Differential Deep Neural Network Accelerator. In *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture*.

[48] Nikolaus Mayer, Eddy Ilg, Philip Häusser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. 2016. A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation. In *Proceedings of 26th IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

[49] Amrita Mazumdar, Thierry Moreau, Sung Kim, Meghan Cowan, Armin Alaghi, Luis Ceze, Mark Oskin, and Visvesh Sathe. 2017. Exploring computation-communication tradeoffs in camera systems. In *Proceedings of the 13th IEEE International Symposium on Workload Characterization*.

[50] M. Menze and A. Geiger. 2015. Object scene flow for autonomous vehicles. In *Proceedings of the 25th IEEE Conference on Computer Vision and Pattern Recognition*.

[51] Donald Michie. 1968. "Memo" functions and machine learning. *Nature*.

[52] Ravi Teja Mullapudi, Andrew Adams, Dillon Sharlet, Jonathan Ragan-Kelley, and Kayvon Fatahalian. 2016. Automatically scheduling halide image processing pipelines. In *Proceedings of the 35th International Conference on Computer Graphics and Interactive Techniques*.

[53] Augustus Odena, Vincent Dumoulin, and Chris Olah. 2016. Deconvolution and Checkerboard Artifacts. *Distill*.

[54] William Pugh and Tim Teitelbaum. 1989. Incremental computation via function caching. In *Proceedings of the 16th ACM Symposium on Principles of Programming Languages*.

[55] Wajahat Qadeer, Rehan Hameed, Ofer Shacham, Preethi Venkatesan, Christos Kozyrakis, and Mark A Horowitz. 2013. Convolution engine: balancing efficiency & flexibility in specialized computing. In *Proceedings of the 40th IEEE Annual International Symposium on Computer Architecture*.

[56] Marc Riera, Jose-Maria Arnau, and Antonio González. 2018. Computation reuse in DNNs by exploiting input similarity. In *Proceedings of the 45th IEEE Annual International Symposium on Computer Architecture*.

[57] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 2018. SCALE-Sim: Systolic CNN Accelerator Simulator. arXiv:1811.02883

[58] D. Scharstein, R. Szeliski, and R. Zabih. 2001. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. In *Proceedings of 1st IEEE Workshop on Stereo and Multi-Baseline Vision*.

[59] Nikolai Smolyanskiy, Alexey Kamenev, and Stanley T. Birchfield. 2018. On the Importance of Stereo for Accurate Depth Estimation: An Efficient Semi-Supervised Deep Neural Network Approach. In *Proceedings of the 31th Conference on Computer Vision and Pattern Recognition Workshops*.

[60] M. Song, J. Zhang, H. Chen, and T. Li. 2018. Towards Efficient Microarchitectural Design for Accelerating Unsupervised GAN-Based Deep Learning. In *Proceedings of the 24th IEEE International Symposium on High Performance Computer Architecture*.

[61] Richard Szeliski. 2010. *Computer vision: algorithms and applications*. Springer Science & Business Media.

[62] Qi Tian and Michael N Huhns. 1986. Algorithms for subpixel registration. *Computer Vision, Graphics, and Image Processing*.

[63] F. Tombari, S. Mattoccia, L. Di Stefano, and E. Addimanda. 2008. Classification and evaluation of cost aggregation methods for stereo correspondence. In *Proceedings of the 8th IEEE Conference on Computer Vision and Pattern Recognition*.

[64] Christos Ttofis and Theocharis Theocharides. 2014. High-quality real-time hardware stereo matching based on guided image filtering. In *Proceedings of the 6th the Conference on Design, Automation & Test in Europe*.

[65] Jianfeng Wang, Cha Zhang, Wenwu Zhu, Zhengyou Zhang, Zixiang Xiong, and Philip A Chou. 2012. 3D scene reconstruction by multiple structured-light based commodity depth cameras. In *Proceedings of the 37th IEEE International Conference on Acoustics, Speech and Signal Processing*.

[66] Wenqiang Wang, Jing Yan, Ningyi Xu, Yu Wang, and Feng-Hsiung Hsu. 2015. Real-time high-quality stereo vision system in FPGA. *IEEE Transactions on Circuits and Systems for Video Technology*.

[67] Claus Weitkamp. 2006. *Lidar: range-resolved optical remote sensing of the atmosphere*. Springer Science & Business.

[68] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning Structured Sparsity in Deep Neural Networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*.

[69] P. N. Whatmough, S. K. Lee, D. Brooks, and G. Wei. 2018. DNN Engine: A 28-nm Timing-Error Tolerant Sparse Deep Neural Network Processor for IoT Applications. *IEEE Journal of Solid-State Circuits*.

[70] P. N. Whatmough, S. K. Lee, M. Donato, H. Hsueh, S. L. Xi, U. Gupta, L. Pentecost, G. G. Ko, D. Brooks, and G. Wei. 2019. A 16nm 25mm2 SoC with a 54.5x Flexibility-Efficiency Range from Dual-Core Arm Cortex-A53 to eFPGA and Cache-Coherent Accelerators. In *Proceedings of the 7th Symposium on VLSI Circuits*.

[71] Paul N. Whatmough, Chuteng Zhou, Patrick Hansen, Shreyas K. Venkataramanaiah, Jae-sun Seo, and Matthew Mattina. 2019. FixyNN: Efficient Hardware for Mobile Computer Vision via Transfer Learning. In *Proceedings of the 2nd Conference on Systems and Machine Learning*.

[72] Bichen Wu, Alvin Wan, Xiangyu Yue, Peter Jin, Sicheng Zhao, Noah Golmant, Amir Gholaminejad, Joseph Gonzalez, and Kurt Keutzer. 2018. Shift: A Zero FLOP, Zero Parameter Alternative to Spatial Convolutions. In *Proceedings of the 31st IEEE Conference on Computer Vision and Pattern Recognition*.

[73] Qingxiong Yang. 2014. Hardware-efficient bilateral filtering for stereo matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[74] Xuan Yang, Mingyu Gao, Jing Pu, Ankita Nayak, Qiaoyi Liu, Steven Emberton Bell, Jeff Ou Setter, Kaidi Cao, Heonjae Ha, Christos Kozyrakis, and Mark Horowitz. 2018. DNN Dataflow Choice Is Overrated. arXiv:1809.04070

[75] Xuan Yang, Jing Pu, Blaine Burton Rister, Nikhil Bhagdikar, Stephen Richardson, Shahar Kvatinsky, Jonathan Ragan-Kelley, Ardavan Pedram, and Mark Horowitz. 2016. A Systematic Approach to Blocking Convolutional Neural Networks. arXiv:1606.04209

[76] Amir Yazdanbakhsh, Kambiz Samadi, Nam Sung Kim, and Hadi Esmaeilzadeh. 2018. GANAX: A Unified MIMD-SIMD Acceleration for Generative Adversarial Networks. In *Proceedings of the 45th ACM/IEEE Annual International Symposium on Computer Architecture*.

[77] Yuhao Zhu, Matthew Mattina, and Paul N. Whatmough. 2018. Mobile Machine Learning Hardware at ARM: A Systems-on-Chip (SoC) Perspective. In *Proceedings of the 1st Conference on Systems and Machine Learning*.

[78] Yuhao Zhu, Anand Samajdar, Matthew Mattina, and Paul Whatmough. 2018. Euphrates: Algorithm-SoC Co-Design for Low-Power Mobile Continuous Vision. In *Proceedings of the 45th ACM/IEEE Annual International Symposium on Computer Architecture*.