Low Complexity, Hardware-Efficient Neighbor-Guided SGM Optical Flow for Low Power Mobile Vision Applications

Ziyun Li, *Student Member, IEEE*, Jiang Xiang, *Student Member, IEEE*, Luyao Gong, *Student Member, IEEE* , David Blaauw, *Fellow, IEEE*, Chaitali Chakrabarti, *Fellow, IEEE*, and Hun Seok Kim, *Member, IEEE*

Abstract— **Accurate, low-latency and energy-efficient optical flow estimation is a fundamental kernel function to enable several real-time vision applications on mobile platforms. This paper presents Neighbor-Guided Semi-Global Matching (NG-fSGM), a new low-complexity optical flow algorithm tailored for low power mobile applications. NG-fSGM obtains high accuracy optical flow by aggregating local matching costs over a semi-global region, successfully resolving local ambiguity in texture-less and occluded regions. Proposed NG-fSGM aggressively prunes the search space based on neighboring pixels' information to significantly lower the algorithm complexity from the original fSGM. As a result, NG-fSGM achieves 17.9× reduction in the number of computations and 8.37× reduction in memory space compared to the original fSGM without compromising its algorithm accuracy. A multicore architecture for NG-fSGM is implemented in hardware to quantify algorithm complexity and power consumption. The proposed architecture realizes NG-fSGM with overlapping blocks processed in parallel to enhance throughput and to lower power consumption. The 8-core architecture achieves 20M pixel/s (66 frames / sec for VGA) throughput with 9.6mm^2 area at 679.2mW power consumption in 28nm node.**

Index Terms — **Optical flow, Semi-global matching, VLSI, Low power, Multi-core accelerator.**

I. INTRODUCTION

REAL TIME, accurate, dense and energy-efficient optical flow estimation is essential for many computer vision applications such as object detection and tracking [1], simultaneous localization and mapping (SLAM) [2][3], and advanced driving assistance system (ADAS) [4]. Optical flow represents the pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer (an eye or a camera) and the scene [5].

Challenges involved in optical flow estimation include ambiguity over occlusion, perspective distortion, transparent objects, low/uniform textures, and repeated patterns. To address these challenges, most existing optical flow algorithms optimize a global energy function in the form of weighted sum of a data term and a prior term [6][7][8][9], as stated in the taxonomy of Baker et al. [10]. Semi-global matching (SGM) [11], an algorithm often used in stereo matching, can resolve local ambiguity by propagating information from neighboring pixels along multiple paths over the entire image. However, directly applying SGM to optical flow estimation is challenging because, unlike stereo matching, the search space for optical flow is two-dimensional. Increase in flow search range will quadratically affect the memory and computation requirement. Therefore, it is typical to employ a coarse-to-fine approach [12] [13] for estimating large optical flow displacements. However, accuracy degradation in hierarchical approaches is inevitable due to resolution loss at higher hierarchical levels and error propagation to lower levels [14]. As an alternative technique, convolutional neural network (CNN) based optical flow estimation schemes have been recently proposed [15][16]. These techniques, however, require a very large number of parameters (tens of millions) and convolution operations (>600k OPs per pixel) [15] making them impractical for mobile applications.

When optical flow computation is implemented on emerging power-critical mobile platforms such as autonomous navigation of unmanned aerial vehicles (UAV), it imposes further "SWaP" [17] constraints limiting the size, weight and power consumption to deliver real-time performance. Conventional general purpose CPU or GPU based solutions are not applicable to meeting stringent SWaP constraints for mobile applications as they suffer from either high power consumption (~100W) [18] for real-time performance or low performance (~1fps) given power constraint.

We originally proposed[2] a new, low-complexity optical flow method; Neighbor-Guided Semi-Global Matching (NG-fSGM) in [19][20]. In this paper, we build upon that work and provide comprehensive parametric analysis of NG-fSGM along with a discussion on several optimization techniques for SWaP constrained MAV applications. The proposed NG-fSGM method is based on SGM [11], a popular concept in stereo matching, and its optical flow version, fSGM [21]. Our objective is to achieve orders of magnitude complexity reduction of the original SGM while maintaining its high accuracy. This goal has been achieved by performing comprehensive algorithm-architecture co-optimization. The algorithm optimization techniques include: (1) aggressively pruning the search space size by exploiting flow similarity of

[2] This work was in part published in ICIP'16 [19] and SiPS'16 [20].

neighboring pixels, (2) approximating cost array aggregation to avoid exhaustive pixel-wise cost computation, (3) partitioning the image into overlapping blocks and initializing boundary flow vectors with temporal prediction to minimize accuracy degradation and overlap overhead at the boundary, and (4) executing the full NG-fSGM algorithm on selective pixels and performing interpolation to construct dense flow field. Along with the proposed algorithm optimization, we designed a multi-core, highly parallel architecture for NG-fSGM in TSMC 28nm GP technology. The accuracy, performance and power of the proposed NG-fSGM algorithm and hardware design are characterized and evaluated. Results reveal that the proposed NG-fSGM method provides dense optical flow with accuracy comparable to the original fSGM while exhibiting significant complexity reduction both in arithmetic/logical operations and in memory size/bandwidth requirements. The proposed energy-efficient hardware architecture with 8 cores achieves a throughput of 20M flow/s (equivalent to VGA 66 fps) with estimated power consumption of 679.2mW in TSMC 28nm GP node.

II. Background

The SGM algorithm was originally proposed by Hirschmüller for stereo matching [11]. It achieves state-of-the-art accuracy by applying dynamic programming based cost function optimization over the entire image. SGM first computes pixel-wise matching costs of corresponding pixels in two frames for all disparities (stereo matching) in the search space. This is followed by cost aggregation along a finite number of paths that penalizes abrupt disparity changes. SGM was applied to optical flow, fSGM in [21] by extending the search space from 1D stereo to 2D optical flow. A brief review of fSGM is included here for completeness.

Step 1: Computation of pixel-wise matching costs C($\boldsymbol{p}$, $\boldsymbol{o}$) between pixel $\boldsymbol{p} = (x, y)$ in the previous image frame and pixel $\boldsymbol{q} = \boldsymbol{p} + \boldsymbol{o}$ in the current image frame, for all flow vectors $\boldsymbol{o} = (u, v)$, where u is the horizontal component and v is the vertical component. The cost function can be based on Rank, Census [22] and mutual information [23].

Step 2: The smoothness constraint on matching costs is applied to penalize abrupt changes of flow vectors among adjacent pixels. The *accumulated cost* $L_r(\boldsymbol{p}, \boldsymbol{o})$ of the pixel $\boldsymbol{p}$ for a flow vector $\boldsymbol{o}$ along a path in the direction r is defined as

$$L_r(\boldsymbol{p},\boldsymbol{o}) = \mathrm{C}(\boldsymbol{p},\boldsymbol{o}) + Z - \min_{\forall \boldsymbol{k}} L_r(\boldsymbol{p}\text{-}\boldsymbol{r},\boldsymbol{k}). \tag{1}$$

The cost regularization summand has the form

$$Z = \min\{L_r(\boldsymbol{p}\text{-}r,\boldsymbol{o}), \min_{|\boldsymbol{i}\text{-}\boldsymbol{o}|^2 \le 2} L_r(\boldsymbol{p}\text{-}r,\boldsymbol{i}) + P_1, \min_{\boldsymbol{j}} L_r(\boldsymbol{p}\text{-}r,\boldsymbol{j}) + P_2\} \tag{2}$$

where P_1 and P_2 are regularization penalties ($P_1 \le P_2$). Instead of the piecewise linear model used in fSGM[21], we adopt a constant penalty model because of its simplicity for VLSI implementation without significant accuracy degradation [24]. The modification penalizes 1-pixel offset flow vectors by a smaller penalty P_1 (smoothness constraint) and all other vectors with >2 pixel offsets by a larger penalty P_2 [11]. The aggregated cost $S(\boldsymbol{p},\boldsymbol{o})$ is the sum of $L_r(\boldsymbol{p},\boldsymbol{o})$ over all paths.

$$S(\boldsymbol{p},\boldsymbol{o}) = \textstyle\sum_{\mathrm{r}} L_r(\boldsymbol{p},\boldsymbol{o}) \tag{3}$$

Step 3: The final flow estimation uses winner-takes-all strategy. The flow vector $\boldsymbol{o}$ with the minimum cost $S(\boldsymbol{p},\boldsymbol{o})$ is selected as the final flow estimation.

Straightforward fSGM implementation poses significant hardware challenges. The complexity of the original SGM method is $O(WHD)$, where W is the image width, H is the image height and $D = d^2$ is the size of the search space per pixel given the one-dimensional search range d. Note that complexity of fSGM increases quadratically with the one-dimensional flow range d, making the algorithm very inefficient for a moderate flow search range (e.g., $D = 10000$ for ±50 pixel search range per dimension). Prior work [25] prunes SGM aggregation results for stereo vision processing to minimize the storage requirement and to reduce the SGM aggregation complexity in forward and backward propagations. However, the technique in [25] still involves exhaustively evaluating the entire search range (1D for stereo) for pixel matching, and thus would dominate the overall complexity when applied to optical flow. In this paper, we introduce a new optical flow algorithm; Neighbor-Guided fSGM (NG-fSGM) whose complexity (both matching and cost aggregation) is independent of 2D search range. Despite its significantly lower complexity and memory footprint, the proposed NG-fSGM still achieves near fSGM accuracy. We also propose several hardware-oriented optimizations, as will be described in the following sections.

III. Neighbor-Guided Semi-global matching

NG-fSGM reduces the complexity by aggressively pruning the search space based on information from neighbors. Using neighborhood information to prune the search space has also been used in [26] and [27] in the context of block matching for motion estimation. We extend the idea to semi-global cost aggregation and also modify flow computation functions to reduce the overall complexity.

A. Flow Subset Selection

It is highly likely that neighboring pixels in the image have an identical or slowly changing flow vector since they tend to belong to the same object, and thus have similar motion. Small flow variation usually occurs due to slanted surface of objects, spinning objects, camera motion, etc. Large flow variations can occur at the boundary of different objects and are typically combined with occlusion and motion discontinuity. NG-fSGM exploits this property by selecting a subset of search space, $\boldsymbol{O_p}$ for each pixel $\boldsymbol{p}$, based on its neighboring pixels' flow results. Computation of pixel-wise matching costs is performed on the subset $\boldsymbol{O_p}$ whose size is much smaller than $D = d^2$. This selection strategy is inspired by PatchMatch [27], in which the search space is initialized to a random set and neighboring pixels exchange 'good guesses'.

The subset $\boldsymbol{O_p}$ selection for each pixel $\boldsymbol{p}$ is guided by its neighboring pixels along every aggregation path r in SGM, as shown in Fig. 1. For pixel $\boldsymbol{p}$, $\boldsymbol{O_p}$ is initially empty. The best N flow vectors in $\boldsymbol{O_{p-r}}$ of previous pixel along path r with the minimum cost $L_r(\boldsymbol{p}-r, \boldsymbol{o})$ are added to $\boldsymbol{O_p}$ to construct the search subset. Pixels correspond to these best N flow vectors are marked by B in Fig. 1. We may choose multiple ($N > 1$ per each path r) best vectors for robustness to cost variation accumulated along a path and also local abnormality of pixel-wise matching cost. Since fSGM applies a low aggregation penalty (P_1 in (2)) when the flow varies smoothly, it is reasonable to add adjacent flow vectors, marked by A in Fig. 1, around each of these best N vectors to $\boldsymbol{O_p}$. Note that selection of A points around each B point is pseudo-random and unbiased. To enable the algorithm to adapt to rapid flow variations (e.g., occlusion and object discontinuity), M random flow vectors are added to the subset, which are marked by R in Fig. 1. Although the number of these random vectors selected for each pixel is small compared with the flow search range, it plays a very important role because randomly found 'good' candidates can propagate to neighboring pixels through forward and backward propagations. NG-SGM propagation starts from image boundary pixels which do not have sufficient number of neighboring pixels. The initial subset for these boundary pixels is thus randomly selected from a uniform distribution.

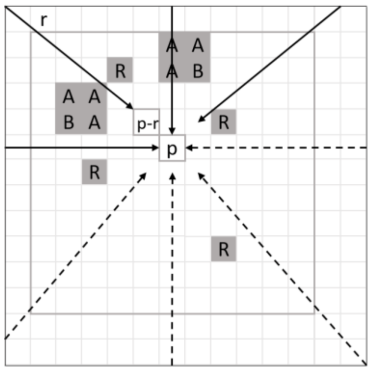


Fig. 1. Illustration of subset selection for the center pixel $\boldsymbol{p}$. The solid arrows represents path directions in forward scan while dashed arrows represents path directions in backward scan. The selected flow vectors, is the combination of B and A, where B corresponds to the N = 2 best flow vectors and 3 A's are selected for each B within a 2x2 window. R corresponds to M = 4 random flow vectors.

Typically, SGM is implemented in two scans, forward and backward, and so SGM aggregation paths are divided into two groups one for each scan. In Fig. 1, the forward scan proceeds from top-left to bottom-right of the image in the raster scan order, while the backward scan processes pixels in the reverse order. As a result, pixel $\boldsymbol{p}$ has different flow vector subset $\boldsymbol{O_{p1}}$ and $\boldsymbol{O_{p2}}$, and aggregated cost $S_1(\boldsymbol{p},\boldsymbol{o})$ and $S_2(\boldsymbol{p},\boldsymbol{o})$ for forward scan with subscript 1 and backward scan with subscript 2, respectively. The overall subset $\boldsymbol{O_p}$ is the union of $\boldsymbol{O_{p1}}$ and $\boldsymbol{O_{p2}}$ and the overall aggregated cost $S(\boldsymbol{p},\boldsymbol{o})$ is the sum of $S_1(\boldsymbol{p},\boldsymbol{o})$ and $S_2(\boldsymbol{p},\boldsymbol{o})$. We propose in Section III-C an approximation strategy to combine forward and backward aggregation. In the backward scan, N best flow vectors with the forward scan minimum cost $S_1(\boldsymbol{p},\boldsymbol{o})$ is added to construct $\boldsymbol{O_{p2}}$ to increase algorithm accuracy.

The flow vector candidates chosen by different aggregation paths may be redundant since neighboring pixels' best vectors can be identical, and the adjacent windows (i.e., A's in Fig.1) can overlap. When candidates are all exclusive (worst case), the total number of vectors in the search subset $\boldsymbol{O_p}$ is $T = NK(P/2+1)+M$, where P is the total (forward and backward) number of aggregation paths, and K is the window size to select adjacent candidates (A points) surrounding each best candidate (B point) guided by a neighbor. Fig. 1 shows an example for K=2×2 window. The complexity of NG-fSGM is $O(WHT)$, which is independent of flow search range ($D = d^2$). With $T << D$, significant (>10x) complexity reduction can be achieved compared to the original fSGM.

B. Pixel-wise matching cost

For pixel-wise matching cost C($\boldsymbol{p},\boldsymbol{o}$), we adopt Hamming distance of Census transform [22]. Census transform has been proven to represent image structure well and to be robust to illumination variations [29].

One possible implementation of SGM is to pre-calculate the Census transform of the entire image and store the Census image in an array of size WHC^2. Here, C^2 denotes the Census transform window size. Storing the Census transformed images poses significant memory overhead because each pixel is represented with a bit string of C^2 bits instead of 8-bit greyscale value. Once Census transformed images are pre-calculated, raw costs can be generated directly by accessing these Census transformed pixels. Although computing the Census transform for all pixels is computationally wasteful, it may lower memory bandwidth per pixel because it can be implemented using a highly efficient sliding window based approach with deterministic memory access patterns.

In the proposed NG-fSGM, only a subset $\boldsymbol{O_p}$ of full flow candidate vectors needs to be evaluated. The calculation of C($\boldsymbol{p},\boldsymbol{o}$) can be performed on-the-fly only when $\boldsymbol{o}$ is selected during the cost aggregation step. For this approach, storing an array of precomputed pixel-wise matching cost C($\boldsymbol{p},\boldsymbol{o}$) is unnecessary. However, memory bandwidth required for Census transform per pixel is significantly higher because an efficient sliding window approach is no longer applicable. In Section III. E, we quantify the tradeoff space between pre-calculating Census transform vs. computing Census transform on selective pixels on-the-fly. Pre-calculating Census transform results in calculating the Census transform for unused pixels whereas computing Census transform on-the-fly loses (because of irregular pixel processing pattern) the computing and memory efficiency of sliding window based calculations.

C. Cost Aggregation and Flow Computation

In order to compute Z in (2) for cost aggregation (1), typical SGM-based methods store $L_r(\boldsymbol{p},\boldsymbol{o})$ in a line buffer array of size WPD. NG-fSGM stores only the best N flow vectors for each path and their costs. Thus, the line buffer array size is reduced to WPN. Since we selectively store the aggregated cost $L_r(\boldsymbol{p},\boldsymbol{o})$, the cost $L_r(\boldsymbol{p}\text{-}r,\boldsymbol{o})$ may not be available for $\boldsymbol{O_{p\text{-}r}}$ along a certain direction r. In that case, the aggregated cost is approximated by assigning

$$L_r(\boldsymbol{p}\text{-}r, \boldsymbol{o}) = \min_{\forall \boldsymbol{j}} L_r(\boldsymbol{p}\text{-}r, \boldsymbol{j}) + P_2 \quad (4)$$

To access forward scan results during the backward scan, the original SGM-based method stores the aggregated cost $S(\boldsymbol{p}, \boldsymbol{o})$ for all searched flow vectors in an array of size *WHD*, and updates the values by accumulating path-wise aggregated cost $L_r(\boldsymbol{p}, \boldsymbol{o})$ obtained during the backward scan. NG-fSGM avoids such a large memory usage by storing only the N best flow vectors $\boldsymbol{B_p}$ per pixel along with their corresponding aggregated cost $S_1(\boldsymbol{p}, \boldsymbol{o})$ from the forward scan. Similar to the $L_r(\boldsymbol{p}, \boldsymbol{o})$ line buffer handling, $S_1(\boldsymbol{p}, \boldsymbol{o})$ is approximated to $\min_{\forall \boldsymbol{j}} S_1(\boldsymbol{p}, \boldsymbol{j}) + P_2$ when it is not available for backward scan aggregation. As a result, the array size for storing $S_1(\boldsymbol{p}, \boldsymbol{o})$ is reduced from *WHD* to *WHN*.

For each pixel $\boldsymbol{p}$ visited during the backward scan, the set of N best vectorst $\boldsymbol{B_p}$, from forward scan and the neighbor-guided vectors from backward scan, $\boldsymbol{O_{p2}}$, may be dissimilar. For vectors whose cost has not been calculated in either the forward or the backward scan, the following rules are applied to approximate the final aggregation cost $S(\boldsymbol{p}, \boldsymbol{o})$: The missing costs $S_1(\boldsymbol{p}, \boldsymbol{o})$ in forward scan are assigned the maximum cost in $\boldsymbol{B_p}$ plus P_2, while the missing costs in backward scan $S_2(\boldsymbol{p}, \boldsymbol{o})$ are assigned the maximum cost in $\boldsymbol{O_{p2}}$. The overall cost $S(\boldsymbol{p}, \boldsymbol{o})$ is the sum of cost from two scans. Finally, the output flow vector $\boldsymbol{o}$ is the one corresponding to the minimum cost $S(\boldsymbol{p}, \boldsymbol{o})$.

D. Post Processing

After the raw flow results are computed, post-processing steps are applied to refine the flow image. We apply a simple 3×3 median filter on both horizontal and vertical components of the flow image to remove errors and smoothen flow fields. If the accuracy requirement is high, a consistency check between previous and current frame (similar to left-right check for stereo [11]) can be applied to create the confidence map.

E. Complexity Analysis

Next, we analyze the complexity of the proposed NG-fSGM with respect to memory size, memory access bandwidth and number of operations. The key parameters in the tables are: image size (W×H), number of best flow vectors (N) per aggregation path, number of random flow vectors (M), number of aggregation paths per scan (P), window size (K) to select adjacent candidates (A points in Fig. 1) surrounding each best candidate (B point in Fig. 1) guided by a neighbor, the maximum flow search range (d^2), and Census transform window size (C×C). The maximum search subset size per pixel (i.e., the number of elements in $\boldsymbol{O_p}$) is (KN(P+1)+M). The Census transform for each pixel requires $(C^2\text{-}1)/8$ bytes while each flow vector requires 4 bytes; 1 byte for the horizontal component, 1 byte for the vertical component, and 2 bytes for storing the (aggregated) cost.

Table I summarizes the memory size, number of operations and memory bandwidth requirement for two possible implementation options: one is selectively computing Census transform on-the-fly for vectors included in the search subset $\boldsymbol{O_p}$, and the other option is to pre-compute and store Census transform for all pixels to benefit from the deterministic sliding window approach. If Census transform is computed on-the-fly, two (previous and current) grayscale input images need to be stored. In addition, a small temporary memory is necessary to store Census transform of matching candidate pixels. Meanwhile, when Census transform is pre-computed, Census transform results for the entire current and previous images are stored in the memory instead of raw greyscale images.

The middle panel of Table I show the number of operations to process an image frame of size W×H, and the right panel of Table I summarizes the memory bandwidth requirement represented by the total number of read/write Bytes for a W×H sized frame. Memory bandwidth requirement also depends on whether Census transform is computed on-the-fly or precomputed. Although precomputing Census allows for an efficient sliding window approach, most Census transform results would not be needed for matching cost evaluation. As can be observed from the table, memory bandwidth is dominated by accesses required to perform pixel-wise matching (Census and Subset Selection rows combined). It is evident that Census transform on-the-fly vs. pre-computing Census transform poses tradeoffs in the number of computations, memory size, and memory bandwidth requirements.

TABLE I. MEMORY SIZE, NUMBER OF OPERATIONS, MEMORY BANDWIDTH REQUIREMENT OF NG-FSGM FOR PROCESSING AN IMAGE OF SIZE (W×H)

	Memory size (Bytes per frame)		***Number of Operations***		***Memory access bandwidth (R/W Bytes per frame)***	
Data Type	***Census on the fly***	***Pre-compute Census***	***Census on the fly***	***Pre-compute Census***	***Census on the fly***	***Pre-compute Census***
Input Images	2WH	0				
Subset Selection			$2(P(KN+M)+(N(P+1)K+M)(2+P)+3d^2)WHS$		$(N(P+1)+M+1)2WH$	
Census Transform	$(KN(P+1)+M+1)(C^2\text{-}1)/8$	$2WH(C^2\text{-}1)/8$	$((N(P+1)K+M)(9+KC^2))WHS$	$2C^2WH$	$(K+8(C\text{-}1/2)^2(N(P+1)+(M+1))+8(C\text{-}1/2)^2)WH$	$WH((NK(P+1)+M+1)2C^2+8)/8$
Pixel-wise Matching cost	$4(KN(P+1)+M)$		$((N(P+1)K+M)(2C^2))WHS$			
Path-wise Aggregated Cost	$4P(KN(P+1)+M)$		$(6P(N(P+1)K+M)(N+3))WHS$		4NPWH	
Summated Aggregated Cost	$4(KN(P+1)+M)$					
Path-wise Best flow	4WPN		$(KN(P+1)+M)P(4N+2)WHS$			
Scan-wise Best flow	4WHN					
Flow Map	2WH		$(4(KN(P+1)+M)N+3(KN(P+1)+M))WHS$		4NWH	

IV. Optimizations for hardware-efficient NG-SGM

NG-fSGM allows optical flow complexity reduction with aggressive search space pruning. However, its complexity could still be excessive for power-constrained real-time mobile applications. According to the analysis provided in Section III, for full HD (1920×1080) resolution, NG-fSGM requires ~20MB of memory, ~0.168TOPs performance and 3.8Tb/s memory bandwidth to achieve a throughput of 30fps. To enable low power, high throughput mobile optical flow estimation with high accuracy, we propose NG-fSGM-specific optimizations. Proposed techniques include: 1) performing overlapping-block based NG-fSGM in parallel 2) initializing flow search space with temporal prediction. 3) performing sparse flow estimation with NG-fSGM and interpolating results to obtain dense flows.

A. *Parallel Block-based NG-fSGM*

We propose parallel block-based NG-fSGM to enable reduction in the overall power consumption by processing each block at a lower frequency and voltage given throughput target. The memory space requirement for block-based approach is proportional to the number of parallel-processed blocks and the block size instead of the image size. Other notable advantages include improved latency and throughput. Latency is proportional to the block size (instead of the image size) and the throughput linearly improves with the number of parallel block processing cores.

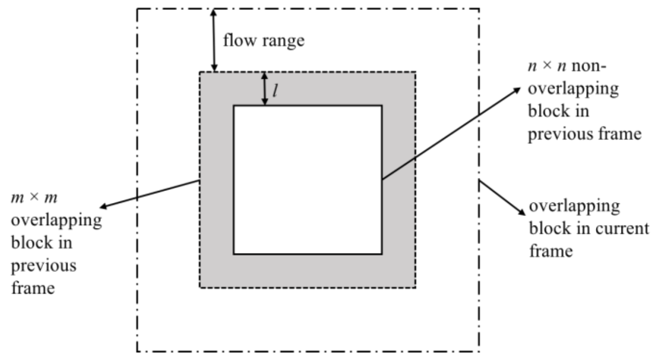

flow range
l
$n \times n$ non-overlapping block in previous frame
$m \times m$ overlapping block in previous frame
overlapping block in current frame

Fig. 2. An example of an $n \times n$ non-overlapping block in the previous frame. The corresponding $m \times m$ overlapping block in the previous frame is obtained by extending l pixels along four sides, and the corresponding overlapping block in the current frame is obtained by further extending d pixel along four sides.

In a naive block-based implementation, input frames are partitioned into non-overlapping blocks and the NG-fSGM algorithm is applied to each block in parallel. This non-overlapping block approach reduces the algorithm accuracy because of several factors. First, for the boundary pixels, NG-fSGM uses random selections to initialize the search subset so it takes some time (in term of pixel propagation) until ‘good’ (correctly guided by neighbors) flow vectors appear in neighbor-guided search subset. Second, since the aggregating paths accumulate information from boundary to inner pixels (because of the raster scan order), the cost aggregation is relatively unreliable at the boundary. Third, the flow smoothness constraint is interrupted at the boundary of two blocks when blocks are non-overlapping.

In order to improve the accuracy of block-based NG-fSGM, we impose flow smoothness constraints across the block boundary. In the proposed scheme, the ‘previous’ (or reference) frame is first divided into $n \times n$ non-overlapping blocks, and then each block is extended by l pixels along four sides, resulting in $m \times m$ overlapping blocks, where $m = n + 2l$. The ‘current’ (or target) frame is divided into overlapping blocks as well, but with block size of $m + 2d$ per dimension, where d is the flow search range per dimension. The output flow map of the entire image is built using the flow map of each $n \times n$ block. Fig. 2 shows an example of a non-overlapping block and its extensions.

Generally, a larger size of non-overlapping block and a larger number of extended pixels result in better flow accuracy, at the expense of higher latency, computational cost, and memory requirement. For a certain image size, the parameter n defines the number of blocks to be processed, and together with l, it defines the architectural complexity that is a function of the memory size, memory bandwidth and the number of operations. The latency is linear in the size of overlapping block, $m = n + 2l$. Table II summarize the effect of parameters n and l on the memory size, number of operations and memory bandwidth (in Bytes) requirements to process a W×H frame. Larger n and l both increase the memory, computation and memory bandwidth per block, though larger n reduces the number of blocks to be processed. While overlapping of the blocks can significantly improve the flow accuracy, it also increases the complexity of computing a whole frame from O(n^2) to O(m^2), where $m = n + 2l$. Detailed analysis on this tradeoff is presented in Section V.

B. *Inertial Flow Vector Prediction Using Sequential Frames*

Large overlapping regions in block-based processing reduce throughput while increasing memory size and bandwidth requirements. We observed that the size of overlapping region can be significantly reduced if the search space of the boundary pixels is initialized with good flow estimates/predictions, replacing random vectors. Our method is inspired by ‘inertial estimates’ proposed in [30]. Assume that each object in the frame moves at a constant velocity. Then the flow of pixels between time frames [t, t + 1] can be estimated/predicted from the flow for [t – 1, t]. Let (i, j) denote pixel position, and let $(u, v)_{(i,j)}$ and $(u', v')_{(i,j)}$ denote flow of pixel (i, j) between time frames [t, t + 1] and [t – 1, t], respectively. Then we assume that the relationship (5) holds.

$$(u, v)_{(i+u', j+v')} = (u', v')_{(i,j)} \tag{5}$$

We use (5) to provide the inertial guided flow estimates. For each pixel in the extended region (grey area in Fig. 2) in the $m \times m$ block, inertial guided flow estimates are more reliable, in general, than neighbor-guided flows especially when the neighbors are closer to the block boundary where flow vectors are randomly initialized. Thus, we replace the guided flow of one neighbor (which is closest to the boundary) and its (K-1) adjacent flow vectors with the inertial guided flow vector and its corresponding (K-1) adjacent flow vectors. The number of operations remains the same though small extra memory space is required for storing the inertial estimates. As we only store inertial estimates for boundary pixels, memory overhead is low (<2%). This approach helps significantly reduce the size of the extended region for overlapped block processing without algorithm accuracy degradation.

C. Sparse-to-Dense Optical Flow Estimation

To further reduce the number of operations of NG-fSGM, we propose to estimate dense optical flow from sparse optical flow using interpolation. In the proposed method, sparse flow vectors are computed by performing NG-fSGM on selective pixels. It is also worth noting that the proposed method is different from conventional subsampling approaches where optical flow is performed on a subsampled image. Instead, our proposed method operates on the full resolution image while the optical flow is only computed on selective pixel positions with patterns shown in Fig. 3

NG-fSGM aggregation and optical flow computation is initially performed on these subsampled pixels only. Once NG-fSGM is complete for selective (subsampled) pixels, dense optical flow of remaining pixels marked white in Fig. 3 is computed by interpolating the result of subsampled (black) pixels in Fig. 3. This approach is similar in spirit to [31]. While in [31], SGM (for stereo) is performed for every pixel, here aggregated costs are updated only on selected (subsampled) pixels; the other pixels have the same aggregated costs as the selected pixels.

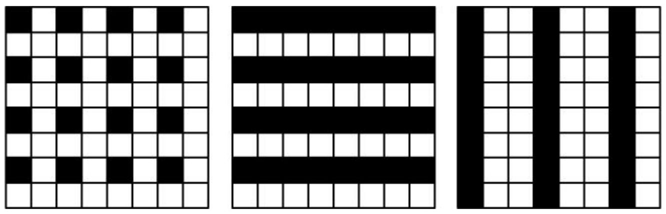


Fig. 3. Sampling pattern examples where grey pixels are sampled. Left: $f_1 = 1/2$, $f_2 = 1/2$; Middle: $f_1 = 1/2$, $f_2 = 1$; Right: $f_1 = 1$, $f_2 = 1/3$.

Different sampling patterns exemplified in Fig. 3 are governed by parameters f_1 and f_2, the horizontal and vertical sampling rates, respectively. The proposed sparse-to-dense NG-fSGM method performs interpolation in two steps. As neighboring pixels tend to have identical or similar motion, the flow vector $\boldsymbol{o_p}$ at pixel $\boldsymbol{p}$ is estimated by the bilinear interpolation of nearest subsampled (i.e., black pixels in Fig. 3) neighbors' optical flow vectors. This approach reduces the required memory size and the number of operations by a factor of f_1f_2. We summarize the impact of these hardware-oriented optimization on the memory size, number of operations, and memory bandwidth in Table II.

V. Complexity - Accuracy Tradeoff Analysis

We conducted comprehensive experiments on the Middlebury [10], KITTI [33] and MPI [32] optical flow benchmark to evaluate optical flow estimation accuracy and hardware implementation complexity. The accuracy is quantified in terms of the endpoint error percentage (EEP) with Middlebury (EEP radius 2) / KITTI (EEP radius 3) benchmark, and average endpoint error (EPE) on MPI dataset. The EEP is the percentage of pixels whose optical flow estimation error radius (error vector magnitude) is larger than a certain threshold (2 or 3 in our case) [10]. The EPE is the averaged error radius of optical flow estimates of all pixels [32]. The memory size, number of computations and memory bandwidth requirements are used to quantify the hardware complexity.

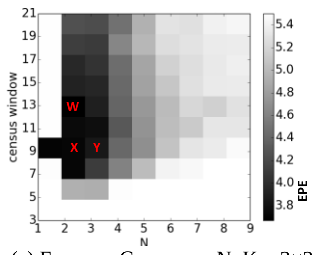
A. Parametric Analysis on NG-fSGM

We evaluate the effect of multiple algorithm parameters, namely, N, M, C, K, P, n, l, f_1, and f_2. Some of the parameters have correlated impact on algorithm accuracy. First, we analyze the impact of N, M, C and K when P is fixed. For now, block-based approach and sparse-to-dense interpolation are disabled to simplify the analysis. In Fig. 4, parameters N and M are enumerated from 0 to 9, and C^2 is evaluated from 5×5 to 19×19. Fig. 4 (a) and (c) visualize the algorithm accuracy for K = 1×1 and 2×2 respectively on Middlebury test in EPE (the darker, the more accurate) when M = 3 and P = 8. Fig. 4 (b) and (d) show the impact of N and M when C = 11, and P = 8 for K = 1×1 and 2×2, respectively.

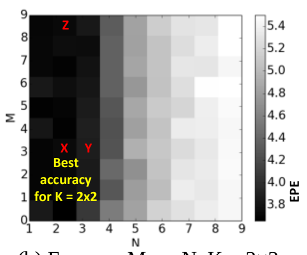
TABLE II. Memory size, Number of Operations, Memory bandwidth of NG-fSGM with optimizations

	Memory size		***Number of Operation***		***Memory access bandwidth***	
Data Type	***Census transform on the fly***	***Pre-compute census map***	***Census transform on the fly***	***Pre-compute census map***	***Census transform on the fly***	***Pre-compute census map***
Input Images	$n^2 + (n+2l+2d)^2$	0				
Subset Selection			$2f_1f_2(P(KN+M)+(N(P+1)K+M)(2+P)+3d^2)WH(n+2l)^2 S/n^2$		$f_1f_2(N(P+1)+M+1)2WH(n+2l)^2/n^2$	
Census Transform	$f_1f_2(KN(P+1)+M+1)(C^2-1)/8$	$(n^2+(n+2l+2d)^2)(C^2-1)/8$	$f_1f_2((N\times(P+1)K+M)(9+KC^2)) WH(n+2l)^2S/n^2$	$f_1f_2C^2WH (n^2+(n+2l+2d)^2)/n^2$	$f_1f_2(K+8(C-1/2)^2N(P+1)+(M+1)+8(C-1/2)^2) WH(n+2l)^2/n^2$	$f_1f_2WH(n+2l)^2/n^2(N(P+1)+M+1)2C^2/8$
Pixel-wise Matching cost	$4f_1f_24(KN(P+1)+M)$		$f_1f_2((N(P+1)K+M)2C^2)WH(n+2l)^2S/n^2$			
Path-wise Aggregated Cost	$4f_1f_2P(KN(P+1)+M)$		$f_1f_2(6P(N(P+1)K+M)(N+3))WH(n+2l)^2S/n^2$		$f_1f_2N4PWH(n+2l)^2/n^2$	
Summated Aggregated Cost	$4f_1f_2(KN(P+1)+M)$					
Path-wise Best flow	$4f_1f_2(n+2l)PN$		$f_1f_2(KN(P+1)+M)P(4N+2)WH(n+2l)^2S/n^2$			
Scan-wise Best flow	$4f_1f_2(n+2l)^2N$					
Flow Map	$2f_1f_2(n+2l)^2$		$f_1f_2(4(KN(P+1)+M)N+3(KN(P+1)+M))WH(n+2l)^2S/n^2$		$f_1f_2N4WH(n+2l)^2/n^2$	
Inertial Map	$8f_1f_2(n+2l)$					

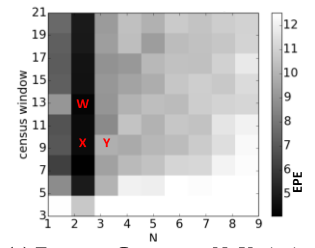
Fig. 4 (a) and (b) show that, when the search window size K is 2×2, the optimal accuracy is obtained with N = 2 or 3. Notice that more best-candidates (larger N) from the neighbor degrades the algorithm accuracy because the smoothness constraints weakens when more (some are incorrect) candidates are admitted. Regarding the Census size, the algorithm accuracy stabilizes when C is larger than 9 but degrades when the Census size is too large (e.g., C ≥ 17) possibly because of dissimilar optical flows within the Census window. Analysis confirms that the number of random vectors (M) is relatively insensitive when other parameters are chosen optimally and $M \geq 1$.



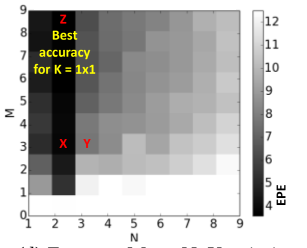
(a) Error vs. Census vs. N, K = 2×2



(b) Error vs. M vs. N, K = 2×2



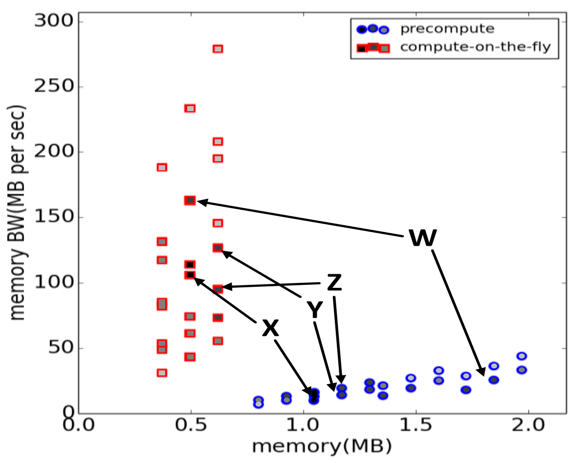
(c) Error vs. Census vs. N, K=1×1



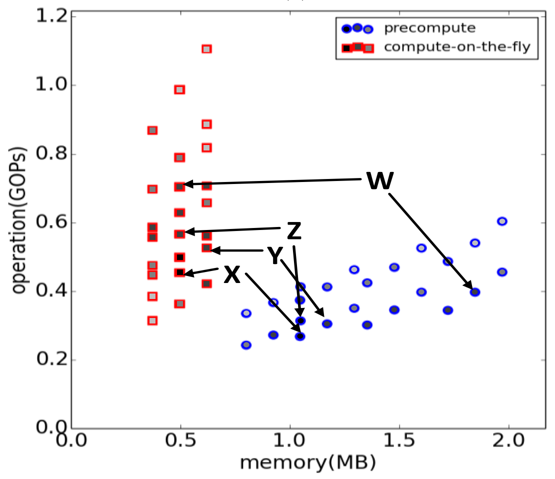
(d) Error vs. M vs. N, K = 1×1

Fig. 4. Error performance analysis with parameter sweep (M, C, K, N).

In Fig. 4 (c) and (d), K is changed from 2×2 to 1×1. It is worth noting that the impact of parameter N, M and C on algorithm accuracy shows similar non-monotonic trend as in Fig. 4 (a) and (b). With a smaller K, more random candidates with M ⩾ 3 are desired to compensate for the reduced number



(a)



(b)

Fig. 5. Performance and complexity tradeoff analysis: (a) Memory BW vs. Memory size vs. Error rate, (b) # of OPs vs. Memory size vs. Error rate.

of neighbor guided candidates, and thus to minimize the

TABLE III. SENSITIVITY ANALYSIS ON PERFORMANCE AND COMPLEXITY

	Trend as variable increases							***Insight***
	Sensitivity analysis on accuracy and complexity	***Census on-the-fly***			***Pre-compute Census***			
		OPs	***Mem Size***	***Mem BW***	***OPs***	***Mem Size***	***Mem BW***	
Census window (C×C)	Accuracy improves as C increases from 3 to 7. Complexity is in general proportional to C^2. Further increasing C > 7 will saturate the accuracy and eventually degrade the accuracy.	$O(C^2)$	$O(C^2)$	$O(C^2)$	$O(C^2)$	$O(C^2)$	O(C)	Improvement from a large C diminishes when C>7 as the larger window tends to contain pixels with distinct optical flows. C quadratically increases memory size and operation. Memory access bandwidth increases linearly if census map is precomputed.
N	Accuracy is not monotonic to N. Modest N (N = 1 or 2) typically works better than larger N values. OPs proportional to $O(N^2)$ while memory requirement is O(N).	$O(N^2)$	$O(N)$	$O(N)$	$O(N^2)$	$O(N)$	$O(N)$	Larger N helps in finding good candidates and thus, improving the time of convergence (in terms of pixels). However, large N may introduce unnecessary ambiguity that lead to errors in flow map.
M	For K=1×1: M > 0 is strongly desired. A larger M > 1 won't provide significant gain. For K >= 2×2, accuracy is insensitive to M.	$O(M^2)$	~0	~0	$O(M^2)$	~0	~0	Larger M can compensate smaller K by introducing additional matching candidates in flow subset. If K is large, extra random positions is unnecessary (M = 0 is acceptable). Impact of M on overall complexity is insignificant compared to other parameters (C, N, K).
Search window (K×K)	K = 2×2 provides better accuracy than K = 1×1 or 3×3. Complexity is proportional to $O(K^2)$.	$O(K^2)$	~0	$O(K^2)$	$O(K^2)$	~0	$O(K^2)$	Optimal K exists for the best accuracy. A small K may result in insufficient number of candidates while a large K may introduce unnecessary ambiguity from too many candidates. A small K can be compensated by a larger M and vice versa. A small K with a larger M is preferred to a larger K with a smaller M because the impact of M on complexity is lower.

algorithm accuracy degradation. Point 'Z' (N = 2, M = 9, C = 9 with K = 1×1) in Fig. 4. (a) has the best accuracy of 3.69% EPE comparable to the point 'X' in Fig. 4 (d) (3.67% EPE, with N = 2, M = 3 , C = 9 and K = 2×2).

The algorithm complexity in terms of memory size, memory bandwidth and operation counts is visualized in Fig. 5 (a) and (b) when K=2×2. Analysis for K=1×1 is omitted as it exhibits a similar trend. The comparison between Census transform on-the-fly and pre-computed Census is also analyzed in the same figures. The algorithm accuracy – complexity tradeoff can be identified by associating 'X, Y, Z, W' points in Fig. 4c, d to the same labeled points in Fig. 5. Points in Fig. 5 are shaded with different levels to represent algorithm accuracy; darker shades represent better algorithm accuracy following the same convention in Fig. 4. One can observe the impact of larger Census size (from point 'X' to 'W') that allows memory size vs. bandwidth vs. operations tradeoff depending on whether Census transform is pre-computed (red) or calculated on-the-fly (blue). Horizontal shift of 'X' $\rightarrow$ 'W' indicates memory size increase while vertical shift implies more memory bandwidth and number of operations.

Notice that all parameters have monotonic relationship with the algorithm complexity while their impact on algorithm accuracy is non-monotonic. Therefore, finding an optimal tradeoff point is a non-trivial task. The impact of parameter C and N on both algorithm accuracy and complexity is in general more significant than that of other parameters. For the pre-computed Census approach, a large C could result in an excessive memory size requirement, as it is a function of C^2. The number of operations and memory requirements for the on-the-fly Census approach is proportional to NC^2. A reasonable complexity-accuracy tradeoff can be made with C = 9, N = 1 and M = 1 given K = 2×2 and P = 8. In this case, computing Census on-the-fly reduces memory size requirement to ~0.5MB at the cost of ~60% more memory bandwidth and computation compared to the pre-computed Census option.

The impact of various parameters on algorithm accuracy and complexity is summarized in Table III. A smaller N implies more aggressive candidate pruning and slower convergence of the flow propagation among neighbors, but also avoids over-constraining smoothness constraints that could lead to flow error. A larger M can compensate for small search window size K and can remain small if N or K is large. Based on the exhaustive analysis on Middlebury dataset, we picked the parameter set that provides balanced hardware complexity and algorithm accuracy. For the rest of the section, we use the following parameters: K = 2×2, C = 9, N = 1, P = 8, M = 1.

Table IV compares the accuracy and complexity of NG-fSGM with the aforementioned parameter set to the original fSGM [21] and Lucas-Kanade (LK) [13], for the Middlebury, KITTI and MPI benchmarks. NG-fSGM with K = 2×2, C=9, N=1, P=8, M=1 is used for all three benchmarks. As Table IV indicates, the proposed NG-fSGM provides significant complexity reduction compared to fSGM since NG-fSGM only evaluates <10% flow vectors and aggressively prunes the others to avoid strict regularization of fSGM. Recall that the complexity of fSGM quadratically increases with the search range d while the proposed NG-fSGM complexity is independent of d. Middlebury dataset used for Table IV has a limited d $\leq$ 32. The complexity gap between fSGM and NG-fSGM is more significant for MPI [32] and KITTI [33] benchmarks that require a larger d. The fSGM and LK complexity for KITTI and MPI reported in Table IV is based on a three-level hierarchical pyramid approach [21] that limits d to 40 (fSGM) or uses a 13x13 search window with 10 iterations (LK) for each level. Non-hierarchical fSGM (d=40) and LK (13x13, 10 iterations) is used for the Middlebury . We observed that NG-fSGM significantly outperforms LK in accuracy at the cost of slightly increased memory area requirement. The number of operations for NG-fSGM is lower than that of LK for all benchmarks.

To visualize the algorithm quality difference, Fig. 6 shows

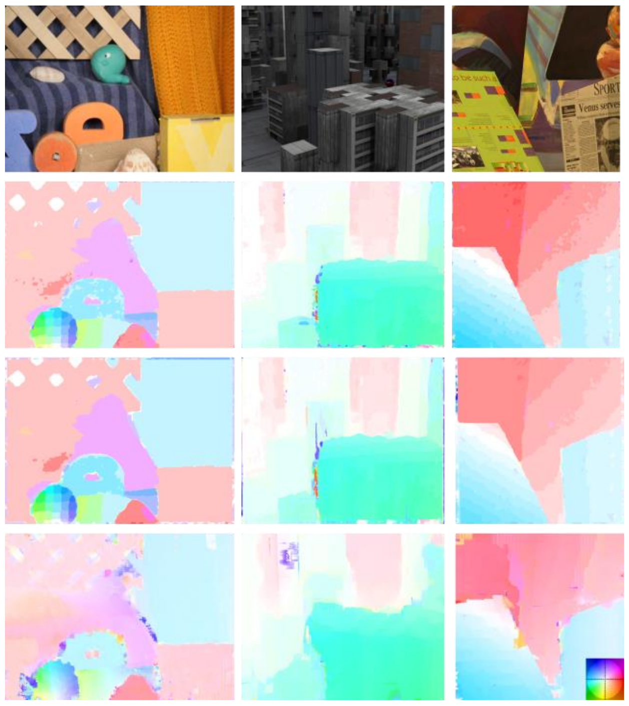


Fig. 6 Colored flow maps using different algorithms. 1st column: RubberWhale; 2nd column: Urban2; 3rd column: Venus. 1st row: input previous frame; 2nd row: NG-fSGM; 3rd row: fSGM; 4th row: Lucas-Kanade. Color legend is at bottom-left corner.

TABLE IV. COMPARISON OF NG-FSGM, FSGM & LUCAS KANADE

Metric	***EEP %***		***EPE***	***Memory Size (MB)***			***Number of Giga Operations***		
Dataset	***Middlebury***	***KITTI***	***MPI***	***Middlebury***	***KITTI***	***MPI***	***Middlebury***	***KITTI***	***MPI***
fSGM	4.54%	10.74%*	-	20.68	342.08**	331.21**	37.53	65.1**	63.48**
Lucas-Kanade	15.78%	28.91%	10.45	1.47	3.83	3.745	3.15	8.24	8.01
NG-fSGM	3.70%	11.37%	7.91	2.47	3.68	3.59	2.10	3.14	3.12

*: Reported in [21] for hierarchical fSGM using a 5×5 median filter.

**: Estimated from hierarchical fSGM [21] with d = 40.

the flow maps obtained from a Middlebury test image using each algorithm. The color of each pixel indicates the direction of the flow whereas the color intensity is proportional to the magnitude of the flow. NG-fSGM (2nd row) and fSGM (3rd row) both outperform LK (4th row). The raw (before post-processing) flow map output of NG-fSGM shows blurry results along object edges but has fewer error patches compared to fSGM. The neighbor guidance in NG-fSGM is less reliable at the object edges when aggressive search space pruning is applied. However, after a simple post processing of 3×3 median filtering, this difference becomes insignificant. Table IV confirms that the overall accuracy of NG-fSGM for Middlebury evaluation is comparable to that of fSGM.

For the KITTI and MPI dataset, the search range is substantially larger; d = 128 for a non-hierarchical approach. Unlike NG-fSGM, a multi-level hierarchical approach is preferred for LK and/or fSGM to limit d (e.g., 40) for each pyramid level. Fig. 7 and 8 show output flow images from KITTI and MPI for qualitative comparison. Proposed non-hierarchical NG-fSGM achieves 11.37% EEP on KITTI benchmark, which significantly outperforms pyramidal LK whose EEP is 28.91%. NG-fSGM exhibits 0.63% degradation compared with a hierarchical fSGM [21] while achieving >10× memory and computation complexity reduction. NG-fSGM achieves 7.91 average EPE (end point error) on MPI test cases significantly outperforming LK whose EPE is 10.45. Evaluation of hierarchical fSGM on MPI benchmark is not reported in [21] and so could not be included.

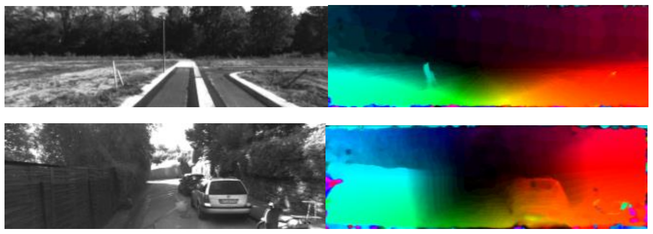


Fig. 7. Colored flow maps using KITTI dataset.

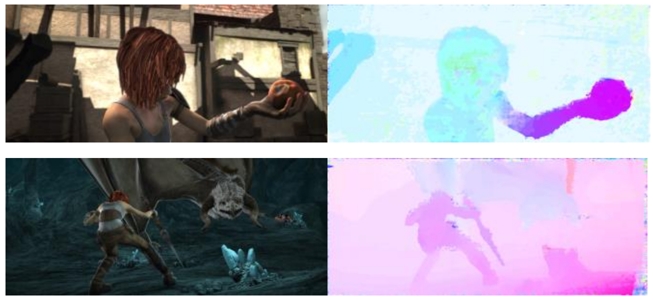


Fig. 8. Colored flow maps using MPI dataset.

B. Overlapping Block-based NG-fSGM

So far, for more direct comparison to other algorithms, NG-fSGM complexity and accuracy has been analyzed without additional hardware optimization techniques. In the following subsections, we discuss the impact of overlapped block-based processing, inertial guidance for flow initialization, and sparse-to-dense flow interpolation to further reduce hardware implementation complexity specifically for NG-fSGM.

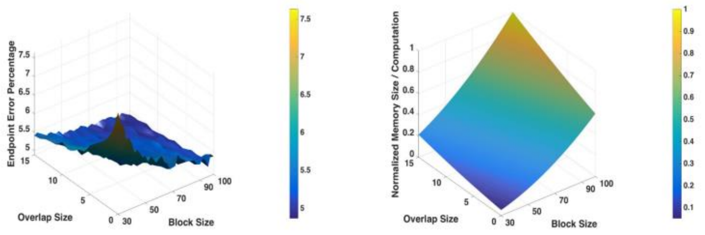


Fig. 9. Left: accuracy of NG-fSGM+B using the Middlebury training dataset. Right: complexity vs. overlap size (l) vs. block size (n).

We evaluate the basic overlapped block-based processing (denoted by NG-fSGM+B) by sweeping parameter n from 30 to 100 and l from 0 to 15. The accuracy is evaluated in EPE (radius = 2 pixels) for the Middlebury dataset. In Fig. 9, it is evident that a larger n or l monotonically improves algorithm accuracy but the gain diminishes (approaches the original algorithm accuracy without block-based processing) when n and l are around 75 and 16, respectively. Fig. 9 also quantifies the memory size increase for supporting larger n or l. We observe that n = 64 provides reasonable tradeoff between algorithm accuracy and complexity. In later subsections we assume n = 64 unless stated otherwise.

C. Block-based NG-fSGM with inertial guidance

We evaluate the effect of the proposed inertial guidance technique on Middlebury dataset. Fig. 10 shows an example of inertial guidance for the flow estimate of the current frame. The average accuracy of inertial estimates from [t – 1, t] is 10.13% in EEP for Middlebury dataset, which is ~6% lower than that of NG-fSGM. This implies that the inertial estimates can provide useful guidance for flow vector initialization in NG-fSGM. Table V shows the impact of inertial guidance where NG-fSGM+BI denotes the method combining overlapped block-based processing and inertial guidance. The inertial guidance consistently improves the accuracy especially for images with a relatively large flow range (Grove3, Urban2 and Urban3). With inertial guidance, the overlap size l can be reduced from 16 to 2 with negligible loss in accuracy (see second and fourth columns of Table V). This also helps to achieve lower architectural complexity; memory size is reduced by 85% from 0.65MB to 0.38MB and the number of operations is reduced by 93% from 0.058 to 0.028 GOPs per block. This significant complexity reduction is feasible because inertial guidance reduces number of overlapping pixels.

Note that block-based NG-fSGM cannot always resolve the ambiguity from multiple flow candidates for images with repeated patterns that span the entire block (e.g., Urban3). NG-fSGM without block partitioning propagates flow vectors globally beyond the block boundary, thus it can resolve local (within a block) ambiguity by aggregating the cost from the region where repeated patterns no longer exist. Inertial guidance combined with block-based NG-fSGM show improved results for resolving local ambiguity (e.g., Urban3) by utilizing additional temporal guidance. However, if images do not have strong local ambiguity, block-based NG-fSGM performs relatively well within each block.

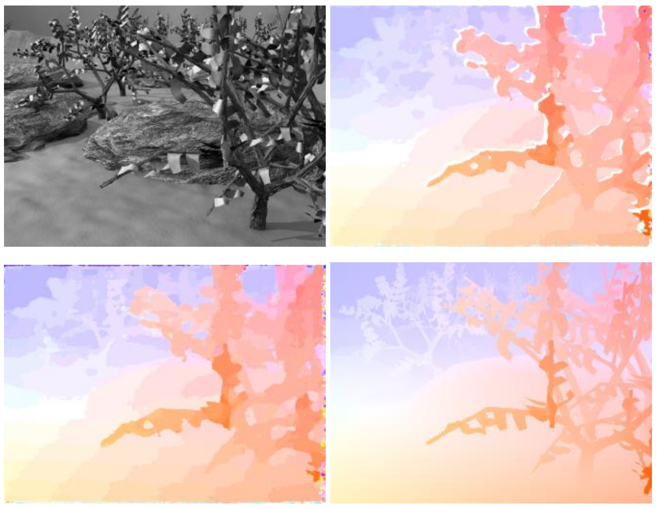


Fig. 10. An example of inertial guidance in Grove 3. Top left: Frame at time *t*. Top right: Inertial estimates from [*t* - 1, *t*]. Bottom left: Output flow using NG-fSGM+BI when *l* = 4. Bottom right: Groundtruth flow.

While the original NG-fSGM (processing the whole image as a single block) has a mean error of 4.43%, NG-fSGM+BI has a mean error of 4.72% with n = 64 and l = 2, exhibiting 0.29% average accuracy degradation. For individual data images, the accuracy difference ranges from -0.09% to 3.14%.

TABLE V. ENDPOINT ERROR PERCENTAGE (EEP) AND COMPLEXITY ON MIDDLEBURY MULTI-FRAME TRAINING DATASET

Image for EEP evaluation	***NG-fSGM***	***NG-fSGM+ B (l=2)***	***NG-fSGM+ B (l=16)***	***NG-fSGM + BI (l=2)***
Grove2	2.03	1.82	1.77	1.74
Grove3	8.35	7.85	7.77	7.66
Hydrangea	0.99	0.80	0.79	0.73
RubberWhale	0.71	0.88	0.67	0.56
Urban2	4.81	5.09	4.45	4.82
Urban3	9.70	18.52	16.14	12.84
EEP Mean	4.43	5.83	4.93	4.72
# of blocks	1	75	75	75
Memory size per block (MB)	2.47	0.38	0.65	0.39
Operations per block (10^9)	2.10	0.028	0.058	0.029
Memory access per block(MB)	88.8	1.28	2.543	1.31

D. Sparse-to-Dense Optical Flow Estimation

The last optimization technique we evaluate is the sparse-to-dense NG-fSGM with different sampling rates f_1 and f_2. The sparse-to-density NG-fSGM is evaluated on Middlebury images with n = 64, and *l* = 16. Table VI confirms that significant reduction in the memory size and number of operations requirement is feasible with only a modest accuracy degradation. The subsampling rate of f_1 = 1/2 and f_2 = 1/2 provides a reasonable tradeoff between accuracy and complexity. The memory size is reduced by 40% and number of operations is reduced by 75% with 1.23% increase in error percentage.

TABLE VI. ENDPOINT ERROR PERCENTAGE (EEP) AND COMPLEXITY ON MIDDLEBURY MULTI-FRAME TRAINING DATASET

	Original NG-fSGM	***NG-fSGM f_1=1/2, f_2 = 1***	***NG-fSGM f_1=1, f_2 = 1/2***	***NG-fSGM f_1=1/2, f_2 = 1/2***
EEP Mean	3.69	4.08	4.20	4.91
Memory size per block (MB)	0.38	0.27	0.27	0.23
Operations per block (10^9)	0.28	0.14	0.14	0.07
Memory access per block(MB)	1.31	0.66	0.66	0.33

E. Post-Processing

Different post processing techniques are evaluated for tradeoff study in low complexity optical flow computation. We analyze and compare the complexity and accuracy between 5×5 median filter and 25×25 weighted median filter (WMF) [34]. Results are shown in Table VII. With pyramidal LK, the weighted median filter outperforms the median filter by 1.5% on KITTI benchmarks. When the raw flow rate is more accurate, the improvement from weighted median filter starts to become marginal compared to a simple median filter. Pyramidal LK is still not able to meet NG-fSGM accuracy even when a 25×25 weighted median filter is applied. Applying this weighted median filter introduces ~25% extra computation for pyramidal LK.

VI. HARDWARE ARCHITECTURE AND PERFORMANCE

We propose a hardware architecture that accelerates the block-based NG-fSGM to achieve high throughput and energy efficiency for power critical real-time mobile systems. We employ Census transform on-the-fly approach instead of pre-computing Census as the memory size overhead for the latter is significant while the computation overhead is manageable for the low power application that we are targeting. Fig. 11 shows the overview of proposed parallel processing

TABLE VII. OUTLIER PERCENTAGE (EEP) WITH DIFFERENT POST PROCESSING SCHEME

	LK		LK with WMF		NG-fSGM		NG-fSGM with WMF	
	EEP %	*Number of Giga Operations*	EEP %	*Number of Giga Operations*	EEP %	*Number of Giga Operations*	EEP %	*Number of Giga Operations*
Midderbury	8.31	3.15	7.41	4.11	3.70	2.10	3.72	3.06
KITTI	28.9	8.24	27.4	9.67	11.3	3.14	11.2	4.57

hardware architecture that employs 8 parallel cores along with a global image buffer of size 0.5MB. The design is scalable to higher resolution images by instantiating a larger memory and more cores in the system.

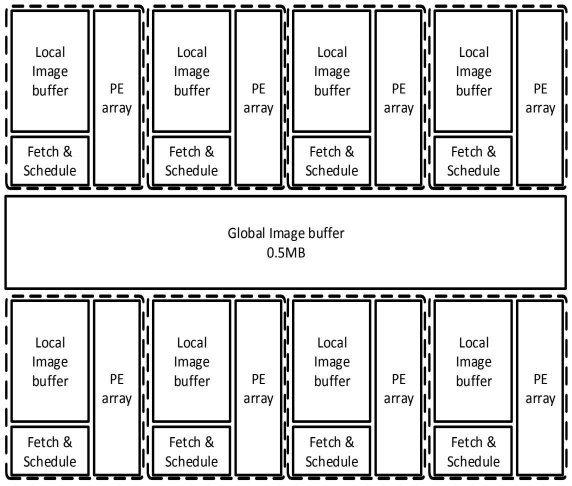


Fig. 11 Architecture of the multicore optical flow processor.

In this 8-core system, each core has its own local image buffer, local processing unit, local control unit and result memory. Raw image is stored in global memory and is transferred sequentially to each core. Each core processes the image block and transfers the results back to the global memory. Memory bandwidth from the global memory to each core is 11MB/s. Local memory inside each core has much high memory bandwidth and will be discussed later.

Fig. 12 shows the overall processing procedure of a single core and Fig. 13 details block diagram of each core. The memory consists of a buffer to store the 88kb flow output and a local image buffer of size 382 kb used to store the image block of size 75×75 with search range (±64 pixels) and overlapping regions (l=2 pixels). Note that by utilizing inertial guidance technique at the block boundary, the overlap region reduced from 16 to 2 pixels resulting in a reduction in the local image buffer size. The fetch-and-schedule finite state machine (FSM) can either fetch potential matching candidates that are guided by inertial estimate of neighboring pixels from the flow memory or fetch random candidates, and store them into the five 4056b shift register arrays. When all potential candidates are fetched, Census transform and pixel-wise matching costs are computed. Fetch & schedule FSM is also programmed to control the traversal pattern in a block so that certain pixels can be omitted to perform the sparse-to-dense NG-fSGM.

The processing unit (PE) array aggregates incoming pixel-wise matching cost with the aggregated costs of previous pixels stored in the row buffer (5850b). When aggregation on a single path finishes, optimal values are written back to the 5850b row buffer, and also forwarded to the fetch-and-scheduler for prefetching. The PE array runs at 20MHz to match the local memory bandwidth, and delivers 2.5M flow vectors per second. With 8 cores in total, the proposed design is able to achieve optical flow processing of over 60fps for VGA streams.

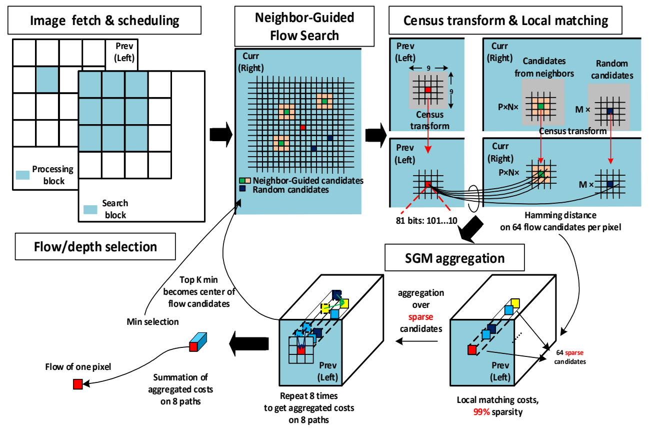


Image fetch & scheduling
Prev (Left)
Processing block
Search block
Neighbor-Guided Flow Search
Curr (Right)
Neighbor-Guided candidates
Random candidates
Census transform & Local matching
Prev (Left)
Census transform
Curr (Right)
Candidates from neighbors
Random candidates
P×N×
M×
Census transform
81 bits: 101…10
Hamming distance on 64 flow candidates per pixel
Flow/depth selection
Top K min becomes center of flow candidates
Min selection
Flow of one pixel
Summation of aggregated costs on 8 paths
Repeat 8 times to get aggregated costs on 8 paths
SGM aggregation
aggregation over sparse candidates
64 sparse candidates
Local matching costs, 99% sparsity

Fig. 12. Processing procedure of a single core

Detailed implementation of a PE array is provided in Figure 14. It consists of 138 PEs (to support up to 138 matching candidates given by $KN(P+1)+M$ where $K=3\times3$, $M=3$, and $P=4$) to enable parallel evaluation and aggregation over all matching candidates. In each PE (gray box), each candidate flow vector is compared with N-best (N=3) flow vectors of previous pixel, and P_1, P_2 (penalties) are added to the raw matching cost as in (2) depending on the distance between flow candidate and previous flow vectors. Aggregation on each path is performed separately. After costs on each matching candidate are aggregated, the aggregated costs are fed to a tree of comparators (path optimal selection unit in Figure 14) to generate 3 path-wise optimal flows vectors. These flow vectors are stored in the row buffer and sent to the fetch-and-schedule FSM for prefetching. Meanwhile, aggregation on another path is performed in parallel with prefetching on a different path. Using this path-interleaved processing with prefetching, performance is improved by 4× compared to a sequential processing without pipelined path interleaving. After aggregation on all paths is completed, path-wise costs are summed together for computing the output optimal flow vector.

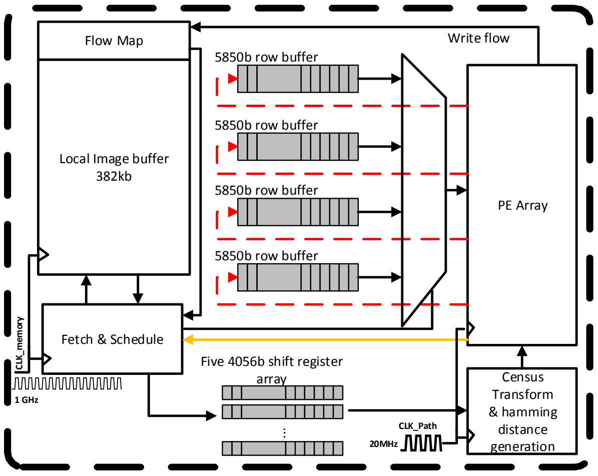


Fig. 13 Architecture of a single core

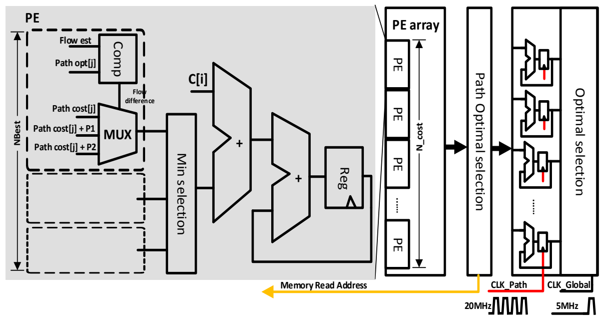


Fig 14. Detailed implementation of a PE array

Table VIII provides the estimated area breakdown of different components of the proposed hardware architecture synthesized in TSMC 28nm HPC technology. VDD of processing elements running at 20MHz can scale down to 0.7V to match the memory access bandwidth while memory banks are still operated at full VDD (0.9V).

TABLE VIII. AEAR AND POWER ESTIMATE OF A SINGLE PE

	Area(um^2)	Power
Image buffer	172380	12.6mW
Flow buffer	87196	56.29uW
PE array + Path optimal selection	234332	18.8mW
5850b row buffer	67106	4.8mW
Census & Hamming distance	205068	5.8mW
Optimal selection	187465	14.2mW
Fetch & scheduler	4280	0.66mW
4056b Register array	45830	3.5mW
Core Total	**1204975**	**74.76mW**

A single core is also place and routed; Fig 15 shows the place and route (P & R) results. The core is split into two parts: the fetch & schedule FSM with the image memory and the PEs with the flow memory. P & R results for both blocks are shown respectively. Table IX provides simulated power results after P & R. We simulate the design with an image block and categorize the power associated with each processing step. Simulation vectors for each block are recorded, and are fed to the P & R tool for accurate power estimation that includes coupling and signal integrity analysis. Overall, the area and power for a single core is 0.56mm^2 and 84.9mW respectively. For the 8-core system, a throughput of 20M pixel/s (2.5M pixels/s per core) or 65 fps in VGA is achieved with power consumption (estimated) of 679.2 mW. Table X provides the comparison with a prior optical flow hardware implementation of LK [35]. Proposed design achieves 4× more optical flow range, significantly better accuracy, 2× more throughput, and 2× energy efficiency compared with [35]. The proposed NG-fSGM with hardware optimizations allows real-time optical flow processing with negligible power overhead compared to ~100W for an MAV [36].

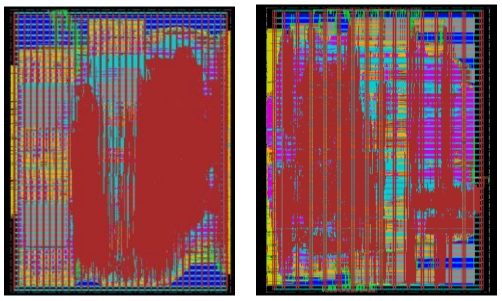


Fig 15. Automated place & route result of a core. (left: image memory with fetch & scheduler, right: aggregation units and flow memory)

TABLE IX. POWER RESULT OF A SINGLE PE POST ROUTE

	Power
Memory read	31.2mW
Local cost computation	10.2mW
Cost aggregation	35.2mW
Optimal flow search	8.3mW
Core Total	**84.9mW**

TABLE X. COMPARISON WITH PRIOR OPTICAL FLOW ASIC

	[35]	Proposed
Algorithm	Pyramid LK	NG-fSGM
Technology	65nm	28nm
Throughput	VGA 30 fps	VGA 60 fps
Optical flow range	-32 ~ 32	-64 ~ 64
Power	600 mW	679 mW

VII. CONCLUSION

This paper presents NG-fSGM, a low complexity method for optical flow. Dramatic complexity reduction is achieved by aggressively pruning the flow vector search space using information from neighbor pixels. The cost aggregation and flow computation steps have been optimized for further complexity reduction. NG-fSGM has been evaluated on the Middlebury and MPI optical flow datasets. The evaluation results show that NG-fSGM has comparable performance with an order of magnitude reduction in complexity compared to a prior work fSGM, and greatly outperforms other similar-costly methods like Lucas-Kanade.

Several hardware optimization techniques were introduced for the NG-fSGM optical flow algorithm. By dividing image into overlapping blocks, the full benefit of parallel processing is exploited with only a small degradation in accuracy compared to the original NG-fSGM. To further reduce the complexity of the overlapping block-based approach, temporal inertial guidance and sparse-to-dense interpolation, methods were proposed. Evaluation on Middlebury datasets shows that proposed block based algorithm has only 0.29% accuracy degradation compared to the original algorithm. The proposed algorithm is mapped onto a multicore architecture its throughput and power consumption evaluated in 28nm node to demonstrate the feasibility for real-time processing on power-critical MAV platforms.

VIII. References

[1] S.Yamamoto, M. Yasushi, S. Yoshiaki i, and J. Miura. "Realtime multiple object tracking based on optical flows." In IEEE International Conference on Robotics and Automation, vol. 3, pp. 2328-2333, 1995.

[2] V. Grabe, H. H. Bülthoff, and P. R. Giordano. "On-board velocity estimation and closed-loop control of a quadrotor UAV based on optical flow." In IEEE International Conference on,Robotics and Automation (ICRA), pp. 491-497, 2012.

[3] G. Bleser, and G. Hendeby. "Using optical flow as lightweight SLAM alternative." In 8th IEEE International Symposium on Mixed and Augmented Reality (ISMAR), pp. 175-176, 2009.

[4] N. Onkarappa, and A.D. Sappa. "Speed and texture: an empirical study on optical-flow accuracy in ADAS scenarios." IEEE Transactions on Intelligent Transportation Systems, 15, no. 1, pp. 136-147, 2014.

[5] B.K.P. Horn,and B. G. Schunck. "Determining optical flow." Artificial Intelligence 17, no. 1-3, pp. 185-203, 1981.

[6] H. Zimmer, A. Bruhn, J. Weickert, L. Valgaerts, A. Salgado, B. Rosen-hahn and H.-P. Seidel, "Complementary optic flow," Energy Minimization Methods in Computer Vision and Pattern Recognition, pp. 207-220, 2009.

[7] V. Lempitsky, S. Roth and C. Rother, "Fusion flow: discrete- continuous optimization for optical flow estimation," Computer Vision and Pattern Recognition, pp. 1-8, 2008.

[8] T. Cooke, "Two applications of graph-cuts to image processing," Digital Image Computing: Techniques and Applications, pp. 498–504, 2008.

[9] M Menze and A. Geiger, "Object Scene Flow for Autonomous Vehicles"; The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3061-3070, 2015

[10] S. Baker, D. Scharstein, J.P. Lewis, S. Roth, M.J. Black and R. Szeliski, "A Database and Evaluation Methodology for Optical Flow," International Journal of Computer Vision, vol. 92, pp. 1-31, 2011.

[11] H. Hirschmueller, "Stereo processing by semiglobal matching and mutual information," Pattern Analysis and Machine Intelligence, Vol. 30, pp. 328–41, 2008.

[12] C. Lei, Y .-H. Y ang, "Optical flow estimation on coarse-to-fine region-trees using discrete optimization," International Conference on Computer Vision, pp. 1562–1569, 2009.

[13] S. Baker and I. Matthews, "Lucas-Kanade 20 Years On: A Unifying Framework", International Journal of Computer Vision, vol. 56(3), pp. 221-225, 2004.

[14] A. Bruhn, J. Weickert and C. Schnörr, "Lucas/Kanade meets Horn/Schunck: combining local and global optic flow methods," International Journal of Computer Vision, Vol. 61(3), pp. 211–231, 2005.

[15] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. van der Smagt, D.Cremers, and T. Brox. "Flownet: Learning optical flow with convolutional networks." In IEEE International Conference on Computer Vision, pp. 2758-2766. 2015.

[16] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A.Dosovitskiy, and T. Brox. "Flownet 2.0: Evolution of optical flow estimation with deep networks." arXiv preprint arXiv:1612.01925 (2016).

[17] Z. Li, Q. Dong, M. Saligane, B. Kempke, S. Yang, Z. Zhang, R. Dreslinski, D. Sylvester, D.Blaauw, and H.-S. Kim. "3.7 A 1920× 1080 30fps 2.3 TOPS/W stereo-depth processor for robust autonomous navigation." In IEEE International Solid-State Circuits Conference (ISSCC), pp. 62-63, 2017.

[18] K. Pauwels, M. Tomasi, J. D. Alonso, E. Ros, and M. M. Van Hulle. "A comparison of FPGA and GPU for real-time phase-based optical flow, stereo, and local image features." IEEE Transactions on Computers, 61, no. 7 , pp.999-1012, 2012.

[19] J. Xiang, Z. Li, D. Blaauw, H.-S. Kim, and C. Chakrabarti. "Low complexity optical flow using neighbor-guided semi-global matching." In IEEE International Conference on Image Processing (ICIP), pp. 4483-4487, 2016.

[20] J. Xiang, Z. Li, H.-S. Kim, and C. Chakrabarti. "Hardware-Efficient Neighbor-Guided SGM Optical Flow for Low Power Vision Applications." In IEEE International Workshop on Signal Processing Systems (SiPS), pp. 1-6., 2016.

[21] S. Hermann and R. Klette, "Hierarchical scan line dynamic programming for optical flow using semi-global matching," Computer Vision-ACCV Workshops, pp. 556-567, 2012.

[22] R. Zabih and J. Woodfill, "Non-parametric local transforms for computing visual correspondance," European Conference on Computer Vision, pp. 151–158, 1994.

[23] P. Viola and W. M. Wells, "Alignment by maximization of mutual information," International Journal of Computer Vision, Vol. 24(2), pp. 137–154, 1997.

[24] B. Christian, P. Pirsch, and H. Blume. "Evaluation of penalty functions for semi-global matching cost aggregation." International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences [XXII ISPRS Congress, Technical Commission I] 39, Nr. B3. Vol. 39. No. B3. Göttingen: Copernicus GmbH, 2012.

[25] H. Heiko, M. Buder, and I. Ernst. "Memory efficient semi-global matching." ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences 3, pp. 371-376, 2012.

[26] C. Stiller, "Motion—Estimation for Coding of Moving Video at 8 kbit/s with Gibbs Modeled Vectorfield Smoothing" Proc. SPIE 1360, Visual Communications and Image Processing '90: Fifth in a Series, 468 (September 1, 1990).

[27] G. de Haan, P. W. A. C. Biezen, H. Huijgen and O. A. Ojo, "True-motion estimation with 3-D recursive search block matching," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 3, no. 5, pp. 368-379, Oct 1993.

[28] C. Barnes, E. Shechtman, A. Finkelstein and D. Goldman, "PatchMatch: a randomized correspondence algorithm for structural image editing" ACM Transactions on Graphics (Proc. SIGGRAPH) 28, 2009.

[29] H. Hirschmu ̈ller and D. Scharstein, "Evaluation of stereo matching costs on images with radiometric differences," IEEE Trans. Pattern Analysis Machine Intelligence, Vol. 31, pp. 1582–1599, 2009.

[30] B.D. Lucas and T. Kanade, "An iterative image registration techinique with an application to stereo vision," IJCAI, Vol. 81, pp. 674-679, 1981.

[31] H. Simon, S. Morales, and R. Klette. "Half-resolution semi-global stereo matching." Intelligent Vehicles Symposium (IV), pp. 201-206, 2011.

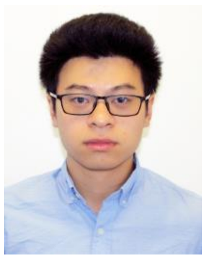
[32] B. Daniel J., et al. "A naturalistic open source movie for optical flow evaluation." European Conference on Computer Vision, Berlin, Heidelberg, pp. 611-625, 2012.

[33] G. Andreas, P. Lenz, and R. Urtasun. "Are we ready for autonomous driving? the kitti vision benchmark suite." IEEE International Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3354-3361, 2012.

[34] M. Ziyang, et al. "Constant time weighted median filtering for stereo matching and beyond." IEEE International Conference on Computer Vision (ICCV), pp. 49-56, 2013.

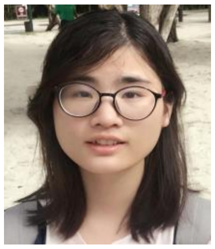
[35] I. Hajime, et al. "A VGA 30-fps optical-flow processor core based on Pyramidal Lucas and Kanade algorithm.", IEEE Asian Solid-State Circuits Conference, ASSCC'., pp. 188-191, 2007.

[36] https://www.dji.com/matrice100/info



Ziyun Li (S'14) received the B.S. degree in electrical and computer engineering from the University of Michigan, Ann Arbor, MI, USA, in 2014, where he is currently pursuing the Ph.D. degree with the Michigan Integrated Circuit Laboratory. His current research interests include highperformance, energy-efficient computer vision/ machine learning processing units to enable next generation intelligent, autonomous navigation system. Mr. Li was a recipient of the Best Paper Award at the 2016 IEEE Workshop on Signal Processing Systems.

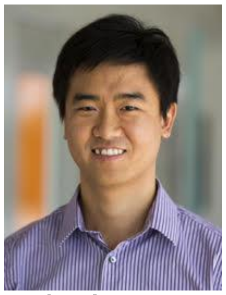
Jiang Xiang received his B.E. in Information Engineering from Xidian University in 2014 and his M.S. in Computer Engineering from Arizona State University in 2017. Currently he works for SAIC Innovation Center, where he is a software engineer in Autonomous Driving team. His research focuses on computer vision, machine learning, and algorithms optimization. He is a recipient of the 2016 SIPS Best Student Paper Award.



Luyao Gong is currently pursuing the Ph.D. degree in electrical and computer engineering with the University of Michigan, Ann Arbor, MI, USA. Her research interests include computer vision, low power image compression and machine learning.

David Blaauw received his B.S. in physics and computer science from Duke University in 1986 and his Ph.D. in computer science from the University of Illinois at Urbana-Champaign in 1991. Until August 2001, he worked for Motorola, Inc. in Austin, TX, where he was the manager of the High Performance Design Technology group and won the Motorola Innovation award. Since August 2001, he has been on the faculty of the University of Michigan, where he is a Professor. He has published over 550 papers, has received numerous best paper awards and nominations, and holds 60 patents. He has investigated adaptive computing to reduce margins and improve energy efficiency using a new approach he pioneered, called Razor, for which he received the Richard Newton GSRC Industrial Impact Award and IEEE Micro annual Top-Picks award. He has extensive research in ultra-low-power computing using subthreshold computing and analog circuits for millimeter sensor systems which was selected by the MIT Technology Review as one of the year's most significant innovations. For high-end servers, his research group introduced so-called near-threshold computing, which has become a common concept in semiconductor design. Most recently, he has pursued research in cognitive computing using analog, in-memory neural-networks. He was general chair of the IEEE International Symposium on Low Power, the technical program chair for the ACM/IEEE Design Automation Conference, and serves on the IEEE International Solid-State Circuits Conference's technical program committee. He is an IEEE Fellow and received the 2016 SIA-SRC faculty award for lifetime research contributions to the U.S. semiconductor industry.

Chaitali Chakrabarti is a Professor with the School of Electrical Computer and Energy Engineering, Arizona State University (ASU), Tempe, and a Fellow of the IEEE. She received the B.Tech. degree in electronics and electrical communication engineering from the Indian Institute of Technology, Kharagpur, India, in 1984, and the M.S. and Ph.D. degrees in electrical engineering from the University of Maryland, College Park, in 1986 and 1990, respectively. Her research interests include VLSI algorithm-architecture codesign of signal processing and communication systems, portable medical imaging, design of low power embedded systems, and emerging memory technology based system design. She has won several best paper awards in signal processing and computer architecture conferences. She is a Distinguished Alumnus of ECE, University of Maryland, College Park, USA, and Indian Institute of Technology, Kharagpur, India.



Hun Seok Kim (S'10 -- M'11) received his B.S. degree from the Seoul National University (South Korea), and M.S. & Ph.D. degrees from the University of California, Los Angeles (UCLA), all in Electrical Engineering. He is currently an assistant professor at the University of Michigan, Ann Arbor. His research focuses on system analysis, novel algorithms, and efficient VLSI architectures for low-power / high-performance wireless communication, signal processing, computer vision, and machine learning systems. Before joining the University of Michigan, Kim worked as a technical staff member at Texas Instruments (2010 – 2014). He is serving as an associate editor of IEEE Transactions on Green Communications & Networking, and IEEE Solid State Circuits Letters. Kim is a recipient of the 2018 Defense Advanced Research Projects Agency (DARPA) Young Faculty Award (YFA).