Energy-Efficient Mobile Computer Vision and Machine Learning Processors

by

Ziyun Li

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical Engineering)
in The University of Michigan
2019

Ziyun Li

liziyun@umich.edu

ORCID iD: 0000-0001-6070-6310

To all the people in my life

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Technology scaling has driven computing devices to be faster, cheaper, and smaller while consuming less power in past decades. However, as technology scaling has become increasingly difficult in recent years, power has become the major constraint in performance, and thus, the improvement in the performance of mobile devices has begun to diminish. Moreover, emerging intelligent mobile systems are demanding increasing computing power. In light of this challenge associated with artificial intelligence (AI), domain-specific architectures are widely believed to be the path to realizing considerable improvements in the efficiency, performance and cost of intelligent mobile systems.

This thesis presents several algorithm, architecture and circuit co-optimized solutions for intelligent and autonomous mobile systems, including vision-based stereo depth, optical flow, simultaneous localization and mapping (SLAM) and convolutional neural network- (CNN)-based image recognition.

Four prototypes are implemented for demonstration and verification. The first two prototypes include a depth estimation processor and a 6D vision processor that enable real-time dense depth and motion perception, respectively. The third prototype is a CNN-SLAM processor that estimates ego-motion for vision-based navigation. Together, these prototypes form a geometric understanding of the environment for mobile systems. The fourth prototype is an ReRAM-CNN processor that enables semantic understanding through machine learning. The work presented in this dissertation exploits various optimizations including parallelism, scheduling, exploiting sparsity and circuit customization to overcome the complexity of these problems for extremely energy-efficient, real-time, robust operation. The impact is significant in the age of AI as mobile systems can become increasingly intelligent in daily life, powered by these proposed solutions.

CHAPTER I

Introduction

1.1 Mobile Applications for Autonomicity and Intelligence

Autonomous and intelligent mobile systems continuously monitor their current surroundings, process sensor information, construct geometric and semantic understandings of the environment and navigate to fulfill a set of tasks without human intervention. Research on developing next-generation autonomous and intelligent mobile systems has been increasingly gaining interest in recent years to make autonomous and intelligent mobility a reality.

One well-known projection of autonomous and intelligent mobile systems is self-driving cars. The automotive industry has shifted towards more autonomous systems that aim to reduce traffic collisions and more intelligent systems that seamlessly interact with drivers. According to a survey, two-thirds of US consumers want advanced vehicle technologies. These include features that help reduce human driver errors, such as lane-keeping assistance and emergency braking, as well as those that improve user experience, such as parking assistance and adaptive cruise control. Fig. 1.1 shows the degree of desire expressed by US consumers for four graduated levels of vehicle automation as defined by the National Highway Traffic Safety Administration. About 67 percent of US consumers have a strong desire for adaptive autonomy features, marking an increase of 11 percentage points over the 2014 results[1]. These

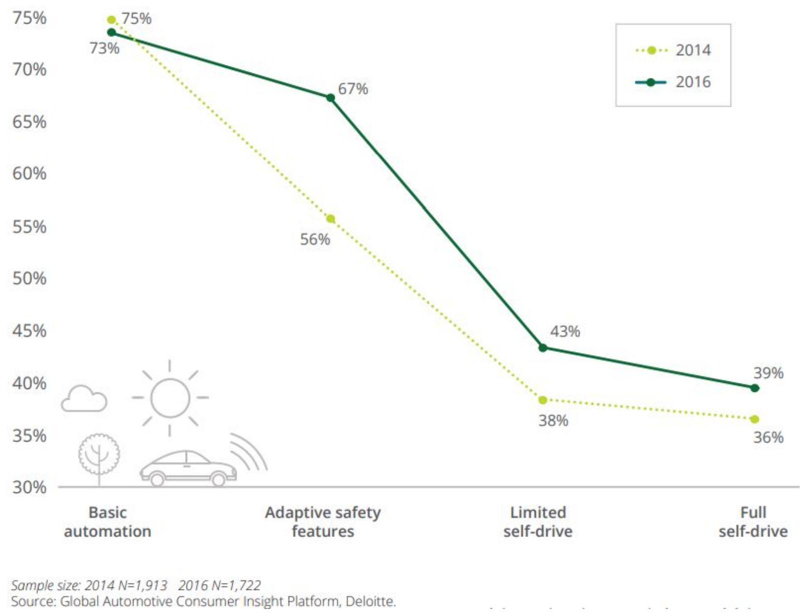


Figure 1.1: Percentage of US consumers interested in different levels of vehicle automation technology

systems are typically referred to as Advanced Driver Assistance Systems (ADAS) and are not fully autonomous. Nowadays, both academia and industry (technology companies such as Google, Tesla, and Uber and automakers such as Ford and Toyota) are moving toward fully self-driving cars.

Another similar projection for autonomous and intelligent mobile systems is unmanned robots. Unmanned aerial vehicles (UAVs) represent one of the most sought-after consumer electronics products after cellphones. According to Deloitte, the number of UAVs bought by consumers or prosumers is expected to reach 300,000 units a year with revenue of more than $300 million[2] in 2017. UAVs are deployed in a widening range of professional and entertainment contexts. For consumers, UAVs are appealing for high-definition photography and can be programmed for "follow-me" footage. For professionals, UAVs can undertake tasks such as agricultural monitoring, search and rescue missions, geological mapping of uncharted territories and delivery of parcels. Leading research groups and industries such as Amazon and DJI are developing smart UAVs that can avoid obstacles, track a person and navigate to make

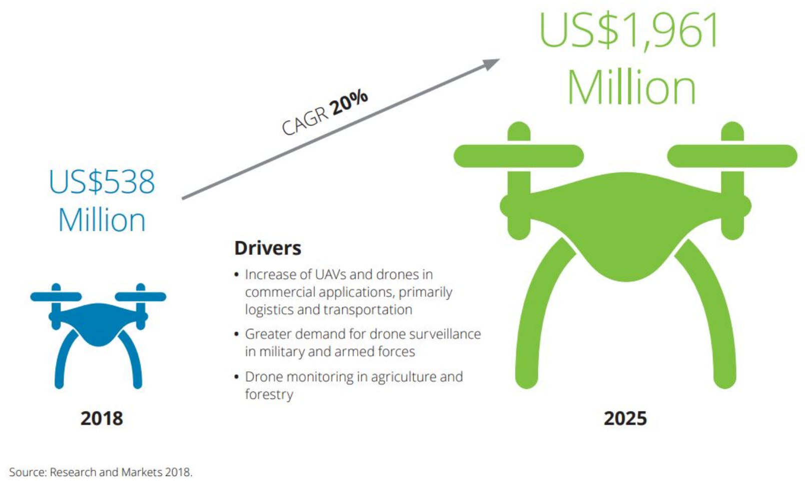


Figure 1.2: Growth of unmanned aircraft system traffic management (UTM) market

their use safe.

1.2 Overview of Mobile Autonomous and Intelligent Systems

Mobile autonomous and intelligent systems monitor the environment continuously through various sensors, including cameras, Lidar, time-of-flight cameras, ultrasonic sensors, Radar, etc. These systems process these sensor inputs to localize themselves and build an understanding of their environment. Although various sensor modalities can be applied in mobile systems, this thesis focuses on a passive camera-based approach because passive cameras are cheap and can easily adapt to new requirements[3, 4] with state-of-the-art computer vision and machine learning algorithms.

As is shown in Fig. 1.3, a state-of-the-art vision-based mobile autonomous system consists of components for acquiring raw sensor data, cognitive processing and control of high-level tasks. The cognitive processing component enables the system to construct a 3D model of the environment, detect/recognize objects and understand the semantics of the scene. These features are then used to complete high-level tasks such as search and rescue, navigation, obstacle avoidance, etc. Various combinations

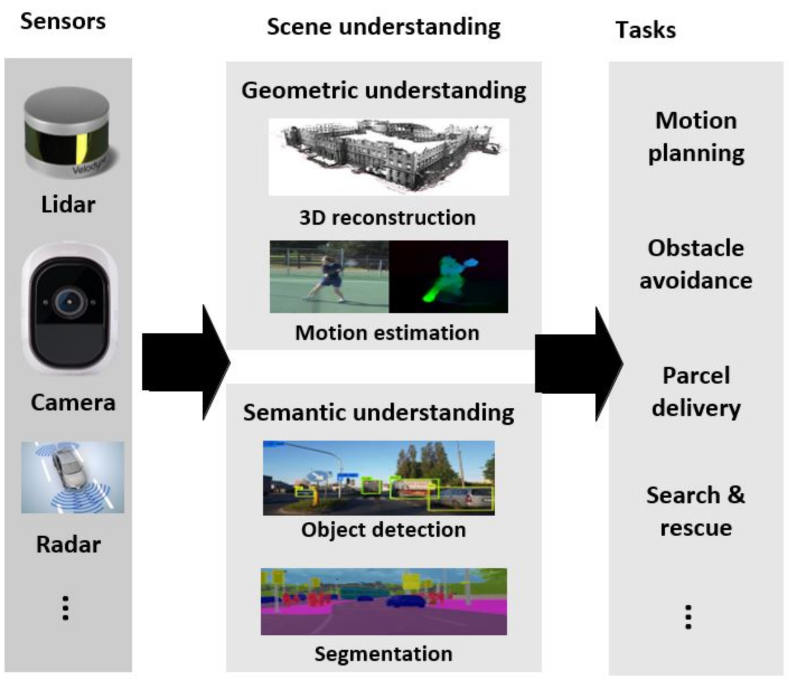


Figure 1.3: Overview of intelligent and autonomous system

of these components have been observed in real-world applications recently, both from academia[5, 6] and industry[7].

1.2.1 Geometry Awareness

Geometric understanding is critical for an autonomous system to localize and estimate its own motion and construct a 3D map of its surroundings. With indoor and GPS-limited environments, passive camera-based approaches are favored for their low cost, accuracy and dense scene reconstruction[8]. Examples of camera-based 3D understandings that are widely studied by the robotics and computer vision community include stereo depth estimation, dense optical flow and simultaneous localization and mapping (SLAM). These algorithms and systems perform vision-based geometric understandings using different approaches, as will be discussed later in Chapter II, Chapter III and IV.

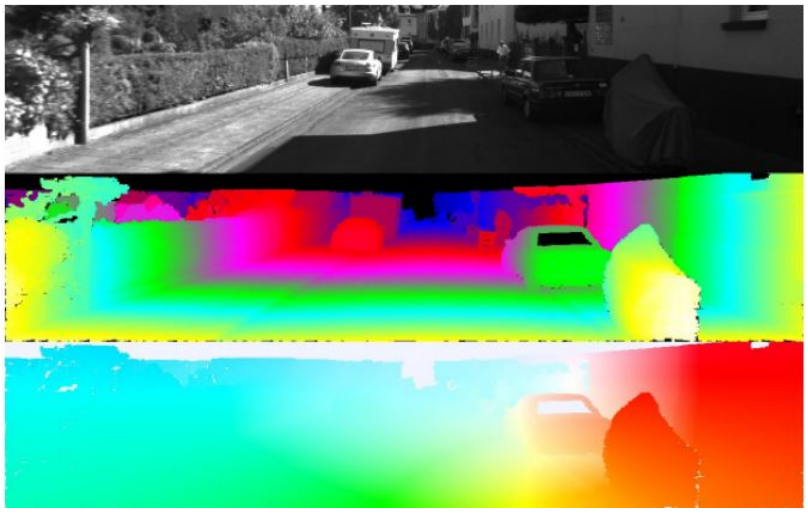


Figure 1.4: Dense stereo and optical flow perception of a scene

1.2.1.1 Stereo Vision and Optical Flow

Stereo vision, which requires extracting 3D information from a scene using multiple cameras, has been thoroughly studied by the computer vision community for decades[9, 10, 11, 12]. The basis for stereo vision is that a single 3D physical location projects to a unique pair of pixels in two observing cameras[9]. Therefore, computational stereo compares the objects in images taken from distinct viewpoints and then extracts their 3D information by examining the relative positions of the common objects (Fig. 1.4, mid). A detailed discussion of stereo vision systems can be find in chapter II.

While stereo vision perceives the 3D structure of a scene, optical flow perceives rich motion information and also plays an integral role in intelligence. Optical flow is defined as the apparent motion of individual pixels in an image (Fig. 1.4, bottom). Similar to computational stereo, calculating optical flow through image sequences has also been an important research topic[13, 14, 15] for decades. Most of these algorithms assume the invariance of pixels under the displacement and track these pixels from one image frame to the next. A detailed discussion of optical flow systems is presented in chapter III.

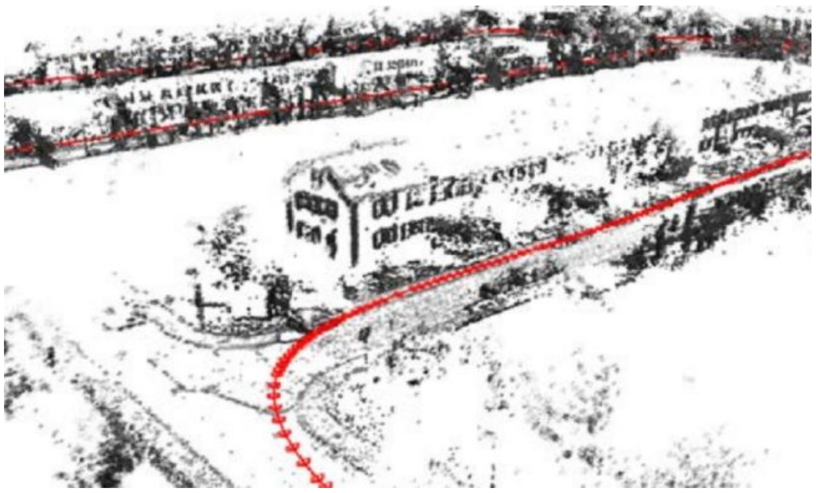


Figure 1.5: SLAM for autonomous navigation

Although algorithms that compute stereo vision and optical flow have matured significantly in recent years, many advances in computational stereo vision and optical flow continue to be made for real-world and real-time demanding applications.

1.2.1.2 Localization and Mapping

The first attempts at estimating a system's ego-motion and incrementally constructing a map of an unknown environment from a sequence of images were made in the 1980s[16]. Since then, SLAM has been extensively studied, resulting in denser 3D reconstruction and improved accuracy and reliability in unconstrained environments[17, 18, 19]. Typically, these SLAM systems compute ego-motion through triangulating the common landmarks in consecutive images. A detailed description of SLAM systems can be found in chapter IV.

1.2.2 Semantic Awareness

Semantic understanding is the capability of an intelligent system to describe the visual content of a scene. Semantic understanding of the environment includes various kernel functions such as object detection, object tracking, and segmentation at different levels of detail. Mobile intelligent systems are required to identify surrounding

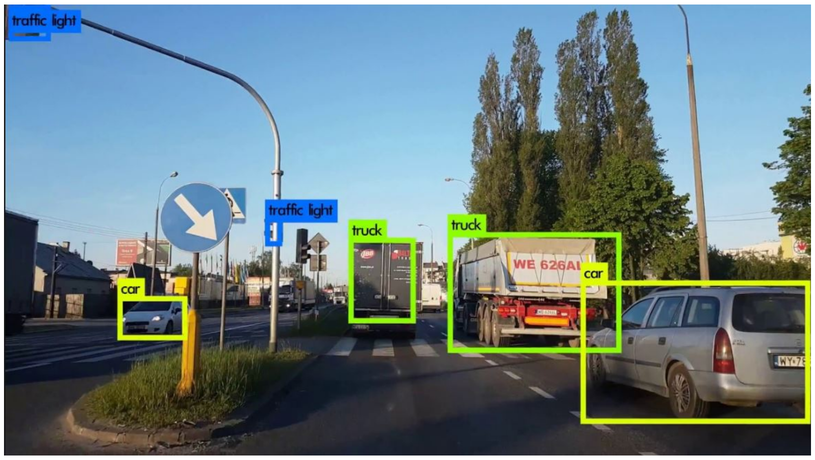


Figure 1.6: Recognizing objects in a scene

objects, avoid collision with obstacles and/or search for a target. Recent advances in computer vision and machine learning have started to reach a human-like level of object recognition, detection, and semantic segmentation by training a very deep neural network[20, 21, 22] with very large datasets[23, 24]. These algorithms and systems that perform vision-based semantic understandings will be discussed later in Chapter V.

1.2.2.1 Object Recognition

Object Recognition is one of the fundamental problems in computer vision that requires a form of semantic understanding of a scene (Fig. 1.4). Many model-based[25, 26, 27] and learning-based[6, 22, 20] algorithms have been proposed over the past decades on object recognition tasks. Mostly, these object recognition systems extract features from an image and then use these features to classify the object into a specific object category. Recently, due to the rapid advances in deep learning, learning-based algorithms have become favored because they surpass human performance in object recognition tasks. A detailed discussion on object recognition with deep learning on can be found in Chapter V.

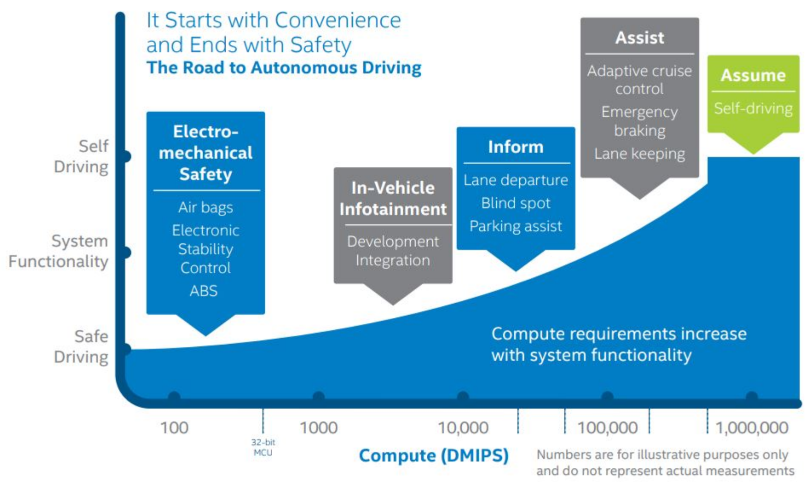


Figure 1.7: Throughput requirement for self-driving cars

1.3 Challenges for Intelligent Mobile Computers

Although recent advances in computer vision and machine learning have paved the way for autonomous and intelligent mobile systems, many practical challenges and limitations remain. This section describes the challenges of processing extremely high dimensional and high bandwidth information in these systems, and the dissertation focuses on solving these practical issues with ASIC solutions.

1.3.1 High Throughput and Low Latency

While operating in a fast changing environment, an autonomous and intelligent system needs to process sensor information quickly and make proper responses. For example, the processing latency of a single frame for an ADAS system must be less than 33 ms[28]. With the increase in camera resolution from VGA to HD and the increasing number of cameras and sensors integrated in cars, approximately 1 GB of data will need to be processed each second in a car's real-time operating system[29]. Fig. 1.7 shows the throughput requirement for autonomous systems: the more intelligent a mobile system, the more stringent the throughput requirement.

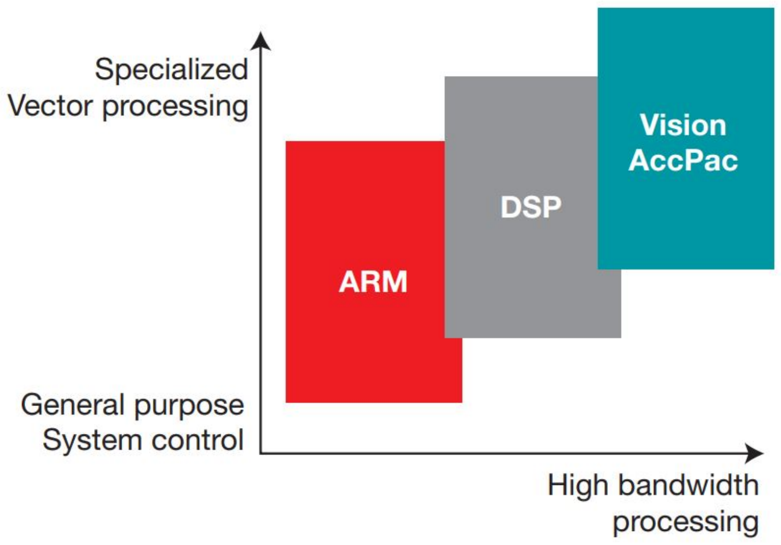


Figure 1.8: Comparison between general purpose and special purpose computer vision solutions

1.3.2 Low Power Consumption

While the computers in autonomous mobile systems must deliver increasing computing power, they also must do so as efficiently as possible. As mobile systems are typically powered with batteries, their limited on-board energy poses a real challenge, which is manifested in their low endurance. For example, computing systems in UAVs and self-driving cars need to provide very high processing capabilities and also use very little power[29].

1.3.3 CMOS Technology Scaling and Moore's Law

Moore's Law has guided the semiconductor industry for 50 years as society has shifted from the PC era to the mobile computing era (Fig. 1.8). The resulting increase in the capability, affordability, energy efficiency, performance and availability of integrated circuits has enabled computer architects to design better computers for various needs. Sophisticated processors and memory hierarchies have been developed to exploit parallelism between instructions without the knowledge of the programmer.

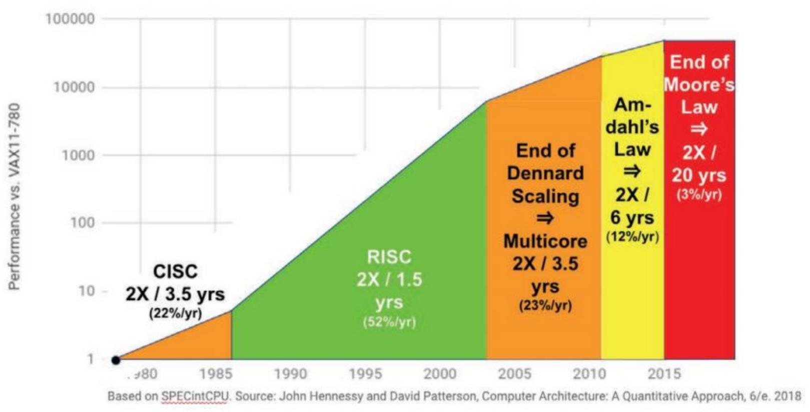


Figure 1.9: Average performance gain for a single program over time versus VAX 11/780 using SPECintCPU

However, as Moore's law is slowing down, and the computing requirement for designing autonomous and intelligent systems is increasing rapidly, architects now widely believe that the only path left for major improvements in performance-cost-energy is designing domain-specific architectures. As shown is Fig. 1.9, special purpose processors achieve higher processing bandwidth compared with general purpose processors. A survey (Fig. 1.10) also revealed that ASIC chips may dramatically expand the addressable market and increase the use of machine learning because they enable applications to use less power and at the same time become more responsive, flexible and capable.

1.4 Thesis Contribution, Organization

This work contributes to the development of mobile computer vision ASICs and domain-specific machine learning processors with low-power consumption, high energy efficiency and high performance, enabling its potential use in intelligent and autonomous mobile systems. This dissertation presents four works in detail.

In Chapter II, this thesis presents an SGM stereo vision processor designed for autonomous navigation of micro aerial vehicle applications with tight SWaP (size, weight and power) constraints. The major contributions of this work include 1)

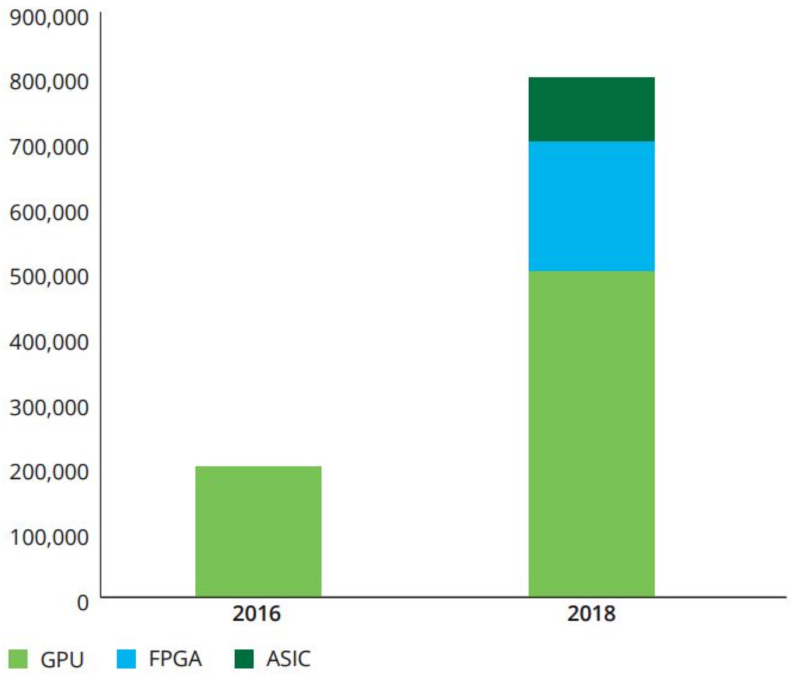


Figure 1.10: Annual minimum sales of ML chips in global data centers (units)

single-chip implementation of the semi-global matching algorithm using overlapping block-based processing to remove the need for an external DRAM storage; 2) a new dependency-hidden diagonal image-scanning stride with 17-stage deeply pipelined implementation providing 512-level depth output at 30 fps FHD resolution; and 3) customized 50 word, 1612 bit parallel ultra-wide SRAM with high (1.64 Tb/s) on-chip memory access bandwidth and low access power (6 mW).

In Chapter III, this thesis presents a realtime optical flow/stereo vision reconfigurable vision processor designed for dense 6D vision perception on low-power mobile platforms such as MAVs and VR/AR systems. The major contributions of this work include 1) a new NG-SGM algorithm that significantly prunes the 2D optical flow search space with negligible performance degradation; 2) single chip implementation of a unified NG-SGM datapath that supports both wide-range (176×176-level) 2D optical flow and 1D stereo vision computation; 3) a customized coalescing crosspoint crossbar that provides high (2.6 Tb/s) on-chip memory access bandwidth; and 4) multi-frequency and voltage partitioning of the design that reduces power consumption without performance degradation.

In Chapter IV, this thesis presents a realtime CNN-SLAM processor designed for six degrees of freedom egomotion and trajectory estimation. It is designed for low-power mobile autonomous navigation systems and VR/AR systems with seamless user interaction. The major contributions of this work include 1) feature extraction and description with a highly parallelized and programmable CNN engine with 32% better accuracy in feature matching than SIFT; 2) aggressively pruned feature matching with temporal pose prediction, triangulation, and address hashing to eliminate 97% of unnecessary matchings; 3) numerically stable fixed-point implementation with function re-organization and pivoting in the linear solver; and 4) hierarchical memory organization, completely eliminating external DRAM accesses for BA optimization.

In Chapter V, this thesis presents a single-chip CNN processor with 30 MB on-chip MLC ReRAM for high performance and energy efficient machine learning/deep learning applications on an AI edge. It is designed for low-power mobile visual recognition systems powered by deep learning. The proposed design is taped out with TSMC 22-nm ULL technology. The major contributions of this work include 1) single-chip implementation of large-scale modern CNNs with 4$\times$4 mesh connected cores; 2) deeply compressed weights with MLC (multi-level cell) ReRAM operation achieving 1 million on-chip parameter storage without external DRAM; and 3) highly parallelized processing with weight/input/output reuse on 2048 MAC units, achieving 0.8 TMACS throughput with 2.5 TOPS/W efficiency.

CHAPTER II

A 1920×1080 30fps 2.3TOPS/W Stereo Depth Processor for Energy-Efficient Autonomous Navigation of Micro Aerial Vehicles

2.1 Introduction

Precise depth estimation is essential to realize autonomous navigation in micro aerial vehicles (MAVs), robots, and self-driving cars. Depth estimation serves as a key kernel function in simultaneous localization and mapping (SLAM), 3D scene understanding and reconstruction, object recognition and obstacle avoidance, as depicted in Fig. 2.1. Real-time reliable autonomous navigation requires the depth estimation to be dense, accurate, wide ranged, and with high performance. Emerging mobile platforms such as MAVs introduce additional "SWaP" (Size, Weight and Power) constraints on depth estimation systems. For mobile applications, the system must be small (e.g., 50 cm^2), lightweight ($<$100 g), fast (30 ms response time), and low power ($<$1 W) [30]. Fig. 2.1 highlights the requirements of real-time stereo depth estimation for MAV applications. LIDAR [31], RADAR [32], Ultrasonic sensor [33] or IR sensor [34] are conventional approaches for depth sensing. IR sensors typically have low resolution and accuracy while ultrasonic sensors have limited ranging distance [35]. Therefore, they are not widely adopted in autonomous systems. While 24G RADAR is accurate and robust, it has limited field of view (30° horizontal angle) so that it still needs other sensors for wide-range, large-scale 3D scene construction. LIDAR is the most frequently used sensor for 360° 3D scene construction in autonomous systems. Fig. 2.2 visualizes the difference between the depth map acquired by LIDAR and that obtained by stereo vision correspondence. Nowadays, even the most advanced LIDAR-based ranging systems suffer from a limited field of view (e.g., 30° vertical view angle, which accounts for the top blackout region of the LIDAR image in Fig. 2.2.), heavy weight (>600 g) and energy consumption (10 W) [31]. In contrast, depth estimation by stereo vision is fast, energy efficient and lightweight when mounted on an MAV platform.

There are prior ASIC implementations of stereo vision depth estimation based on various algorithms [36, 37, 28, 38, 39]. These designs [36, 37, 28, 38, 39] employ hardware-oriented algorithmic optimizations to enable depth estimation in real-time systems, but there are deficiencies associated with these ASIC designs. Some uses local matching [36] or aggressively truncated global algorithms [37], which results in inferior quality. Other works limit their disparity range to 32 or 64 pixels and therefore fail to support industrial standard automotive scene benchmarks [36, 37, 28, 38, 39]. Semi-global matching(SGM)-based FPGA implementation is demonstrated in [40], but it is not applicable in robust navigation of power constrained MAV platforms because of its limited performance (30 fps for 320×240 QVGA) and high power consumption (3 W for QVGA). Prior advanced driver assistance system SoCs [41, 42, 43] are not favorable for SGM because of the memory bandwidth bottleneck. Due to the big memory requirement of SGM, prior methods [37, 28, 38] use external DRAM to store intermediate results, leading to limited frame rate and low power efficiency. Several alternative approaches have been explored to reduce the high complexity of global methods [9] using dynamic programming [44], belief propagation [45, 46], flow vector search space pruning [47], and pseudo-random flow candidate selection [48].

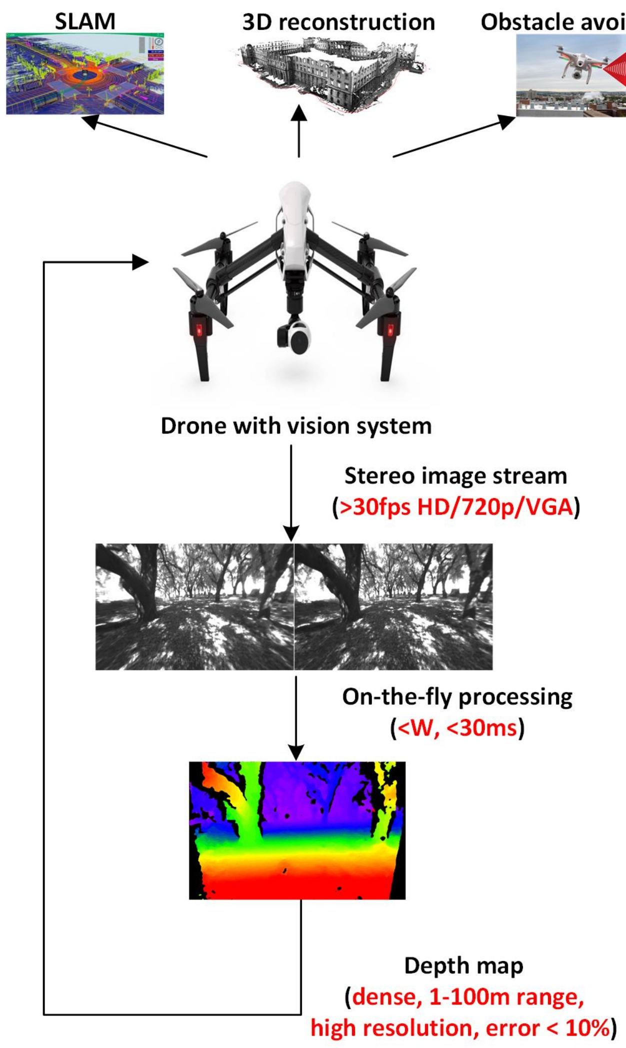


Figure 2.1: Drone with stereo vision system and its applications.

However, SGM (and its variations) is clearly one of the most widely used algorithms in industrial standard benchmarks, such as KITTI [49].

In this work [50], we first study the computation, memory, and bandwidth bottlenecks of the SGM algorithm and propose algorithm-architecture co-optimization techniques that significantly reduce the hardware cost with negligible accuracy degradation. We propose a deeply pipelined hardware architecture with a dependency-resolving scan to handle the critical-path data dependency in the algorithm and to expressively improve throughput. We also introduce a custom designed dual-port 8T SRAM that leverages the unique memory access characteristics of the SGM algorithm

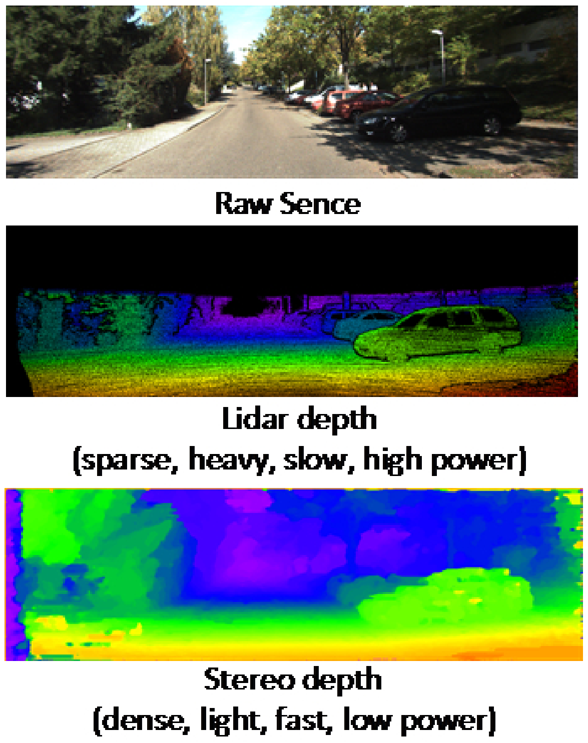

Raw Sence
Lidar depth
(sparse, heavy, slow, high power)
Stereo depth
(dense, light, fast, low power)

Figure 2.2: Comparison between stereo depth and Lidar based depth estimation.

to enable ultra-high bandwidth (1.64 Tb/s) and energy-efficient on-chip memory access. The fabricated chip employs a standard USB3.0-compliant interface, allowing effortless integration with a wide range of commercial off-the-shelf stereo cameras and general purpose mobile application processors (APs). Integrated with a ZED camera [51] and ODRIOD-5422 mobile AP on the ODRIOD-XU4 [52], the fabricated chip was successfully mounted on a quadcopter MAV for system demonstration in realistic flight scenarios. The chip delivers 512 levels of stereo depth for each pixel at full HD (1920×1080) resolution with real-time 30 fps throughput, while consuming 836 mW.

2.2 Overview of Stereo Vision Algorithms

2.2.1 Local approach

Vision-based depth estimation is computed by stereo correspondence. As shown in Fig. 2.3, a point P in the real world will be horizontally displaced at the pixel positions p and q in stereo images because the left and right cameras are placed apart by distance b. This horizontal displacement between a pixel in the left image p= (x,y) and its matching pixel in the right image q= (x',y) is defined as disparity (x'-x). The depth Z is inversely proportional to the disparity as in

$$Z = f\frac{b}{x\prime - x} \tag{2.1}$$

where f represents the focal length of the camera and b is the baseline of stereo camera system.

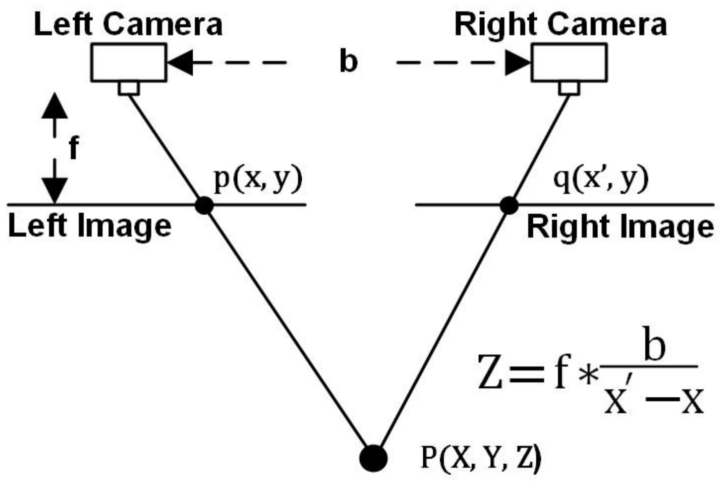


Figure 2.3: Illustration of principals in stereo vision.

The most straightforward approach to compute disparity is local matching. As shown in Fig. 2.4, local matching compares each pixel (e.g., the white pixel in the far left image) with all its matching candidates and then finds the best matching pixel (e.g., the black pixel in the next image to the right) associated with the minimum matching cost within the search range (depicted by the white bar in Fig. 2.4). Typically, to enhance robustness, local matching is performed based on a window that consists of a group of pixels surrounding the matching pixel. The same step is applied to determine the disparity of all the pixels. An example disparity map resulting from using local matching is shown in Fig. 2.4. In the local approach, every pixel in the image can be processed independently in parallel to improve throughput.

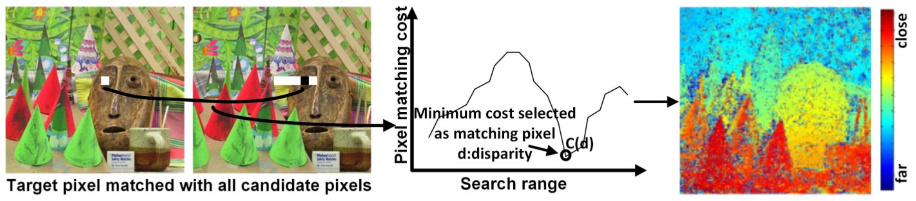

Target pixel matched with all candidate pixels
Pixel matching cost
Minimum cost selected as matching pixel
d:disparity
C(d)
Search range
close
far

Figure 2.4: Local matching method of stereo depth estimation.

The accuracy of a local approach is unreliable since it typically fails to resolve ambiguities in many challenging but realistic scenarios such as occlusions, texture-less regions, transparency and repetitive patterns. As shown in Fig. 2.5, almost all of the pixels of the wall on the right are saturated and texture-less due to strong illumination. The disparity results obtained by a local approach on this texture-less region will be completely incorrect, as shown in Fig. 2.5, because many matching pixels will appear as identical with the same cost. As seen in other less challenging regions in Fig. 2.5, the disparity map derived from using a local approach has substantial noise that cannot be easily removed by advanced post processing.

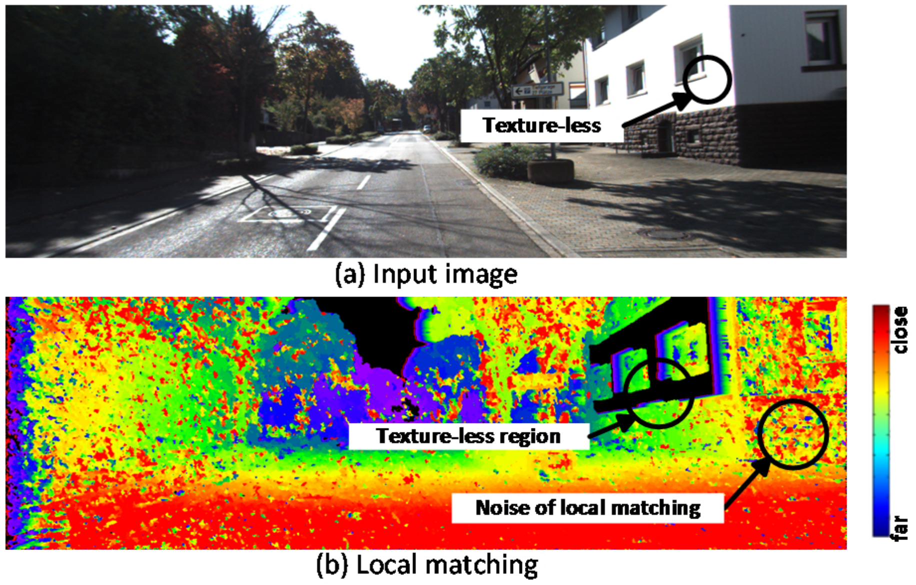


Figure 2.5: Problems of local matching.

2.2.2 Semi-global matching and its complexity

Recently, various global methods [11, 53, 54] have been proposed to improve accuracy. In these global algorithms, information from neighbouring pixels is (semi-)globally propagated to the current processing pixel to enhance the correspondence matching accuracy. The SGM algorithm introduced in [55] is one of the most popular global methods. SGM is favoured for its robustness and high accuracy under various scenarios. SGM has been validated to achieve good accuracy in various industrial standard benchmarks [49]. In particular, it effectively handles low texture regions with its dynamic programming-based global optimization of the disparity over the entire image. Fig. 2.6 visualizes the output difference between the local sum of the absolute difference (SAD) [11] algorithm and SGM, clearly illustrating the higher quality obtained with SGM.

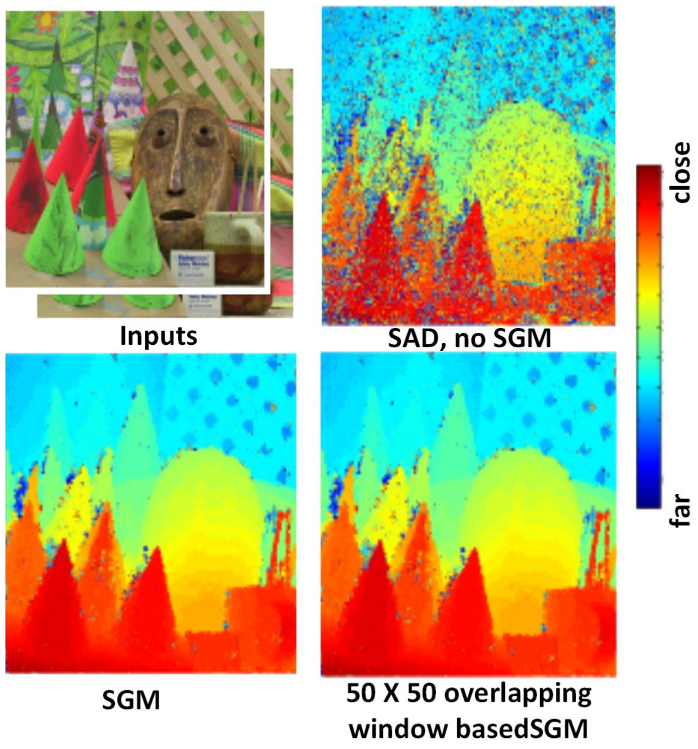


Figure 2.6: Comparison between local matching, original SGM and overlapping-block based SGM.

SGM consists of three steps: 1) pixel-wise matching cost computation; 2) semi-global aggregation; and 3) disparity selection. To compute the pixel-wise matching cost, N×N census transform [56, 57] is performed on both left and right image. Census transform $I_L(p)$ of a pixel p is computed by comparing the grayscale intensity of center pixel p with all of its neighboring pixels within the N×N window. As a result, each pixel in the image is converted to a bit string of the length N^2-1. We use a 7×7 census for our design, and each pixel is represented by a 48-bit string as a result of the census transform. The pixel-wise matching cost $C(p, \mathbf{d})$ for a pixel p with a disparity $\mathbf{d}$ is evaluated by the Hamming distance [58] between the census transformed left image pixel $I_L(p)$ and the right image pixel $I_R(p - \mathbf{d})$ as shown in 2.2 where $|.|_H$ denotes the Hamming distance.

$$C(p, \mathbf{d}) = |I_L(p) - I_R(p - \mathbf{d})| \tag{2.2}$$

This operation is repeated for all disparity candidates per pixel. Since each pixel will have 128 matching candidates, the local matching costs evaluation results in a cube of dimension H×W×128, as shown in Fig 2.7, where H×W is the image height × width in the number of pixels. For each pixel, a 128-entry depth vector is generated as the local matching cost associated with 128 disparities.

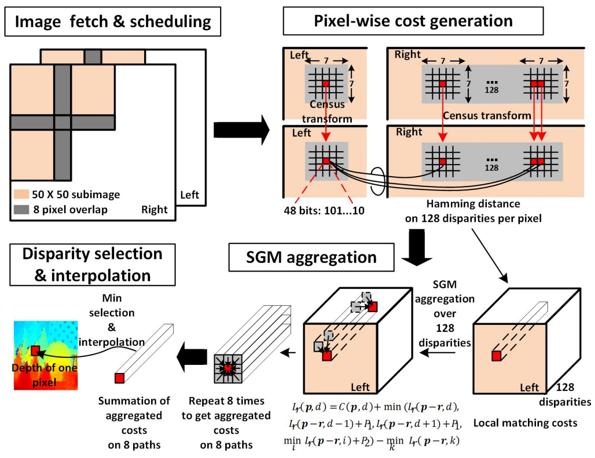


Image fetch & scheduling
50 X 50 subimage
8 pixel overlap
Left
Right
Pixel-wise cost generation
Left
Right
7
128
Census transform
Census transform
Left
Right
48 bits: 101...10
Hamming distance on 128 disparities per pixel
Disparity selection & interpolation
SGM aggregation
Min selection & interpolation
Depth of one pixel
Summation of aggregated costs on 8 paths
Repeat 8 times to get aggregated costs on 8 paths
$L_r(\mathbf{p}, d) = C(\mathbf{p}, d) + \min(L_r(\mathbf{p}-\mathbf{r}, d), L_r(\mathbf{p}-\mathbf{r}, d-1)+P_1, L_r(\mathbf{p}-\mathbf{r}, d+1)+P_1, \min_i L_r(\mathbf{p}-\mathbf{r}, i)+P_2) - \min_k L_r(\mathbf{p}-\mathbf{r}, k)$
SGM aggregation over 128 disparities
Left
128 disparities
Local matching costs

Figure 2.7: SGM algorithm processing flow.

Global aggregation is then performed on local matching costs. SGM aggregation takes the current processing pixel p and propagates information from neighbouring pixels along eight paths r over the entire image using equation 2.3 as depicted in Fig. 2.8. The term $L_r(p, \mathbf{d})$ denotes the aggregated cost for a given pixel p with disparity d along path r. The aggregated costs of neighbouring pixels associated with the same (d) or similar ($d\pm1$ and $d\pm2$) disparities are merged into the aggregated cost for the current pixel with zero or small (P_1,P_2) penalty.

$$L_r(p,\mathbf{d}) = C(p,\mathbf{d}) + \min L_r(p-r,\mathbf{d}), L_r(p-r,\mathbf{d}-1)+P_1, L_r(p-r,\mathbf{d}+1)+P_1, min_i L_r(p-r,i)+P_2 - \min_k L_r(p-r,k) \tag{2.3}$$

Eventually, 'good' disparities with low matching costs (when neighbouring pixels all see smaller matching costs in general) will propagate from far positions to the current pixel through recursive aggregation (equation 2.3). This is particularly useful for propagating 'good' matching candidates for the center of a low texture region from texture-rich boundary pixels.

The SGM aggregation is performed along 8 paths (the size of the r set is 8) separately, as shown in Fig. 2.8, and the aggregated costs on 8 paths are summated together as in 2.4.

$$S(p,\mathbf{d}) = \sum_r L_r(p,\mathbf{d}) \tag{2.4}$$

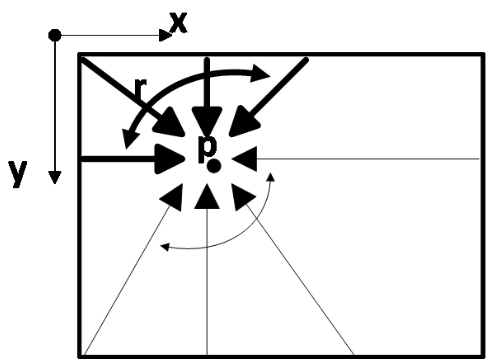


Figure 2.8: Path aggregation diagram.

The disparity $\mathbf{d}$ with the minimum summated costs $S(p,\mathbf{d})$ is eventually selected as the integer level of disparity for the processing pixel p. To obtain a sub-integer pixel disparity precision, we select three minimums from all of the summated costs $S(p, \mathbf{d})$ for a given p and perform a bilinear interpolation [59] on these three minimums. This generates an additional 2-bit sub-integer pixel disparity resolution and eventually generates 512 levels of depth (disparity) for each pixel.

Although SGM provides superior accuracy compared with local approaches, it poses significant hardware challenges. The original SGM requires massive computation (2TOP/s), extremely high bandwidth (38.6 Tb/s), and very large memory (386 MB) for 30 fps full HD resolution. Therefore, when realized in general-purpose computing platforms, it leads to very low frame rates and energy efficiency. Specifically, this SGM complexity translates to 20 sec runtimes for a full HD image pair on a 3 GHz CPU with >35 W power consumption [60]. Although server/mobile GPUs achieve higher energy efficiency, they still consume a few Joules to process a single full HD frame with 5 fps throughput [61]. Table 2.1 provides a comparison of the estimated performance, power and memory for different platforms. To resolve these challenges and address "SWaP" requirements of MAVs, we propose a highly optimized ASIC solution attained via a cross-layer optimization conducted across algorithm, micro-architecture, and circuit levels.

	CPU (intel i7)	GPU (TITAN X)	GPU (Tegra)	FPGA
Performance @ FHD	0.05fps	29fps	2.3fps	11fps
Power	~50W	~250W	~11W	~8W
Energy per frame	>35J	8.4J	4.24J	0.8J
External DRAM	yes	yes	yes	yes

Table 2.1: Low efficiency on CPU/GPU/FPGA platforms.

2.3 Algorithm, Architecture and Circuit Optimizations

A high-level summary of our cross-layer optimizations designed to tackle the challenges of SGM is illustrated in Table 2.2. Firstly, strong data parallelism in the algorithm is exploited so that the processor computes 128 local costs, aggregates 128 disparities, and accumulates 4 paths all in parallel. Secondly, instead of processing the whole image frame, we propose an overlapping block-based processing to eliminate the very large on-chip memory requirement and to achieve a single-chip SGM implementation without off-chip DRAM accesses. Moreover, a dependency-resolving scan with a 16-stage deep pipeline is proposed to hide the data dependency and improve throughput by 3×. Finally, we also custom designed an ultra-high bandwidth dual port SRAM that leverages unique memory access patterns of SGM for high performance and energy-efficient memory access.

Challenges / Optimization	Massive computation	Inter-pixel Dependency	Large Memory	High memory bandwidth
Parallel processing: Compute 128 local costs in parallel Aggregate 128 disparities in parallel Aggregate 4 path in parallel	√			√
16 stage deep pipeline		√		√
Dependency resolving scan		√		
Overlapping block based processing			√	
Overlapping block based processing				√

Table 2.2: Summary of SGM challenges and algorithm-architecture-circuit optimizations.

2.3.1 Algorithm: Overlapping Block-based SGM Processing

The original SGM algorithm consists of forward and backward scans as shown in Fig. 2.8, where each scan aggregates costs for each pixel along 4 paths. This two-scan approach is unavoidable to allow 8-path aggregation over the entire image frame.

The partial (4 paths) aggregated costs (12 bit each) for every pixel are stored in the memory during the forward scan and then later combined with the backward scan results for the remaining 4 paths. This 2-scan imposes significant on-chip memory and bandwidth requirement for storing 128 (number of disparities) aggregated costs (16 bits each) for every (2 M for full HD) pixel in the image. Therefore, the memory requirement of SGM is not scalable to various image resolutions, and a single full HD image pair will require 386 MB storage for temporary aggregated costs. A prior work [62] reduced the amount of temporary memory usage without significant accuracy degradation by selectively storing sparse aggregated costs. Similarly, we only store three disparities associated with the three minimum summated costs for each pixel. This allows the temporary memory in our SGM implementation to be independent of the disparity search range (the number of disparities evaluated per pixel). However, the temporary memory size still depends on the image size, and storing sparse aggregated costs with their associated disparities for a full HD image would still require 20 MB of memory. This memory requirement will significantly degrade energy efficiency if it is mapped to external DRAM.

To further reduce the on-chip memory requirement and to eliminate the need for external DRAM, we first evaluate the sensitivity of accuracy with different overlapping window size on 194 KITTI cases. While the original SGM achieve 6.5 outlier, overlapping blocks of 200×200, 150×150, 100×100, 50×50 achieve 6.61%, 6.62%, 6.75% and 7% outliers respectively. From the evaluation, we observe that inter-pixel correlation diminishes when pixel pairs are more than 50 pixels apart. Therefore, instead of processing the whole image, the proposed design uses an overlapping block-based processing to partition the input image into units of 50×50 pixel overlapping blocks to minimize on-chip memory size, as shown in Fig. 2.7 top-left. Adjacent blocks are overlapped by 8 pixels to allow cost aggregation across block boundaries. This technique achieves 95.4% memory reduction for storing intermediate aggregation costs for a full

HD image. We evaluate this technique with standard Middlebury [63] and KITTI [49] benchmarks. Fig. 2.6 shows a side-by-side qualitative comparison of this block-based SGM and the original SGM on Middlebury test case, which yield almost identical results. Fig. 2.9 presents the accumulated density function evaluated on 194 realistic KITTI automotive test cases. The proposed overlapping blocked-based SGM suffers only 0.5% outlier percentage degradation compared with the original SGM throughout 194 KITTI evaluation cases. An outlier is a pixel that has a disparity error of more than 3 integer levels.

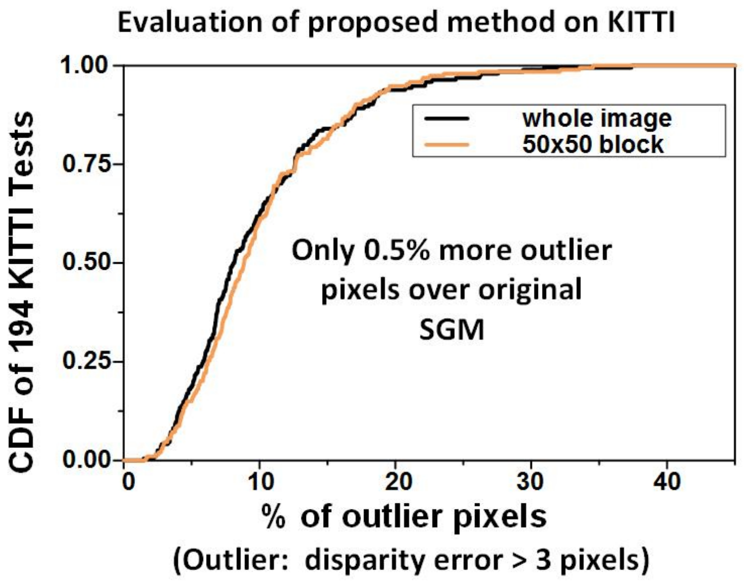


Figure 2.9: Quantitative evaluation with overlapping-block based SGM over 194 KITTI test images.

2.3.2 Energy-efficient Hardware Architecture

The proposed block-based SGM processing procedure is shown in Fig. 2.7, and the chip architecture is shown in Fig. 2.10. One 32-bit parallel interface streams input image data and processing instructions into the chip, and the other 32-bit parallel interface is used to stream the final disparity results off the chip. The control registers and on-chip input images are memory mapped and can be accessed with a

USB interface through an external USB-to-parallel converter [64]. To maximize the input bandwidth, a streaming mode is supported so that input/output image data are streamed to/from the chip continuously. The block-partitioned left and right images are stored in two on-chip interleaved image (ping-pong) buffers (30 Kb each). Processing is concurrently performed with input/output image blocks streaming to achieve real-time performance.

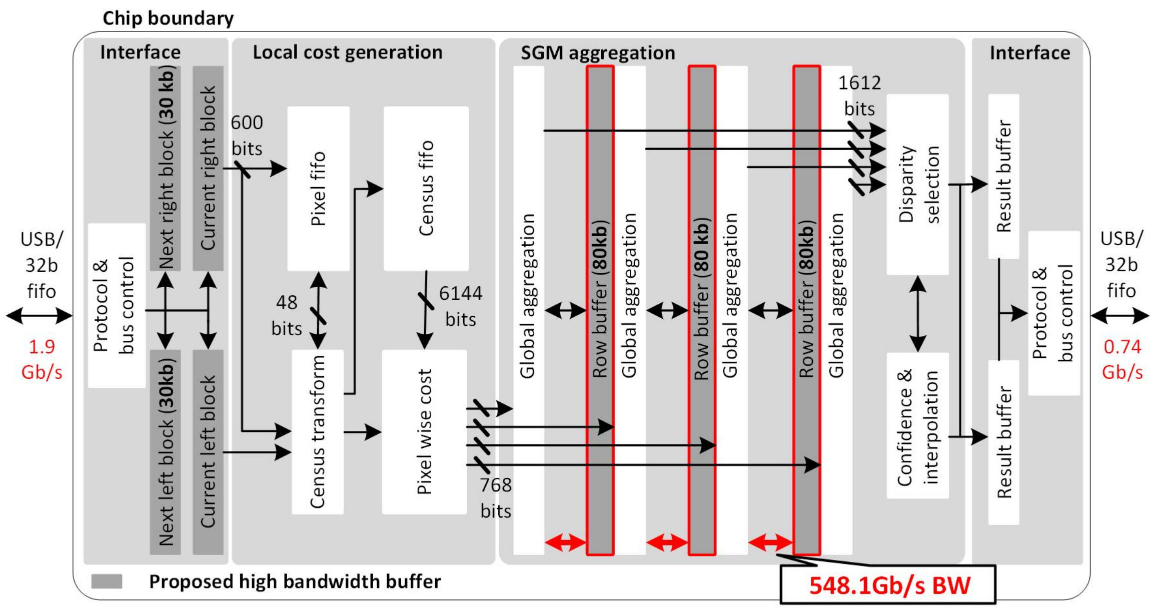


Figure 2.10: Hardware architecture of energy efficient SGM.

As the first step, 7×7 census transformations are performed on the processing pixel as well as its matching candidates at 128 different disparity locations using their surrounding (7×7 window) pixels. This census transform computing on-the-fly scheme would result in 6000 compare operations for every processing pixel and thus have poor energy efficiency. We observe that 127 out of 128 census transformed matching candidates of previous pixels overlap with the census transformed pixels of the current processing pixel when processing proceeds. Therefore, an on-chip 128-entry circular FIFO is employed to eliminate redundant census transforms and to store the pre-computed census results. A total of 127 census transformed matching candidates (48-bit each) are read directly from the on-chip FIFO as shown in Fig. 2.11. At each cycle when a new pixel is pushed into the pipeline, only one new census transform is performed and pushed into the FIFO. This census FIFO eliminates 98% of redundant census transforms. In simulation, storing census transforms in a FIFO and preloading them (5.1 pJ/pixel including memory accesses) achieves 2.8× better energy efficiency compared with on-the-fly re-computing census (14.2 pJ/pixel) for every pixel.

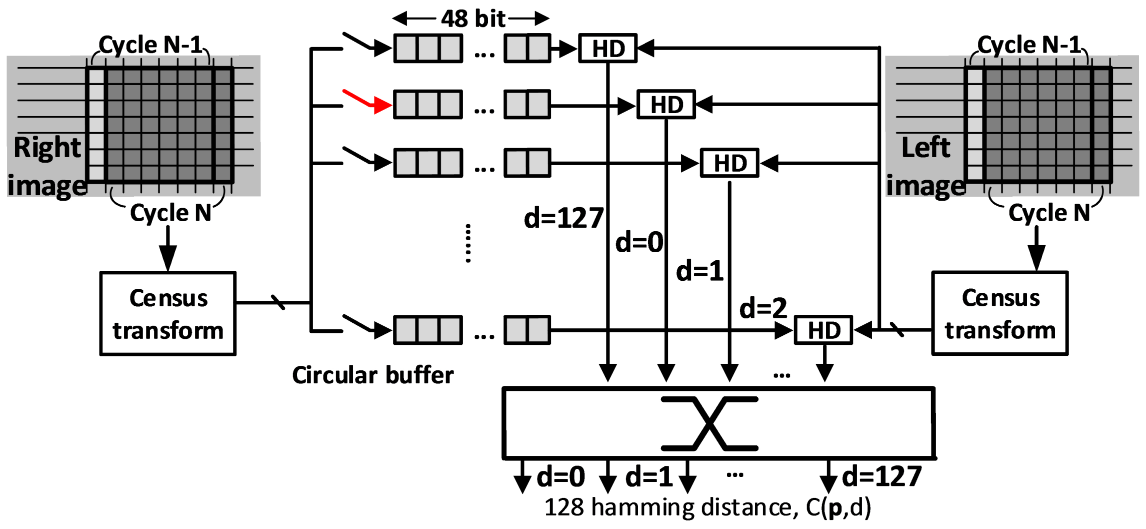


Figure 2.11: Rotating FIFO based local cost generation.

The census-transformed pixel in the left image and the census-transformed pixels in the right image at 128 different disparity locations are then compared in parallel. This produces 128 Hamming distances (6 bits each) for each pixel that represent the 'local' pixel-wise matching cost vector for the 128 disparities; $C(p,d)$. These 128 local costs are all sent to 4 parallel aggregation units for SGM aggregation. Each aggregation unit is equipped with a high-bandwidth buffer and aggregates 128 disparity locations in parallel, accumulating costs over 4 different paths. Massive parallelism in aggregation shown in Fig. 2.12 helps us achieve high throughput and energy efficiency. The tree-structured selection unit identifies the best 3 aggregated costs and disparities. These 3-best aggregated costs for each pixel are stored in the on-chip memory. Once the forward scan completes, the backward scan is performed in the similar fashion. Aggregation results that are discarded (except for 3-best results) during the forward scan are combined with a constant penalty with the backward scan results. The final best disparity candidates are selected based on the 8-path aggregated costs. Finally, bilinear interpolation is performed, and the 512-level (7-bit integer and 2-bit fractional) disparity results are stored in two interleaved result buffers.

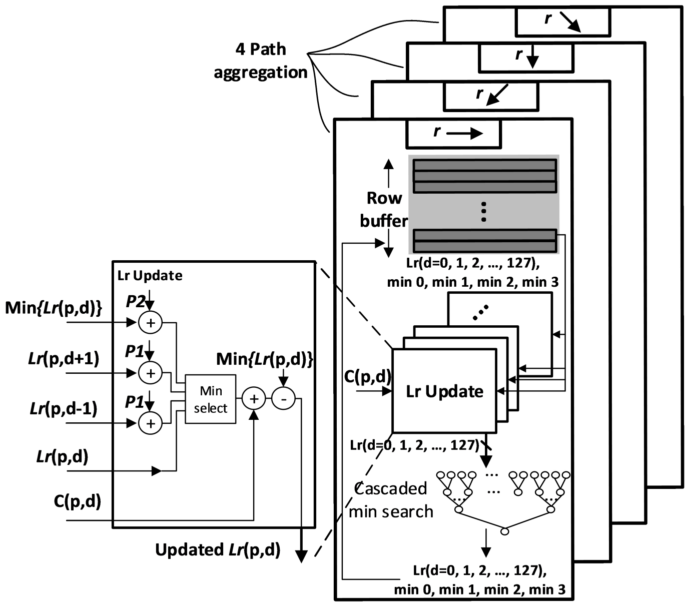


Figure 2.12: Implementation of 4 path aggregation.

2.3.3 Dependency-resolving scan, Pipeline and Forwarding

In the proposed highly parallelized cost aggregation, each aggregation unit has its own ultra-high bandwidth row buffer (marked in dark gray in Fig. 2.10). During each clock cycle, each aggregation unit reads the 128 aggregated costs from each neighbouring pixel (4 neighbours as shown in Fig. 2.13) from the buffer and writes the 128 aggregated costs of the current pixel to the buffer. However, this straightforward implementation would result in data dependency because the aggregation of the current pixel depends on the results of the neighbour that is processed in the previous cycle.

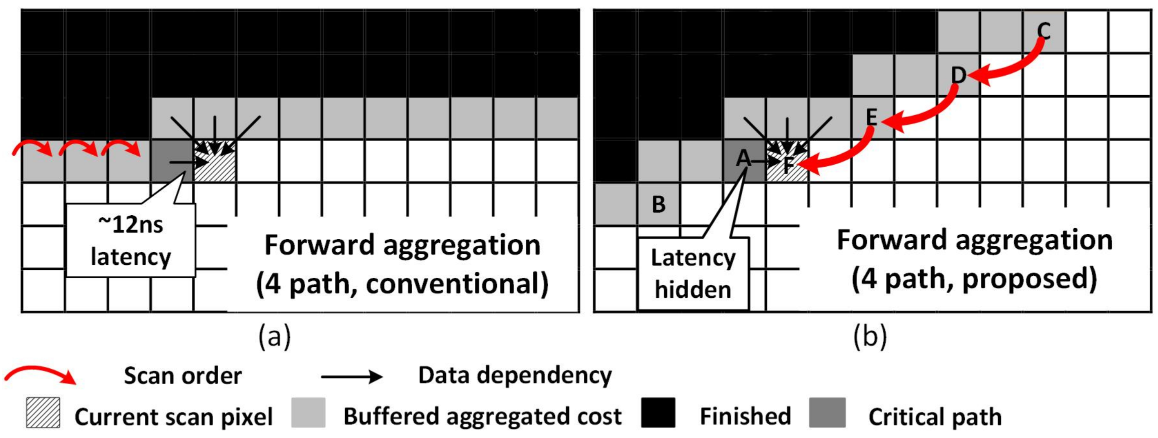


Figure 2.13: Illustration of conventional raster scan and proposed diagonal scan.

As discussed earlier, SGM is implemented with a forward and a backward raster scan, with each scan performing aggregation along 4 paths (Fig. 2.8). However, following this conventional raster scan order results in data dependency because the previous pixel must complete its computation before the current pixel can be aggregated. As shown in Fig. 2.13), the forward scan aggregates the results from its 4 neighbours marked with arrows. Aggregation in the top-left, top, top-right does not lead to dependency because those pixels belong to the last row and are ready before processing the current pixel. Data dependency is from the left neighbouring pixel along the raster scan path, and processing of the current pixel must wait until its left neighbour finishes aggregation. This data dependency dominates the critical path, limiting the clock frequency and voltage scalability for low power operation.

We therefore propose a dependency-resolving scan in which pixel processing proceeds diagonally (Fig. 2.13). Now the original single cycle data dependency extends to 5 cycles because of the diagonal scan. This allows pipelining the aggregation unit with 5 cycles and resolving inter-pixel dependency in a deep pipeline. Fig. 2.14 shows the proposed 16 stage deep pipeline for SGM processing. With the diagonal scan, there are 5 cycles between pixel A and F during which we can process the other 4 pixels (B, C, D, E). When F is fetched into the pipeline, the aggregated costs of previous pixels (light gray & dark gray) are already computed and stored in high bandwidth custom SRAMs. The critical path data from pixel A is forwarded to pixel F in the pipeline. This mechanism enables aggressive pipelining with a 4 ns clock frequency, yielding a 3× performance gain compared with that of the conventional raster scan. Moreover, because the data processed for pixel A are forwarded in the pipeline, this successfully eliminates unnecessary row buffer, leading to an extra 25% memory reduction. The 16-stage deeply pipelined design operates at a relatively low frequency (200MHz) to minimize the energy overhead of tremendous parallel pipeline registers if the design has more pipeline stages with higher frequency. As shown in Fig. 2.13, our design leverages parallelism in cost aggregation by running 4 paths in parallel on 4 aggregation units. Each aggregation unit contains 128 processing elements and 512 selection units, resulting in a total throughput of 1.882 TOP/s. Each OP is defined as 8-bit integer operation including add, subtract, compare and memory access.

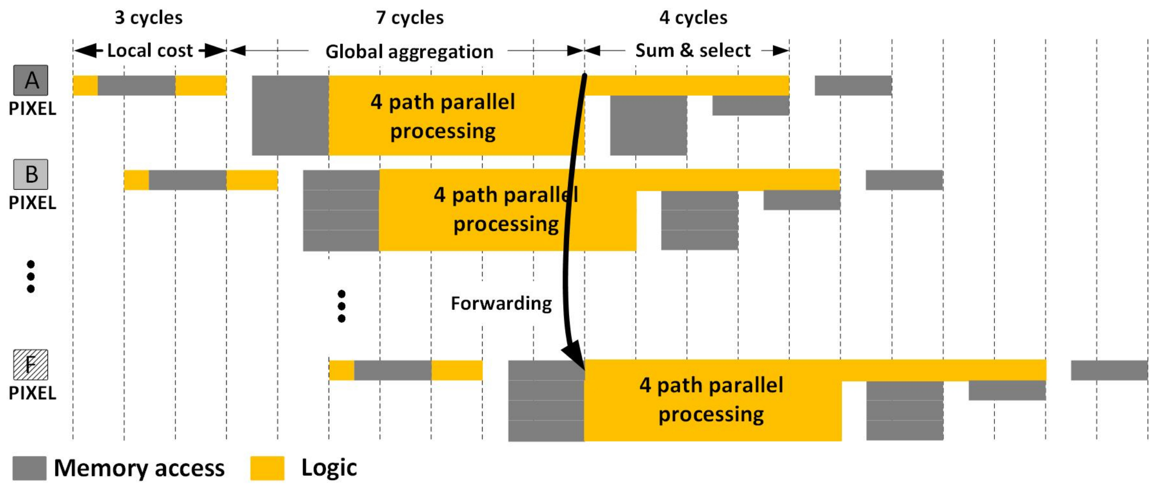


Figure 2.14: Pipelining and forwarding in SGM processing.

2.3.4 Custom Designed High Bandwidth 8T-SRAM

In the proposed design, each aggregation unit has its own ultra-high bandwidth row buffer. For each row buffer, 128 aggregated costs (12 bit each) are read and written simultaneously in a single cycle at 170 MHz. This translates to a total memory bandwidth of 1.64 Tb/s for the 3 row buffers accessed in parallel. This bandwidth would incur significant chip area and power overhead if realized with compiled SRAMs as a large number of banks and redundant peripherals are unavoidable due to the limited word length of compiled SRAMs. In simulation, Instead, we propose a custom-designed dual port SRAM to cope with this challenging memory access characteristic of SGM.

Because of the design's highly parallelized structure, the row buffer has a very unconventional aspect ratio: there are only 50 words in the buffer, but each word is 1612 bits wide. This motivates the proposed high bandwidth custom SRAM that provides enhanced area/power efficiency of SGM that was previously unattainable by general-purpose computing platforms. Fig. 2.15 shows the block diagram of the customized high-bandwidth SRAM. We partition the row buffer into 4 banks, and each bank has 50 words with a word size of 403 bits. All four banks are accessed in parallel with concurrent read and write functions, realizing 1612-bit dual port access.

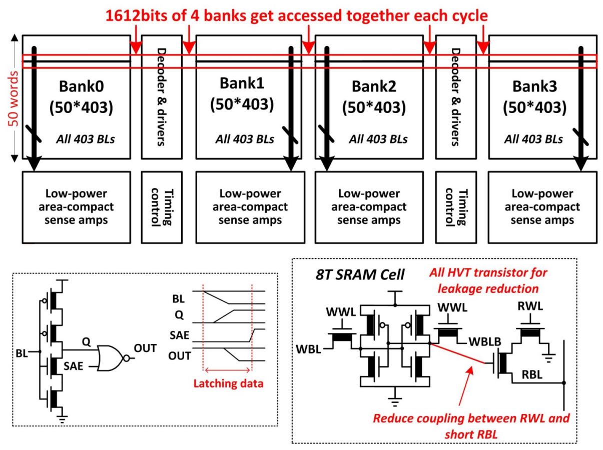
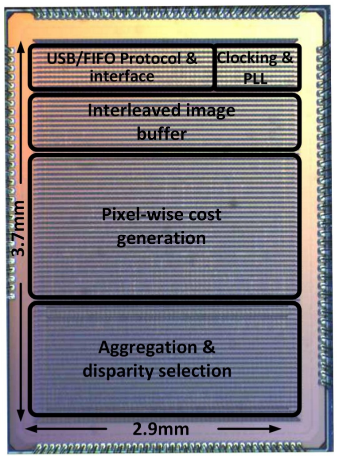

1612bits of 4 banks get accessed together each cycle
50 words
Bank0 (50*403)
All 403 BLs
Decoder & drivers
Bank1 (50*403)
All 403 BLs
Bank2 (50*403)
All 403 BLs
Decoder & drivers
Bank3 (50*403)
All 403 BLs
Low-power area-compact sense amps
Timing control
Low-power area-compact sense amps
Low-power area-compact sense amps
Timing control
Low-power area-compact sense amps
BL
Q
SAE
OUT
BL
Q
SAE
OUT
Latching data
8T SRAM Cell
All HVT transistor for leakage reduction
WWL
WWL
RWL
WBL
WBLB
RBL
Reduce coupling between RWL and short RBL

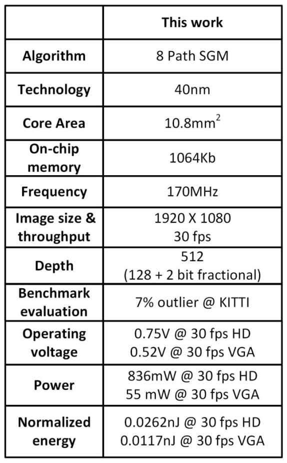
Figure 2.15: Block diagram and circuit of proposed high bandwidth 8T SRAM.

The very unbalance aspect ratio of this custom SRAM results in a massive number of very short bit lines (50 μm each) and very long word lines (380 μm each). Therefore, unlike conventional 8T cells, we propose swapping the position of the conventional 8T SRAM read transistor stack to avoid directly connecting the read access transistor to the read bit line (RBL). This approach effectively reduces coupling between the read word line (RWL) and the short, low capacitance RBL. In spice simulation with 0.9V nominal voltage, the coupling from RWL to RBL is reduced from 18mV to 2mV when read access transistor stack is flipped. Fig. 2.15 also shows the bit cell circuit in the bottom right panel. To reduce leakage power in the 40-nm technology, the custom 8T bit cell uses HVT transistors. The stacked skewed inverter-based sense amplifier and the timing of the SRAM read operation is shown in the bottom left of Fig. 2.15. Output latches are transparent during the RBL evaluation phase to ensure the correct memory read operation. Employing conventional sense amplifiers for 1612 bit lines would lead to significant area overhead. Therefore, skewed inverters perform RBL voltage sensing to achieve better area efficiency. Compared with conventional sense amplifiers, skewed inverters reduce the area overhead by 2.8$\times$. The low capacitance on the short BL allows the proposed SRAM to reliably operate at 200 MHz with a supply voltage as low as 0.6 V, further improving the energy efficiency. Overall, each 80-Kb SRAM is measured to consume only 6 mW with 548.1 Gb/s bandwidth at 200 MHz. Three banks operate at 200MHz with concurrent 1612-bit read and write operations, achieving 1.64 Tb/s access bandwidth with 18 mW power consumption.

2.4 Chip Measurements

Fig. 2.16 shows a die photo with a summary of the test chip performance. This work is fabricated in TSMC 40-nm GP process with 10.8 mm^2 chip area. TSMC 40-nm GP process has low nominal voltage (0.9V), and high performance (low Vth), which meets our design target. The fabricated chip successfully produces 512 levels of depth in full HD (1920×1080) resolution with real-time 30 fps performance with 170 MHz core frequency and consumes 836 mW from a 0.75 V supply. The depth image outputs produced by the chip using KITTI [49] automotive scenes are shown in Fig 2.17. Notice that the depth information of the cars in the shadow is successfully obtained. Large (>100 pixels) disparity frequently occurs at close distances, and the proposed processor is able to generate an accurate depth map over the entire image due to its 512 levels of resolution. The proposed chip achieves 7% outlier pixels running 194 KITTI evaluation images. Fig. 2.18 shows typical chip measurement results from Middlebury indoor scenes.


USB/FIFO Protocol & interface
Clocking & PLL
Interleaved image buffer
Pixel-wise cost generation
Aggregation & disparity selection
3.7mm
2.9mm



	This work
Algorithm	8 Path SGM
Technology	40nm
Core Area	10.8mm^2
On-chip memory	1064Kb
Frequency	170MHz
Image size & throughput	1920 X 1080 30 fps
Depth	512 (128 + 2 bit fractional)
Benchmark evaluation	7% outlier @ KITTI
Operating voltage	0.75V @ 30 fps HD 0.52V @ 30 fps VGA
Power	836mW @ 30 fps HD 55 mW @ 30 fps VGA
Normalized energy	0.0262nJ @ 30 fps HD 0.0117nJ @ 30 fps VGA

Figure 2.16: Die photo and summary of performance.

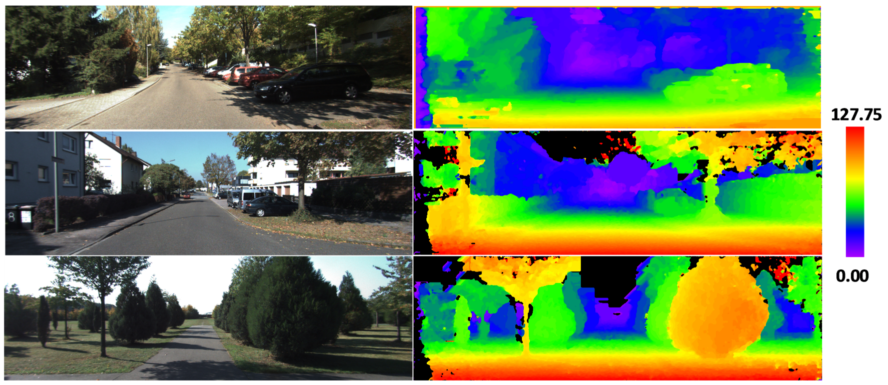


Figure 2.17: Measured result with KITTI tests.

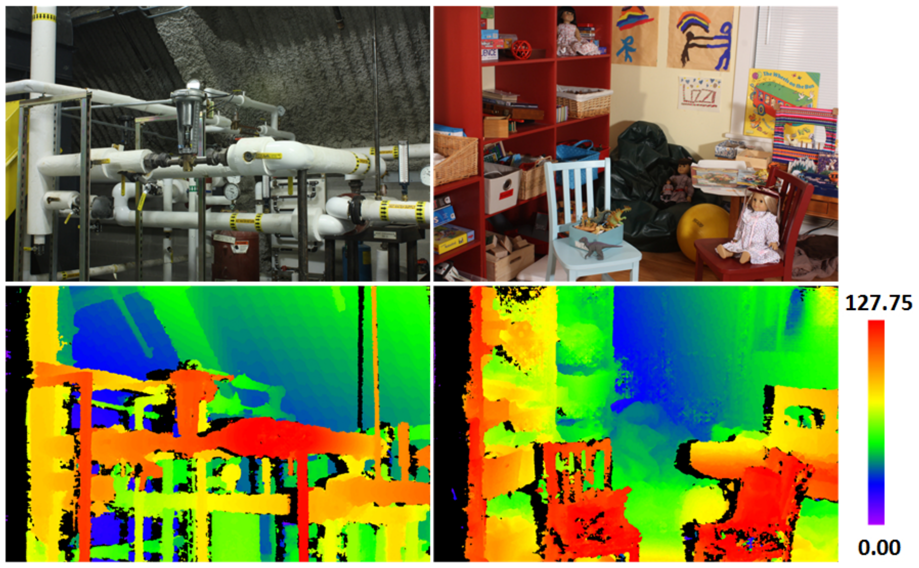
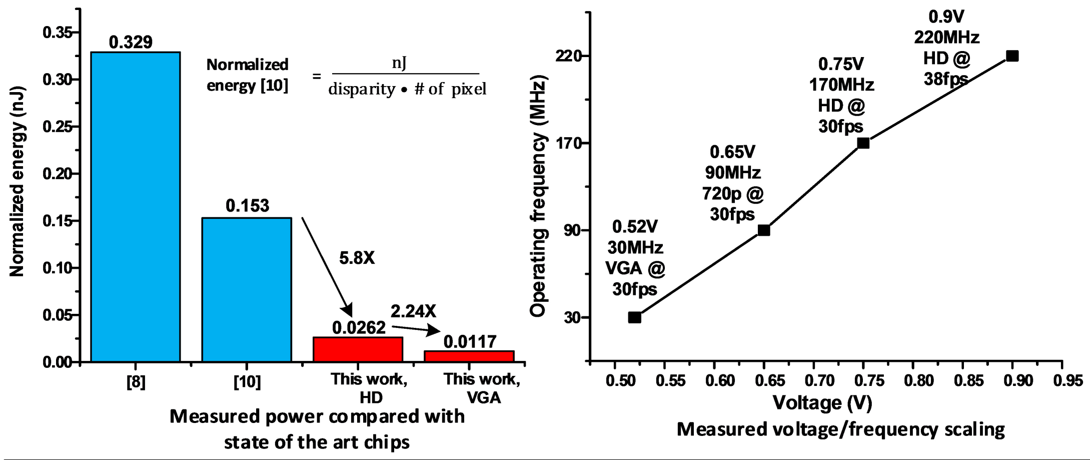


Figure 2.18: Measured result with Middlebury tests.

Fig. 2.19 shows the measured voltage and frequency scaling of the chip and provides a comparison with prior works. Compared with other state-of–the-art chips, this work implements SGM depth with 512 disparity levels, resulting in 8× improvement. It exhibits only 7% outliers in the KITTI benchmark, whereas other chips have limited disparity search ranges that are insufficient to run the industrial standard KITTI benchmark. The chip is programmable and supports various frame rates and image resolutions. It consumes 836 mW at 30 fps full HD. Power scales to 55 mW at 30 fps VGA at low voltage (0.52 V). Normalized energy is a FoM used in [38] and [37]. This work achieves 5.8× better FoM (energy per pixel per disparity) compared with other state-of-the-art works at 30 fps full HD resolution. Normalized energy scales to 0.0117 nJ at 30 fps VGA resolution, yielding 2.2× higher efficiency. Fig. 2.19 top-right shows the frequency and voltage scaling. The maximum chip performance is 38 fps for full HD resolution.



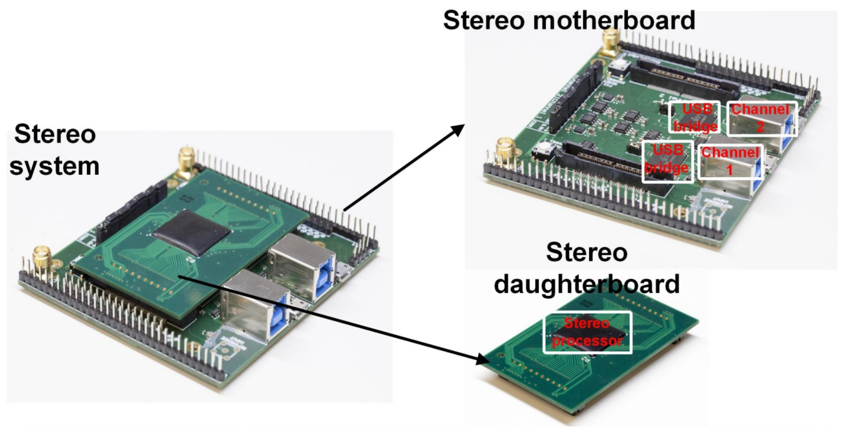
	VLSI 2004[7]	ISSCC 2015[10]	ISSCC 2016[8]	This work
Method	SAD	5-View BP	Truncated SGM	SGM
Technology	180nm	40nm	65nm	40nm
Chip area	25mm^2	22mm^2	16mm^2	10.8mm^2
On-chip memory	N.A	352Kb	3946.9Kb	1064Kb
Frequency	125MHz	215MHz	250MHz	170MHz
Throughput & image size	320 X 240 30fps	1920 X 1080 30fps	1280 X 720 30fps	1920 X 1080 30fps
Depth level	N.A	64 (32 + 1 bit fractional)	64	512 (128 + 2 bit fractional)
Accuracy (Outlier % on KITTI)	Not reported due to limited depth range			7%
Operating voltage	1.8 V	0.9 V	1.2V	0.75V @ 30 fps HD 0.52V @ 30 fps VGA
Power	N.A	1019mW (includes DRAM power)	582mW (excludes DRAM power)	836mW @ 30 fps HD 55 mW @ 30 fps VGA
Normalized energy*	N.A	0.153nJ	0.329nJ	0.0262nJ @ 30 fps HD 0.0117nJ @ 30 fps VGA

*** Normalized energy [10] - equation shown in top left**

Figure 2.19: Voltage & frequency scaling of the design and comparison with state-of-the-art chips.

2.5 System Integration and Evaluation

To demonstrate a complete system, the chip is integrated with a camera general processing (GPP) system and mounted on a real-time quadcopter platform. A small, light custom board is designed and fabricated to satisfy the 'SWaP' requirements for system integration on MAVs. Fig. 2.20 provides the board specs. Our system consists of the stereo daughterboard with the chip on top (covered with black epoxy) and a motherboard with two Cypress USB bridges where USB signals are converted to the 32-bit parallel interface. Fig. 2.21 shows the measurement setup and complete stereo system. The real-time image streams captured by the ZED stereo camera [51] are rectified, block-partitioned into 50×50 blocks by a Samsung Exynos-5422 processor on the ODRIOD-XU4[52] board, and then transmitted to the stereo processor through the input USB3.0 interface. Instructions are also sent to the chip with the same input USB3.0 interface that sustains the total 1.8 Gb/s bandwidth. The processed real-time depth images along with the 'confidence' side-information on each pixel are streamed back via the other (output) USB3.0 interface exhibiting 0.8 Gb/s bandwidth. Each 50×50 block is processed concurrently when the next block is being transmitted and stored on the on-chip interleaved image buffers. This technique minimizes camera-chip-depth latency. The real-time demonstration platform mounted on a quadcopter is shown in Fig. 2.22. At 0.9 V nominal voltage, the real-time VGA (full HD) frame processing latency of the stereo processor is 4.1 ms (26 ms), which is sufficient for real-time flight control. Table 2.3 shows the measured system power breakdown. The stereo vision board consume 20% of the system power. Fig. 2.23 shows qualitative results measured from our own quadcopter scene. As seen in the left image, strong illumination on a sunny day leads to saturation of the sky and grass, however the chip still generates accurate depth maps for navigation control.



Weight	Size	Throughput	Power
65g	50cm^2	30fps full HD	836mW @HD

Figure 2.20: Stereo system setup and summary.

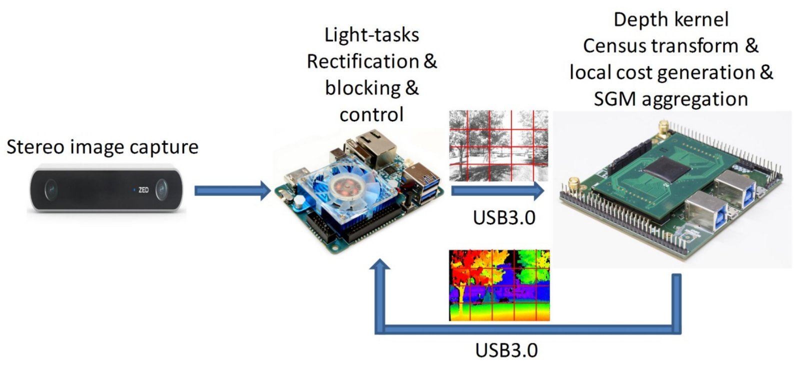

Stereo image capture
Light-tasks Rectification & blocking & control
Depth kernel Census transform & local cost generation & SGM aggregation
USB3.0
USB3.0

Figure 2.21: Chip measurement setup with stereo system.

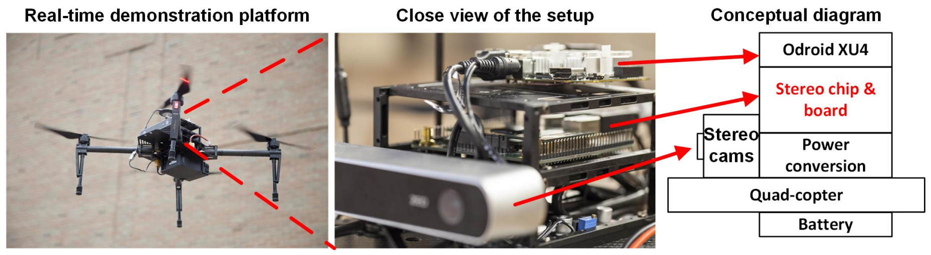


Figure 2.22: Real-time quadcopter demonstration platform.

	Stereo vision processor	2 Cypress FX3 USB bridge	Odroid XU4	ZED stereo camera
Power(W)	836mW	288mW	3.4W	1.9W*

*estimated from ZED camera [24]

Table 2.3: Measured system power break down.

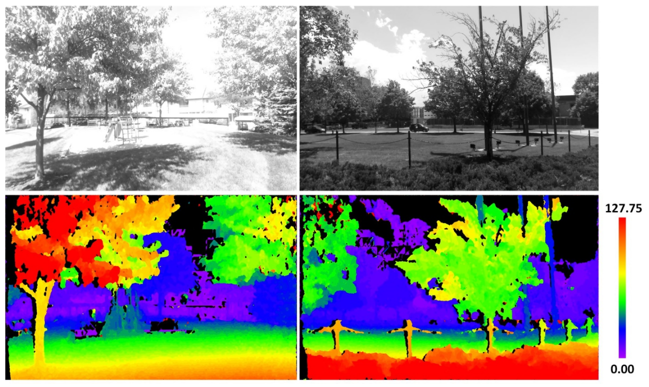


Figure 2.23: Measured HD result with quad-copter.

2.6 Summary

We present a single-chip, accurate, high performance, energy efficient depth estimation processor using the SGM algorithm for autonomous MAV applications. The fabricated processor generates 512 levels of depth in full HD (1920×1080) resolution with real-time 30 fps throughput consuming 836 mW from a 0.75 V supply in 40 nm CMOS. The chip reports 7% outliner on industry standard KITTI evaluation. The overlapping block-based processing achieves 95.4% memory reduction, eliminating the need for external DRAM at the cost of only 0.5% accuracy degradation. The proposed image-scanning stride with 16-stage deeply pipelined implementation yields 3× performance gain, 25% additional memory reduction and enables processing 512 level depth output at 30 frames per second for full HD resolution. Customized ultra-wide SRAM enables 1.64 Tb/s on-chip memory access bandwidth with 18 mW power consumption. The chip is measured with industry standard benchmarks. A complete stereo system is built and demonstrated on a quadcopter for realistic real-time operations.

CHAPTER III

A 1920×1080 25 fps 2.4 TOPS/W Low Power 6D Vision Processor for Unified Optical Flow and Stereo Depth with Semi-Global Matching

3.1 Introduction

In addition to the real-time high-performance and energy-efficient stereo vision system described in Chapter II, optical flow describes the 3D motion of surroundings and is also widely considered as a kernel function of autonomous navigation in micro aerial vehicles (MAVs), robots, and self-driving cars (Fig. 3.1). Stereo depth perceives 3D structure and optical flow tracks 3D motion field of the environment. Together, optical flow and stereo vision enable dense 6 Dimensional (6D) perception (3D coordinate and 3D motion) and are fundamentals for mobile autonomous system.

Real-time accurate and dense perception of 3D coordinates and apparent motion serves as a kernel function in simultaneous localization and mapping (SLAM), scene understanding/reconstruction, object tracking, and obstacle avoidance. A broad range of applications, including autonomous navigation of MAVs, require the depth and optical flow perception to be high resolution (e.g., dense FHD), accurate, wide range, low cost, and real-time with a high frame rate (e.g., >20 fps). Moreover, emerging miniaturized MAV applications impose additional stringent 'SWaP' (size, weight, and power) [30] constraints, for example <50 cm^3, <50 g, and <1 W. Although LIDAR [31] systems are widely used in autonomous systems and can provide accurate 3D depth, they cannot capture the motion information of objects in the scene. Moreover, LIDAR systems are difficult to miniaturize and do not produce dense results in high resolution. To supplement or replace LIDAR, camera and vision-based techniques have been widely investigated [49, 39, 40, 37, 28, 36, 65]. However, the high computational complexity of vision processing (especially optical flow) has been a major challenge for wide adoption in low power and low cost applications. To satisfy this technology need, we introduce a new 6D vision processor that, to our best knowledge, demonstrates for the first time real-time dense depth and optical flow computation with <1 W power consumption at FHD. This power budget does not impose significant overhead to the overall system that typically includes cameras (118.5mW each)[66], a mobile application processor (2W)[66], and a neuronal network accelerator (450mW)[67].

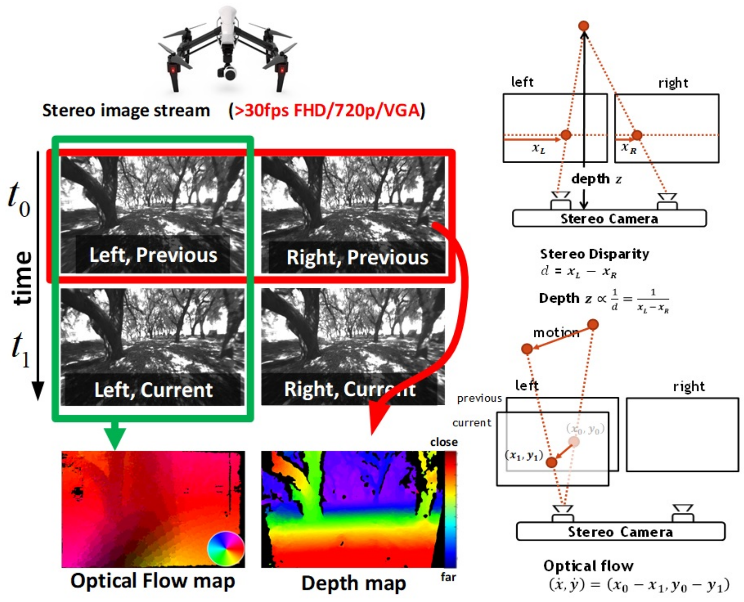


Figure 3.1: Optical flow and depth estimation on autonomous MAVs.

As described in Chapter II, with stereo cameras, the 3D depth of a pixel in the left image is inversely proportional to the horizontal one-dimensional (1D) displacement between the pixel and its matching pixel on the epipolar line of the right image [68] (Fig. 3.1, top right). 3D motion of a pixel in the current image is proportional to the 2D displacement between the pixel and its matching pixel in the next frame [69] (Fig. 3.1, bottom right). Unlike stereo depth, optical flow requires a 2D search to find the corresponding match as a projected point can move both in horizontal and vertical directions on the 2D image [69]. In this work, we combine the stereo depth and optical flow problems as 1D and 2D matching problems under the same semi-global matching (SGM framework) between images pairs. The complexity of the search problem is quadratically increased as the searching dimension augments from 1D (depth) to 2D (optical flow).

To process stereo depth and optical flow without quadratically increased complexity, we proposed a new, low-complexity optical flow method: Neighbour-Guided Semi-Global Matching (NG-SGM) in [48]. In this chapter, we provide comprehensive parametric analysis of NG-SGM along with a discussion on several optimization techniques in both algorithm and hardware for SWaP constrained MAV applications. The proposed NG-SGM method is based on SGM [55], a popular concept in stereo matching, and its optical flow version, fSGM [70]. We are able to achieve orders of magnitude complexity reduction of the original SGM while maintaining its high accuracy. With proposed NG-SGM approach, we studied the computation, memory, and bandwidth bottlenecks of the NG-SGM algorithm and proposed an algorithm, architecture, and circuit-level co-optimized design that significantly reduces the hardware cost. We also designed a unified architecture and a custom design crossbar circuit to accelerate both the stereo depth and optical flow while maintaining similar complexity with the NG-SGM approach.

The algorithm optimization techniques include: (1) aggressive decrease of searching space size by exploiting flow similarity of neighbouring pixels, (2) cost array aggregation approximation to avoid exhaustive pixel-wise cost computation, (3) image partitioning into overlapping blocks and boundary flow vectors initialization with temporal prediction to minimize accuracy degradation and overlap at the boundary, (4) execution of the full NG-fSGM algorithm on selective pixels and application of interpolation to construct dense flow field.

Although NG-SGM greatly reduces the complexity of stereo depth and optical flow processing compared to the original SGM, it still raises large implementation challenges for real-time FHD processing. Facing these implementation challenges, we custom-designed chip architecture and circuits. To mitigate irregular and redundant memory access patterns in NG-SGM, we introduce a new custom-designed 16×16-mesh, high-bandwidth (128 b/access, 4.08 Tb/s peak), 2-cycle pipelined coalescing crossbar with built-in memory access merging at each crosspoint. These coalescing crossbars are tightly integrated with 64 on-chip rotating buffers to maximize on-chip memory re-usability for a wide 2D searching range. In addition, a deeply pipelined hardware architecture with a skewed diagonal image scanning stride efficiently resolves the variable length inter-pixel dependency, significantly reducing the critical path for more aggressive clock frequency and voltage scaling. The fabricated chip employs two standard USB3.0-compliant interfaces, allowing effortless integration with a wide range of commercial off-the-shelf stereo cameras and general purpose mobile application processors. The chip supports a wide search range of 176×176 pixels to enable dense optical flow or stereo depth on FHD image pairs with real-time 25 fps/30 fps (flow/depth) throughput, consuming only 760 mW in 28 nm CMOS.

The content of this work in Chapter III was published previously in collaboration with Xiang Jiang [71].

3.2 Overview of 6D Vision Algorithms

The 3D depth of a point is inversely proportional to the horizontal 'spatial' displacement of that point between the left and right camera image frames taken at the same time. The optical flow of a point, on the other hand, is obtained by the 'temporal' displacement between the previous and current frames from the same camera as depicted in Fig. 3.1. The 3D motion of the object is proportional to the optical flow and inversely proportional to its depth. By combining 3D depth and 3D motion, 6D perception can be constructed [72].

3.2.1 Local Matching Algorithm

The census transform [73] is widely used for evaluating the correspondence between pixels. The $N \times N$ census transform converts each pixel to a bit stream of length $N^2 - 1$ to represent the intensity comparison between the center pixel and its surrounding $N^2 - 1$ pixels. The Hamming distance between two census transformed pixels is the pixel correspondence. This is typically referred as the 'local' matching cost because the matching of each pixel pair can be evaluated independently using only local pixels. In local approaches, the matching position is obtained by selecting the pixel with the minimum local cost. This can be generalized to a 1D search for stereo matching and 2D search for optical flow as shown in Fig. 3.1. Local approaches in general are unreliable [74] since they often fail to resolve ambiguities in many challenging scenarios such as occlusions, texture-less regions, transparency, and repetitive patterns where 'global' contexts are required to find the proper match. For example, the pixels of the wall on the right in Fig. 3.2 are saturated and texture-less due to strong illumination. The local approach applied to this region is mostly incorrect as shown in Fig. 3.2 because many pixels can have identical matching cost.

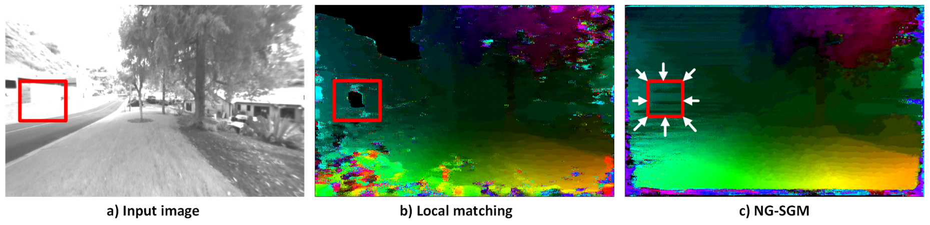


Figure 3.2: Comparison between local matching and SGM matching.

3.2.2 Semi-Global Matching Algorithms

As is described in Chapter II, the SGM algorithm was originally proposed by Hirschmüller for stereo matching[55]. 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 [57] 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 $C(p, \mathbf{o})$ between pixel $p = (x, y)$ in the previous image frame and pixel $q = p + \mathbf{o}$ in the current image frame, for all flow vectors $\mathbf{o} = (u, v)$, where u is the horizontal component and v is the vertical component. The cost function can be based on Rank, Census [58] and mutual information [57].

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

$$L_r(p, \mathbf{o}) = C(p, \mathbf{o}) + Z - \min_k L_r(p - r, k) \tag{3.1}$$

The cost regularization summand has the form

$$Z = \min L_r(p - r, \mathbf{o}), \min_{|\mathbf{i}-\mathbf{o}|^2 \leq 2} L_r(p - r, \mathbf{i}) + P_1, \min_j L_r(p - r, j) + P_2 \tag{3.2}$$

where P_1 and P_2 are regularization penalties ($P_1 \leqslant P_2$). Instead of the piecewise linear model used in fSGM[57], we adopt a constant penalty model because of its simplicity for VLSI implementation without significant accuracy degradation [75]. 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[55]. The aggregated cost $S(p, \mathbf{o})$ is the sum of $L_r(p, \mathbf{o})$ over all paths.

$$S(p, \mathbf{o}) = \sum_r L_r(p, \mathbf{o}) \tag{3.3}$$

Step 3: The final flow estimation uses winner-takes-all strategy. The flow vector $\mathbf{o}$ with the minimum cost $S(p, \mathbf{o})$ 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 [62] 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 [62] 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 chapter, 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.

3.3 Neighbor-guided Semi-Global Matching Algorithm

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 [76] and [77] 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.

3.3.1 Flow Subset Selection

It is highly likely that neighboring pixels on 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-SGM exploits this property by selecting a subset of search space, O_p for each pixel p, based on its neighboring pixels' flow results. Computation of pixel-wise matching costs is performed on the subset O_p whose size is much smaller than $D = d^2$. This selection strategy is inspired by PatchMatch [78], in which the search space is initialized to a random set and neighboring pixels exchange 'good guesses'.

The subset O_p selection for each pixel p is guided by its neighboring pixels along every aggregation path r in SGM, as shown in Fig. 3.3. For pixel p, O_p is initially empty. The best N flow vectors in O_{p-r} of previous pixel along path r with the minimum cost $L_r(p - r, \mathbf{o})$ are added to O_p to construct the search subset. Pixels correspond to these best N flow vectors are marked by B in Fig. 3.3. We may choose multiple ($N > 14$ 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 3.1) when the flow varies smoothly, it is reasonable to add adjacent flow vectors, marked by A in Fig. 3.3, around each of these best N vectors to 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. 3.3. 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.

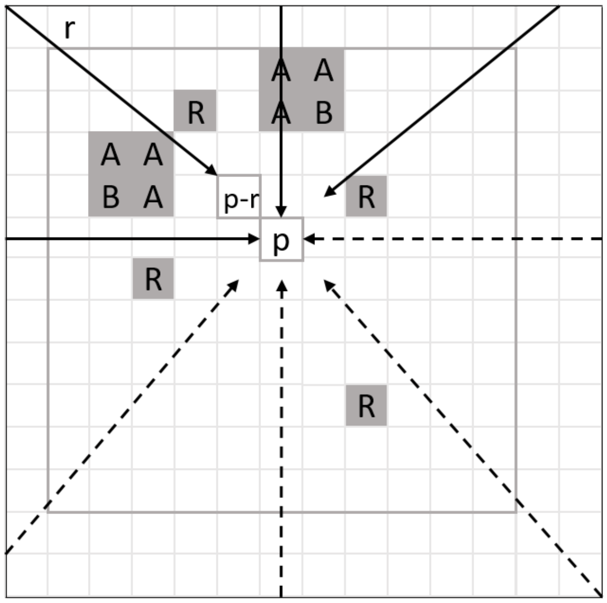


Figure 3.3: Illustration of subset selection for the center pixel p.

Typically, SGM is implemented for two scans, forward and backward, therefore SGM aggregation paths are divided into two groups one for each scan. In Fig. 3.3, 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 p has different flow vector subset O_{p1} and O_{p2}, and aggregated cost $S1(p, \mathbf{o})$ and $S2(p, \mathbf{o})$ for forward scan with subscript 1 and backward scan with subscript

2, respectively. The overall subset O_p is the union of O_{p1} and O_{p2} and the overall aggregated cost $S(p, \mathbf{o})$ is the sum of $S1(p, \mathbf{o})$ and $S2(p, \mathbf{o})$. We propose in Section 3.3 an approximation strategy to combine forward and backward aggregation. In the backward scan, N best flow vectors with the forward scan minimum cost $S_1(p, \mathbf{o})$ is added to construct 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. 3.3) can overlap. When candidates are all exclusive (worst case), the total number of vectors in the search subset 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. 3.3 shows an example for $K = 2 \times 2$ window. The complexity of NG-SGM is $O(WHT)$, which is independent of flow search range ($D = d^2$). With $T \ll D$, significant (>10×) complexity reduction can be achieved compared to the original fSGM.

3.3.2 Pixel-wise Matching Cost

For pixel-wise matching cost $C(p, \mathbf{o})$, we adopt Hamming distance of Census transform [58]. Census transform has been proven to represent image structure well and to be robust even with illumination variations [73].

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-SGM, only a subset Op of full flow candidate vectors needs to be evaluated. The calculation of $C(p, \mathbf{o})$ can be performed on-the-fly only when $\mathbf{o}$ is selected during the cost aggregation step. For this approach, storing an array of precomputed pixel-wise matching cost $C(p, \mathbf{o})$ is unnecessary. However, memory bandwidth required for Census transform per pixel is significantly higher because an efficient sliding window approach is no longer applicable. We also 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.

3.3.3 Cost Aggregation and Flow Computation

In order to compute Z in equation 3.2 for cost aggregation referring to equation 3.1, typical SGM-based methods store $L_r(p, \mathbf{d})$ in a line buffer array of size WPD. NG-SGM 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 $L_r(p, \mathbf{o})$, the cost $L_r(p - r, \mathbf{o})$ may not be available for O_{p-r} along a certain direction r. In that case, the aggregated cost is approximated by assigning

$$L_r(p, \mathbf{o}) = \min_j L_r(p - r, j) + P_2 \tag{3.4}$$

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

For each pixel p visited during the backward scan, the set of N best vectors B_p, from forward scan and the neighbor-guided vectors from backward scan, 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 $S(p, \mathbf{o})$: The missing costs $S_1(p, \mathbf{o})$ in forward scan are assigned the maximum cost in B_p plus P_2, while the missing costs in backward scan $S_2(p, \mathbf{o})$ are assigned the maximum cost in O_{p2}. The overall cost $S(p, \mathbf{o})$ is the sum of cost from two scans. Finally, the output flow vector $\mathbf{o}$ is the one corresponding to the minimum cost $S(p, \mathbf{o})$.

3.3.4 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 [55]) can be applied to create the confidence map.

3.4 Optimizations for Hardware Efficient NG-SGM Algorithm

NG-SGM allows optical flow complexity reduction with aggressive search space pruning. However, its complexity could still be excessive for power-constrained real-time mobile applications. For full HD (1920×1080) resolution, NG-SGM 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-SGM-specific optimizations. Proposed techniques include: 1) performing overlapping-block based NG-SGM in parallel 2) initializing flow search space with temporal prediction 3) performing sparse flow estimation with NG-SGM and interpolating results to obtain dense flows.

3.4.1 Parallel Block-based NG-SGM

We propose parallel block-based NG-SGM 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.

In a naive block-based implementation, input frames are partitioned into non-overlapping blocks and the NG-SGM 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-SGM 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-SGM, 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. 3.4 shows an example of a non-overlapping block and its extensions.

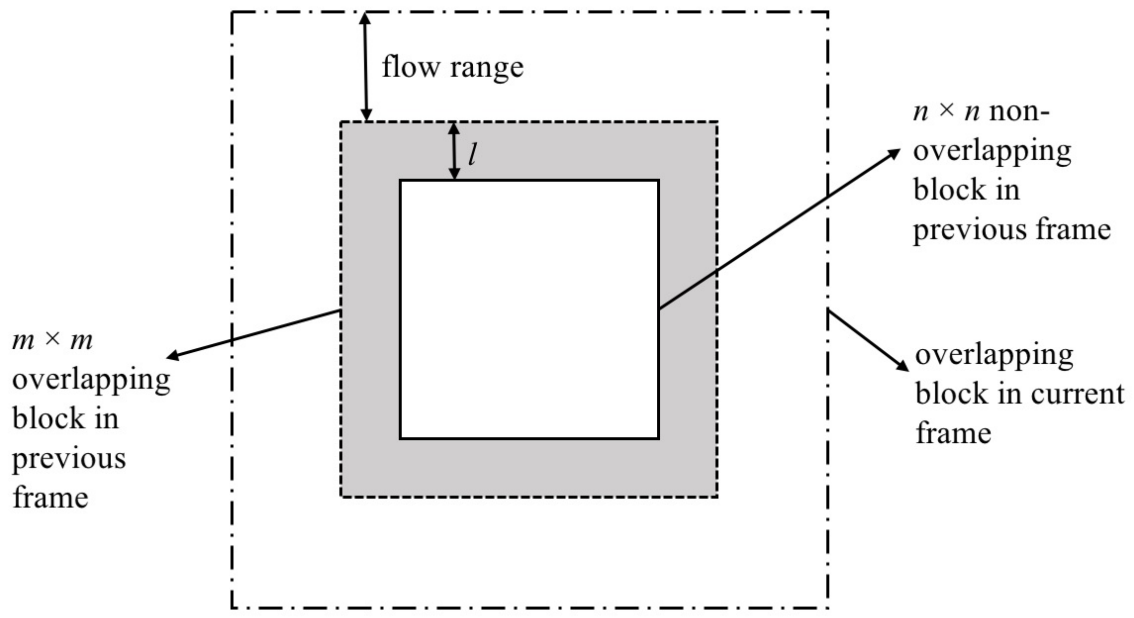

flow range
l
$n \times n$ non-overlapping block in previous frame
$m \times m$ overlapping block in previous frame
overlapping block in current frame

Figure 3.4: An example of an n×n non-overlapping block in the previous frame.

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 3.1 summarizes the effect of parameters n and l on the memory size, number of operations and memory bandwidth (in Bytes) requirements to process a $W \times H$ 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 3.5.

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.

Table 3.1: Memory size, number of operations, memory bandwidth of NG-fSGM with optimizations.

3.4.2 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 [79]. 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 3.5 holds.

$$(u, v)_{(i+u', j+v')} = (u', v')_{(i,j)} \tag{3.5}$$

We use 3.5 to provide the inertial guided flow estimates. For each pixel in the extended region (grey area in Fig. 3.4) 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.

3.4.3 Sparse-to-Dense Optical Flow Estimation

To further reduce the number of operations of NG-SGM, 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-SGM 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. 3.5

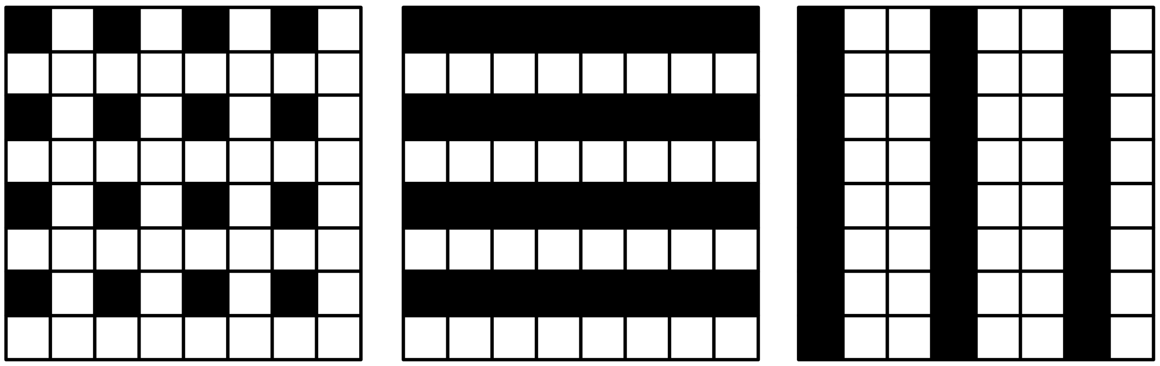


Figure 3.5: Sampling pattern examples where grey pixels are sampled.

NG-SGM aggregation and optical flow computation are initially performed on these subsampled pixels only. Once NG-fSGM is complete for selective (subsampled) pixels, dense optical flow of remaining pixels marked in white in Fig. 3.5 is computed by interpolating the result of subsampled (black) pixels in Fig. 3.5. This approach is similar in spirit to [80]. While in [80], 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.

Different sampling patterns exemplified in Fig. 3.5 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 $\mathbf{o}_p$ at pixel p is estimated by the bilinear interpolation of nearest subsampled (i.e., black pixels in Fig. 3.5) neighbors' optical flow vectors. This approach reduces the required memory size and the number of operations by a factor of $f_1 f_2$. We summarize the impact of these hardware-oriented optimization on the memory size, number of operations, and memory bandwidth in Table 3.1.

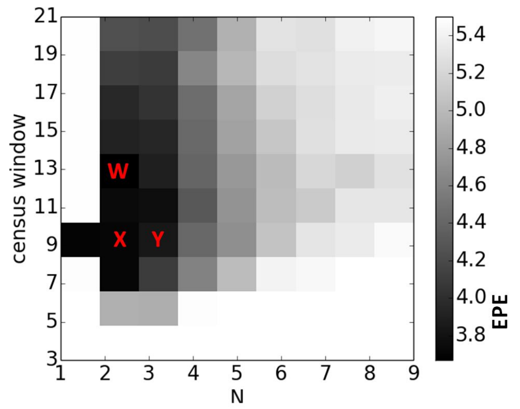
3.5 Evaluation of Hardware Oriented Optimizations

We conducted comprehensive experiments on the Middlebury[63], KITTI[49] and MPI[81] 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)[63]. The EPE is the averaged error radius of optical flow estimates of all pixels[81]. The memory size, number of computations and memory bandwidth requirements are used to quantify the hardware complexity.

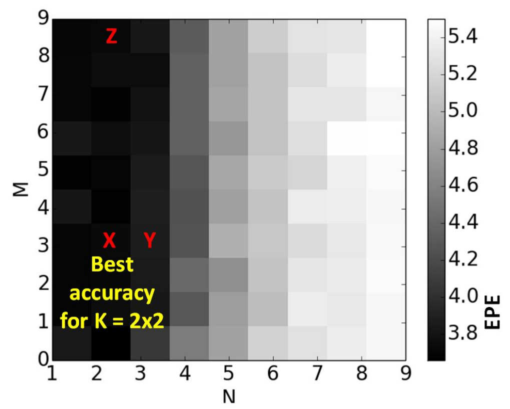
3.5.1 Parametric Analysis on NG-SGM

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. 3.6, parameters N and M are enumerated from 0 to 9, and C_2 is evaluated from 5×5 to 19×19. Fig. 3.6 (a) and (c) visualize the algorithm accuracy for $K = 1 \times 1$ and 2×2 respectively on Middlebury test in EPE (the darker, the more accurate) when $M = 3$ and $P = 8$. Fig. 3.6 (b) and (d) show the impact of N and M when $C = 11$, and $P = 8$ for $K = 1 \times 1$ and 2×2, respectively.

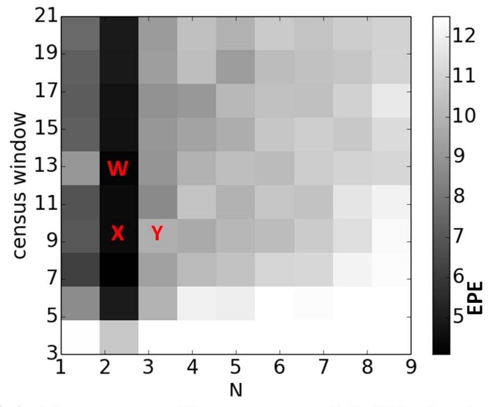
Fig. 3.6 (a) and (b) show that, when the search window size K is 2×2, the optimal accuracy is obtained with $N = 2$ or 3. Notice that more best-candidates (larger N) from the neighbor degrades the algorithm accuracy because the smoothness constraints weakens when more (some are incorrect) candidates are admitted. Regarding the Census size, the algorithm accuracy stabilizes when C is larger than 9 but de-



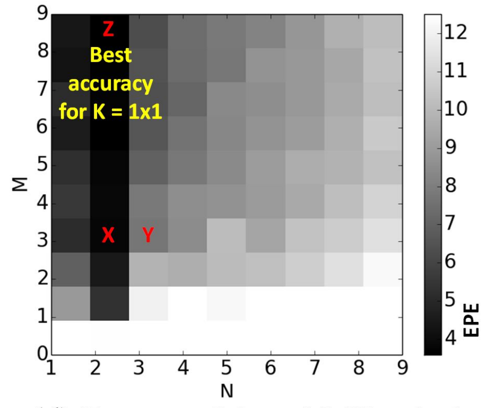
(a) Error vs. Census vs. N, K = 2×2



(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

Figure 3.6: Error performance analysis with parameter sweep (M, C, K, N).

grades when the Census size is too large (e.g., $C \geq 17$) 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 $M \geq 1$.

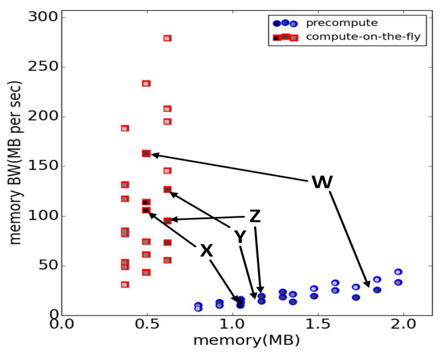
In Fig. 3.6 (c) and (d), K is changed from 2×2 to 1×1. It is worth noting that the impact of parameter N, M and C on algorithm accuracy shows similar non-monotonic trend as in Fig. 3.6 (a) and (b). With a smaller K, more random candidates with $M \geq 3$ are desired to compensate for the reduced number of neighbor guided candidates, and thus to minimize the algorithm accuracy degradation. Point'Z' ($N = 2$, $M = 9$, $C = 9$ with $K = 1 \times 1$) in Fig. 3.6 (a) has the best accuracy of 3.69% EPE comparable to the point 'X' in Fig. 3.6 (d) (3.67% EPE, with $N = 2$, $M = 3$,

$C = 9$ and $K = 2 \times 2$).

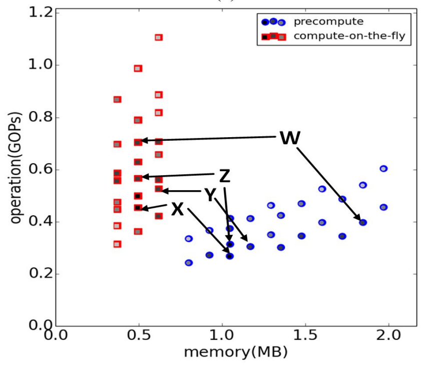
The algorithm complexity in terms of memory size, memory bandwidth and operation counts is visualized in Fig. 3.7 (a) and (b) when $K = 2 \times 2$. Analysis for $K = 1 \times 1$ is omitted as it exhibits a similar trend. The comparison between Census transform on-the-fly and pre-computed Census is also analyzed in the same figures. The algorithm accuracy – complexity tradeoff can be identified by associating 'X, Y, Z, W' points in Fig. 3.6 (c), (d) to the same labelled points in Fig. 3.7. Points in Fig. 3.7 are shaded with different levels to represent algorithm accuracy; darker shades represent better algorithm accuracy following the same convention in Fig. 3.7. 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'->'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 $K = 2 \times 2$ 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 3.1. A smaller N implies more aggressive candidate pruning and



(a)



(b)

Figure 3.7: Performance and complexity tradeoff analysis: (a) Memory BW vs. Memory size vs. Error rate, (b) # of OPs vs. Memory size vs. Error rate.

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: $K = 2 \times 2$, $C = 9$, $N = 1$, $P = 8$, $M = 1$.

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.

Table 3.2: Comparison of NG-fSGM, fSGM & Lucas-Kanade.

Table 3.2 compares the accuracy and complexity of NG-SGM with the aforementioned parameter set to the original fSGM[70] and Lucas-Kanade (LK)[69], for the Middlebury, KITTI and MPI benchmarks. NG-fSGM with $K = 2 \times 2$, C=9, N=1, P=8, M=1 is used for all three benchmarks. As Table 3.4 indicates, the proposed NG-SGM provides significant complexity reduction compared to fSGM since NG-SGM 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 3.4 has a limited $d \leq 32$. The complexity gap between fSGM and NG-SGM is more significant for MPI[81] and KITTI[49] benchmarks that require a larger d. The fSGM and LK complexity for KITTI and MPI reported in Table 3.4 is based on a three-level hierarchical pyramid approach[70] that limits d to 40 (fSGM) or uses a 13×13 search window with 10 iterations (LK) for each level. Non-hierarchical fSGM (d=40) and LK (13×13, 10 iterations) is used for the Middlebury . We observed that NG-SGM significantly outperforms LK in accuracy at the cost of slightly increased memory area requirement. The number of operations for NG-SGM is lower than that of LK for all benchmarks.

To visualize the algorithm quality difference, Fig. 3.8 shows 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-SGM (2nd row) and fSGM (3rd row) both outperform LK (4th row). The raw (before post-processing) flow map output of NG-SGM shows blurry results along object edges but has fewer error patches compared to fSGM. The neighbor guidance in NG-SGM 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 3.2 confirms that the overall accuracy of NG-SGM for Middlebury evaluation is comparable to that of fSGM.

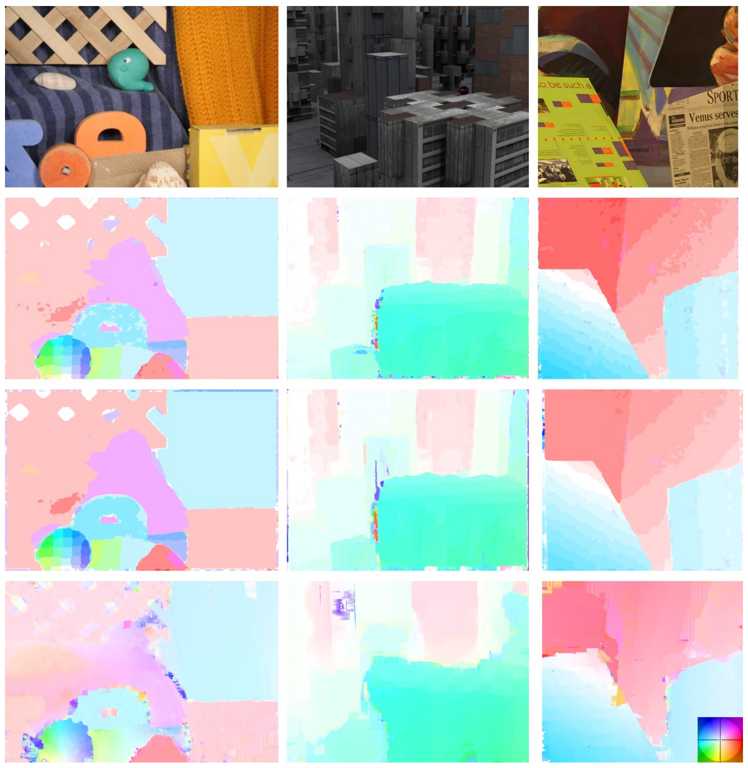


Figure 3.8: Colored flow maps using different algorithms.

For the KITTI and MPI dataset, the search range is substantially larger; $d = 128$ for a non-hierarchical approach. Unlike NG-SGM, a multi-level hierarchical approach is preferred for LK and/or fSGM to limit d (e.g., 40) for each pyramid level. Fig. 3.9 and 3.10 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-SGM exhibits 0.63% degradation compared with a hierarchical fSGM[70] while achieving >10× memory and computation complexity reduction. NG-SGM 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 [70] and so could not be included.

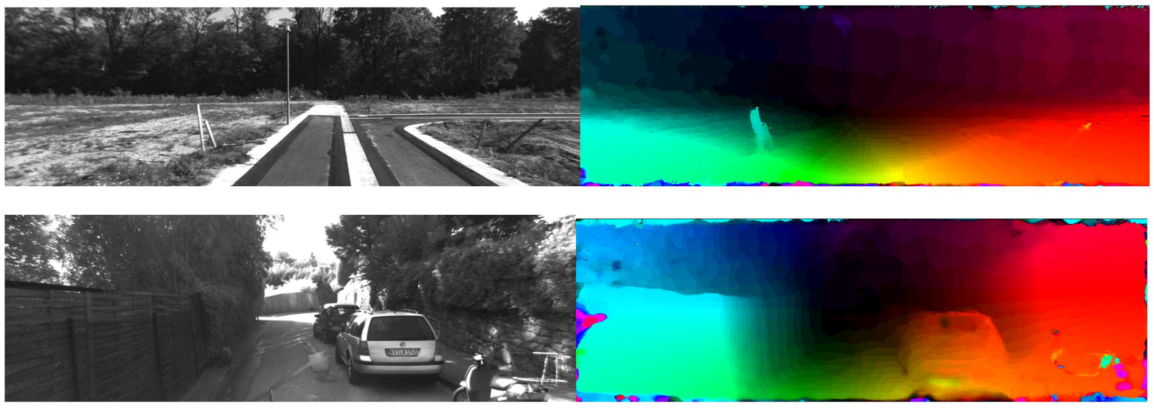


Figure 3.9: Colored flow maps using KITTI dataset.

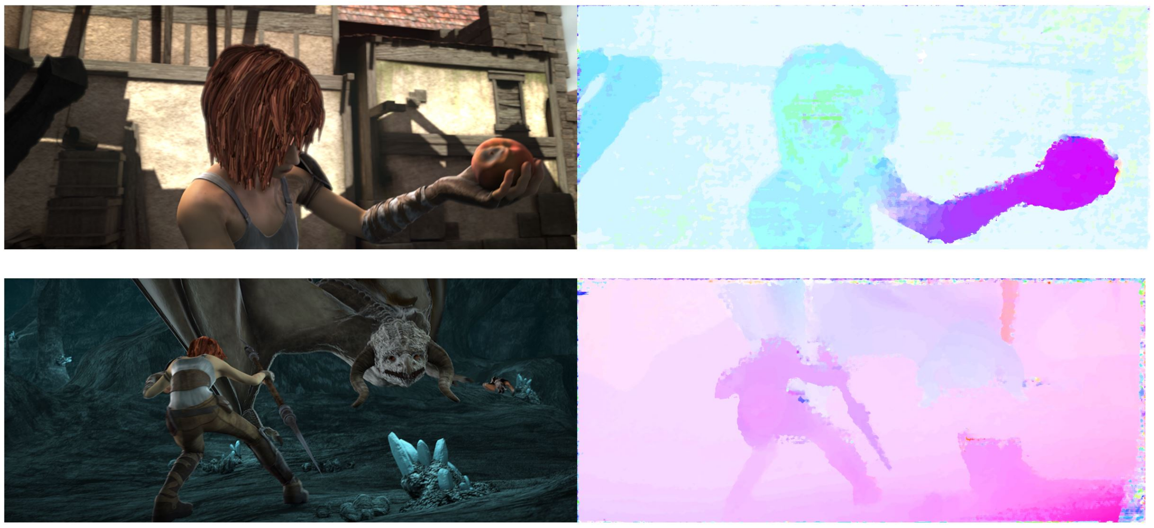


Figure 3.10: Colored flow maps using MPI dataset.

3.5.2 Overlapping Block-based NG-SGM

So far, for more direct comparison to other algorithms, NG-SGM complexity and accuracy have 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-SGM.

We evaluate the basic overlapped block-based processing (denoted by NG-SGM+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. 3.11, 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. 3.11 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.

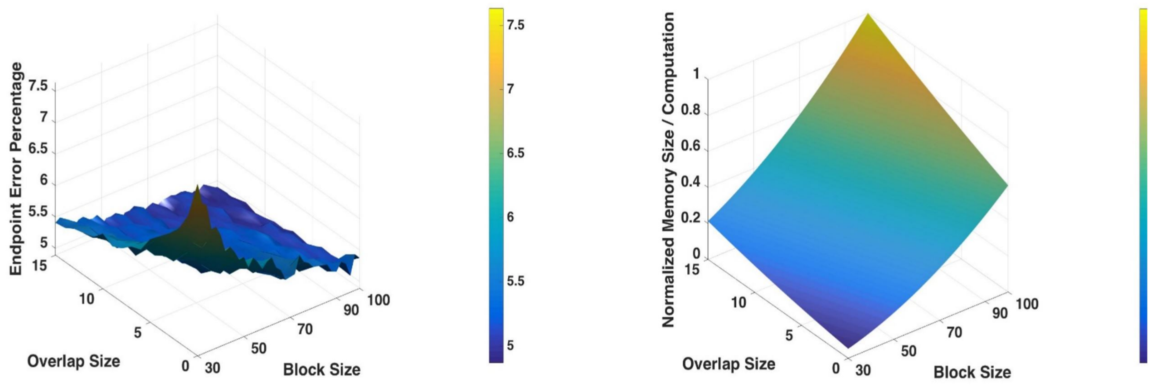


Figure 3.11: Left: accuracy of NG-SGM+B using the Middlebury training dataset. Right: complexity vs. overlap size (l) vs. block size (n).

3.5.3 Block-based NG-fSGM with inertial guidance

We evaluate the effect of the proposed inertial guidance technique on Middlebury dataset. Fig. 3.12 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-SGM. This implies that the inertial estimates can provide useful guidance for flow vector initialization in NG-fSGM. Table 3.3 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 3.5). 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.

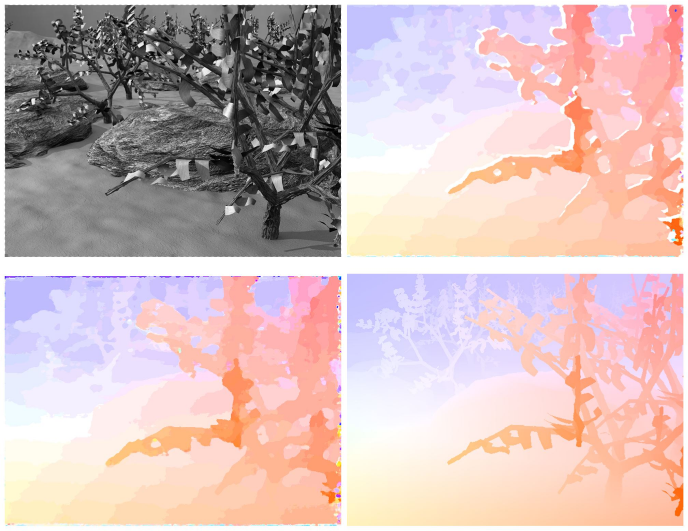


Figure 3.12: An example of inertial guidance. Top left: Input frame. Top right: Inertial estimates. Bottom left: NG-SGM+BI. Bottom right: Groundtruth.

Note that block-based NG-SGM 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.

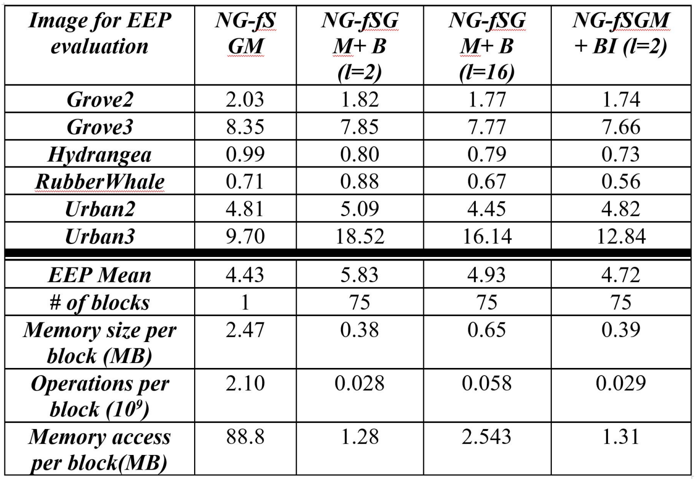


Image for EEP evaluation	NG-fSGM	NG-fSGM+ B (l=2)	NG-fSGM+ B (l=16)	NG-fSGM + BI (l=2)
Grove2	2.03	1.82	1.77	1.74
Grove3	8.35	7.85	7.77	7.66
Hydrangea	0.99	0.80	0.79	0.73
RubberWhale	0.71	0.88	0.67	0.56
Urban2	4.81	5.09	4.45	4.82
Urban3	9.70	18.52	16.14	12.84
EEP Mean	4.43	5.83	4.93	4.72
# of blocks	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

Table 3.3: Endpoint error percentage (EEP) and complexity on Middlebury multi-frame training dataset.

While the original NG-SGM (processing the whole image as a single block) has a mean error of 4.43%, NG-SGM+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%.

3.5.4 Sparse-to-Dense Optical Flow Estimation

The last optimization technique we evaluate is the sparse-to-dense NG-SGM with different sampling rates f_1 and f_2. The sparse-to-density NG-SGM is evaluated on Middlebury images with $n = 64$ and $l = 16$. Table 3.4 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.

	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

Table 3.4: Endpoint error percentage (EEP) and complexity on Middlebury multi-frame training dataset.

3.5.5 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)[82]. Results are shown in Table 3.5. 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-SGM accuracy even when a 25×25 weighted median filter is applied. Applying this weighted median filter introduces ≈25% extra computation for pyramidal LK.

	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

Table 3.5: Outlier percentage (EEP) with different post processing schemes.

3.6 Architecture and Circuit Implementations

Although proposed NG-SGM algorithm greatly reduces the complexity of stereo depth and optical flow processing compared to the original SGM, it still poses large implementation challenges for real-time FHD processing. First, it requires very high memory bandwidth of about 2.6 Tb/s with highly irregular access patterns because of the dynamic search space pruning of the NG-SGM algorithm. Second, NG-SGM involves a large memory footprint of about 4 MB to enable a wide-range 2D search for optical flow. Third, the variable-latency data dependency from neighboring pixels makes the control of pipelined massive parallel processing more challenging. To make SGM feasible for low power real-time 6D vision, we also custom designed a hardware architecture that accelerates the NG-SGM to achieve high throughput and energy efficiency for power critical real-time mobile systems.

3.6.1 Overview of NG-SGM Datapath

NG-SGM realized on our 6D vision processor involves five steps (Fig. 3.13): 1) Fetching, block-partitioning and scheduled transfer of image blocks; 2) aggressively pruned neighbor-guided flow candidate search; 3) pixel-wise matching cost computation; 4) cost aggregation of the sparsely populated local cost cuboid; and 5) optical flow selection and guidance propagation to neighbor pixels. To compute the pixel-wise local matching cost, we selectively apply a 9×9 census transform on the matching candidate pixel pairs. Each candidate pixel is transformed to a bit string of the length 80. The local matching cost $C(p, d)$ for a pixel at the location p with the displacement vector d is evaluated by the Hamming distance[79] between the census transformed pixel pair at p and $p - \mathbf{d}$ on the current and target image pair, respectively.

We repeat this operation on all candidates in the pruned search space, constructing a cuboid that is sparsely populated with local matching costs for each pixel as shown in Fig. 3.13. Because of NG candidate pruning, there are at most 64 non-zero costs per

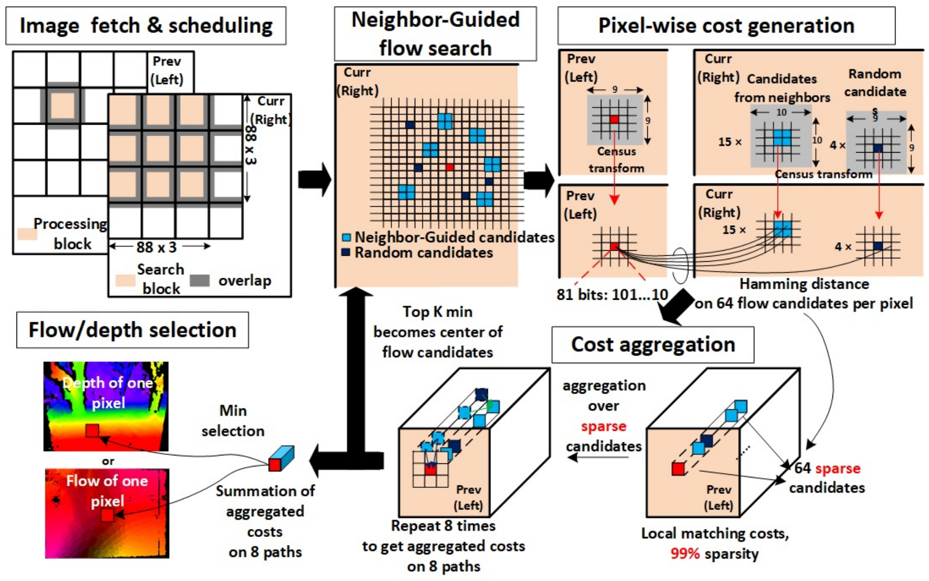


Image fetch & scheduling
Prev (Left)
Curr (Right)
88 x 3
Processing block
88 x 3
Search block
overlap
Neighbor-Guided flow search
Curr (Right)
Neighbor-Guided candidates
Random candidates
Pixel-wise cost generation
Prev (Left)
Census transform
Curr (Right)
Candidates from neighbors
Random candidate
Census transform
Prev (Left)
Curr (Right)
81 bits: 101...10
Hamming distance on 64 flow candidates per pixel
Top K min becomes center of flow candidates
Flow/depth selection
Depth of one pixel
or
Flow of one pixel
Min selection
Summation of aggregated costs on 8 paths
Repeat 8 times to get aggregated costs on 8 paths
Cost aggregation
aggregation over sparse candidates
64 sparse candidates
Local matching costs, 99% sparsity

Figure 3.13: Data path of the proposed NG-SGM 6D vision algorithm.

pixel in the 3D cost cuboid. We then perform SGM aggregation following equation 3.1 on matching costs using 4 paths (rs) for the forward scan and another 4 paths for the backward scan (Fig. 3.14). In total, NG-SGM aggregation is performed on 8 paths for every pixel as shown in Fig. 3.14.

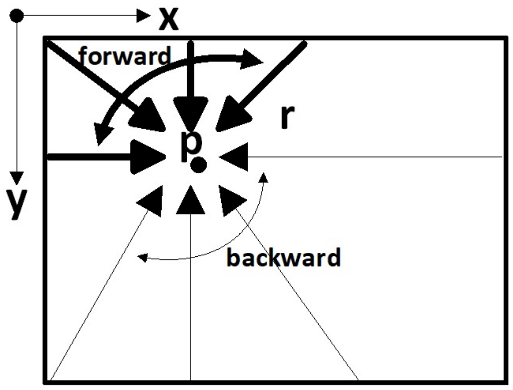


Figure 3.14: 8 path aggregation of NG-SGM.

When the cost $C(p, \mathbf{d})$ is unavailable for some ds because of candidate pruning, it is replaced by max$L_{rstored}(p, \mathbf{d}) + P_2$. Finally, for each pixel, we get a vector of summated cost as in equation 3.3.

The vector d with the minimum summated cost $S(p, \mathbf{d})$ is the final displacement (either optical flow or stereo disparity) vector for the current pixel. At the same time, we also select the best 3 candidates (marked as A in Fig. 3.13) for each aggregation path and hand this information to neighboring pixels to guide candidate pruning. The maximum number of pruned candidates per pixel is 64, which consists of 3(best candidates from each path) $\times$ 4(surrounding pixels for each best candidate) $\times$ 5(4 paths per scan and the forward scan result) + 4 random candidates. Application of NG-SGM for 1D stereo depth matching has not been discussed in the literature [48, 47], but it is straightforward.

3.6.2 Interface and Architecture Overview

Fig. 3.15 shows the chip architecture and datapath overview. On-chip control registers and 64 on-chip rotating input image buffers are memory mapped and can be accessed with a USB3.0 interface through an external USB-to-parallel converter [64]. During operation, one 32-bit parallel interface streams input images and processing instructions into the chip, and the other 32-bit parallel interface streams the final optical flow or depth disparity images off from the chip. A streaming mode is supported so that the input/output images are streamed on/off the chip continuously to maximize the bandwidth of the interface. The block-partitioned previous (or left) and current (or right) frames for computing optical flow (or stereo) are stored in 64 on-chip rotating image buffers (20 Kb each, 1280 Kb in total). Optical flow and stereo processing are performed concurrently when the input/output image blocks are transferred to support real-time streaming operation.

As the first step of NG-SGM, aggressively pruned matching candidates are selected based on guidance from neighboring pixels. Hamming distances of census transformed pixels at $d = 64$ different sparse optical flow/disparity locations are then compared in parallel. This produces 64 pixel-wise matching costs $C(p, \mathbf{d})$ for the 64 candidate ds. These pixel-wise matching costs are all sent to 4 parallel aggregation units for

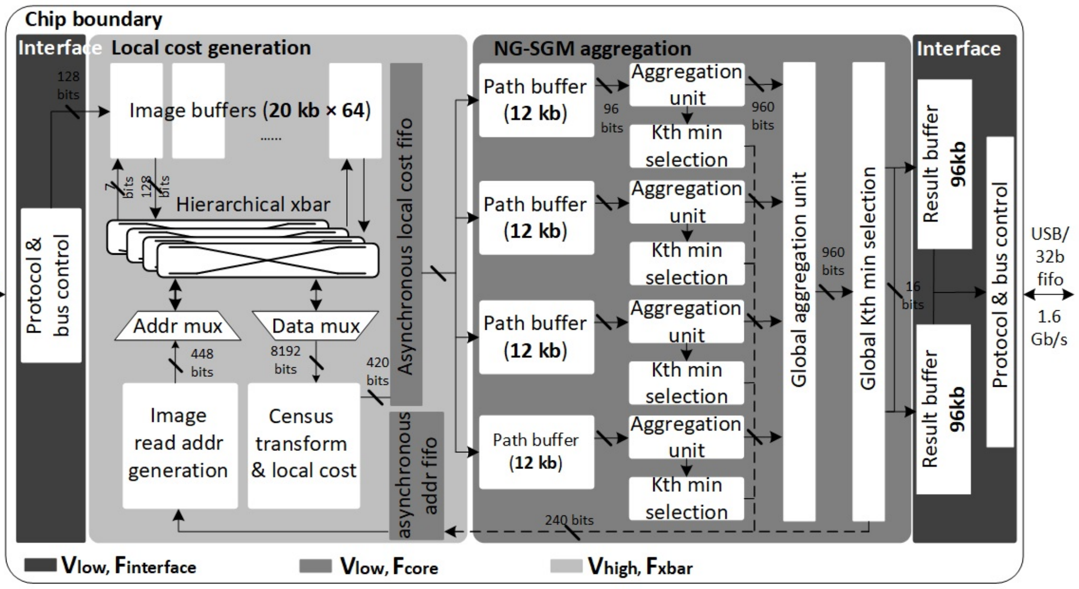


Figure 3.15: Chip architecture of 6D vision processor.

NG-SGM aggregation. Each aggregation unit is equipped with a small 12 Kb buffer to aggregates 64 candidates for each path in parallel. In each aggregation unit, the 64 candidates are divided into 19 clusters (15 clusters guided by neighbors and 4 random candidates). Candidates in a neighbor guided cluster have at most ± 1 displacement vector difference. To simplify evaluating 3.1, only one candidate in that cluster is compared with the three best displacement vectors of the previous neighboring pixel, and P_1, P_2 (penalties) are added to the aggregated cost depending on the distance between the candidate and the previous displacement vectors. The aggregated costs of the remaining candidates in that cluster are directly obtained based on the position of the evaluated candidate. This eliminates 75% of redundant comparisons and saves 68% power on cost aggregation. Also, it helps parallelizing the aggregation of 4 paths with 64 candidates per path to achieve higher throughput and energy efficiency (Fig. 3.15).

Each aggregation unit is then followed by tree-structured selection units to identify the best 3 aggregated costs and corresponding displacement vectors, ds, for each aggregation path, while there are 4 paths for either forward or backward propagation. As shown in Fig. 3.16, the top-1 minimum cost is selected based on the comparison of a full radix-2 tree. The second minimum is selected among the (green) elements that were compared and lost to the top-1 minimum. The third minimum is selected among the (green and blue) elements that were compared and lost to the top-1 and top-2 minimums. This implementation saves 52% energy consumed in candidate selection compared with implementing three full radix-2 trees. These path-wise optimal optical flows and costs are sent to the next processing pixel to guide its search space pruning. After the costs are aggregated on each path, aggregated costs over 4 different paths are accumulated, resolving the sparse overlap of candidates among multiple paths. Then a tree structured selection unit identifies the best 3 aggregated costs and corresponding ds from the accumulated costs. These 3 best aggregated costs for each pixel are stored in the on-chip 96 kB result buffer. The processor first completes these steps for each pixel following the skewed-diagonal forward scan. The backward scan is performed in a similar fashion but in the reverse order. The aggregation results that are discarded (all except the 3 best results) during the forward scan are replaced with the maximum stored accumulated cost in the backward scan. The final best candidates are selected based on the minimum cost from the aggregation of 8 paths. Finally, the selected d vectors are stored in two interleaved output buffers.

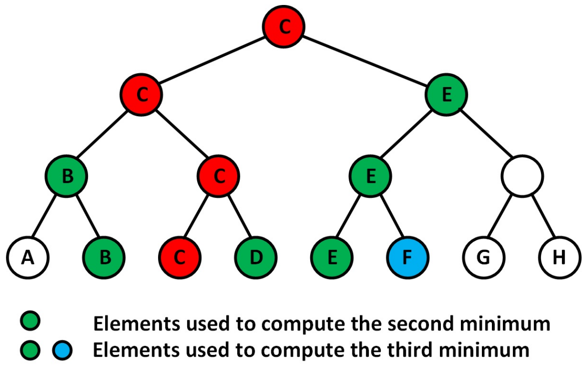

C
C
E
B
C
E
A
B
C
D
E
F
G
H
Elements used to compute the second minimum
Elements used to compute the third minimum

Figure 3.16: Tree structured minimum selection unit.

3.6.3 Memory Architecture

Adopting the technique in [50], the processor only stores the 3 best accumulated costs for every pixel after the forward scan and then later combines the entry with the backward scan results for the remaining 4 paths. Despite this simplified 2-scan approach, NG-SGM would still require $\approx$13 MB of on-chip memory for the accumulated cost storage to support FHD processing. To reduce the on-chip memory requirement, eliminate the need of using external DRAM and provide a single chip solution, the proposed design uses block-based processing to partition the input image into units of 88$\times$88-pixel overlapping blocks. The inter-pixel correlation in NG-SGM aggregation diminishes when pixel pairs are more than 88 pixels apart[83]. As shown in Fig. 3.14 top left, adjacent blocks are overlapped by 24 pixels to allow cost aggregation across block boundaries.

Image buffering in on-chip memory is rather straightforward in stereo depth computation[50] because the matching direction is 1D unidirectional with respect to the current processing pixel location. In contrast, optical flow processing involves 2D omnidirectional (around the current pixel) search, incurring significant image buffering overhead for block-based processing. The same image blocks are read multiple times to evaluate 2D displacement if a raster scan progression is used. Therefore, we propose a rotating image buffer scheme, shown in Fig. 3.17, that consists of 64 SRAMs to buffer 16 image blocks. Each 88$\times$88-pixel image block occupies 4 SRAM banks. As shown in Fig. 3.17, the block processing follows the row-alternating raster scan order where the 2 dark gray blocks are fetched on chip to replace black blocks while the chip processes the red block. This approach maximizes on-chip memory reuse and reduces interface bandwidth by 2$\times$ at the cost of 28% larger on-chip memory.

Unlike stereo SGM, computing census transform for the entire image is wasteful as NG-SGM evaluates only a small set of candidates. Thus, the proposed architecture selectively computes 9$\times$9 census transform only for sparse matching candidates.

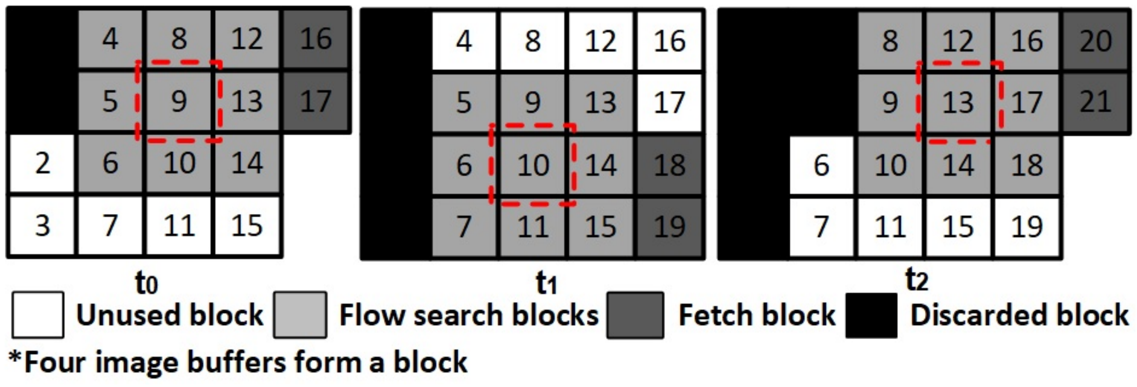


Figure 3.17: On-chip rotating buffer scheme.

Note that computing census for sparse points on-the-fly does not necessarily lead to lower energy consumption compared to full census computation for the entire image because the latter can benefit from the deterministic sliding window approach for the maximum memory reuse[50]. However, when the density of the selected pixels is low, as in NG-SGM, we have observed that on-the-fly census computation is desirable despite the highly irregular memory access pattern. As shown in Fig. 3.18, selectively computing census transform on-the-fly is associated with 34% energy reduction and 10× memory reduction compared with precomputing census transform.

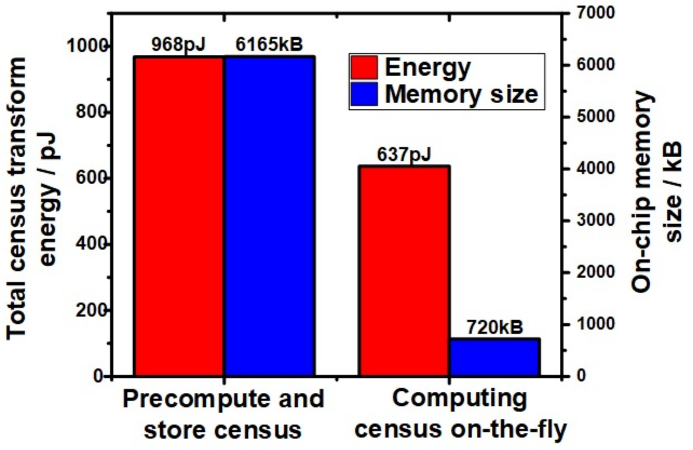


Figure 3.18: Memory size & power comparison between precomputing census vs. computing census on-the-fly.

Although on-the-fly census transform significantly reduces the memory bandwidth requirement, it still imposes a challenge because matching candidates are dynamically selected depending on neighboring pixels' guidance. Moreover, as each census transform requires 9×9 pixel block accesses, 1824 pixel accesses are required to evaluate all candidates for a single pixel. To reduce the number of memory word accesses and improve access energy efficiency, we group 4×4 pixels into a single memory word as shown in Fig. 3.19. In addition, we made a modification to NG-SGM so that the cluster of 4 candidates consisting of one best candidate (red in Fig. 3.19 and A in Fig. 3.3) and the surrounding adjacent candidates (blue in Fig. 3.19 and B in Fig. 3.3) is always confined within the 4×4 pixel memory word block. This can be guaranteed by adjusting the adjacent candidate locations based on the best candidate position. With this approach, the memory access for on-the-fly census computation is simplified, and exactly 9 memory word accesses are sufficient to fetch all the pixels necessary to process the census transform of the cluster of 4 NG candidates.The algorithm evaluation of NG-SGM includes the impact of this modification, which turns out to be negligible.

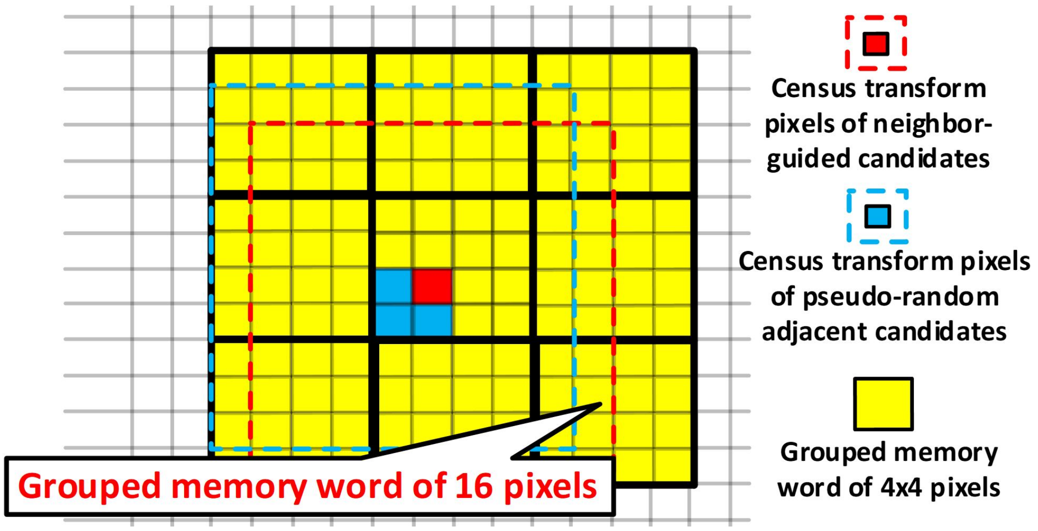

Census transform pixels of neighbor-guided candidates
Census transform pixels of pseudo-random adjacent candidates
Grouped memory word of 4x4 pixels
Grouped memory word of 16 pixels

Figure 3.19: Memory grouping for high-bandwidth access.

3.6.4 Pipelining Architecture

In the proposed highly parallelized cost aggregation, each aggregation unit (one per aggregation path) has its own row buffer, storing 3 minimum aggregated costs and their corresponding displacement vectors from the previous pixel. During each clock cycle, each aggregation unit aggregates costs over 64 sparsely scattered displacement vector locations and selects the 3 best displacement vectors (per aggregation path) to guide the NG-SGM search pruning of the next pixel. When each pixel is processed in the raster scan order, it incurs severe data dependency because neighbor guidance is only available when all aggregation and best candidate selection of previous pixels completes. Moreover, this severe inter-pixel dependency issue is aggravated by variable processing latency in resolving irregular (based on dynamic neighbor guidance) image buffer memory access collisions.

This challenge is mitigated by adopting a dependency-resolving and variable latency tolerant deeply pipelined architecture inspired by [50]. The forward and backward scan of NG-SGM performs aggregation along 4 paths, indicated by black arrows in Fig. 3.14 and 3.20. In the proposed pipeline architecture, the pixel processing proceeds in a skewed-diagonal fashion, as shown in Fig. 3.20, where the pixel processing step is in alphabetical order. When 'G' is fetched into the pipeline, the aggregated costs of previous pixels (light gray & dark gray) are already computed and stored in high bandwidth SRAMs. Consequently, the original single cycle data dependency from the left adjacent pixel in the raster scan processing extends to 7 cycles in the proposed skewed-diagonal scan. This significantly relaxes the critical path length of the pipeline and provides slack for the variable latency of candidate matching, cost aggregation, and the flow selection up to the maximum of 7 cycles to resolve inter-pixel dependency. The proposed pipelining achieves a 3 ns critical path, yielding a 4× performance gain compared with that of the conventional raster scan. Moreover, because of the improved throughput with deep pipelining, further energy reduction is achieved via more aggressive voltage and frequency scaling.

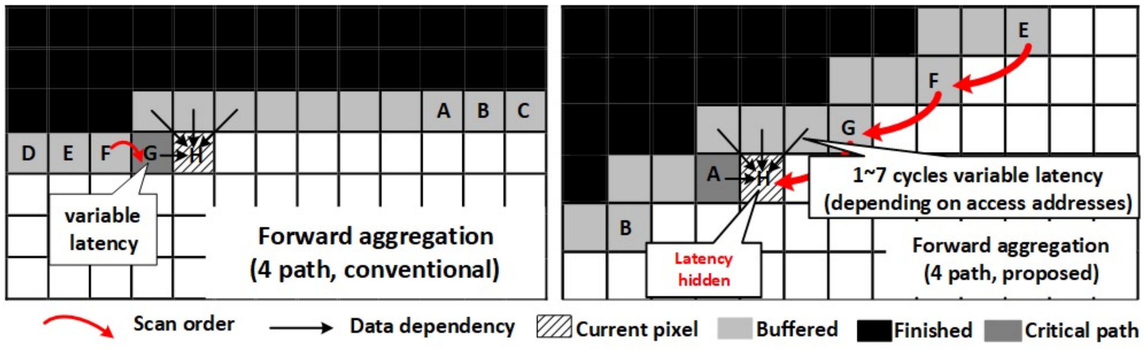

A
B
C
D
E
F
G
H
variable latency
Forward aggregation (4 path, conventional)
E
F
G
A
H
B
1~7 cycles variable latency (depending on access addresses)
Latency hidden
Forward aggregation (4 path, proposed)
Scan order
Data dependency
Current pixel
Buffered
Finished
Critical path

Figure 3.20: Variable-latency critical path hidden diagonal scan.

3.6.5 Multiple frequency and voltage processing

The proposed design is partitioned into 3 power and frequency domains as shown in Fig. 3.15 to balance throughput among different modules while improving energy efficiency. Optical flow processing with the NG-SGM algorithm at 25 fps FHD requires 2.6 Tb/s irregular memory access governed by NG search space pruning. These memory accesses cannot be easily parallelized due to the dynamic and data-dependent nature of the neighbor guidance. Meanwhile, once the local matching costs are evaluated, cost aggregation and candidate selection over 4 paths can be highly parallelized and processed at the same time. Based on this inherent throughput mismatch, we partition the design into three voltage-frequency domains shown in Fig. 3.15. The image buffer access, census transform and local cost generation modules are placed in the high voltage (up to 1.3 V) and high frequency (500 MHz) domain to provide 2.6 Tb/s memory access bandwidth and improve throughput. The highly parallelizable NG-SGM cost aggregation and candidate vector selection modules are placed in the low voltage (0.7 V) and low frequency (200 MHz) domain to balance the computation throughput with memory bandwidth and save energy. Parallel interfaces are placed in a separate voltage (0.7 V) and frequency (100 MHz) domain to balance the interface data bandwidth and processing throughput. The F_{xbar} and F_{core} clocks in Fig. 3.15 are generated with two separate on-chip VCOs while $F_{interface}$ is divided from F_{core}. Fig. 3.21 shows the power distribution between different blocks in the proposed architecture, where memory access (labeled 'xbar', 'mem access', 'census transform' in Fig. 3.21) consumes 74% while highly parallelized processing (other items in Fig. 21) consumes 26% of the overall power. The power of NG-SGM aggregation and selection is reduced by 27% compared with that of a single voltage design without any performance degradation.

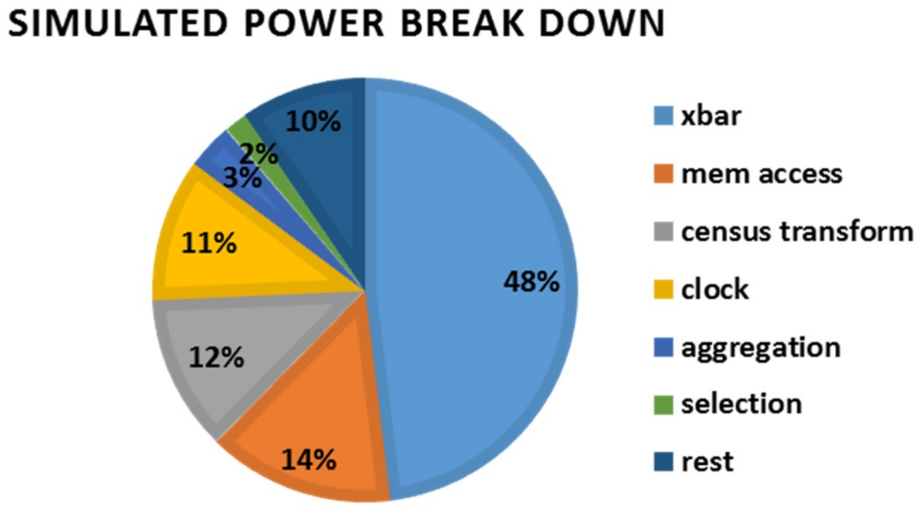


Figure 3.21: Power break down of different modules.

3.6.6 High-Bandwidth, Coalescing Crosspoint Crossbar

The NG-SGM algorithm unified for stereo depth and optical flow requires 2.6 Tb/s memory bandwidth for the image buffer to compute on-the-fly census transform of pruned candidates. The memory access patterns are irregular and unpredictable due to the dynamic nature of neighbor guidance. However, we observe that many of these memory accesses frequently overlap because neighboring pixels often agree on the same matching candidates and provide the same guidance. In simulation, only 30% of the memory accesses are unique on average. To exploit this property while maintaining >2 Tb/s on-chip memory access bandwidth, we propose a new crossbar circuit that efficiently coalesces redundant memory accesses.

Fig. 3.22 shows the architectural block diagram of the proposed high-bandwidth, 2-cycle pipelined coalescing crossbar. The crossbar has an SRAM-like layout with crosspoints (Xpoints) connected in a 16×16 mesh. Query queues are connected on the right side, and 16 memory banks are connected at the top and bottom of the crossbar. Each queue holds $\leqslant$ 11 entries to avoid queue overflow in the worst case (no coalescing). The memory system hierarchically connects 4 such custom-designed coalescing crossbars for accessing 64 on-chip image rotating buffers (Fig. 3.15). The data width of the crossbar is 128 bit, and the crossbar operates at 500 MHz.

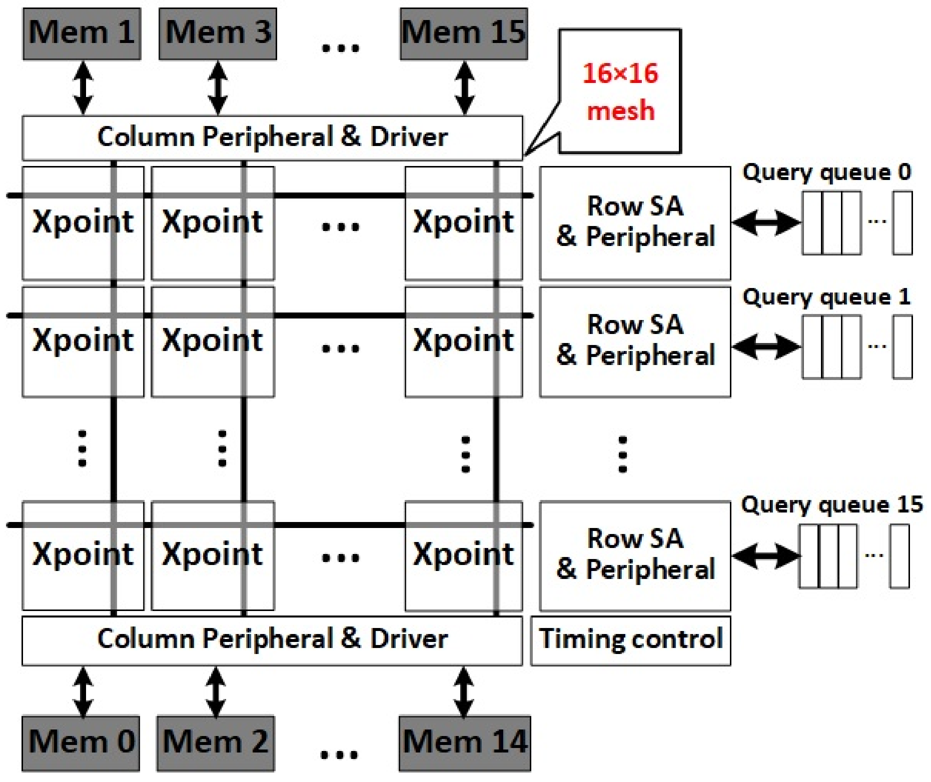


Figure 3.22: Block diagram of the coalescing crossbar.

In the first cycle, arbitration is performed when multiple queries try to access the same memory bank. In the proposed crossbar, each crosspoint locally handles arbitration and remembers collision. If a collision occurs, the query with the higher priority wins arbitration and sends its address to the memory. The query with the lower priority loses arbitration and is suppressed. In the second cycle, the data requested by the wining query is fetched, and the request from the wining query is fulfilled. At the same time, the crosspoint that blocked a losing query broadcasts its data and its access address to all losing queries to check if any coalescing memory access exists in its queue. If any query in the queue of the losing query has a coalescing memory access, it consumes the broadcasted data and eliminates itself from the queue, coalescing the two requests and removing the redundant memory access.

Complexity of the arbitration logic increases quadratically with the number of input or output ports, making conventional CMOS MUX/NOR-based design very inefficient for a large number of ports[84]. Inspired by [85] and to achieve an area/power efficient solution, we customized the arbitration circuits to perform arbitration and detect collision with dynamic logic on bitlines. Fig. 3.23 shows the detailed arbitration circuit at each crosspoint and the timing of its control logic. The timing control of the crossbar is verified through Monte Carlo simulations considering PVT variations to ensure sufficient margins for correct functionality. The proposed dynamic logic on bitlines does not suffer from charge sharing because the discharge transistor on each bitline are not stacked. There are 16 arbitration lines, one per input channel. All lines are pre-charged high during operation. Then each crosspoint discharges all arbitration bitlines whose indices are smaller than its query index and therefore have lower priority. Then the crosspoint examines the bitline with its own index to see if it won or lost the arbitration. Each crosspoint remembers the collision and stores it in the local register if the query loses. The winning crosspoint then sends the query address to its memory. All queries including losing queries receive the same broadcasted data from memory as well as the address of the winning query. If the queue has a matching address, the broadcast data is returned, and the query is removed from the queue, completing coalescing.

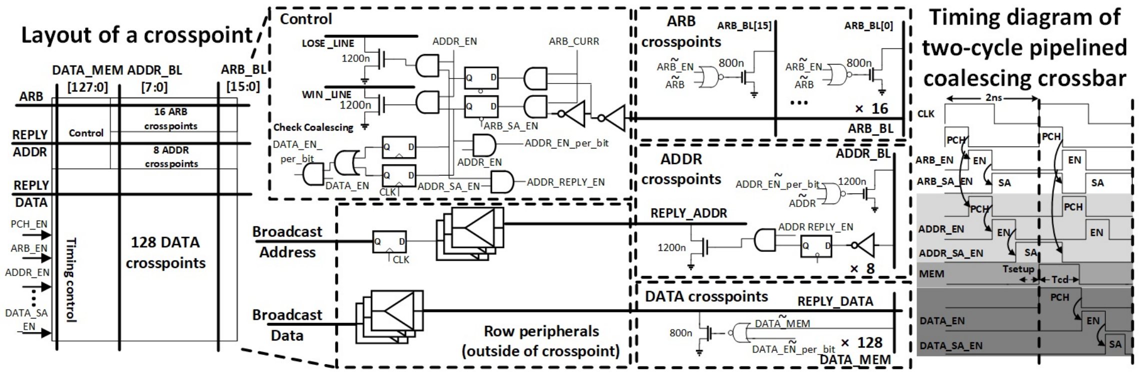


Figure 3.23: Circuits and timing diagram of a coalescing crosspoint.

With 64 SRAM banks, a single 162×64 crossbar (162 is the total number of memory accesses issued for on-the-fly census transform) would be ideal for maximizing performance. However, the area and energy consumption of a crossbar also increase quadratically when the number of input/output ports of a crossbar increases. Therefore, instead of using large crossbars, we instantiate four smaller 16×16 coalescing crossbars in the design hierarchically so that each handles the accesses of 16 memory banks. Fig. 3.24 and Fig. 3.25 show the tradeoff (based on simulation) between different crossbar design choices. Compared with a single 64×64 coalescing crossbar, the proposed design achieves 3.1× power reduction and 3.4× area reduction. Compared to an ideal case of using a single 162×64 crossbar with unlimited resources, the proposed design achieves a comparable throughput with 3.8× reduced area and 3.6× reduced power. Moreover, the proposed coalescing crossbar yields 54% higher performance compared to a regular crossbar without coalescing. Overall, this proposed approach enables 2.6 Tb/s average bandwidth from 4 instances of high-bandwidth, 2-cycle pipelined coalescing crossbars. Fig. 3.26 summarizes the relative power consumption (normalized to the final optimized power) of various components of the system. It also quantifies the power reduction from the proposed circuit and architecture optimization techniques.

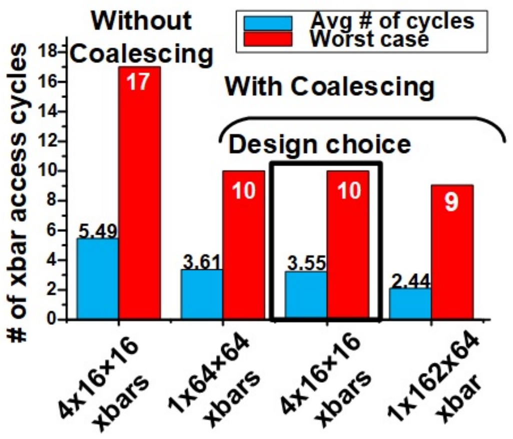


Figure 3.24: Comparison of different crossbar architectural options.

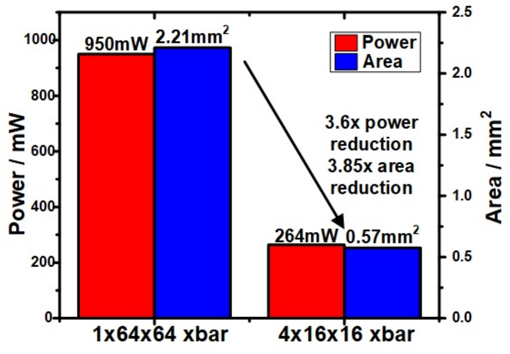


Figure 3.25: Comparison of different crossbar architectural options.

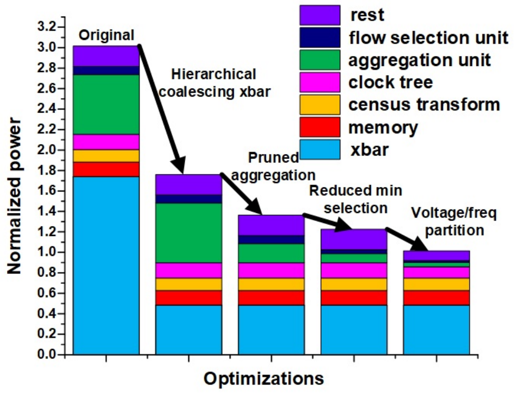


Figure 3.26: Relative power consumption of various components of the system, and the impact of various optimization techniques.

3.7 Chip Measurement Results

Fig. 3.27 shows a die photo, and Table 3.6 shows the performance summary of the fabricated chip. This work is fabricated in TSMC 28-nm HPC process with 9.3-

mm2 chip area. The host mini-computer (Odroid-XU4) performs block-partition of images and streams real-time input to the processor through USB3.0 interfaces via Cypress FX3 USB-to-parallel bridge chips. The parallel interface on the processor serves 2 USB bridge chips in parallel. The host sends control instructions to the chip via the USB interface shared with the input image streaming. The 6D vision output from the chip is streamed via a separate dedicated USB channel. The fabricated chip successfully produces optical flow and stereo depth with 176×176 and 1×176 pixel search ranges, respectively, in FHD (1920×1080) resolution. The real-time operation is confirmed at 25 fps (optical flow)/30 fps (stereo) FHD processing with 180 MHz core (F_{core}) and 500 MHz memory frequency (F_{xbar}) consuming 760 mW combined from 0.9 V and 1.3 V supplies. Fig. 3.28 shows the measured optical flow and depth results for the 194 KITTI automotive benchmark, achieving 11.37% outlier for optical flow, 7.52% outlier for depth. An outlier is defined as a pixel that has a displacement error of more than 3 integer levels (disparity) or 2 integer levels horizontally or vertically (optical flow). The chip is also measured with over 1000 JPL-captured MAV benchmarks, and the qualitative results are shown in Fig. 3.28.

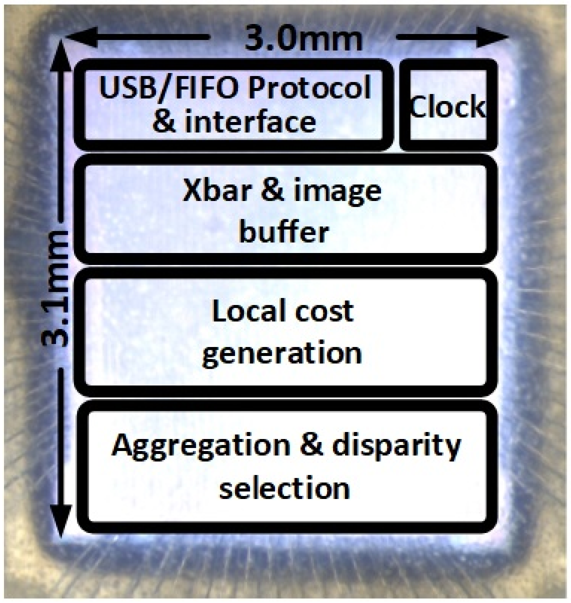

3.0mm
3.1mm
USB/FIFO Protocol & interface
Clock
Xbar & image buffer
Local cost generation
Aggregation & disparity selection

Figure 3.27: Die photo.

	This work
Method	NG-SGM
Technology	28nm
Chip area	9.3mm^2
On-chip memory	1568Kb
Frequency	F_{core}: 180MHz, F_{xbar}: 500MHz
Throughput & image size	1920 X 1080 @ 25fps (flow) 1920 X 1080 @ 30fps (depth)
Search Range	Depth & Optical flow
	176x176=30976
Accuracy (Outlier %)	4.7% @ Middlebury
Operating voltage	Vlow: 0.9V Vhigh: 1.3V @ 30 fps HD Vlow: 0.58V Vhigh: 0.75V @ 30 fps VGA
Power	760mW @ 25 fps HD (flow) / 30 fps HD (depth) 62 mW @ 30 fps VGA (flow)/ 40 fps VGA (depth)
Normalized energy* (depth)	0.069nJ @ 30 fps HD (depth) 0.029nJ @ 30 fps VGA (depth)
Normalized energy* (Optical flow)	0.0048nJ @ 25 fps HD (flow) 0.0022nJ @ 30 fps VGA (flow)

$$\text{Normalized energy [6]} = \frac{\text{nJ}}{\text{Search range} \bullet \text{\# of pixel}}$$

Table 3.6: Summary of performance.

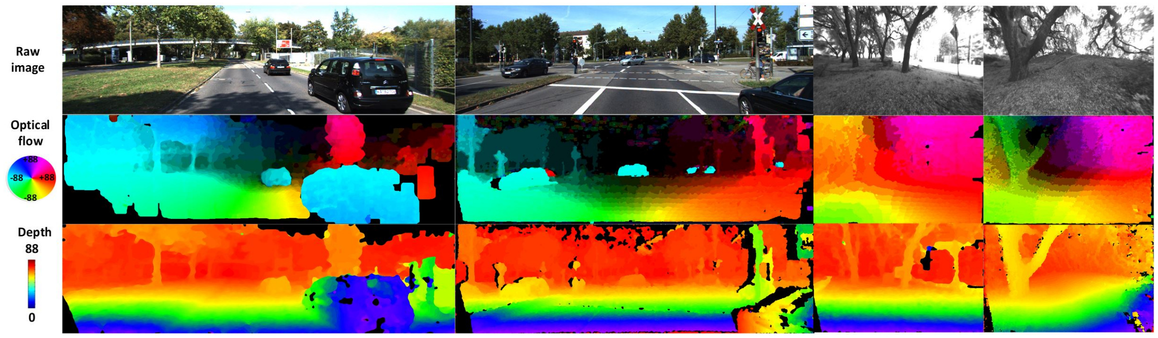


Figure 3.28: Measured output from KITTI dataset and from JPL MAV captured images.

Fig. 3.29 shows the measured throughput and energy efficiency trade-off with varying F_{xbar} and F_{core} of two different voltage-frequency domains. The optimal energy point occurs when F_{xbar} is ≈2.7× greater than F_{core}. Comparison with prior works is provided in Table 3.7. Only the proposed design can support optical flow computation. We achieve 1.5× higher energy efficiency for stereo processing compared to our prior chip [50], optimized for stereo processing only. Normalized energy is the FoM proposed in [38] and adopted in [38, 28, 50]. The FoM applied to the optical flow shows >100× improvement compared to the stereo processing of prior chips. Fig. 3.31 shows this FoM comparison, with 1.5× better energy efficient at FHD for stereo processing and >100× better efficiency for optical flow compared with prior works[38, 28, 50], which are stereo vision-only ASICs. The chip is programmable and supports various frame rates and image resolutions. Fig. 3.30 shows the voltage and frequency scaling over different frame resolutions and frame rates. It consumes 760 mW to process optical flow at 25 fps/stereo at 30 fps FHD, while the power consumption scales to 62 mW at 30 fps VGA operating at lower voltages (0.72 V, 0.65 V).

	ISSCC 2015[22]	ISSCC 2016[6]	ISSCC 2017[1]	This work
Method	5-View BP	Truncated SGM	SGM	NG-SGM
Technology	40nm	65nm	40nm	28nm
Chip area	22mm^2	16mm^2	10.8mm^2	9.3mm^2
On-chip memory	352Kb	3946.9Kb	1064Kb	1568Kb
Frequency	215MHz	250MHz	170MHz	Fcore: 180MHz, Fxbar: 500MHz
Throughput & image size	1920 X 1080 30fps	1280 X 720 30fps	1920 X 1080 30fps	**1920 X 1080 @ 25fps (flow) 1920 X 1080 @ 30fps (depth)**
Search Range	**Stereo depth only**			**Depth & Optical flow**
	1x64	**1x64**	**1x128**	**176x176=30976**
Accuracy (Outlier %)	Not reported due to limited depth range		7% @ KITTI	**4.7% @ Middlebury, 11.7% @ KITTI**
Operating voltage	0.9 V	1.2V	0.75V @ 30 fps HD 0.52V @ 30 fps VGA	**Vlow: 0.9V Vhigh: 1.3V @ 30 fps HD Vlow: 0.58V Vhigh: 0.75V @ 30 fps VGA**
Power	1019mW (includes DRAM power)	582mW (excludes DRAM power)	836mW @ 30 fps HD 55 mW @ 30 fps VGA	**760mW @ 25 fps HD (flow) / 30 fps HD (depth) 62 mW @ 30 fps VGA (flow)/ 40 fps VGA (depth)**
Normalized energy* (depth)	0.153nJ	0.329nJ	0.104nJ @ 30 fps HD 0.047nJ @ 30 fps VGA	**0.069nJ @ 30 fps HD (depth) 0.029nJ @ 30 fps VGA (depth)**
Normalized energy* (Optical flow)	**Optical flow NOT supported**			**0.0048nJ @ 25 fps HD (flow) 0.0022nJ @ 30 fps VGA (flow)**

Table 3.7: Comparison with prior art.

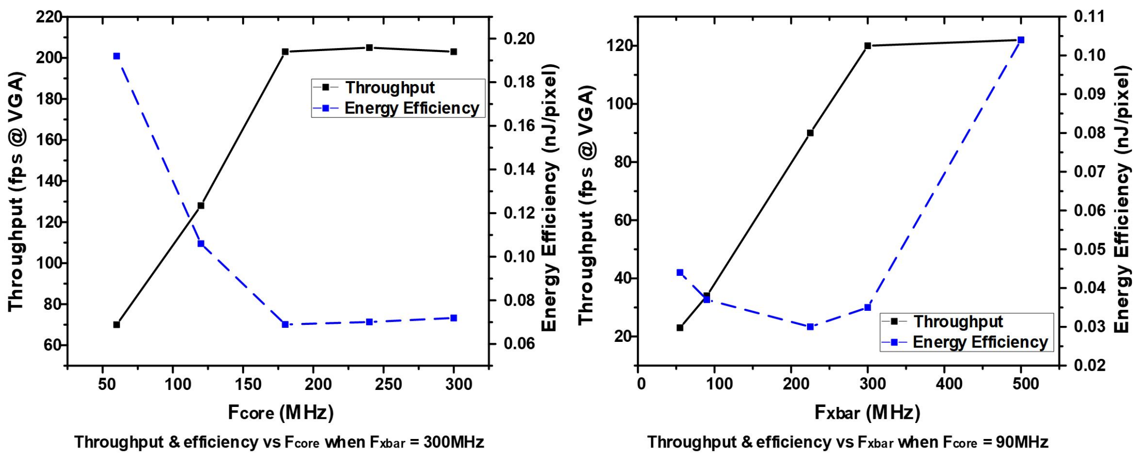


Figure 3.29: Measured throughput and energy efficiency with different frequencies

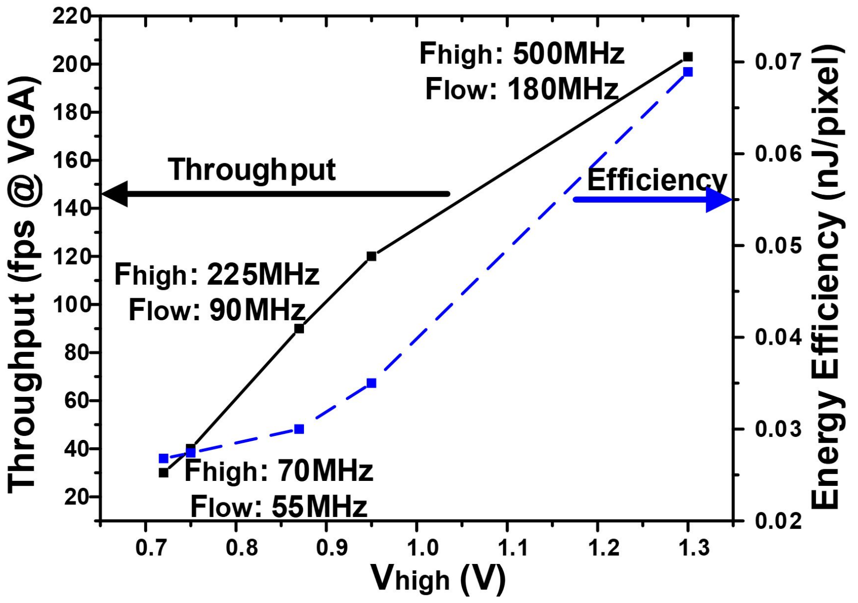


Figure 3.30: Measured voltage & frequency scaling of proposed design.

3.8 Summary

This work presents a single-chip, accurate, high performance, energy efficient unified optical flow and stereo depth 6D vision processor using the NG-SGM algorithm for low power computer vision applications. The fabricated 6D vision processor generates

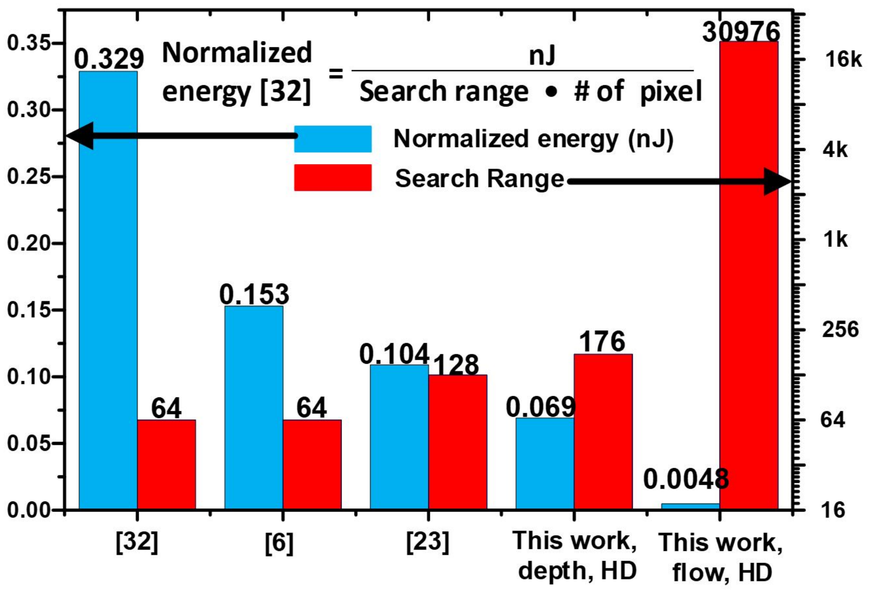


Figure 3.31: Comparison of measured FoM with prior art.

dense optical flow and depth with a wide search range of 176 pixels per dimension in FHD resolution with real-time 25 fps throughput for optical flow and 30 fps throughput for stereo depth, consuming only 760 mW in 28-nm TSMC HPC CMOS. The chip reports 11.7% outlier accuracy on industry standard KITTI automotive optical flow evaluation. The proposed rotating on-chip image buffer scheme reduces the interface bandwidth by 2× compared with raster scan progression and enables real-time streaming operation at the cost of 28% larger on-chip memory size. The proposed dependency-resolving image-scanning stride with deeply pipelined implementation yields 4× performance gain. The customized coalescing crosspoint crossbar yields 2.6 Tb/s on-chip bandwidth, efficiently mitigating irregular memory access patterns from the dynamic neighbor guidance of NG-SGM. A complete optical flow and stereo processing is built and demonstrated for realistic scenes in real-time operations.

CHAPTER IV

A 879GOPS, 243mW 80fps VGA Fully Visual CNN-SLAM Processor for Wide Range Autonomous Exploration

4.1 Introduction

As is described in Chapter. II and Chapter. III, stereo vision system provides dense 3D perception of surroundings and optical flow perceives 3D motion field of the environment. These two functions behave as kernel functions in many autonomous navigation systems. However, other than geometric environment perception, autonomous navigation systems finally needs to estimate the ego-motion based on the visual perception. In Chapter IV, we will further introduce a CNN-SLAM processor that enables energy-efficient ego-motion estimation in real-time for autonomous navigation systems and VR/AR(virtual reality/augment reality) system.

Simultaneous localization and mapping (SLAM) continuously estimates an agent's trajectory for all six degrees of freedom (6 DoF) and constructs a 3D map of unknown surroundings. It is a fundamental kernel that provides delightful user interaction with head-mounted augmented/virtual reality devices and autonomous navigation of micro aerial vehicles and self-driving cars (Fig. 4.1). A noticeable recent trend in visual SLAM is to apply computation- and memory-intensive convolutional neural networks

(CNNs) that outperform traditional hand-designed feature-based methods to achieve accurate tracking[86]. For each video frame, CNN-extracted features are matched with stored keypoints to estimate the agent's 6-DoF posture by solving a perspective-n-points (PnP) non-linear optimization problem. The agent's long-term trajectory over multiple frames is refined by a bundle adjustment process, which involves a large-scale (120 variables) non-linear optimization. Visual SLAM requires massive computation (>250GOP/s) in the CNN-based feature extraction, matching as well as data-dependent dynamic memory access and control flow with high-precision operations, which throws out significant low-power design challenges. Software implementations are impractical, resulting in 0.2s runtime with a 3GHz CPU+ GPU system with >100MB memory footprint and >100W power consumption. Prior ASICs have implemented either an incomplete SLAM system [87] that lacks estimation of ego-motion or employed a simplified (non-CNN) feature extraction and tracking [87, 88, 89] that limits SLAM quality and range. A recent ASIC [89] augments visual SLAM with a high-precision inertial measurement unit (IMU), simplifying the computational complexity but adding significant additional power consumption.

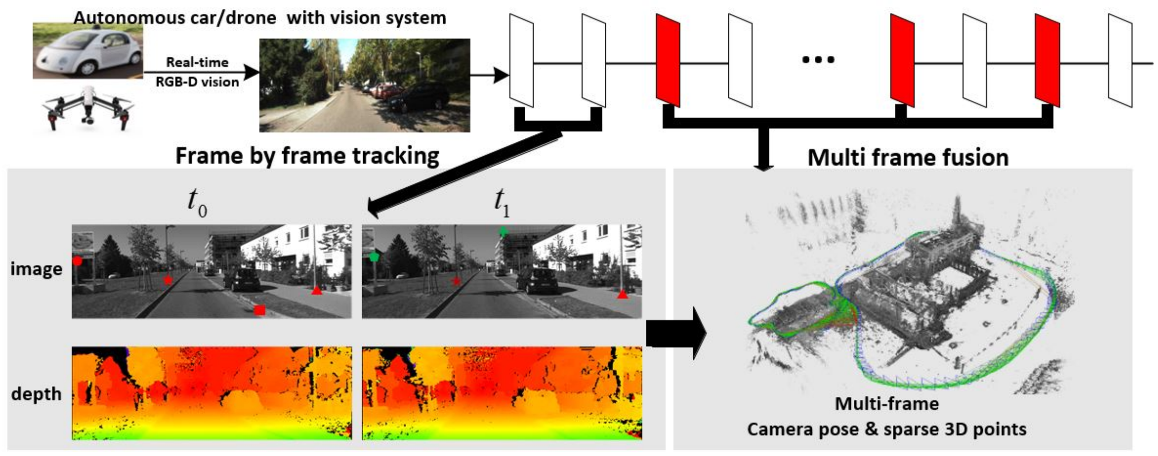


Figure 4.1: Application of real-time SLAM processing.

This chapter presents an accurate, low-power, real-time CNN-SLAM processor that, for the first time, implements full-visual SLAM on a single chip. The proposed major design features are:

1) feature extraction and description with a highly parallelized and programmable CNN engine with 32% better accuracy in feature matching than SIFT;

2) aggressively pruned feature matching with temporal posture prediction, triangulation, and address hashing to eliminate 97% of unnecessary matchings;

3) numerically stable fixed-point implementation with function re-organization and pivoting in the linear solver;

4) hierarchical memory organization, completely eliminating external DRAM accesses for BA optimization.

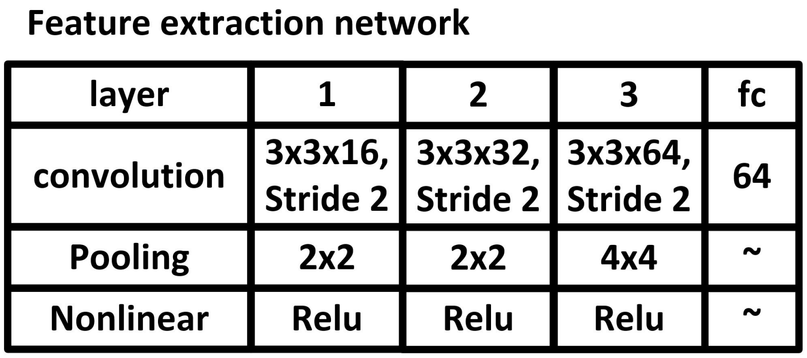
By applying these optimizations, we propose the Visual SLAM ASIC to report performance tested on the industrial standard KITTI benchmark that renders large-scale realistic automobile trajectory over 1km. The proposed design supports localization and mapping of >1000 keypoints/frame on VGA (640×480) resolution in real-time at 80fps, consuming 243.6mW from a 0.9V supply in 28nm CMOS.

4.2 Datapath of CNN-SLAM Algorithm

In this section we describe overall procedure of the proposed SLAM system, consisting of keypoint description, CNN-based feature description, frame-by-frame PnP and multi-frame BA. PnP processing computes the 6-DoF pose (represented by a combination of 3D translation T and 3D rotation R) for each frame by analyzing the projected coordinate differences of matched keypoints in two subsequent frames. CNN-extracted keypoints are matched with others in the previous frame to establish correspondence of the same points in the environment. The 6-DoF pose is obtained by solving a non-linear optimization to minimize the 2D reprojection error of these matched points. After acquiring the pose of the current frame, all newly added keypoints are projected onto 3D real-world coordinates for PnP processing of the next frame. A frame that contains >50% new keypoints is registered as a keyframe. To refine the long-term trajectory of the agent (camera), BA is performed over the last 20 keyframes whenever a new keyframe is registered.

4.2.1 Feature Extraction and Matching

We extract DoF corners with 6 levels Gaussian pyramidal images. For image resolutions from 640×480 to 1226×370 pixels, we typically found 1000 ~1500 keypoints per frame. For higher resolution or a frame with more than 1536 keypoints detected, we divide each scale level into a 16×16 grid, extracting at most 4 corners per cell to ensure an homogeneous distribution of keypoints across the entire frame. All retained keypoints and their surrounding 16×16 image patch are passed to a CNN to compute the feature descriptor (the network is shown in Fig. 4.2). The CNN used for feature extraction is pre-trained on Middlebury Multi-View stereo dataset[90] and is tested on KITTI[49] automotive dataset, as is shown in Fig. 4.3. Proposed CNN-based feature description achieves 32% better feature matching accuracy compared with widely used descriptors such as SIFT[91].


Feature extraction network

layer	1	2	3	fc
convolution	3x3x16, Stride 2	3x3x32, Stride 2	3x3x64, Stride 2	64
Pooling	2x2	2x2	4x4	~
Nonlinear	Relu	Relu	Relu	~

Figure 4.2: Feature description network.

4.2.2 Perspective-n-Points

Once feature vectors of each keypoints are extracted, we predict the camera pose of the current frame with a constant acceleration model to prune the feature matching

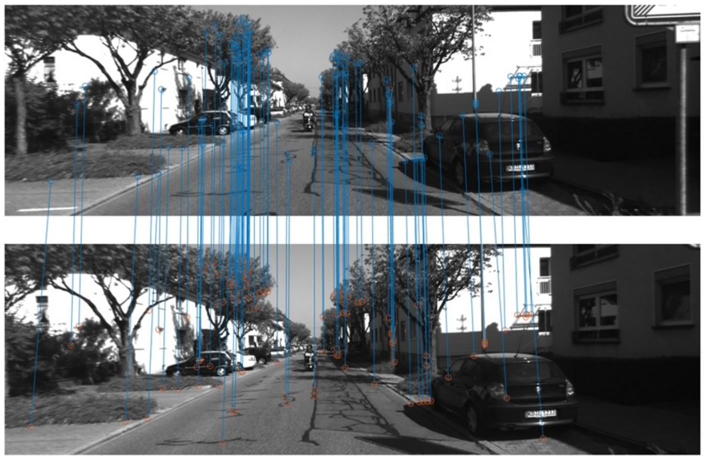


Figure 4.3: Example of CNN feature matching with proposed pre-trained feature description network.

space and guide the feature match. Euclidean distance is used to match the feature vectors. Matching candidates within the prune search space that has the lowest euclidean distance is identified as the best match. Best matches are also filtered with a programmable threshold to ensure robust matching results. Typically, we find 400 matching keypoints on KITTI dataset[49]. We then perform PnP optimization to find a camera pose for the current frame. The PnP algorithm estimates the camera pose between two consecutive frames using motion-only bundle adjustment. As shown in Fig. 4.4, given each 3D world coordinates and 2D image coordinates of the keypoints, motion-only bundle adjustment minimizes the reporjection error while finding out the current camera pose with equation 4.1,

$$\min_{R,t} \sum_{j=1}^{M} \rho(||(RX_j + t) - x_j||) \tag{4.1}$$

where R is the 3×3 rotation matrix, t is the translator vector of the current frame and X_j/x_j are world/image coordinates of the keypoint respectively. ρ is a non-linear function to ensure the robustness and we use Humble loss[92] in our system. The PnP problem is solved with Levenbury-Marquardt[93] method in our case.

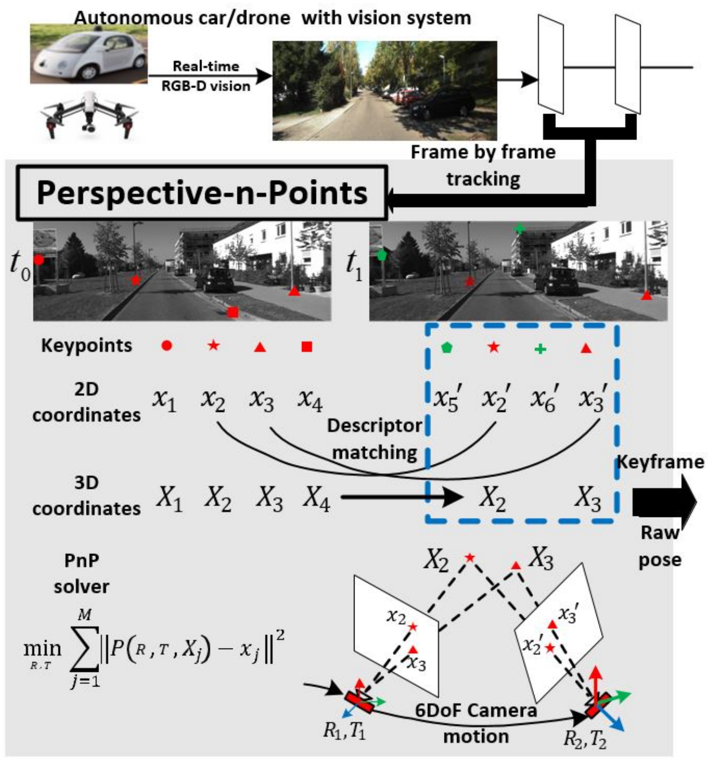


Figure 4.4: Pose estimation of each frame.

4.2.3 Local Bundle Adjustment

Once we have an initial estimation of the camera pose of the current frame from PnP and an initial set of keypoint matches between current and previous frames, we can look up the matches into the older frames and acquire more map point correspondences. Local bundle adjustment at time t optimizes the set of 3D points and camera poses which have at least one detected projection in frames t_n ... t_1 and t.

$$\min_{R_i,t_i} \sum_{i=1}^{N} \sum_{j=1}^{M} ||(R_i X_j + t) - x_{ij}|| \tag{4.2}$$

Where R_i and t_i corresponds to a 3×3 rotation matrix and a 3×1 translator vector of each keyframe, and X_{ij}/x_{ij} are world/image coordinates of the keypoint respectively. The local bundle adjustment can be also solved with Levenbury-Marquardt[93] method in our case. To bound the complexity of our design, the number of appearance of a keypoint is set to be less than 8 keyframes and a keyframe is registered if >50% new keypoints are detected from previous keyframe (shown in Fig. 4.5).

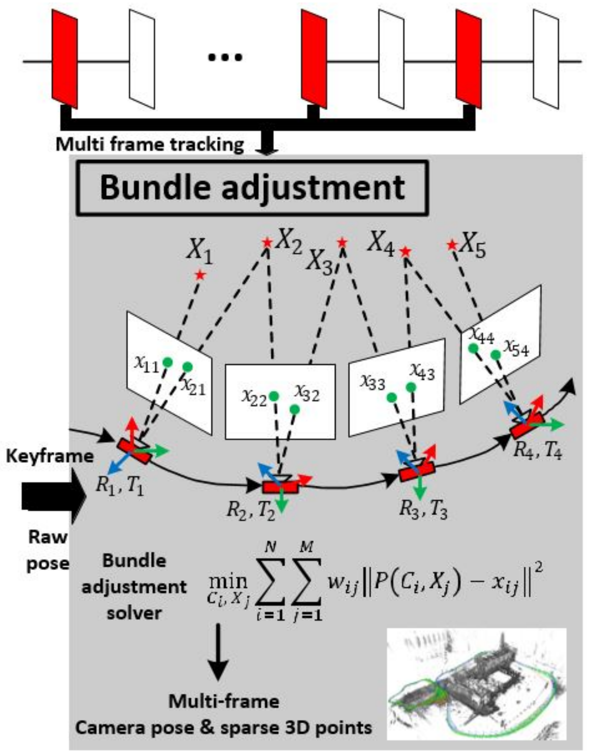

Multi frame tracking
Bundle adjustment
X_1 X_2 X_3 X_4 X_5
x_{11} x_{21} x_{22} x_{32} x_{33} x_{43} x_{44} x_{54}
Keyframe
R_1, T_1 R_2, T_2 R_3, T_3 R_4, T_4
Raw pose
Bundle adjustment solver
$\min_{C_i, X_j} \sum_{i=1}^{N} \sum_{j=1}^{M} w_{ij} \| P(C_i, X_j) - x_{ij} \|^2$
Multi-frame Camera pose & sparse 3D points

Figure 4.5: Pose optimization over multiple keyframes.

4.3 Architecture of Proposed CNN-SLAM Processor

Fig. 4.6 represents the overall architecture of the proposed SLAM processor, including a programmable CNN engine, a PnP engine and a BA engine. The CNN engine has a mesh of 512 MAC units in 8 clusters (each with a grid of 8 × 8 MAC units), a 192kB memory for CNN weights/activations, and two interleaved 32kB image buffers for stream processing (see also Fig. 4.6). The CNN uses 8-bit weights and does not require any off-chip weight loading. If the current frame is identified as a keyframe, the list of 3D real-world and 2D projected coordinates of keypoints will be sent to the graph memory in the BA engine (Fig. 4.6 bottom right), where iterative Levenberg–Marquardt (LM) optimization is applied over the last 20 keyframes.

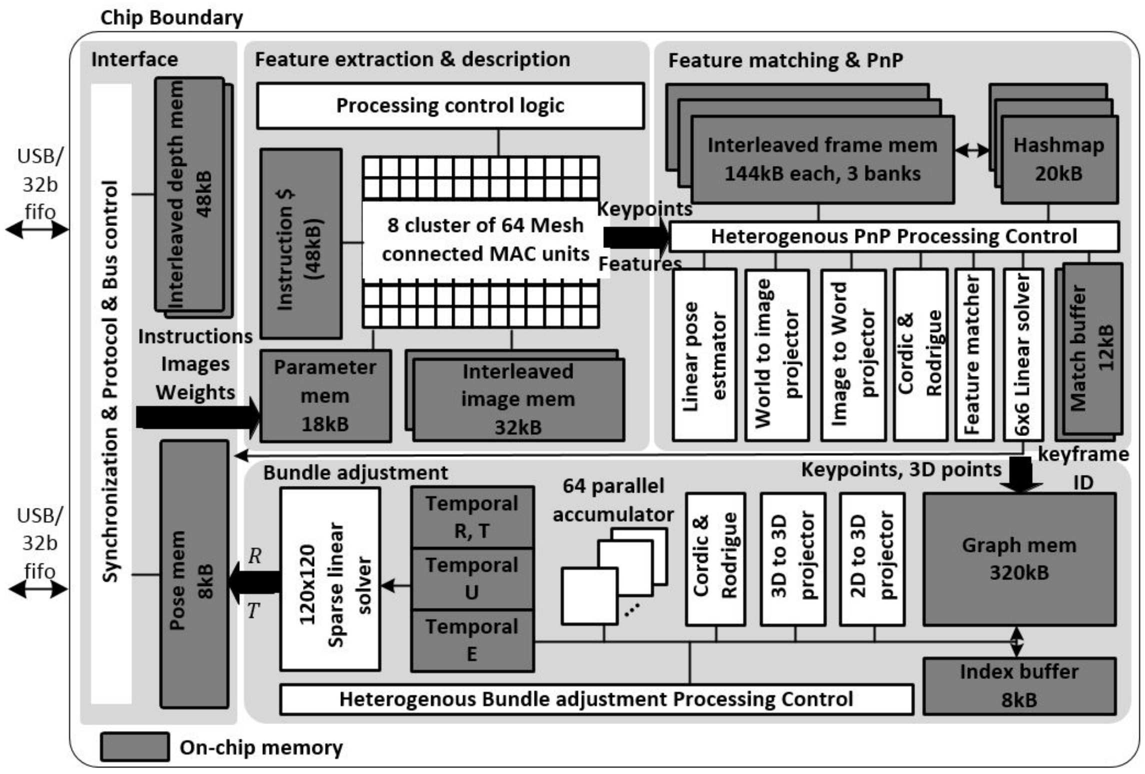


Figure 4.6: Chip architecture of proposed CNN-SLAM processor.

4.3.1 Programmable CNN Engine

Fig. 4.7 details the architecture of the programmable CNN engine. The CNN engine has a mesh of 512 8-bit multiply/24-bit accumulate MAC units in 8 clusters (each with a grid of 8 × 8 MAC units). 512 mesh-connected MAC units allow parallel processing of intensive CNN operations. Moreover, processing 8 input channels, 8 output channels and 8 pixels in parallel maximizes the data re-usability in the MAC array. Weights are first transferred to small 16kB weight buffers for frequent accesses. 4 bank partitioning of weight buffer provides 512bit/cycle memory access bandwidth with high access energy efficiency. 128kB local buffer stores input and output activation. High data bandwidth from weight buffer and local buffer ensures the full utilization of the 512 MAC units. 48kB instruction memory stores the processing sequence of the keypoint detection algorithm and feature extraction network.

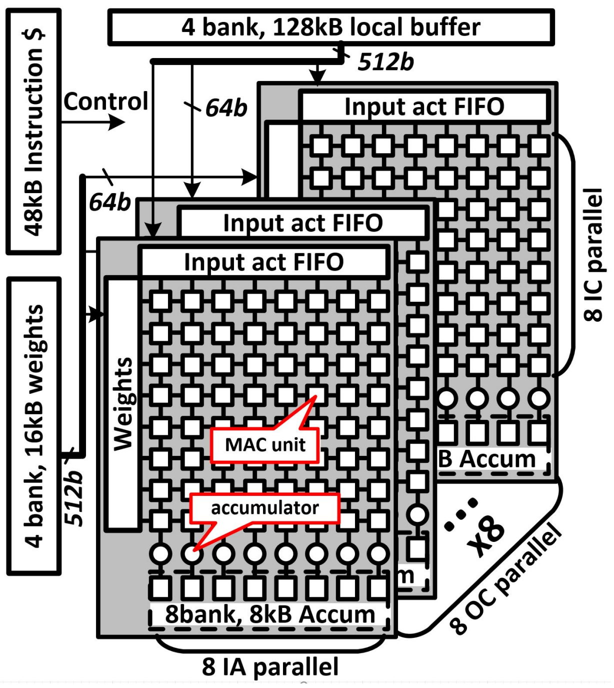


Figure 4.7: Architecture of programmable CNN unit.

Fig. 4.8 details the operation of the programmable CNN engine. First, 1D convolution are performed with shifting input activations (IAs) using a row of 8 MAC units with k (kernel size) cycles. This operation is repeated on a second and third row of IAs to complete the 2D convolution. Pooling and extrema detection are performed in the same fashion as 1D convolution in the MAC array. Partial 2D convolution outputs are accumulated locally in each MAC unit. Using 8 clusters of 8 $\times$ 8 MAC arrays, 8 consecutive input channels are convolved in parallel to accumulate partial output activations with 8 accumulators, and 8 consecutive output channels are processed in parallel to maximize reuse of the same IAs.

Moreoever, as is shown in Fig. 4.9, the proposed architecture computes convolution, pooling, and ReLU together rather than treating them as separate layers with buffering inputs-outputs in between. This reduces the instruction count, maximizes local re-usability and improves energy efficiency.

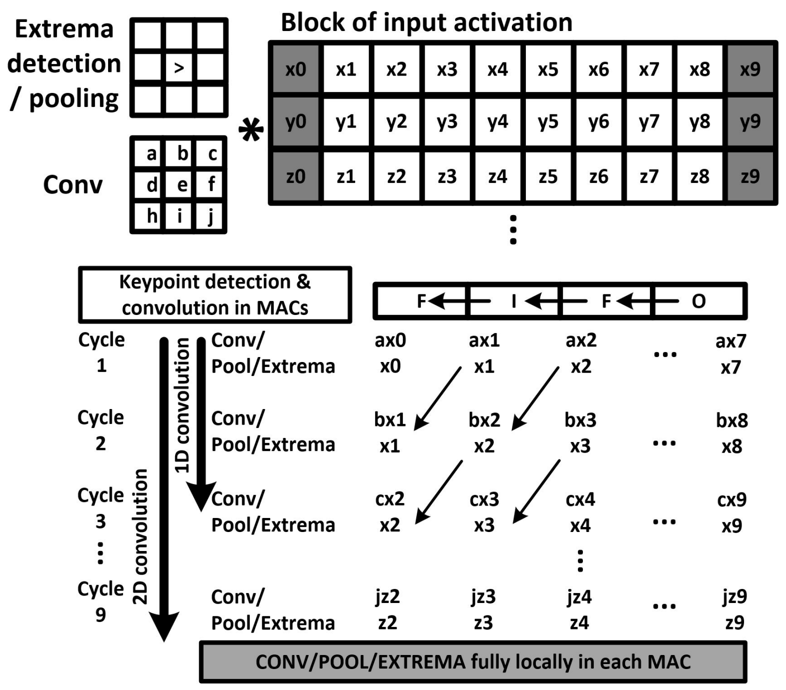

Extrema detection / pooling
>
Conv
a b c
d e f
h i j
*
Block of input activation
x0 x1 x2 x3 x4 x5 x6 x7 x8 x9
y0 y1 y2 y3 y4 y5 y6 y7 y8 y9
z0 z1 z2 z3 z4 z5 z6 z7 z8 z9
Keypoint detection & convolution in MACs
F I F O
Cycle 1
Cycle 2
Cycle 3
Cycle 9
1D convolution
2D convolution
Conv/ Pool/Extrema
ax0 x0
ax1 x1
ax2 x2
ax7 x7
bx1 x1
bx2 x2
bx3 x3
bx8 x8
cx2 x2
cx3 x3
cx4 x4
cx9 x9
jz2 z2
jz3 z3
jz4 z4
jz9 z9
CONV/POOL/EXTREMA fully locally in each MAC

Figure 4.8: Processing flow of a CNN layer.

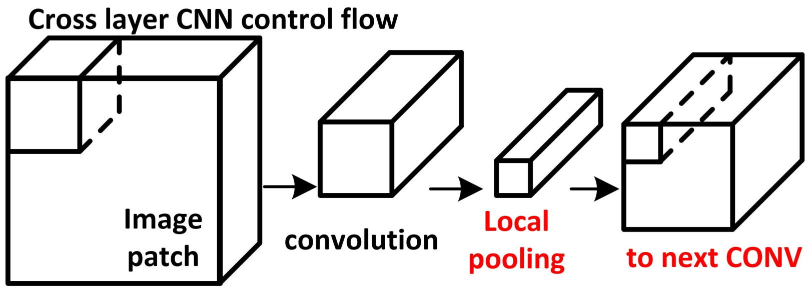


Figure 4.9: Cross layer processing flow of multiple CNN layers.

4.3.2 Feature Matching and PnP Engine

Keypoints and features extracted from the CNN engine are transferred to PnP Engine for frame-based pose estimation. As each frame contains >1000 keypoints, enumerating all possible matchings ($\approx 1000\times1000$) is extremely costly. Aiming at this challenge, we propose a prediction-based pruned keypoint matching scheme depicted in Fig. 4.10. We assume the velocity changes between consecutive frames are constant and use a locally linearised camera movement model to predict the new pose of the current frame from the poses of the previous two frames (Fig. 4.10). We then only re-project the keypoints of current frame onto previous frame. Extracted keypoints from previous frame that are adjacent to the projected keypoints from the current frame are treated as matching candidates. Feature matching is only performed on these selected matching candidates. Based on this pose prediction, the search range for each keypoint is reduced to $\approx \pm$ 24 pixels/dimension on the 2D projected image. This eliminates 97% of unnecessary matchings with negligible accuracy degradation (<0.1%).

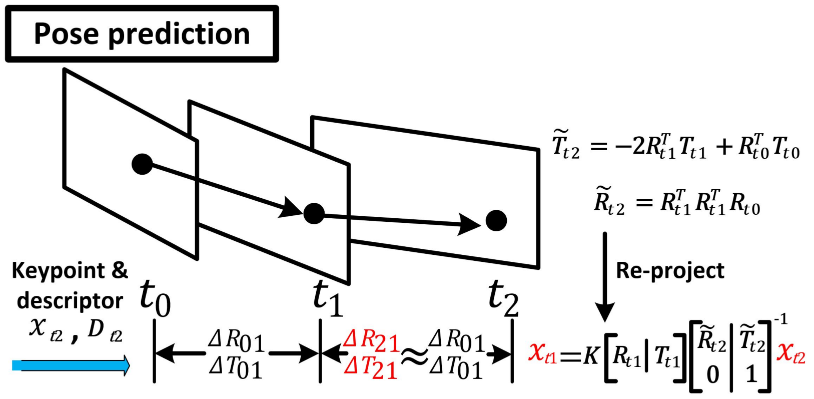

Pose prediction
$\widetilde{T}_{t2} = -2R_{t1}^{T}T_{t1} + R_{t0}^{T}T_{t0}$
$\widetilde{R}_{t2} = R_{t1}^{T}R_{t1}^{T}R_{t0}$
Re-project
Keypoint & descriptor
x_{t2}, D_{t2}
t_0
t_1
t_2
ΔR_{01} ΔT_{01}
ΔR_{21} ΔT_{21} $\approx$ ΔR_{01} ΔT_{01}
$x_{t1} = K[R_{t1}|T_{t1}]\begin{bmatrix}\widetilde{R}_{t2} & \widetilde{T}_{t2}\\ 0 & 1\end{bmatrix}^{-1} x_{t2}$

Figure 4.10: Search space pruning with pose prediction.

Prediction-based matching efficiency is further improved by employing a hierarchical memory system (492kB) using keypoint position-based hashing to store/load feature descriptors and 3D coordinates. As shown in Fig. 4.11, the image is partitioned into grid of 16×16 pixels and each 16×16 block may contain 4 features at maximum. We then use the predicted keypoints location to directly select the matching candidates from previous frame. The hash input is the keypoint location, and its output points to the keypoint entry in the memory where stores a list of feature descriptors and 3D coordinates in adjacent locations.

After the matching candidates are selected and reading from the frame memory, we perform L2 based feature matching. Each feature has 64 elements, and the feature matching cost is calculated by 8 parallel processing units, taking up to 8 cycles. Additionally, we deploy the early termination so that we stop and move to next

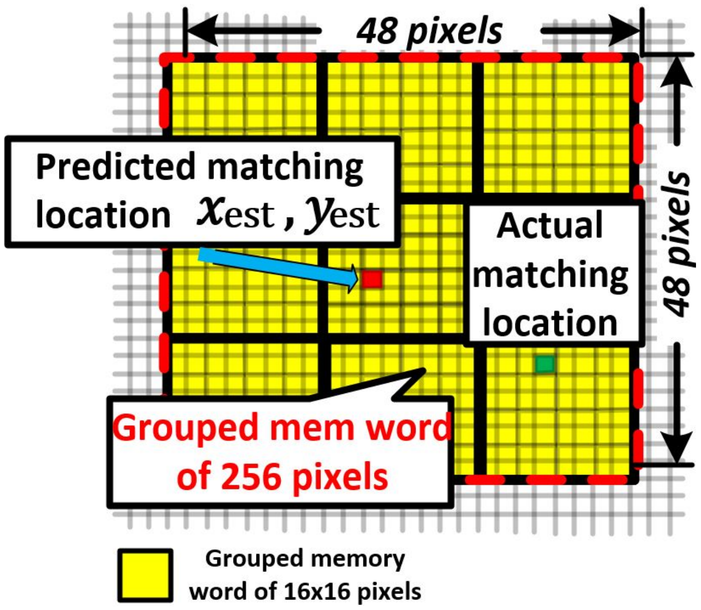

48 pixels
48 pixels
Predicted matching location x_{est}, y_{est}
Actual matching location
Grouped mem word of 256 pixels
Grouped memory word of 16x16 pixels

Figure 4.11: Grid based search of keypoints.

candidate if the L2 matching distance is larger than a programmable threshold. Early termination cuts down to ≈4.8 cycles when feature elements mismatch in early stages (Fig. 4.12).

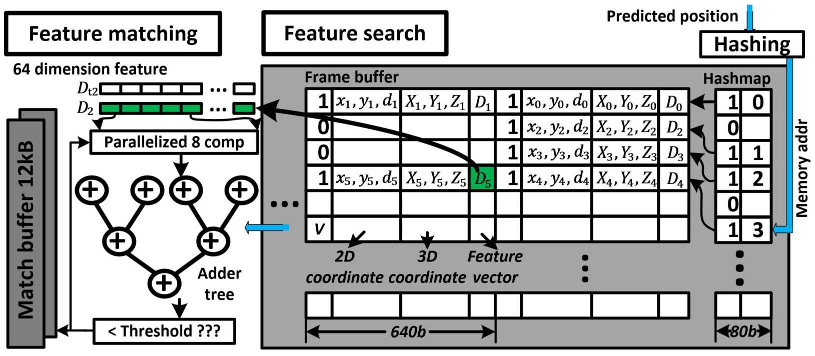


Figure 4.12: Hash based feature access and early termination of feature matching.

4.3.3 Local BA Engine

BA processing involves 20 keyframes and 4096 keypoints stored in the hierarchical graph memory (Fig. 4.13). Because each keypoint appears on multiple keyframes, each keypoint entry is structured to contain a single 3D coordinate and multiple 2D projected coordinates associated with different keyframe IDs as shown in Fig. 4.13.Although local bundle adjustment is performed over 20 keyframes, only maximum of 8 consecutive appearances of a keypoint are stored. This yields 52% additional memory reduction. A separate FIFO serves to eliminate and insert keypoints into the graph memory, while matched keypoints are merged into a single entry.

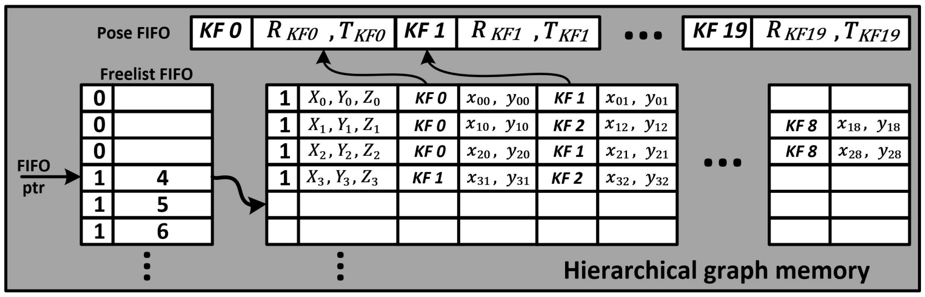


Figure 4.13: Graph memory for local bundle adjustment

In each BA iteration, the keypoints' 3D coordinates are updated according to the frame pose, and very small increments are applied to numerically compute the Jacobian matrix. Because of this increment, computing Jacobians requires an extremely large dynamic range and high precision, thus, prior works [88, 89] used double precision floating point operations. We reformulate the reprojection step so that common offset is subtracted before normalizing the projected 2D points into homogeneous coordinates. This allows a 32-bit fixed point implementation with $\approx$ 40% energy reduction while maintaining numerical stability in computing the Jacobian. After linearization, we construct a sparse Hessian matrix that only has non-zero 6×6 diagonal submatrix that relate to the 6-DoF pose of each keyframe as shown in Fig. 4.14. Thanks to the sparse and deterministic matrix structure, we block partition the Hessian and solve each 6×6 submatrix sequentially instead of solving the full 120×120 linear system. We apply pivoting to Gaussian elimination (GE) to maintain numerical stability in the linear system solver as shown in Fig. 4.15. With pivoting, non-zero elements in other rows are eliminated with the maximum element of the selected row. Row shuffling is performed before back substitution (BS), and GE/BS share 6 parallel computing units.

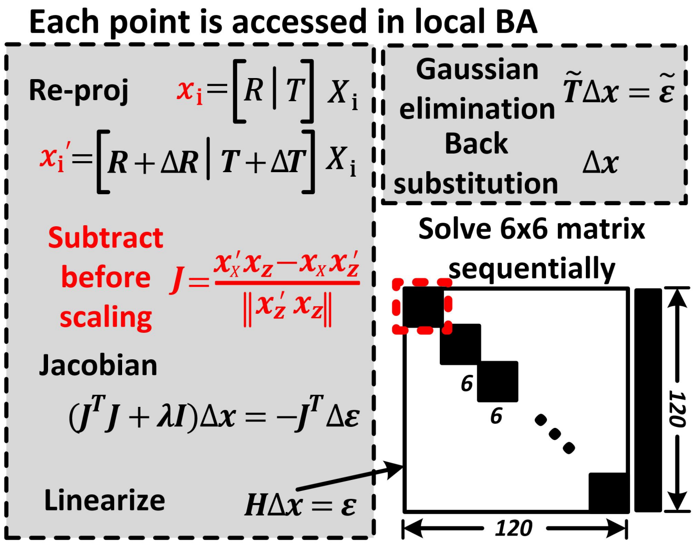

Each point is accessed in local BA
Re-proj
$x_i = [R|T] X_i$
$x_i' = [R+\Delta R | T+\Delta T] X_i$
Subtract before scaling
$J = \frac{x_x' x_z - x_x x_z'}{\|x_z' x_z\|}$
Jacobian
$(J^T J + \lambda I)\Delta x = -J^T \Delta\varepsilon$
Linearize
$H\Delta x = \varepsilon$
Gaussian elimination
$\tilde{T}\Delta x = \tilde{\varepsilon}$
Back substitution
Δx
Solve 6x6 matrix sequentially
6
6
120
120

Figure 4.14: Reformulation and fixed point implementation of sparse matrix solver.

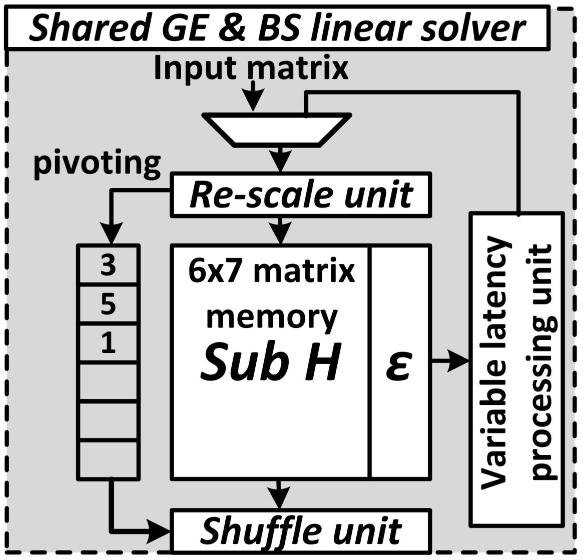


Figure 4.15: Numerically stable matrix solver with gaussian eliminating and pivoting.

4.4 Measurement Results

The proposed SLAM processor is fabricated on 28nm HPC CMOS. Die photo is shown in Fig. 4.16 and performance summary is shown in Table. 4.1. Real-time image input and 6-DoF SLAM output are streamed using a USB3.0 interface. Fig. 4.17 shows the trajectory produced by the chip for KITTI automobile scenes with >1000 images and >500m range. Fig. 4.18 shows frequency/energy scaling of the chip across voltage. At 0.9V nominal voltage, the real-time VGA frame processing latency is 12.5ms. It achieves 97.9% accuracy in translation, 99.34% in rotation on KITTI evaluation rendering large scale automotive scenes over 1km. The chip consumes 243.6mW to process 80fps VGA images at 3.6TOPS/W, marking a 15× improvement in performance and 1.44× in energy efficiency over listed prior works. Moreover, prior works use hand-crafted features or off-chip IMU and do not support large scale KITTI automotive evaluation. Now power reduces to 61.8mW for VGA images at 30fps at 0.63V, yielding 48% additional energy efficiency.

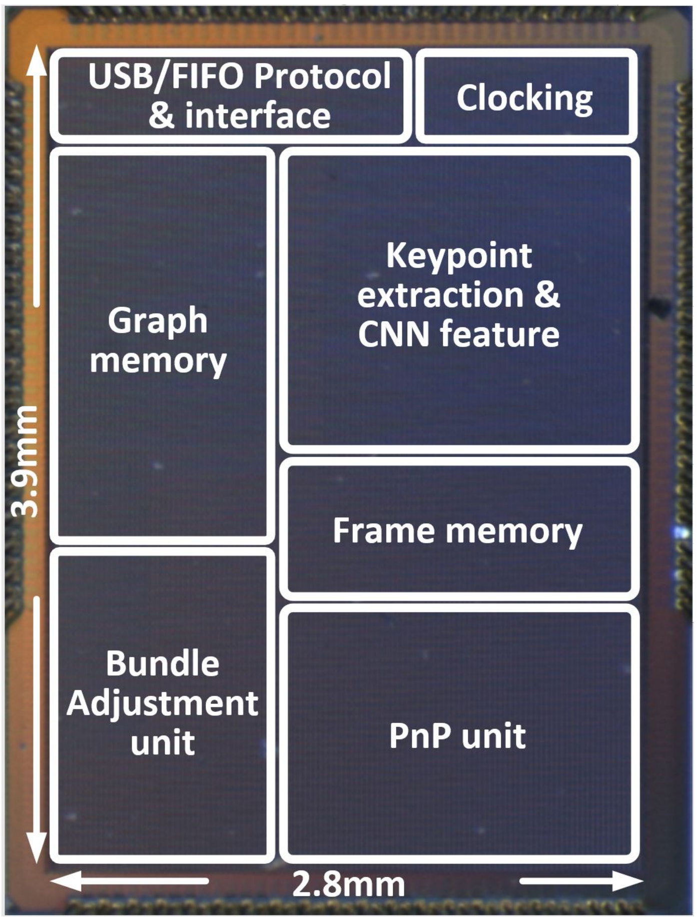


Figure 4.16: Die photo.

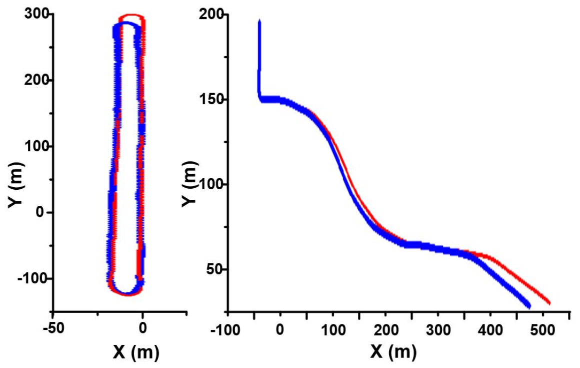


Figure 4.17: Measured trajectory on KITTI dataset.

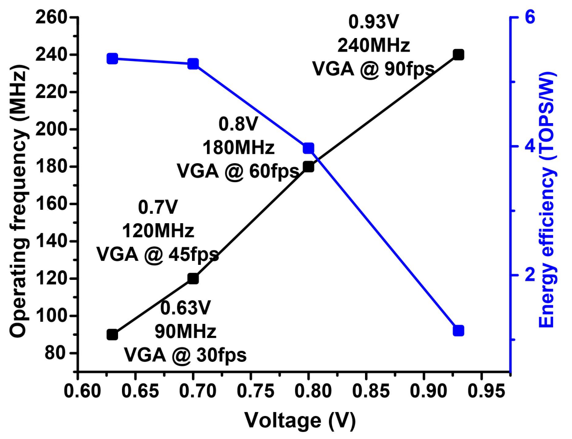


Figure 4.18: Measured voltage and frequency scaling of the design.

4.5 Summary

We designed an energy-efficient CNN-SLAM processor to enable low-power vision based navigation for mobile devices. Proposed processor is fabricated and measured in TSMC 28nm HPC technology. Comparison with prior works is shown in Table. 4.2. Proposed design implements full visual SLAM pipeline with CNN-based feature in a single chip. This design also supports localization and mapping of >1000 key-points/frame on VGA (640×480) resolution in real-time at 80fps, consuming 243.6mW from a 0.9V supply. It achieves 1.44× better energy efficiency over prior works.

	This work
Vision frontend	**CNN feature**
Pose tracking	**PnP**
Graph optimization	**Local BA**
Technology	**28nm**
Chip area	**10.92mm^2**
On-chip memory	**1126kB**
Frequency	**240MHz**
Throughput & image size	**640 X 480, 80fps**
Operation range	**± 500m**
Track points / frame	**1000**
Operating voltage	**0.9V**
Performance	**879.6 GOPS @ 0.9V, 215MHz** **329.8 GOPS @ 0.63V, 90MHz**
Power	**243.6 mW @ 0.9V** **61.75 mW @ 0.63V**
Energy efficiency	**3.6 TOPS/W @ 0.9V** **5.34 TOPS/W @ 0.63V**

Table 4.1: Summary of performance.

		This work	**TVLSI 2013[3]**	**JSSC 2015[4]**	**VLSI 2018[5]**
Complete Visual SLAM system	**Vision frontend**	√ CNN feature	√ Marker recognition	×	√ Harris feature
	Pose tracking	√ Visual	×	√ Visual	× Inertial sensor
	Graph optimization	√ Local BA	×	×	√ Local BA
Technology		28nm	180nm	65nm	65nm
Chip area		10.92mm^2	28.75mm^2	3.25mm^2	20mm^2
On-chip memory		1126kB	N.A	96KB	854kB
Frequency		240MHz	200MHz	133MHz	83.3MHz
Throughput & image size		640 X 480, 80fps	640 X 480, 100fps	640 X 480, 95fps	752 X 480, 90fps
Operation range		± 500m	Not reported due to limited range		
Track points / frame		1000	9	33	200
Operating voltage		0.63-0.9V	1.8V	1.2V	1.2V
Performance		879.6 GOPS @ 0.9V, 215MHz 329.8 GOPS @ 0.63V, 90MHz	153.6 GOPS	20.2 GOPS	10.5-59.1 GOPS
Power		243.6 mW @ 0.9V, 215MHz 61.75 mW @ 0.63V, 90MHz	413 mW	27 mW	24 mW* (excluding IMU power)
Energy efficiency		3.6 TOPS/W @ 0.9V, 215MHz 5.34 TOPS/W @ 0.63V, 90MHz	0.37 TOPS/W	0.75 TOPS/W	0.43-2.5 TOPS/W

Table 4.2: Performance comparison with state-of-the-art chips

CHAPTER V

A 0.8TMACS, 2.5TOPS/W Energy Efficient Re-configurable CNN Processor with 30MB Embedded MLC ReRAM

5.1 Introduction

As is discussed in chapter I, semantic understanding of the surrounding environment is key to autonomous and intelligent systems. Deep neural networks (DNNs), which were proposed back in the 1960s (Fig. 5.1), are the cornerstone of modern artificial intelligence (AI) because of their unprecedented accuracy on many computer vision tasks. They are widely applied to extract the semantic of a scene through various kernel functions such as image recognition[23], semantic segmentation[6] and image-to-text synthesis[94].

The next wave in the AI revolution is the deployment of deep learning in mobile systems to perform challenging tasks under real-world constraints. However, existing hardware and infrastructure cannot provide satisfying performance and energy efficiency for emerging deep learning-based applications because of the excessive computation and large memory footprints in recent DNN models. As is shown in Fig. 5.2, state-of-the-art DNN models typically comprise more than 10 million parameters and require more than 10 GOP per inference, which translates to more than 50 MB on-

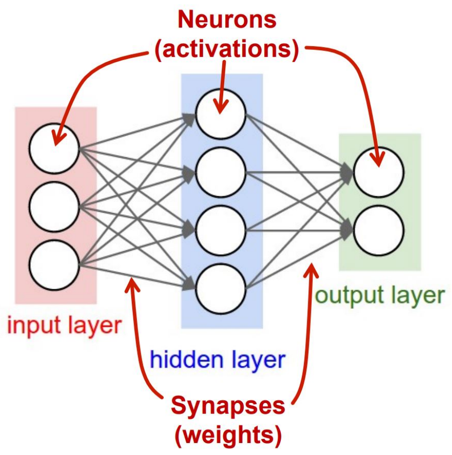


Figure 5.1: Model of a neuron network

chip storage and more than 300 GOPS throughput for real-time 30 fps operation. Thus, there is a growing demand for designing high-performance, energy-efficient, re-configurable software-hardware systems for mobile AI applications.

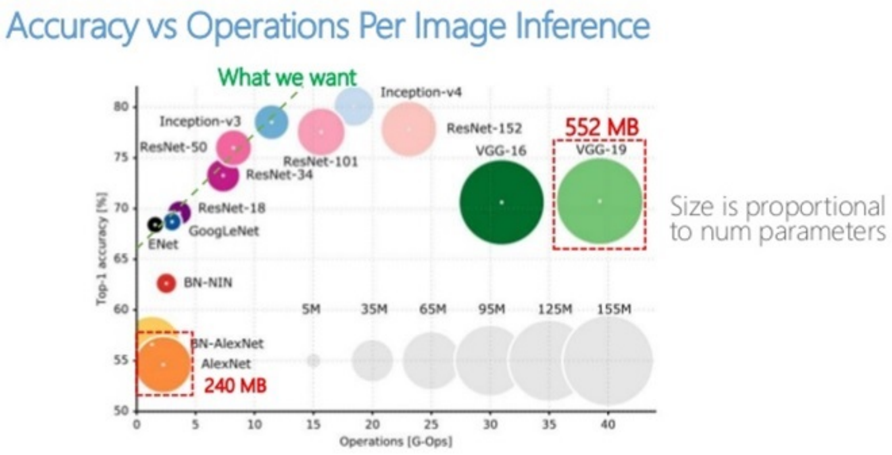


Figure 5.2: Memory and computation requirement for state-of-the-art CNN models

To address these issues, compressed NN models have been proposed to reduce the model size and computation dramatically for mobile applications. However, these compressed models sacrifice accuracy and robustness compared with original models[95, 96]. In addition to the compressed NN models, many other ASICs have been proposed recently to accelerate deep learning on mobile platforms [97, 98, 99, 100, 101, 102, 103]. Various optimization techniques have been explored in these designs, including dataflow optimizations, precision reduction, bit-serial operation, etc. Combining these techniques, state-of-the-art NN processors achieve more than 100 GOPS performance and 2 TOPS/W efficiency during inference. Fig. 5.3 details the performance, efficiency and precision of these digital ASICs.

However, most of these digital chips [97, 98, 99, 100, 101, 102, 103] adopt the DRAM-NPU- (neural processing unit)-style processing architecture for computing large models (Fig. 5.4). Weights and input activations (IAs) are transferred on chip for processing while computed output activations are transferred back to the DRAM. As the processing on the NPU is extensively optimized, transferring data on/off the NPU becomes a major bottleneck in the overall system because of frequent data access on external DRAMs. To reduce the off-chip data/parameter accesses, a few works [104, 102] have proposed to store all parameters on chip. However, these works suffer from either limited memory capacity (only 100 kB of weights are supported in [104]) or high access energy due to using low density SRAMs.

To address these challenges, this chapter presents a high-performance, energy-efficient, re-configurable CNN processor with 30 MB embedded multi-level cell (MLC)

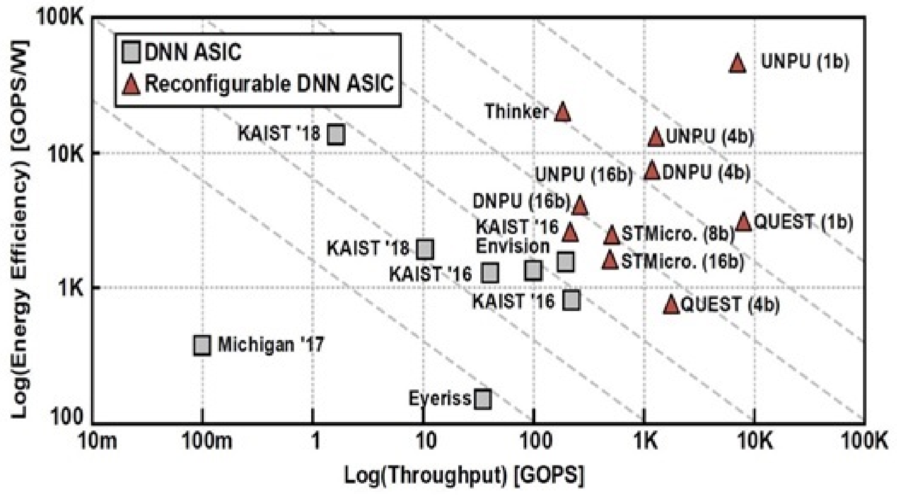


Figure 5.3: Performance and energy efficiency of state-of-the-art NPUs

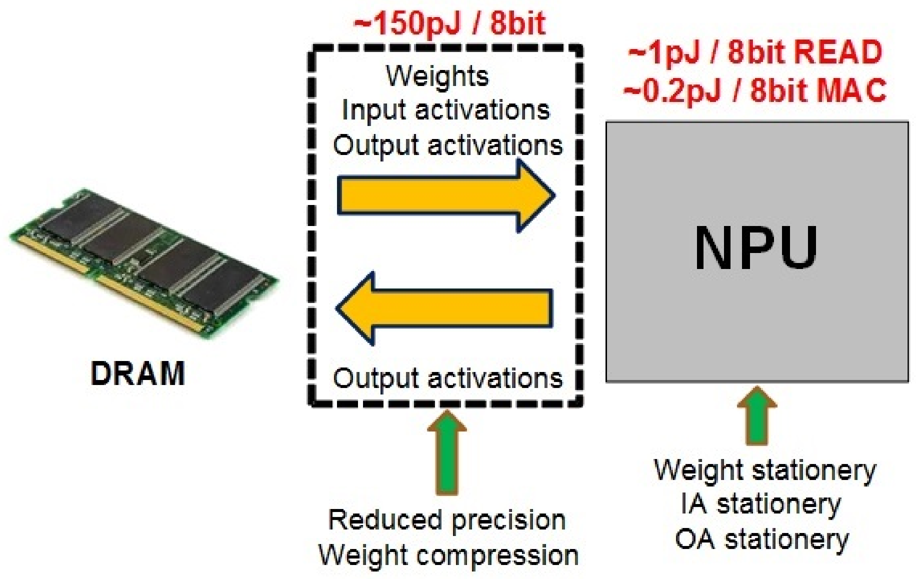


Figure 5.4: Conventional DRAM-NPU architecture for mobile inference

ReRAM on a single chip. The proposed design targets large DNN model inference, and the major design features are as follows:

1) 30 MB compact ReRAM memory using MLC for on-chip non-volatile weight storage, eliminating the need for external DRAMs;

2) Deep compressed and Huffman-coded weights and runlengths to store 50-layer ResNet on a single chip;

3) Distributed memory and compute architecture to maximize data reuse and reduce on-chip data movement (Fig. 5.2 and Fig. 5.3); and

4) Outer product-based matrix multiplication to improve MAC utilization.

By applying these optimizations, we designed a re-configurable ReRAM CNN processor that can process >50 layers ResNet on a single chip, removing the need for external memories. The proposed design fabricated in TSMC 22-nm ULL technology supports real-time DNN inference with 0.8 TMACS throughput and 2.5 TOPS/W energy efficiency.

5.2 Architecture of Proposed ReRAM-CNN Processor

In this section, the overall architecture of the proposed ReRAM-CNN processor is described. Moreover, the detailed architecture of a single PE as well as the decompression engine in the PE are also discussed in this chapter.

5.2.1 16 mesh connected PEs

As shown in Fig. 5.5, the chip consists of 4×4 mesh-connected processing elements (PEs) and a global shared memory. Each PE has its local memory for buffering the input/output activations, ReRAM memory for non-volatile parameter storage, MAC units for highly parallelized processing and instruction memory for controlling the layer functions. In the mesh, each PE has both read and write access to its own local memory but only read access to its neighboring PEs' local memories. The global shared memory is 8 Mb. It supports parallel write and read access if the accesses are pre-partitioned to different memory banks. Moreover, the shared global memory coalesces accesses. The shared memory broadcasts data to a row of, a column of, or all 16 PEs if the requested addresses of the PEs coalesce. In simulation, broadcasting data to coalescing requests results in $4\times$ latency reduction when multiple PEs are fetching the same IA from the global shared memory.

During a layer function, a PE first loads a chunk of IAs from the global shared memory to its local memory. The PE's neighbors can also share its input activation because of the local connectivity between PEs. The PE then processes the layer function on the chunk of inputs with local stored weights. After all output activations are computed, the PE moves the output chunk back to the shared global memory. Each PE may process different data and may execute different instructions, which can lead to a variable processing latency. Therefore, synchronization is necessary to ensure correct layer operations when PEs are collaborating. In the proposed design, the PEs can be synchronized between a row of, a column of, or all 16 PEs.

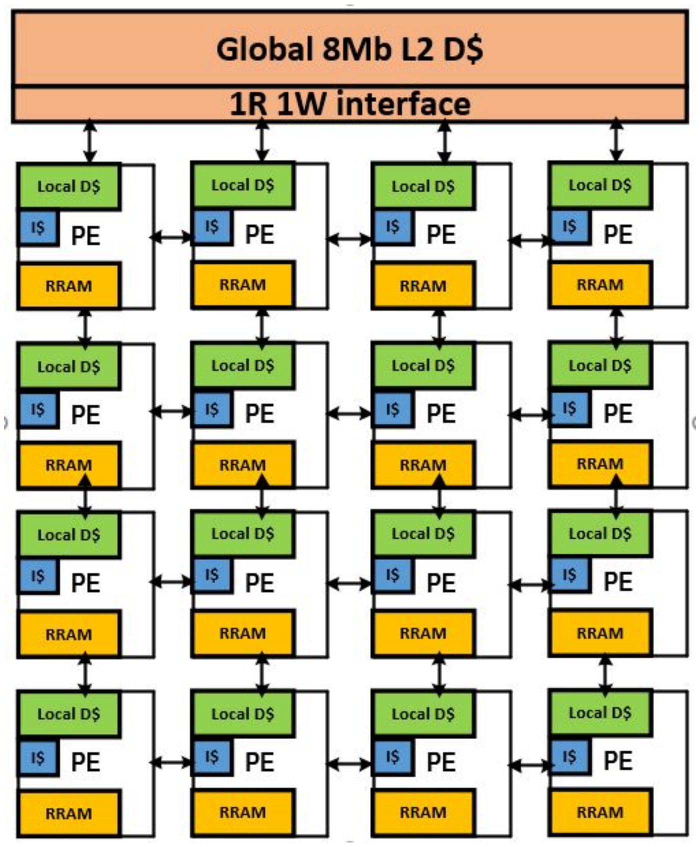


Figure 5.5: Overall architecture of proposed ReRAM-CNN processor

5.2.2 Architecture of a single PE

Fig. 5.6 details the design of a single PE in the 4 × 4 mesh architecture. The architecture of a single PE is similar to the programmable CNN engine described in chapter IV. Each PE has a mesh of 128 8-bit multiply/32-bit accumulate MAC units in 4 clusters (each with a grid of 4 × 8 MAC units). In total, 16 PEs have 2048 MAC units on chip, enabling massive parallel processing capability under compute-intensive CNN operations. Moreover, similar to chapter. IV, each PE processes 4 input channels, 4 output channels and 8 pixels in parallel to maximize the data re-usability in the MAC array. Each PE has its private 18-Mb ReRAM for parameter storage. During the CNN operation, weights are first read from the ReRAM, decompressed through the decompression engine and transferred to small 2-kB interleaved weight buffers for frequent accesses. The processing happens concurrently when the weights are decompressed to maximize the throughput. Accessing the small 1-kB weight memory bank provides 128-bit/cycle memory access bandwidth with high access energy efficiency. The 2 bank, 256-kB local buffer stores input and output activation with 256-bit/cycle access bandwidth. Both the high data bandwidth from the weight buffer and the local buffer ensure the full utilization of the 128 MAC units.

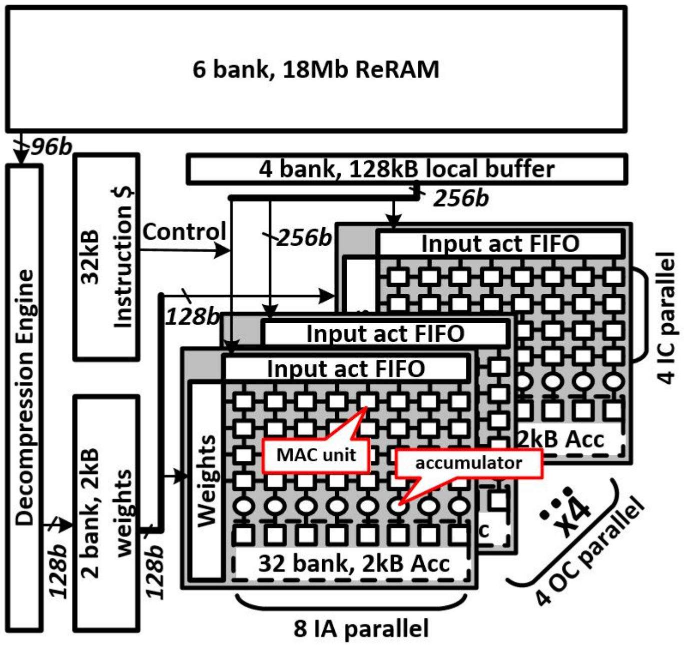


Figure 5.6: Architecture of a processing element.

Moreover, instructions from the 32-kB instruction memory control the processing sequence and synchronization of the NN algorithm. Furthermore, 256-bit Very long Instruction Word (VLIW) instructions are used to control the processing of the MAC units under hundreds of cycles without explicit instruction decoding in each cycle. Fig. 5.7 details the ISA of the proposed ReRAM-CNN processor. The proposed architecture and instruction set support various layer functions, such as convolution, pooling, matrix multiplication, ReLU, etc., as well as flexible layer partition strategies. Data concatenation and scaling can also be achieved through MOV, ADD instructions.

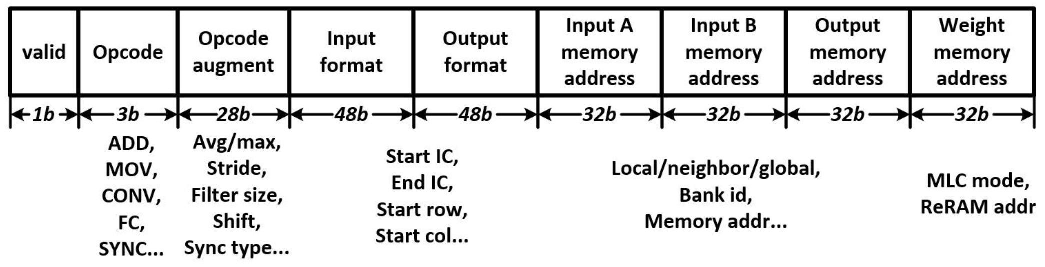


Figure 5.7: ISA of the proposed ReRAM-CNN processor.

5.2.3 Decompression Engine

Each PE is coupled with a decompression engine to decompress the weights stored in the ReRAM. Each decompression engine consists of Huffman tables for both weights and runlength codes as well as a shared parallel look-up table- (LUT)-based decoder. Decompressing Huffman-encoded weights and runlengths involves very long critical paths and thus dominates the pipeline. Therefore, instead of decoding the Huffman tree serially, we decode 4 bits in parallel to improve the performance (Fig. 5.8). This results in storing full 4-bit subtrees in the design (Fig. 5.9). Decompressing weights in parallel reduces the clock cycle of a single PE to 2.5 ns. Moreover, these Huffman tables are stored in each PE and are programmed through the interface. To minimize the programming overhead, multiple PEs can be programmed together if they share the same table.

As is shown in Fig. 5.10, compressed weights are stored in the ReRAM as packets. Each packet is variable in length and is split into multiple ReRAM words. Each packet contains a layer specification and Huffman-coded weights and runlength codes. Each packet only contains Huffman-coded weights and runlengths for 4 input and 4 output channels. The layer specification consists of the offset and location for the entire package. With this configuration, static ReRAM errors can be identified and fixed during the program time. The weights translated from incorrect ReRAM words can be overridden with a following correct packet.

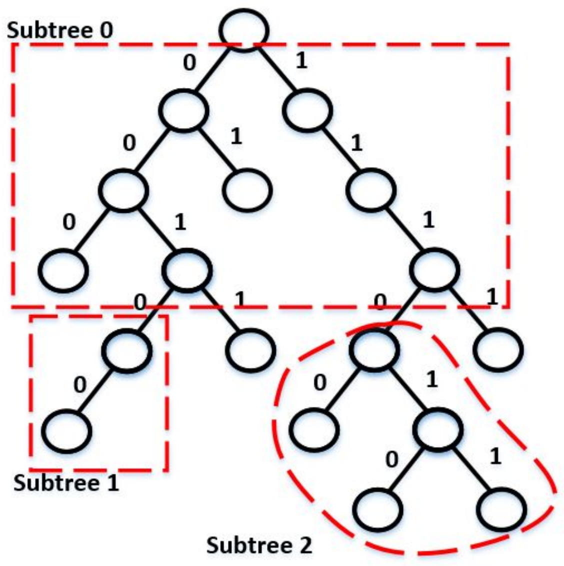


Figure 5.8: Parallel huffman decoder using full subtrees.

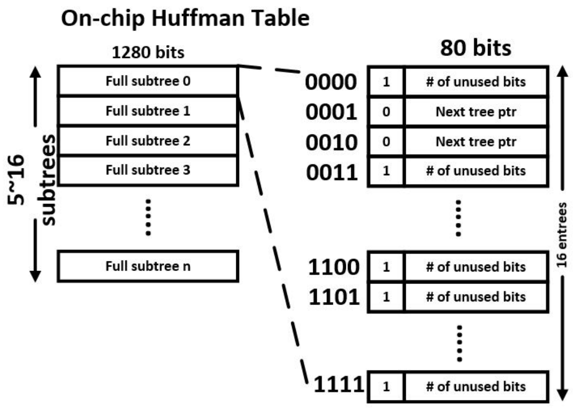


Figure 5.9: On-chip huffman table for decompression.

5.3 Dataflow of Proposed ReRAM-CNN Processor

The proposed architecture and ISA support flexible mapping of a network for efficient hardware execution. This section discusses the various energy-efficient dataflows that are supported in the proposed architecture.

5.3.1 Data Reuse for Efficient CNN Processing Across Layers

One example of mapping a network layer to the architecture is shown in Fig. 5.11. Straightforward mapping splits the IAs by different input channels and sends them

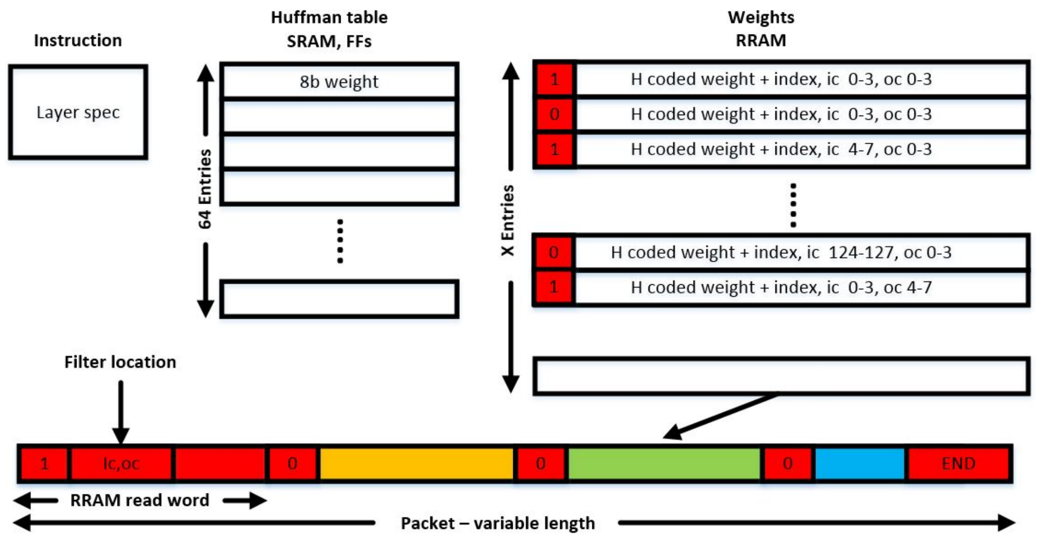

Instruction
Layer spec
Huffman table
SRAM, FFs
8b weight
64 Entries
Weights
RRAM
X Entries
1
H coded weight + index, ic 0-3, oc 0-3
0
H coded weight + index, ic 0-3, oc 0-3
1
H coded weight + index, ic 4-7, oc 0-3
0
H coded weight + index, ic 124-127, oc 0-3
1
H coded weight + index, ic 0-3, oc 4-7
Filter location
1
Ic,oc
0
0
0
END
RRAM read word
Packet – variable length

Figure 5.10: On-chip compressed weight storage on ReRAM.

to different PEs. Note that only 2 PEs are shown in the figure for simplification. The weights are pre-partitioned on these 2 PEs, and each PE is programmed to compute different input channels through instructions. Because of the limited local memory capacity in each PE, the IAs are partitioned into 8 × 8 blocks for processing. After the processing of an 8 × 8 block finishes, the selected PE merges the results from neighboring PEs hierarchically and then writes to the global memory. These 2 PEs then move onto the next 8 × 8 block. Splitting the weights by output channels works in a similar manner and thus is not described here.

Moreover, this architecture also supports mapping different layers to different PEs. As is shown in Fig. 5.12, a bottleneck layer is mapped onto the architecture. Each PE processes a different layer in a pipelined fashion. Similar to an input channel split, IAs are partitioned into 8 × 8 blocks for processing. Each PE processes the current 8 × 8 block of activations, passes output activations to the neighboring PE through a local connection and then fetches the next 8 × 8 input block from the global memory. This mapping scheme enables data reuse across layers but can only be selectively used for efficiency because the valid output activation field decreases after each convolution depending on the kernel size. However, this partition is efficient

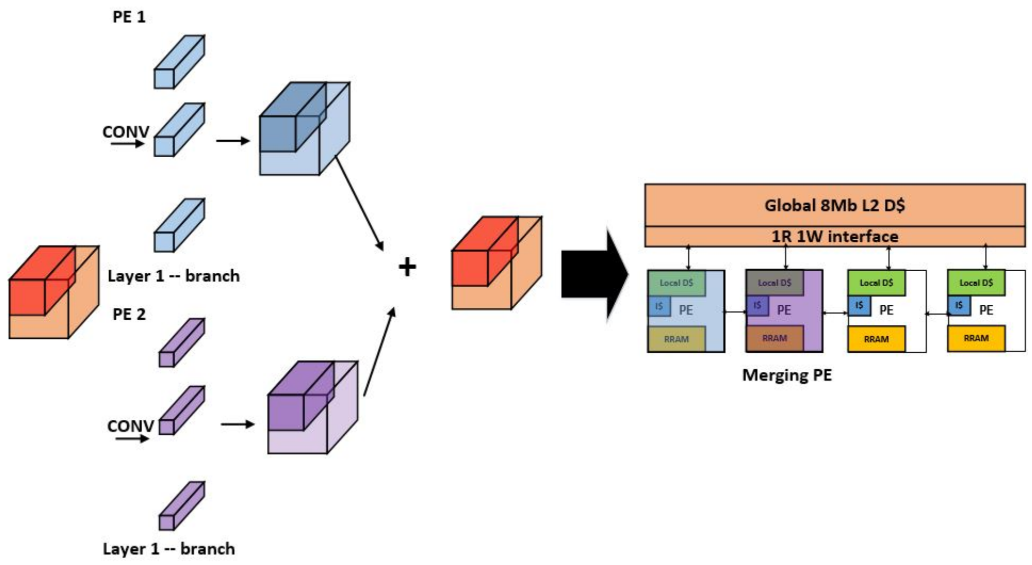


Figure 5.11: Split convolution onto multiple PEs by input channels.

specifically for bottleneck layers, which are widely used in state-of-the-art residual networks [22].

5.3.2 Data Reuse for Efficient CNN Processing Within Convolution Layer

Fig. 5.13 details the operation of the convolution layer on a PE. Similar to chapter IV, a 1D convolution is performed with shifting IAs using a row of 8 MAC units with k (kernel size) cycles. This operation is repeated on a second and third row of

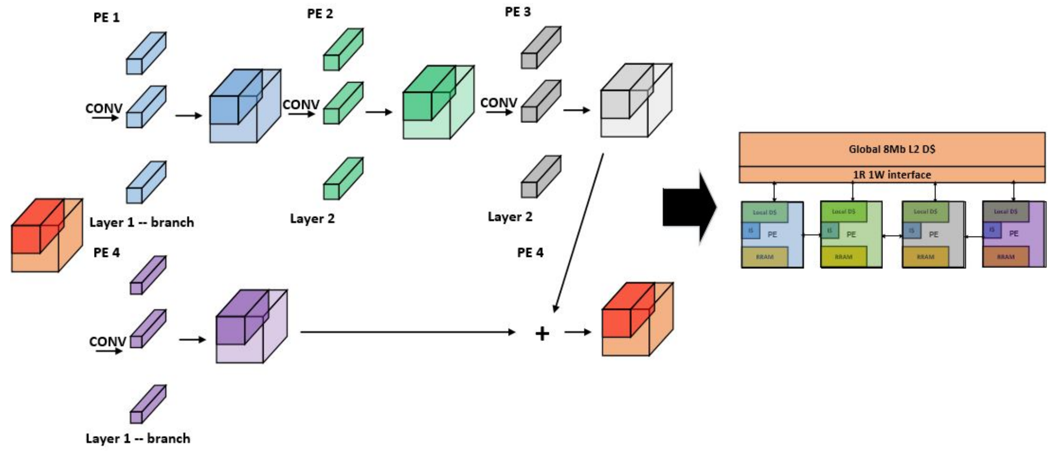


Figure 5.12: Split different layer onto multiple PEs.

IAs to complete the 2D convolution. Max pooling and average pooling are performed in the same fashion as convolution in the 128 MAC units. Partial products in the 2D convolution are fully accumulated locally in each MAC unit. Using 4 clusters of 8 × 4 MAC arrays, 4 consecutive input channels are convolved in parallel to accumulate partial output activations with 4 accumulators, and 4 consecutive output channels are processed in parallel to maximize the reuse of the same IAs.

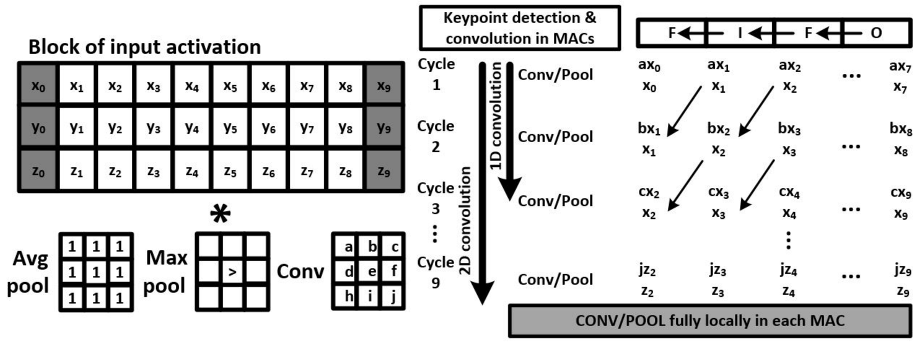

Block of input activation
Keypoint detection & convolution in MACs
F I F O
Cycle 1
Cycle 2
Cycle 3
Cycle 9
1D convolution
2D convolution
Conv/Pool
CONV/POOL fully locally in each MAC
Avg pool
Max pool
Conv

Figure 5.13: Processing flow of convolution on MAC units.

5.3.3 Data Reuse for Sparse FC Processing

The compressed weights for fully connected layers are very sparse, with less than 15% density, while the IAs for the fully connected layers are densely populated. To efficiently compute the fully connected layer and skip all zero multiplications, we deploy the outer product-based matrix multiplication for fully connected layers. Each element of the IAs is multiplied with sparse non-zero weights from ReRAM in each PE. Partial outputs are then accumulated and stored in the accumulators depending on the location of the non-zero weights. Similar to other layer functions, sparse fully connected layer operation is also partitioned on multiple PEs for highly parallelized processing.

5.3.4 Model Compression

To enable single chip implementation for large networks, we leverage an idea from state-of-the-art deep compression schemes [105] for compressing an NN model. However, the compression scheme also must be co-designed for maximizing performance and efficiency with the architecture. The convolution layers typically require less bandwidth on decompressing weights because each weight can be reused over multiple cycles. Therefore, the weights for the convolution layers are pruned and non-linearly quantized with 64 weight centroids and runlength coded using 5-bit runlengths. Because of the high bandwidth required on fully connected layers, weights for fully connected layers are only pruned without any encoding. On average, the proposed compression scheme achieves 7.3 bit per non-zero weight on convolution layers and 13 bit per non-zero weight on FC layers. Table. 5.1 shows an example of applying the compression on AlexNet. Pruning reduces the model size of the convolution layers by 63% and fully connected layers by 85%. Runlength coding and Huffman coding further improves the compression rate by 42%. After restoring the compressed weights, the NN model accuracy only drops by 0.1% under ImageNet [24] evaluation, which is negligible.

	# of parameters	Weight bitwidth	Index bitwidth	Top-5 Accuracy	Compression rate
Baseline	**Conv layers: 2.332 million FC layers: 59 million**	**FP32**	**-**	**77.2%**	**-**
Pruning	**Conv layers: 0.872 million FC layers: 8.862 million**	**FP32**	**-**	**77.1%**	**-**
Pruning + Quantization	**Conv layers: 0.872 million FC layers: 8.862 million**	**INT8**	**-**	**77.0%**	**-**
Pruning + Quantization + Runlength coding	**Conv layers: 0.872 million FC layers: 8.862 million**	**INT8**	**INT5**	**77.0%**	**15.57x**
Pruning + Quantization + Runlength coding + huffman coding	**Conv layers: 0.872 million FC layers: 8.862 million**	**5.2**	**2.1**	**77.0%**	**27.72x**

Table 5.1: Example of apply compression scheme on AlexNet.

5.4 Simulation and Results

The ReRAM-CNN processor is prototyped on TSMC 22-nm ULL technology. The layout of the chip is shown in Fig. 5.14, and the performance summary is shown in Table. 5.2. The image input and programming instructions are streamed using a USB3.0 interface. The inference output is streamed through another USB3.0 interface. In simulation, the chip consumes 640 mW at 400 MHz logic frequency under 0.8 V supply. The throughput of the chip is 0.88 TMACS, marking an energy efficiency of 2.5 TOPS/W with 8 bit arithmetic. Moreover, prior work has limited on-chip memory and needs high bandwidth external DRAM for buffering parameters and intermediate outputs. The proposed design is estimated to achieve 15% improvement in performance over the listed prior works.

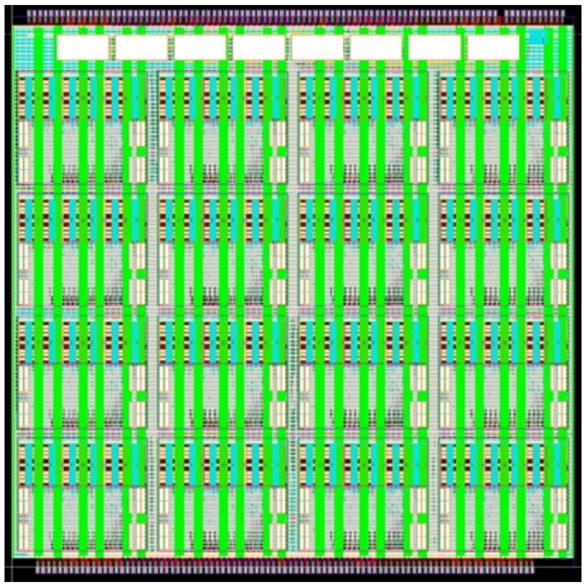


Figure 5.14: Chip layout.

5.5 Summary

We designed an energy-efficient CNN-SLAM processor to enable efficient single chip inference of a large NN model for mobile devices. The proposed processor is

	Proposed
Technology	22ULL with eReRAM
On-chip RAM	36MB
External DRAM	No
# of MACs	2048, 8x8bit
Voltage / V	1.2 ReRAM, 0.8 logic
Frequency / MHz	400 logic, 200 ReRAM
Efficiency / TOPS/W	2.5
Performance / GMACS	0.8 @ 8bit
Power / mW	640
Area / mm^2	42

Table 5.2: Performance summary of the chip

	ISSCC'16 Eyeriss	ISSCC'17 DCNN	ISSCC'17 Envision	ISSCC'17 DNPU	ISSCC'18 UNPU	ISSCC'18 QUEST	VLSI'18 Sticker	Proposed
Technology	65GP	28FD-SOI	28FD-SOI	65GP	65GP	40LP	65GP	22ULL with eReRAM
On-chip RAM	192kB	5.625MB	148kB	290kB	256kB	96MB	170kB	36MB
External DRAM	Yes	Yes	Yes	Yes	Yes	No	Yes	No
# of MACs	168 16x16bit	288 16x16bit	256 16x16bit	768 4x16bit	2304 1x16bit	12288 1x1bit	256 8x8bit	2048 8x8bit
Voltage / V	0.82-1.17	0.575-1.1	1.05	0.77-1.1	0.63-1.1	1.1	0.67-1.0	1.2 ReRAM 0.8 logic
Frequency / MHz	100-250	200-1175	200	200	200	300	20-200	400
Efficiency / TOPS/W	0.26 @ 16bit (exclude DRAM)	2.93 @ 8bit (exclude DRAM)	1 @ 16bit (exclude DRAM)	1 @ 16bit (exclude DRAM)	50.6 @ 1bit (exclude DRAM)	2.27 @ 1bit	0.411 @ 8bit (exclude DRAM)	2.5 @ 8bit
Performance / GMACS	0.042 @ 16bit	0.34 @ 16bit	0.102 @ 8bit	0.15 @ 16bit	0.69 @ 8bit	0.98 @ 4bit	0.102 @ 8bit	0.8 @ 8bit
Power / mW	250	41	44	279	297	3300	248.4	640
Area / mm^2	12.25	34	1.87	16	16	122	12	42

Table 5.3: Performance comparison with state-of-the-art NN processors

prototyped and simulated in TSMC 22-nm ULL eReRAM technology. A comparison with prior works is shown in Table. 5.3. The proposed design supports a large NN model inference with more than 1 million parameters in a single chip. This design achieves 0.8 TMACS throughput in real time, consuming 640 mW from a 0.8 V supply. It achieves 0.15% better performance over prior works.

CHAPTER VI

Conclusions and Future Work

6.1 Contributions

This dissertation focuses on mobile domain-specific computer vision and machine learning processors with low-power consumption, high energy efficiency and high performance, providing potential uses in intelligent and autonomous mobile systems. As we approach the limit of Moore's law for technology scaling, it is challenging to develop mobile general-purpose processors for diverse intelligent mobile applications due to power constraints. In order to solve such difficult problems, we studied, learned and analyzed various kernels of intelligent and autonomous mobile systems including 3D stereo vision perception, 3D optical flow perception, SLAM and deep learning. We proposed novel solutions to optimize mobile vision and machine learning processing from different perspectives in algorithms, architectures and circuits. The proposed solutions were implemented in four prototype systems for demonstration and verification.

The first prototype is an SGM stereo vision processor designed for autonomous navigation of micro aerial vehicle (MAV) applications with tight size, weight and power (SWaP) constraints. The fabricated processor generates 512 levels of depth in full HD (1920×1080) resolution with real-time 30 fps throughput consuming 836 mW from a 0.75 V supply in 40-nm CMOS. Customized ultra-wide SRAM enables 1.64

Tb/s on-chip memory access bandwidth with 18 mW power consumption. The chip is measured with industry standard benchmarks. A complete stereo system is built and demonstrated on a quadcopter for realistic real-time operations.

The second prototype is a real-time optical flow/stereo vision reconfigurable vision processor designed for dense 6D vision perception on low-power mobile platforms such as MAVs and VR/AR systems. The fabricated 6D vision processor generates dense optical flow and depth with a wide search range of 176 pixels per dimension in FHD resolution with real-time 25 fps throughput for optical flow and 30 fps throughput for stereo depth, consuming only 760 mW in 28-nm TSMC HPC CMOS. The proposed dependency-resolving image-scanning stride with deeply pipelined implementation yields a 4$\times$ performance gain. The customized coalescing crosspoint crossbar yields 2.6 Tb/s on-chip bandwidth. A complete optical flow and stereo processing system is built and demonstrated for realistic scenes in real-time operation.

The third prototype is a real-time CNN-SLAM processor designed for six degrees of freedom egomotion and trajectory estimation for MAV and VR/AR applications. The proposed processor is fabricated and measured in TSMC 28-nm HPC technology. The design implements full visual SLAM pipeline with CNN-based features in a single chip. This design also supports localization and mapping of >1000 keypoints/frame on VGA (640$\times$480) resolution in real-time at 80 fps, consuming 243.6 mW from a 0.9 V supply.

The fourth prototype is a large-scale CNN processor with 30 MB on-chip MLC ReRAM and 2 k MACs for high performance and energy-efficient machine learning/deep learning applications on mobile platforms. The proposed processor is fabricated in TSMC 22-nm ULL technology. The design implements 100-million weight CNN model inference in a single chip.

6.2 Future directions

The work presented in this dissertation improves use cases of mobile computing systems by identifying and optimizing kernel functions for next-generation intelligent and autonomous applications. The impact of this work is significant in this era of rapid development of AI, when computers are becoming intelligent, and mobile systems (drones, self-driving cars) are becoming autonomous. However, many issues still remain to be explored and solved in future work. First, domain-specific or application-specific processors might involve significant design effort but lack sufficient volume for mass production. This would, therefore, increase the design costs. Secondly, balancing the performance, power and flexibility will be important in the design. Algorithms keep improving, and therefore sufficient programmability is required for current designs to support further need for new algorithms. Thirdly, memory and control logic have become a bottleneck to improving energy efficiency in computing systems these days. Domain/application-specific designs simplify the control logic, making memory power/latency/bandwidth new performance bottlenecks for many applications that require big data. Managing more volatile and non-volatile memory effectively and distributing compute near memory is becoming increasingly important.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Deloitteauto. Deloitte automotive whitepaper. `https://www2.deloitte.com/content/dam/insights/us/articles/3565_Race-to-autonomous-driving/DR20_The\%20race\%20to\%20autonomous\%20driving_reprint.pdf`. Accessed: 2018-11-09.

[2] DeloitteUAV. Deloitte uav whitepaper. `https://www2.deloitte.com/content/dam/Deloitte/global/Images/infographics/gx-eri-managing-the-evolving-skies.pdf`. Accessed: 2018-11-09.

[3] Shashank Dabral, Sanmati Kamath, Vikram Appia, Mihir Mody, Buyue Zhang, and Umit Batur. Trends in camera based automotive driver assistance systems (adas). In *Circuits and Systems (MWSCAS), 2014 IEEE 57th International Midwest Symposium on*, pages 1110–1115. IEEE, 2014.

[4] Mihir Mody, Pramod Swami, Kedar Chitnis, Shyam Jagannathan, Kumar Desappan, Anshu Jain, Deepak Poddar, Zoran Nikolic, Prashanth Viswanath, Manu Mathew, et al. High performance front camera adas applications on ti's tda3x platform. In *High Performance Computing (HiPC), 2015 IEEE 22nd International Conference on*, pages 456–463. IEEE, 2015.

[5] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European Conference on Computer Vision*, pages 834–849. Springer, 2014.

[6] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2980–2988. IEEE, 2017.

[7] Gauto. Google automotive whitepaper. `https://storage.googleapis.com/sdc-prod/v1/safety-report/waymo-safety-report-2017-10.pdf`. Accessed: 2018-11-09.

[8] Amrthesis. Amr thesis. `file:///C:/Users/liziyun/Downloads/1052124202-MIT.pdf`. Accessed: 2018-11-09.

[9] Myron Z Brown, Darius Burschka, and Gregory D Hager. Advances in computational stereo. *IEEE transactions on pattern analysis and machine intelligence*, 25(8):993–1008, 2003.

[10] Henry Harlyn Baker. Depth from edge and intensity based stereo. Technical report, STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, 1982.

[11] Yuichi Ohta and Takeo Kanade. Stereo by intra-and inter-scanline search using dynamic programming. *IEEE Transactions on pattern analysis and machine intelligence*, (2):139–154, 1985.

[12] W Eric L Grimson. Computational experiments with a feature based stereo algorithm. 1984.

[13] Padmanabhan Anandan. A computational framework and an algorithm for the measurement of visual motion. *International Journal of Computer Vision*, 2(3):283–310, 1989.

[14] Alireza Bab-Hadiashar and David Suter. Robust optic flow computation. *International Journal of Computer Vision*, 29(1):59–77, 1998.

[15] Edward H Adelson and J Anthony Movshon. Phenomenal coherence of moving visual patterns. *Nature*, 300(5892):523, 1982.

[16] Hans P Moravec. Obstacle avoidance and navigation in the real world by a seeing robot rover. Technical report, STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, 1980.

[17] Richard A Newcombe and Andrew J Davison. Live dense reconstruction with a single moving camera. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1498–1505. IEEE, 2010.

[18] Abraham Bachrach, Samuel Prentice, Ruijie He, Peter Henry, Albert S Huang, Michael Krainin, Daniel Maturana, Dieter Fox, and Nicholas Roy. Estimation, planning, and mapping for autonomous flight using an rgb-d camera in gps-denied environments. *The International Journal of Robotics Research*, 31(11):1320–1343, 2012.

[19] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments. *The International Journal of Robotics Research*, 31(5):647–663, 2012.

[20] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[21] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[23] Jia Deng, Alexander C Berg, Kai Li, and Li Fei-Fei. What does classifying more than 10,000 image categories tell us? In *European conference on computer vision*, pages 71–84. Springer, 2010.

[24] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[25] Anuj Mohan, Constantine Papageorgiou, and Tomaso Poggio. Example-based object detection in images by components. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (4):349–361, 2001.

[26] Paul Viola, Michael Jones, et al. Rapid object detection using a boosted cascade of simple features. *CVPR (1)*, 1:511–518, 2001.

[27] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *international Conference on computer vision & Pattern Recognition (CVPR'05)*, volume 1, pages 886–893. IEEE Computer Society, 2005.

[28] Kyuho Jason Lee, Kyeongryeol Bong, Changhyeon Kim, Jaeeun Jang, Kyoung-Rog Lee, Jihee Lee, Gyeonghoon Kim, and Hoi-Jun Yoo. A 502-gops and 0.984-mw dual-mode intelligent adas soc with real-time semiglobal matching and intention prediction for smart automotive black box system. *IEEE Journal of Solid-State Circuits*, 52(1):139–150, 2017.

[29] Intelauto. Intel automotive whitepaper. `http://les-svc.org/wp-content/uploads/2015/06/2016-05-18-Intel-automotive-autonomous-driving-vision-paper.pdf`. Accessed: 2018-11-09.

[30] William R Davis, Bernard B Kosicki, Don M Boroson, and DE Kostishack. Micro air vehicles for optical surveillance. *Lincoln Laboratory Journal*, 9(2):197–214, 1996.

[31] Velodyne LiDAR LLC. Puck hi-res lidar. `https://velodynelidar.com/vlp-16-hi-res.html`. Accessed: 2018-11-09.

[32] Josef Wenger. Automotive radar-status and perspectives. In *Compound Semiconductor Integrated Circuit Symposium, 2005. CSIC'05. IEEE*, pages 4–pp. IEEE, 2005.

[33] TA Krouskop, DR Dougherty, FS Vinson, et al. A pulsed doppler ultrasonic system for making noninvasive measurements of the mechanical properties of soft tissue. *J Rehabil Res Dev*, 24(2):1–8, 1987.

[34] HC Wikle Iii, S Kottilingam, RH Zee, and BA Chin. Infrared sensing techniques for penetration depth control of the submerged arc welding process. *Journal of materials processing technology*, 113(1-3):228–233, 2001.

[35] Hyunggi Cho, Young-Woo Seo, BVK Vijaya Kumar, and Ragunathan Raj Rajkumar. A multi-sensor fusion system for moving object detection and tracking in urban driving environments. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 1836–1843. IEEE, 2014.

[36] Masanori Hariyama and Michitaka Kameyama. Vlsi processor for reliable stereo matching based on window-parallel logic-in-memory architecture. In *VLSI Circuits, 2004. Digest of Technical Papers. 2004 Symposium on*, pages 166–169. IEEE, 2004.

[37] Kyuho J Lee, Kyeongryeol Bong, Changhyeon Kim, Jaeeun Jang, Hyunki Kim, Jihee Lee, Kyoung-Rog Lee, Gyeonghoon Kim, and Hoi-Jun Yoo. 14.2 a 502gops and 0.984 mw dual-mode adas soc with rnn-fis engine for intention prediction in automotive black-box system. In *Solid-State Circuits Conference (ISSCC), 2016 IEEE International*, pages 256–257. IEEE, 2016.

[38] Hong-Hui Chen, Chao-Tsung Huang, Sih-Sian Wu, Chia-Liang Hung, Tsung-Chuan Ma, and Liang-Gee Chen. 23.2 a 1920× 1080 30fps 611 mw five-view depth-estimation processor for light-field applications. In *Solid-State Circuits Conference-(ISSCC), 2015 IEEE International*, pages 1–3. IEEE, 2015.

[39] Junyoung Park, Seungjin Lee, and Hoi-Jun Yoo. A 30fps stereo matching processor based on belief propagation with disparity-parallel pe array architecture. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 453–456. IEEE, 2010.

[40] Stefan K Gehrig, Felix Eberli, and Thomas Meyer. A real-time low-power stereo vision engine using semi-global matching. In *International Conference on Computer Vision Systems*, pages 134–143. Springer, 2009.

[41] Jun Tanabe, Sano Toru, Yutaka Yamada, Tomoki Watanabe, Mayu Okumura, Manabu Nishiyama, Tadakazu Nomura, Kazushige Oma, Nobuhiro Sato, Moriyasu Banno, et al. 18.2 a 1.9 tops and 564gops/w heterogeneous multicore soc with color-based object classification accelerator for image-recognition applications. In *Solid-State Circuits Conference-(ISSCC), 2015 IEEE International*, pages 1–3. IEEE, 2015.

[42] Junyoung Park, Injoon Hong, Gyeonghoon Kim, Youchang Kim, Kyuho Lee, Seongwook Park, Kyeongryeol Bong, and Hoi-Jun Yoo. A 646gops/w multiclassifier many-core processor with cortex-like architecture for super-resolution

recognition. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2013 IEEE International*, pages 168–169. IEEE, 2013.

[43] Yasuki Tanabe, Masato Sumiyoshi, Manabu Nishiyama, Itaru Yamazaki, Shinsuke Fujii, Katsuyuki Kimura, Takuma Aoyama, Moriyasu Banno, Hiroo Hayashi, and Takashi Miyamori. A 464gops 620gops/w heterogeneous multicore soc for image-recognition applications. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*, pages 222–223. IEEE, 2012.

[44] Youngsu Kim, Sungchan Park, Chao Chen, and Hong Jeong. Real-time architecture of stereo vision for robot eye. In *Signal Processing, 2006 8th International Conference on*, volume 1. IEEE, 2006.

[45] Chao-Chung Cheng, Chung-Te Li, Chia-Kai Liang, Yen-Chieh Lai, and Liang-Gee Chen. Architecture design of stereo matching using belief propagation. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 4109–4112. IEEE, 2010.

[46] Yu-Cheng Tseng, Nelson Yen-Chung Chang, Tian-Sheuan Chang, et al. Low memory cost block-based belief propagation for stereo correspondence. In *ICME*, pages 1415–1418. Citeseer, 2007.

[47] Jiang Xiang, Ziyun Li, Hun Seok Kim, and Chaitali Chakrabarti. Hardware-efficient neighbor-guided sgm optical flow for low power vision applications. In *Signal Processing Systems (SiPS), 2016 IEEE International Workshop on*, pages 1–6. IEEE, 2016.

[48] Jiang Xiang, Ziyun Li, David Blaauw, Hun Seok Kim, and Chaitali Chakrabarti. Low complexity optical flow using neighbor-guided semi-global matching. In *Image Processing (ICIP), 2016 IEEE International Conference on*, pages 4483–4487. IEEE, 2016.

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

[50] Ziyun Li, Qing Dong, Mehdi Saligane, Benjamin Kempke, Shijia Yang, Zhengya Zhang, Ronald Dreslinski, Dennis Sylvester, David Blaauw, and Hun Seok Kim. 3.7 a 1920$\times$ 1080 30fps 2.3 tops/w stereo-depth processor for robust autonomous navigation. In *Solid-State Circuits Conference (ISSCC), 2017 IEEE International*, pages 62–63. IEEE, 2017.

[51] Yingcai Bi, Jiaxin Li, Hailong Qin, Menglu Lan, Mo Shan, Feng Lin, and Ben M Chen. An mav localization and mapping system based on dual realsense cameras. In *Int. Micro Air Vehicles, Conf. Competitions, Nat. Univ. Singapore, Singapore, Tech. Rep*, 2016.

[52] Jovan Ivković, Alempije Veljović, Branislav Ranđelović, and Vladimir Veljović. Odroid-xu4 as a desktop pc and microcontroller development boards alternative. In *Proc. 6th Int. Conf.(TIO)*, 2016.

[53] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on pattern analysis and machine intelligence*, 23(11):1222–1239, 2001.

[54] Jian Sun, Nan-Ning Zheng, and Heung-Yeung Shum. Stereo matching using belief propagation. *IEEE Transactions on pattern analysis and machine intelligence*, 25(7):787–800, 2003.

[55] Heiko Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on pattern analysis and machine intelligence*, 30(2):328–341, 2008.

[56] Daniel I Barnea and Harvey F Silverman. A class of algorithms for fast digital image registration. *IEEE transactions on Computers*, 100(2):179–186, 1972.

[57] Paul Viola and William M Wells III. Alignment by maximization of mutual information. *International journal of computer vision*, 24(2):137–154, 1997.

[58] Ramin Zabih and John Woodfill. Non-parametric local transforms for computing visual correspondence. In *European conference on computer vision*, pages 151–158. Springer, 1994.

[59] Cosmin D Pantilie and Sergiu Nedevschi. Sort-sgm: Subpixel optimized real-time semiglobal matching for intelligent vehicles. *IEEE Transactions on Vehicular Technology*, 61(3):1032–1042, 2012.

[60] Heiko Hirschmuller. Accurate and efficient stereo processing by semi-global matching and mutual information. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 807–814. IEEE, 2005.

[61] Daniel Hernandez-Juarez, Alejandro Chacón, Antonio Espinosa, David Vázquez, Juan Carlos Moure, and Antonio M López. Embedded real-time stereo estimation via semi-global matching on the gpu. *Procedia Computer Science*, 80:143–153, 2016.

[62] Heiko Hirschmüller, Maximilian Buder, and Ines Ernst. Memory efficient semi-global matching. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 3:371–376, 2012.

[63] Simon Baker, Daniel Scharstein, JP Lewis, Stefan Roth, Michael J Black, and Richard Szeliski. A database and evaluation methodology for optical flow. *International Journal of Computer Vision*, 92(1):1–31, 2011.

[64] Cypress Semiconductor Corporation. Ez-usb fx3:superspeed usb controller. `http://www.cypress.com/file/140296/downloadkernel.htm`, 2018. Accessed: 2018-09-25.

[65] Stephan Gehrke, Kristian Morin, Michael Downey, Nicolas Boehrer, and Thomas Fuchs. Semi-global matching: An alternative to lidar for dsm generation. In *Proceedings of the 2010 Canadian Geomatics Conference and Symposium of Commission I*, volume 2, 2010.

[66] Intel Inc. D435 camera datasheet. `https://software.intel.com/en-us/realsense/d400/intel-realsense-depth-camera-d400-series-datasheet`. Accessed: 2018-11-09.

[67] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, 2017.

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

[69] Simon Baker and Iain Matthews. Lucas-kanade 20 years on: A unifying framework. *International journal of computer vision*, 56(3):221–255, 2004.

[70] Simon Hermann and Reinhard Klette. Hierarchical scan-line dynamic programming for optical flow using semi-global matching. In *Asian Conference on Computer Vision*, pages 556–567. Springer, 2012.

[71] Li-Xuan Chuo, Yao Shi, Zhihong Luo, Nikolaos Chiotellis, Zhiyoong Foo, Gyouho Kim, Yejoong Kim, Anthony Grbic, David Wentzloff, Hun-Seok Kim, et al. 7.4 a 915mhz asymmetric radio using q-enhanced amplifier for a fully integrated $3\times 3\times 3$mm 3 wireless sensor node with 20m non-line-of-sight communication. In *Solid-State Circuits Conference (ISSCC), 2017 IEEE International*, pages 132–133. IEEE, 2017.

[72] Clemens Rabe, Thomas Müller, Andreas Wedel, and Uwe Franke. Dense, robust, and accurate motion field estimation from stereo image sequences in real-time. In *European conference on computer vision*, pages 582–595. Springer, 2010.

[73] Heiko Hirschmuller and Daniel Scharstein. Evaluation of stereo matching costs on images with radiometric differences. *IEEE transactions on pattern analysis and machine intelligence*, 31(9):1582–1599, 2009.

[74] Yibing Yang, Alan Yuille, and Jie Lu. Local, global, and multilevel stereo matching. In *Computer Vision and Pattern Recognition, 1993. Proceedings CVPR'93., 1993 IEEE Computer Society Conference on*, pages 274–279. IEEE, 1993.

[75] Christian Banz, Peter Pirsch, and Holger Blume. Evaluation of penalty functions for semi-global matching cost aggregation. In *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences [XXII ISPRS Congress, Technical Commission I] 39 (2012), Nr. B3*, volume 39, pages 1–6. Göttingen: Copernicus GmbH, 2012.

[76] Christoph Stiller. Motion estimation for coding of moving video at 8 kbit/s with gibbs-modeled vectorfield smoothing. In *Visual Communications and Image Processing'90: Fifth in a Series*, volume 1360, pages 468–477. International Society for Optics and Photonics, 1990.

[77] Gerard De Haan, Paul WAC Biezen, Henk Huijgen, and Olukayode A Ojo. True-motion estimation with 3-d recursive search block matching. *IEEE transactions on circuits and systems for video technology*, 3(5):368–379, 1993.

[78] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics (ToG)*, 28(3):24, 2009.

[79] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. 1981.

[80] Simon Hermann, Sandino Morales, and Reinhard Klette. Half-resolution semi-global stereo matching. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 201–206. IEEE, 2011.

[81] Daniel J Butler, Jonas Wulff, Garrett B Stanley, and Michael J Black. A naturalistic open source movie for optical flow evaluation. In *European Conference on Computer Vision*, pages 611–625. Springer, 2012.

[82] Ziyang Ma, Kaiming He, Yichen Wei, Jian Sun, and Enhua Wu. Constant time weighted median filtering for stereo matching and beyond. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 49–56, 2013.

[83] Ziyun Li, Jiang Xiang, Luyao Gong, David Blaauw, Chaitali Chakrabarti, and Hun Seok Kim. Low complexity, hardware-efficient neighbor-guided sgm optical flow for low power mobile vision applications. *IEEE Transactions on Circuits and Systems for Video Technology*, 2018.

[84] Terry Tao Ye. *On-chip multiprocessor communication network design and analysis*. PhD thesis, stanford university, 2003.

[85] Sudhir Satpathy, Korey Sewell, Thomas Manville, Yen-Po Chen, Ronald Dreslinski, Dennis Sylvester, Trevor Mudge, and David Blaauw. A 4.5 tb/s 3.4 tb/s/w 64× 64 switch fabric with self-updating least-recently-granted priority and quality-of-service arbitration in 45nm cmos. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*, pages 478–480. IEEE, 2012.

[86] Keisuke Tateno, Federico Tombari, Iro Laina, and Nassir Navab. Cnn-slam: Real-time dense monocular slam with learned depth prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, 2017.

[87] Jae-Sung Yoon, Jeong-Hyun Kim, Hyo-Eun Kim, Won-Young Lee, Seok-Hoon Kim, Kyusik Chung, Jun-Seok Park, and Lee-Sup Kim. A unified graphics and vision processor with a 0.89 μw/fps pose estimation engine for augmented reality. *IEEE Trans. VLSI Syst.*, 21(2):206–216, 2013.

[88] Injoon Hong, Gyeonghoon Kim, Youchang Kim, Donghyun Kim, Byeong-Gyu Nam, and Hoi-Jun Yoo. A 27 mw reconfigurable marker-less logarithmic camera pose estimation engine for mobile augmented reality processor. *IEEE Journal of Solid-State Circuits*, 50(11):2513–2523, 2015.

[89] Amr Suleiman, Zhengdong Zhang, Luca Carlone, Sertac Karaman, and Vivienne Sze. Navion: A fully integrated energy-efficient visual-inertial odometry accelerator for autonomous navigation of nano drones. In *2018 IEEE Symposium on VLSI Circuits*, pages 133–134. IEEE, 2018.

[90] Steven M Seitz, Brian Curless, James Diebel, Daniel Scharstein, and Richard Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *null*, pages 519–528. IEEE, 2006.

[91] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[92] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[93] Jorge J Moré. The levenberg-marquardt algorithm: implementation and theory. In *Numerical analysis*, pages 105–116. Springer, 1978.

[94] Jyoti Aneja, Aditya Deshpande, and Alexander Schwing. Convolutional image captioning.

[95] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and$<$ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[96] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[97] Dongjoo Shin, Jinmook Lee, Jinsu Lee, and Hoi-Jun Yoo. 14.2 dnpu: An 8.1 tops/w reconfigurable cnn-rnn processor for general-purpose deep neural networks. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 240–241. IEEE, 2017.

[98] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, 2016.

[99] Giuseppe Desoli, Nitin Chawla, Thomas Boesch, Surinder-pal Singh, Elio Guidetti, Fabio De Ambroggi, Tommaso Majo, Paolo Zambotti, Manuj Ayodhyawasi, Harvinder Singh, et al. 14.1 a 2.9 tops/w deep convolutional neural network soc in fd-soi 28nm for intelligent embedded systems. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 238–239. IEEE, 2017.

[100] Bert Moons, Roel Uytterhoeven, Wim Dehaene, and Marian Verhelst. 14.5 envision: A 0.26-to-10tops/w subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm fdsoi. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 246–247. IEEE, 2017.

[101] Jinmook Lee, Changhyeon Kim, Sanghoon Kang, Dongjoo Shin, Sangyeob Kim, and Hoi-Jun Yoo. Unpu: A 50.6 tops/w unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision. In *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*, pages 218–220. IEEE, 2018.

[102] Kodai Ueyoshi, Kota Ando, Kazutoshi Hirose, Shinya Takamaeda-Yamazaki, Junichiro Kadomoto, Tomoki Miyata, Mototsugu Hamada, Tadahiro Kuroda, and Masato Motomura. Quest: A 7.49 tops multi-purpose log-quantized dnn inference engine stacked on 96mb 3d sram using inductive-coupling technology in 40nm cmos. In *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*, pages 216–218. IEEE, 2018.

[103] Zhe Yuan, Jinshan Yue, Huanrui Yang, Zhibo Wang, Jinyang Li, Yixiong Yang, Qingwei Guo, Xueqing Li, Meng-Fan Chang, Huazhong Yang, et al. Sticker: A 0.41-62.1 tops/w 8bit neural network processor with multi-sparsity compatible convolution arrays and online tuning acceleration for fully connected layers. In *2018 IEEE Symposium on VLSI Circuits*, pages 33–34. IEEE, 2018.

[104] Suyoung Bang, Jingcheng Wang, Ziyun Li, Cao Gao, Yejoong Kim, Qing Dong, Yen-Po Chen, Laura Fick, Xun Sun, Ron Dreslinski, et al. 14.7 a 288μw programmable deep-learning processor with 270kb on-chip weight storage using non-uniform memory hierarchy for mobile intelligence. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 250–251. IEEE, 2017.

[105] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.