A High Throughput Multi-scale Optical Flow Architecture And Its Application Towards Cloud Tracking

A thesis submitted

in partial fulfillment for the award of the degree of

Doctor of Philosophy

by

Bibin Johnson

Department of Avionics

Indian Institute of Space Science and Technology

Thiruvananthapuram, India

December 2019

Certificate

This is to certify that the thesis titled ***A High Throughput Multi-scale Optical Flow Architecture And Its Application Towards Cloud Tracking*** submitted by **Bibin Johnson**, to the Indian Institute of Space Science and Technology, Thiruvananthapuram, in partial fulfillment for the award of the degree of **Doctor of Philosophy** is a bonafide record of the original work carried out by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. J Sheeba Rani
Associate Professor

Dr. Manoj B.S
Professor & Head

Place: Thiruvananthapuram
Date: December 2019

Declaration

I declare that this thesis titled ***A High Throughput Multi-scale Optical Flow Architecture And Its Application Towards Cloud Tracking*** submitted in partial fulfillment for the award of the degree of **Doctor of Philosophy** is a record of the original work carried out by me under the supervision of **Dr. J Sheeba Rani**, and has not formed the basis for the award of any degree, diploma, associateship, fellowship, or other titles in this or any other Institution or University of higher learning. Keeping with the ethical practice in reporting scientific information, due acknowledgements have been made wherever the findings of others have been cited.

Place: Thiruvananthapuram
Date: December 2019

Bibin Johnson
(SC13D019)

This thesis is dedicated to God, Family, Mentors & My soulmate Ashwathy

Acknowledgements

"*Thank You, God. For giving me the strength, knowledge, ability and opportunity to undertake this research study and to persevere and complete it satisfactorily. I am truly grateful for your exceptional love and grace during this entire journey.*"

First and foremost, I am extremely thankful to Dr. J Sheeba Rani for allowing me to work under her supervision. Her selfless guidance, patience and moral support have made me more passionate towards my research and escorted my vision into a wider dimension. She has inculcated the quality of researcher in me by giving an ample amount of freedom towards my research work.

Besides my PhD thesis supervisor, I would like to thank my doctoral committee members: Dr. Manoj B.S, Dr. Deepak Mishra, Dr. Moosath K.S, Dr. N. Selvaganesan, Dr. Sai Subrahmanyam Gorthi, Prof. M. Balakrishnan (IIT Delhi), Prof. Kolin Paul (IIT Bombay), Dr. Amit Acharyya (IIT Hyderabad) for their constant inspiration, thought-provoking comments, and critical technical reviews throughout the research work.

I am immensely thankful to Dr. V. K. Dadhwal, the Director and Dr. K. S. Dasgupta, the former Director for providing me with a wonderful opportunity for pursuing PhD at IIST. Furthermore, I would also like to deliver my kind gratitude and respect to Dr. Raju K. George, Dean R&D, Dr. Kuruvilla Joseph, Dean Student Activities and Dr. A. Chandrasekar, Dean Academic and Dr. Y.V.N. Krishna Murthy, Registrar for providing me adequate facilities that enabled me to complete my PhD thesis.

My sincere thanks also go to all the faculties of Department of Avionics and staff members (technical and nontechnical), especially Mr. Nidheesh Ravi and Ms. Preetha T., as well as IIST library and information services for their direct or indirect kind support during the course of the PhD thesis. Further, I would like to thank the IIST administration for assuring my pleasant and memorable stay at IIST. This work would not have been possible without the support of staff members from the Department of Avionics, R&D Section, Student Affairs and Finance Section of IIST Trivandrum. I really appreciate their patience and thank them all for the support. I also acknowledge the Department of Space, Government of India for providing the scholarship.

I take this opportunity to thank Satellite Meteorology and Oceanography Research and Training program and Training provided by Semi-Conductor Laboratory (SCL) on Satellite

Image processing and VLSI design flow under ISRO. It was a great opportunity to work with my enthusiastic colleagues Ms. Gayathri R Prabhu, Mr. Sachin Thomas, Mr. Saroop R Pillai, Mr. Nimin Thomas, Mr. Jiljo K Moncy and Mr. Thomas James Thomas as they have always inspired me to work hard towards my goal and I am extremely thankful to them.

I would like to profoundly thank the wonderful IIST-Trivandrum campus for providing a calm and nature-friendly environment that has always made me think positively as well as regain my momentum in work. I am equally thankful to the organizing committee of VLSI Design Conference 2014, which was held in IIT Bombay, for awarding me with fellowship to attain all the tutorials and paper presentations as they have inspired me during the initial phase of my PhD.

I am extremely thankful to my parents and wife Ashwathy for their unconditional support and love during crests & troughs of my research work. It gives me immense pleasure to thank my sister and cousin brothers for their profound support, love and care.

Bibin Johnson

Abstract

Optical flow (OF) computation for fast moving objects is an integral part of many vision systems. It uses an OF constraint to measure the relative displacement of all pixels between two consecutive frames of a video sequence. Most of the high-performing OF algorithms in the literature are based on variational OF algorithm proposed by Horn and Shunck (HS). HS formulates the OF computation as a global optimization problem which can be solved by iteratively minimizing the cost functional using a numerical solver. The OF constraint used as data term in the cost functional is valid only for capturing slow-moving objects. Hence the OF constraint is replaced by a non-linear image differencing constraint with a multi-scale or coarse to fine warping approach to compute large displacement. The large displacement OF computation finds a lot of application in high data rate applications ranging from vision aided robots to the near real-time analysis of atmospheric clouds. Realization of such a system requires high-speed computation of OF with deterministic latency and negligible accuracy loss.

The CPU implementation of variational multi-scale OF algorithm for high-resolution images in real-time is restricted due to the high data rate, the serial execution of pyramid levels, and the high memory bandwidth required for buffering and retrieving the image pyramids and the intermediate flow vectors. Hence this research work identifies the bottlenecks of the variational multi-scale OF algorithm and devises various architecture modification to enhance the throughput while reducing the resource utilization and power consumption. There are three challenges in the variational multi-scale OF algorithm, (1) the large number of arithmetic operations involved in the computation of every flow values, (2) the presence of various feedback loops corresponding to the number of pyramid levels, flow refinements and solver iterations, (3) the high memory bandwidth required for buffering and retrieving intermediate flow vectors and the huge amount of storage needed for buffering the image pyramids as well as the intermediate flow vectors.

These limitations are overcome by making various hardware adaptations to the variational multi-scale OF algorithm which involves the restriction of the complex and resource-intensive flow refinement loop to a single iteration without much loss in accuracy, utilizing a Jacobi solver whose current pixel value depends only on the neighbourhood pixels of the previous iteration instead of a Successive Over Relaxation (SOR) solver, eliminating

some of the complex repetitive arithmetic logic present in each solver iterations to a single arithmetic logic before the solver stage to reduce the resource utilization. The work proposes a variable fixed point time-sharing architecture for the modified algorithm and utilizes parallel architectures for solver, gradient, denoising and interpolation modules to improve the throughput while reducing the resource consumption. It also introduces three different memory banking schemes with customized access pattern for the pyramid, warping and flow resizing stage to improve the system throughput while minimizing the storage requirement. The proposed variational multi-scale OF architecture achieves a frame rate of 306 fps for High Definition (HD) image which is the highest when compared to the state of the art architectures. It makes use of 169 super-scalar units with 702 deep pipelines to achieve a throughput of 395 Giga Operations Per Second (GOPS) with a computation density of 21.5 GOPS/Watt on a Xilinx Virtex 7 device. The work also proposes various fixed and floating point hardware variants of the variational multi-scale OF algorithms to analyse the tradeoff in terms of area, power consumption and accuracy.

The variational multi-scale OF architecture is further analysed in terms of resource utilization and the flow accuracy, which leads to the design of the improved architecture for solver and flow filtering stage. The solver stage consumes the highest area in the proposed variational multi-scale OF architecture. So in order to reduce resource utilization of solver stage, different types of solvers are analysed to come up with a high throughput Red-Black Successive Over-Relaxation (RBSOR) solver architecture. The proposed RBSOR solver architecture is utilized for implementing a variational OF architecture. The proposed variational OF architecture is deeply pipelined to achieve high throughput and provides better accuracy at the cost of a lesser number of iteration compared to other solver implementations. It computes OF for Ultra High Definition (UHD) frames at 48 fps reaching a throughput of 406 Megapixels/s achieving a power efficiency of 43 Giga Operations Per Second/Watt (GOPS/Watt) on a Xilinx Virtex-7 device while operating at 412 MHz.

The flow denoising at each pyramid level of the variational multi-scale OF architecture with fewer solver iterations has a significant impact on the OF accuracy. Hence a more accurate and edge-preserving Bilateral Filter (BF) architecture is considered for flow denoising. But the large computational complexity and poor performance of the BF algorithm motivated to design a High Throughput Bilateral Filter (HTBF) architecture. The unfolding of the architecture utilizing the separability and symmetry property of the filter kernel is explored to reduce the computational complexity. The architecture utilizes a streaming variance compute module to dynamically adapt the range filter coefficients in accordance with the varying noise level. The proposed HTBF architecture can denoise UHD flow fields

at 53 fps on a Xilinx Virtex-7 FPGA device. The proposed fixed and floating point variants of the HTBF architecture achieves a power efficiency of 318 GOPS/W at 470 MHz and 37 Giga Floating-point Operations Per Second/Watt (GFLOPS/W) at 190 MHz respectively.

The proposed multi-scale architecture with improved RBSOR and HTBF subsystems is best suited for embedded vision applications, but the current work focuses on the design of a hardware accelerator for cloud/cyclone tracking and analysis based on a selective choice of various computer vision algorithms and the aforementioned architectures. It aids in taking necessary precaution against extreme climatic effects, helps to study the cloud system interactions using near-real-time satellite data. The software implementation of the proposed cloud analysis framework is computationally intensive and requires large processing time. This motivated to the design of a hardware architecture for cloud accelerator achieving a throughput of 71 fps for HD frames on Xilinx Virtex UltraScale+ device interfaced to host PC via Peripheral Component Interconnect Express (PCIe).

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

OF	Optical Flow
CPU	Central Processing Unit
PC	Personal Computer
GPGPU	General Purpose Graphic Processing Unit
FPGA	Field Programmable Gate Array
DSP	Digital Signal Processor
MPPA	Massively Parallel Processor Array
ASIC	Application Specific Integrated Chip
VLSI	Very Large Scale Integration
HS	Horn and Shunck
LK	Lucas and Kanade
HD	High Definition
FHD	Full High Definition
UHD	Ultra High Definition
QHD	Quad High Definition
VDMA	Video Direct Memory Access
HDMI	High Definition Media Interface
DDR	Double Data Rate
TX	Transmitter
RX	Receiver
FMC	FPGA Mezzanine Card
GOPS	Giga Operations Per Second
GFLOPS	Giga Floating point Operations Per Second
HTBF	High Throughput Bilateral Filter
HEBF	High Efficiency Bilateral Filter
BF	Bilateral Filter
SAHTBF	Self Adaptable High Throughput Bilateral Filter

MRBF	Medium Range Bilateral Filter
RBSOR	Red Black Successive Over Relaxation
SOR	Successive Over Relaxation
PCIe	Peripheral Component Interconnect Express
ME	Motion Estimation
HSOF	Horn and Shunck Optical Flow
PE	Processing Element
CNN	Convolutional Neural Network
PDE	Partial Differential Equation
HDL	Hardware Description Language
RTL	Register Transfer Level
CUDA	Compute Unified Device
API	Application Program Interface
SM	Streaming Multiprocessor
SP	Scalar Processor
RISC	Reduced Instruction Set Computer
TIR	Thermal Infrared
WV	Water Vapour
VIS	Visible
VHRR	Very High Resolution Radiometer
GFS	Global Forecast System
NCEP	National Centre for Environmental Prediction
WRF	Weather Research and Forecasting Model
NCAR	National Centre for Atmospheric Research
IMD	Indian Meteorological Department
AMV	Atmospheric Motion Vector
TC	Tropical Cyclone
SoC	System On Chip
FPS	Frames Per Second
HLS	High Level Synthesis
LP	Linear Pipeline
SP	Segment Pipeline
TSMOF	Time Sharing Multi-scale Optical Flow
HEOF	High Efficiency Optical Flow
HTMOF	High Throughput Multi-scale Optical Flow

HPMOF	High Performance Multi-scale Optical Flow
FB	Feed Back
MAC	Multiply Accumulate
AVG	Averaging
RAM	Random Access Memory
ROM	Read Only Memory
CLB	Configurable Logic Device
AAE	Average Angular Error
AEE	Average Endpoint Error
CBG	Combined Brightness Gradient
V2C	Vector to Colour
HPOF	High Performance Optical Flow
HTOF	High Throughput Optical Flow
DFG	Data Flow Graph
PSNR	Peak Signal to Noise Ratio
MSSIM	Maximum Structural Similarity
AWGN	Additive White Gaussian Noise
FLP	Floating Point
FXP	Fixed Point
CAS	Cloud Analysis Software
UTC	Coordinated Universal Time
MSL	Mean Sea Level
NWP	Numerical Weather Prediction
JTWC	Joint Typhoon Warning Centre

Nomenclature

$\mathrm{I}_1(\mathrm{X})$, $\mathrm{I}_2(\mathrm{X}+h)$	: Consecutive image frames
∇	: Gradient operator
$\nabla I = (I_x, I_y, I_t)$	: Spatial and temporal gradients
$\nabla I_{2w} = (I_{2wx}, I_{2wy})$	: Warped image gradients
$h = (u, v)$	: Flow field with their components
(u_{gt}, v_{gt})	: Ground truth of flow field
$\overline{u}, \overline{v}$	: Velocity averages
du, dv	: Velocity increments
α	: Regularization parameter
$W_{\#}, H_{\#}$	: Pyramid size
$P_{\#}, Q_{\#}$	: Macro-block size
η	: Down-sampling factor
D	: Maximum pixel displacement
L	: Number of pyramid levels
σ^2	: variance
σ_{range}	: standard deviation of range filter
σ_{domain}	: standard deviation of domain filter
$E(h)$	: Energy functional
ϵ	: Small value greater than zero
$a_{\#\#}$	: Coefficients of linear equations
$q_{\#}$	: Variables of linear equations
$b_{\#}$	: Constant terms of linear equations
ω	: Convergence factor
η	: Upsampling factor
(l, k)	: Warping non-integer coordinates
(t_x, t_y)	: Warping integer coordinates
$K \times K$	: Filter kernel size

(i, j) : Pixel / flow Indices
$f(x, y)$: Noisy flow field
$\hat{g}(x, y)$: Denoised flow field
$r(x, y)$: regularizer term
Ω : Filter neighbourhood
$S_{\#}$: Number of segments
$P_{\#}$: Number of Parallel
f_int : Internal frequency
f_samp : Sampling frequency
N_{cyc} : Number of cycles
N_{solv} : Number of solver iterations
N_{out} : Number of warping iterations
N_{cost} : Number of cost computations
N_{store} : Number of stored locations
N_{ops} : Number of operations
$thresh\#$: Threshold value
G : Image intensity
$I_{thresh}(x, y)$: Image undergone thresholding
$J(c_i^t, c_j^{t+1})$: Tracking cost
$O(c_i^t, c_j^{t+1})$: Overlap cost
$C(c_i^t, c_j^{t+1})$: Centroid cost
$S(c_i^t, c_j^{t+1})$: Size cost
N_{comm} : Number of common pixels between the frames
A_i^t : Size/area of cloud segments
X_i^t : Centroid of each cloud segment
$w_{\#}$: Weight of cost criteria
$c_{\#}$: Coefficient of domain filter
reg_i : Initial value of reciprocal function

Chapter 1

Introduction

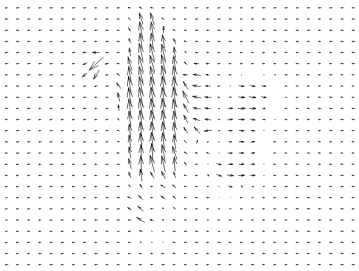
Motion is an important cue which enables the artificial systems to interact with their surroundings, infer the structure of the environment, detect objects and so on. Since the objects are composed of pixels, their motion can be captured by computing the corresponding pixel displacements. The pixel displacement across the image frames can be small or large based on the speed of the objects. The pixel displacement is computed using Motion Estimation (ME) techniques which are broadly classified into a) Feature tracking b) Block matching and c) Optical flow. The feature tracking is a two stages algorithm containing feature selection and feature tracking. The pixels or regions with strong texture information are usually selected as feature points. Then these regions are tracked across frames based on a certain rule to infer their motion. The feature tracking process is treated as an optimization process which minimizes the residue error in a given window. The feature tracking produces a sparse flow field as only distinct feature points are considered. Another category of ME is based on block matching algorithm which is widely used in video compression and media processing applications. The performance of the block matching algorithm is determined by various factors including block size selection, distance metric, searching strategy etc.

With the introduction of variational Optical Flow (OF) algorithm by Horn and Schunck (HS) [1], it became one of the most common methods for performing dense and accurate ME. The Horn and Shunck Optical Flow (HSOF) algorithm treats the OF computation as a global optimization problem which can be solved using the calculus of variation [2]. The HS cost functional is formulated as a weighted average of a data term and a smoothness term. The data term is based on OF constraint, which assumes the pixel intensity remains invariant under motion whereas the smoothness term assumes that the neighbouring pixels also undergo similar motion as that of the object. The HSOF algorithm minimizes the cost functional using a direct or numerical iterative solver to compute the relative displacement of all pixels between two frames. The computed OF can be represented either by displace-

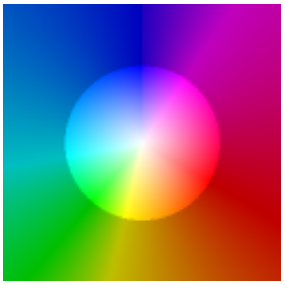
ment vectors or by colour code representation to provide an intuitive perception of actual motion. The Fig. 1.1 (c) and (e) shows the motion field computed across two consecutive images (a) and (b) from Middlebury database. The colour-code representation (e) is a dense visualization of the flow field based on the colour wheel defined in (d). It aids for a better visual perception of difference in the flow field of the neighbouring pixels by associating a colour hue and saturation to the direction and magnitude of the flow vector respectively.

(a) Backyard frame 10

(b) Backyard frame 11



(c) Quiver flow



(d) Colour code

(e) Colour-coded flow

Figure 1.1: Colour code and vector format representation of OF computed from Backyard image pair in the Middlebury database.

Several modifications have been proposed to original variational HS model in the literature during the past three decades, most of the top performing algorithm is based on the variational OF algorithm [3]. The original HSOF algorithm is extremely sensitive to large pixel displacement because the OF constraint is based on the first order Taylor approximation. This leads to undesired results especially in the cases where objects tend to have large displacements. In order to handle large displacement, the OF constraint of HSOF algorithm is replaced by an image differencing model, which leads to a highly non-linear formulation of HS cost functional [4]. The direct computation of non-linear HS cost functional by Euler Lagrange method at fine resolution leads to suboptimal results. A better

initialization leads to faster convergence, which is achieved by multi-scale or coarse to fine approach which downsamples the input image to create a pyramid stack. It help in localizing the search over fewer number of pixels at the coarsest scale which is used to initialize next finer scale thus increasing the effective motion range which is critical for most of the real-world applications. The computation of large displacement OF finds a lot of applications in flow-based action recognition [5], obstacle avoidance, video surveillance [6], vehicle speed estimation [7], accurate solar irradiance forecast [8], tracking the inter/intra motions of the cloud/cyclone systems and analysing the variation of long term cloud patterns [9].

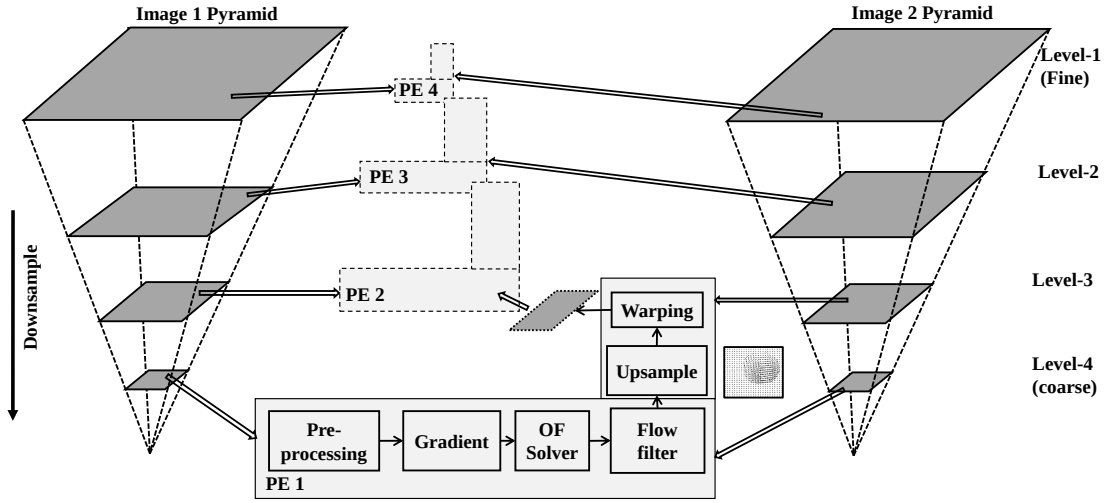


Figure 1.2: Block diagram of a variational multi-scale OF algorithm having four different PE operating on a consecutive image sequence.

Fig. 1.2 shows the block diagram of variational multi-scale OF system. The algorithm constructs a 4 level image pyramids for two consecutive images, the number of pyramid levels depends on the maximum pixel displacement that can be captured accurately by the algorithm. Each pyramid level utilizes a basic Processing Element (PE) which contains pre-processing, gradient computation, solver, flow filtering and image warping modules to compute the variational OF. The OF computation for multi-scale architecture starts at the coarsest level (level-4). A Gaussian filter is utilized in the pre-processing stage to smoothen the input image, this is then fed to a spatiotemporal gradient stage for computing spatial and temporal gradients. The OF is computed by iteratively minimizing the non-linear HS cost functional using an iterative solver. It is denoised using a flow filter and upscaled to the next finer level (level-3). The warping transforms the co-ordinate of the second frame

with the upsampled flow to compensate for the displacement at every level. This helps in reducing the motion range to a single scale OF filter range. This process is repeated for all levels until the finest level (level-1) is reached. The warping and upsampling module are unused in level 4 as the initial flow value are 0 and level 1 respectively. Due to the algorithm complexity and huge computational requirement, the design of variational multi-scale OF architecture with real-time [10] performance and sub-pixel accuracy [11] for embedded vision applications is still an active research area.

1.1 Implementation platforms

The factors which limit the high throughput implementation of variational multi-scale OF algorithm, while achieving low power consumption need to be analysed before the selection of implementation platform. The serial execution of the algorithm limits the achievable parallelization since each level of the OF is to be computed only after the completion of previous level OF. Hence the software implementation of variational multi-scale OF is typically on the order of seconds per frame which prevents many real-time applications of OF. The algorithm operates on a large amount of data (High Definition (1280×720) frames at 176 fps) and involves a large number of computations to estimate motion at a single pixel to find accurate OF. This computational complexity accounts to the fact that the multi-scale variational OF algorithm has L pyramid levels with each level having N_{solv} number of solver iterations. Also, since the non-linear multi-scale OF model involves complex mathematical operation, this needs to be simplified in accordance with the selected platform. A software profiling of variational multi-scale OF algorithm identifies solver as the most computationally intensive part other than the flow filtering, warping and interpolation stages. It also consumes a large amount of memory for buffering the intermediate results and storing the images of the pyramid stack. The available memory bandwidth for a particular platform also limits the achievable performance.

There exist several platforms like CPU, General Purpose Graphics Processing Unit (GPGPU), Field Programmable Gate Arrays (FPGAs), Digital Signal Processors (DSPs), which can solve the above-mentioned challenges of variational multi-scale OF algorithm at a granular level. Matlab is primarily intended for numerical computing, providing the simplest platform for modelling the variational multi-scale OF algorithm with minimum design time. A more practical and faster solution is provided by the software implementation of variational multi-scale OF [12, 13, 14, 4] algorithm on CPU. But is limited to reach real-time performance due to the complex mathematical operations, iterative loops, high

resource utilization and low memory bandwidth.

The performance improvement can be explored on a GPGPU [15] using CUDA trading off the increased power consumption. There exists diverse GPGPU [16, 17] implementation of multi-scale variational OF approaches. However, a GPGPU implementation has higher power consumption and cannot provide a deterministic latency in terms of the number of clock cycles, which are the most demanding requirements in embedded and portable applications. The DSP or FPGA based solutions achieve a very high level of parallelism in operations with a reduced clock frequency of about two orders of magnitude less than a conventional processor or a GPGPU device. The DSPs are more suitable for low power applications while FPGAs are best suited for high-performance applications and serve as the initial prototyping platform in the development of ASIC solutions [18]. The FPGA-based platform was selected for its low power, small size and increasing computation capability which are all essential for embedded vision applications. The research work explores the enhancement of throughput, reduction in resource utilization and efficient memory organisation of multi-scale variational OF methods using optimization in architectural and algorithmic level.

1.2 Objectives and Scope

Scope of the work is on the design of a VLSI architecture for variational multi-scale OF to compute the dense and accurate motion of large displacement objects. The work explores different variational multi-scale OF architecture and its internal subsystems in terms of accuracy and hardware complexity. The research work focuses on hardware friendly adaptation of variational multi-scale OF algorithm and its time-sharing VLSI architecture prototyped on FPGA device. The proposed techniques are particularly designed for grey scale image sequences. The problem addressed in the work is on how to overcome the complexity of variational multi-scale OF algorithm for hardware implementation. Most of the conventional methods based on CPU/GPGPU implementations are not suitable for real-time and low power applications.

The objective of work is to realize a high throughput variational multi-scale OF architecture which can compute large displacement of OF with deterministic latency and negligible accuracy loss in real-time. In order to achieve this, the work involves the analysis of variational multi-scale OF architecture subsystems to understand a) resource requirement of iterative Jacobi solver implementations and proposes methods to reduce the number of iteration making design area efficient b) effect of flow accuracy on the selection of flow

filter in each pyramid level and proposing improved filter architecture without affecting the design throughput. Another goal is to design an accelerator based on the high throughput variational multi-scale OF architecture to speed up the computation of fast moving clouds and cyclones from satellite images or sky cameras which aids the disaster management organizations to mitigate the impact on life and property by taking necessary precautions.

1.3 Our Contribution

The research work focuses on the design of a high throughput variational multi-scale OF architecture for computing dense and accurate OF of fast moving objects in embedded and portable vision applications. The major contributions of the work are the following,

- The work proposes a highly pipelined variational multi-scale OF architecture for capturing large displacement in real-time. It involves the design of a variable fixed point time-sharing architecture to minimize resource utilization and utilizes parallel and unfolding architectures of solver, gradient, denoising and interpolation modules to maximize the throughput. The architecture also introduces three different memory banking schemes with customized access pattern for the pyramid, warping and flow resizing stage to improve the system throughput while minimizing the storage requirement.

- Another focus of work is to explore the internal subsystems of variational multi-scale OF architecture to improve throughput and area efficiency. It involves the design of a high throughput hardware architecture for RBSOR solver consuming lesser resources than Jacobi solver with a similar number of iterations. Followed by the design of a high throughput BF architecture which helps in improving the computed flow accuracy at each pyramid level of the variational multi-scale OF architecture with less number of solver iterations.

- The validation of proposed variational multi-scale OF architecture with improved RBSOR and HTBF subsystems is performed by designing a high throughput hardware accelerator interfaced to host PC via Peripheral Component Interconnect Express (PCIe) for clouds/cyclone analysis and tracking from noisy satellite imagery. It involves various hardware adaptations and utilizing high throughput architectures for pre-processing, cloud motion computation, segmentation, labelling and tracking.

1.4 Outline

This work investigates OF computation in several aspects, with a focus on the design of variational multi-scale OF architecture for large displacement computation with its subsystems and application towards cloud/cyclone tracking. The rest of the thesis is structured as follows:

Chapter 2 discusses the different OF algorithms and the main elements of the variational OF algorithm. The chapter gives a brief introduction to the general notion of OF, and the general cost function used to compute displacements. It also analyses the challenges faced in the real-time implementation of variational multi-scale OF architecture in different platforms. Additionally, this chapter touches on the applications of OF.

Chapter-3 investigates the design of high throughput variational multi-scale OF architecture for fast moving objects. This chapter introduces the hardware adaptation of variational multi-scale OF algorithm, followed by the design of a variable fixed point time-sharing architecture and dedicated banking schemes with their implementation details.

Chapter-4 compares different iterative solvers and come up with high throughput and fast converging RBSOR solver architecture with the implementation details. Followed by a design of high throughput BF architecture to improve the accuracy of the computed flow field.

Chapter-5 discusses the design of a high throughput hardware accelerator interfaced to host PC via PCIe for cloud/cyclone tracking. It also includes hardware implementation details regarding pre-processing, segmentation and tracking stages. The chapter concludes with experiments on synthetic and real-world satellite images showing their applicability in cloud/cyclone tracking.

Chapter-6 concludes this thesis. In this chapter, the work presented is summarized with the proposed improvements. A detailed discussion on future improvements is also included in this chapter.

Chapter 2

Preliminary study towards a Hardware Implementation

This chapter presents a brief description of the existing OF techniques with an emphasis on variational OF algorithm based on HSOF. This chapter discusses the modification in the variational OF algorithm to handle large pixel displacement. This part focuses on the challenges faced in the real-time implementation of variational multi-scale OF architecture in different platforms. It also gives brief information about the application of large displacement OF such as embedded vision systems, hardware accelerators for cloud/cyclone analysis with the benchmarking datasets for analysing the architecture performance.

2.1 OF constraint model

Let $I(X)$ or $I(x, y, t)$ represent the intensity of the pixel (x, y) in the video sequence at time t. The brightness constancy assumption states that the pixel intensities between two consecutive frames $I_1(X)$ and $I_2(X + h)$ of a video sequence remain constant over time as given in equation (2.1), where $h = (\frac{\Delta x}{\Delta t}, \frac{\Delta y}{\Delta t}) = (u, v)$ corresponds to the flow vectors in horizontal and vertical direction respectively.

$$I_1(X) = I_2(X + h) \tag{2.1}$$

By expanding the equation (2.1) using Taylor series [1] to get equation (2.2).

$$I_2(X + h) = I_1(X) + I_x u + I_y v + I_t + H.O.T \tag{2.2}$$

where I_x, I_y are the spatial gradients, I_t corresponds to temporal gradient and Higher

Order Terms (H.O.T) are often set to 0. Substituting the equation (2.2) in equation (2.1) leads to OF constraint given by equation (2.3).

$$I_x u + I_y v + I_t = 0 \tag{2.3}$$

However, the OF constraint is underconstrained with two unknowns u and v, which leads to the aperture problem. The aperture problem describes the ambiguity of the inferred motions when observing local image structures [19]. The local observation only specifies the component of motion perpendicular to the edge boundary. Hence, any information related to the component of motion parallel to the edge is lost, which results in the ambiguity. Researchers have proposed various OF methods [3] by encoding a priori information of the flow field along with the OF constraint to make the problem well posed.

2.2 Classification of OF algorithm

Generally, optical flow techniques are classified into a) Differential-based b) Region-based c) Feature-based d) Energy-based e) Phase-based and f) Convolutional Neural Network (CNN)-based techniques.

2.2.1 Differential methods

The differential technique or gradient-based approach computes the spatial and temporal derivatives of the image brightness [20] to estimate the velocity of the object. The differential techniques are classified into global and local methods, based on the difference in spatial coherence imposed by either local or global constraints.

Variational method

The HSOF algorithm is considered as the global variational method, which formulates the OF as the minimizer of a certain energy functional having a data term and smoothness term. The introduction of global smoothness term helps to solve the aperture problem. It is based on the assumption that neighbouring pixels belong to the same object will undergo a similar motion. The energy model uses quadratic functionals based on the assumption that the image noise and the flow derivatives are expected to follow a Gaussian distribution and hence is sensitive to the presence of noise, illumination changes and occlusions. The data term takes the form of high-order constraints like gradient constancy [14], Hes-

sian constancy and Laplacian constancy in [21]. The gradient constancy together with the OF constraint as data term in HSV colour space is utilized in [22] to handle illumination changes. The work [23] introduce a more suitable data term based on a binary weighted map to switch between gradient constancy or brightness constancy.

Smoothness term is another part of the cost functional which assumes similar motion for pixels in the neighbourhood while preserving discontinuities at motion boundaries. The original smoothness term proposed in the HS algorithm ignores flow discontinuities and over-smooth motion boundaries. Hence an image-driven isotropic and anisotropic smoothness terms are introduced in [24] and [25], to suppress smoothing across image boundaries. The isotropic term avoids over-segmentation of textured structures [26], while an anisotropic term [27] achieves smoothing along flow discontinuities while producing fewer discontinuities. The work [27] constructs a joint image and flow driven smoothness term to avoid artefacts of over-smoothing and over-segmentation. The spatial-smoothness can be extended to the temporal domain based on the assumption that flow varies smoothly and slowly over time. The temporal smoothness is achieved by considering a temporal flow gradient [28, 27], or considering a temporal coherence along the object motion trajectory [29, 30].

The global cost functional can be minimized by continuous optimization [31, 32] like gradient descent [27] and variational method [1, 14] or a discrete optimization [11] using patch matching [33] or graph cuts algorithms [34]. The discrete optimization method approximates the continuous solutions space to enable a complete search of the state space. The differentiation of the energy functional is not performed in discrete optimization methods and hence it can handle a wide variety of data and regularization [35]. But these methods are limited in terms of accuracy and efficiency by the size and number of the label space [36].

Local differential method

The Lucas and Kanade (LK) [37] proposed a parametric model to capture the motion within a local neighbourhood. The model signifies the relation between the brightness and motion of each pixel in a local neighbourhood. The OF is computed by performing a least-square minimisation of the given set of equations for all the pixels in the local neighbourhood. The selection of a small neighbourhood leads to insufficient information leading to the aperture problem whereas the selection of large neighbourhood leads to the inclusion of pixels from other motion surfaces. The local methods are not able to compute OF in homogeneous regions as well as regions with motion discontinuities [38]. The implementation of non-

variational LK [37] algorithms on GPU can be found in [39, 40]. The absence of smoothness term in the LK algorithm made the computation simpler and hence is found to be suitable for high-speed VLSI and FPGA implementations in [41] and [42] respectively.

2.2.2 Region matching

The region-based technique relies on finding the best match between two patches in the consecutive images. The best match is obtained by identifying the largest correlation across shifted patches [43]. It is more robust than differential methods in the presence of noise and works well with downsampled images. The OF computed by region matching method lacks sub-pixel accuracy, and hence need to perform additional flow refinement to improve the flow accuracy.

2.2.3 Feature matching

The feature-based technique is a two-step process, in which the first step retrieves the sparse discriminative features like corners and edges from successive images. Followed by the feature matching step to compute the sparse OF. Even though the flow field is sparse, it produces a robust field by ignoring ambiguous areas. The feature-based OF methods are widely used for large displacement matching [44]. If the selected object lacks discriminative features or is disappeared in the subsequent frame, the computed motion field will be very sparse.

2.2.4 Frequency methods

The frequency-based technique calculates OF in the Fourier domain using velocity-tuned filters. The frequency-based techniques are classified into Energy based method and Phase base method.

Energy based method

The energy-based methods compute the OF based on the energy of the velocity tuned filters. The energy of continuous motion in space corresponds to the orientation of the plane in the spatiotemporal frequency domain with the orientation corresponding to the velocity [45]. The quadrature Gabor filter described for motion perception [46] is modified to compute OF [47].

Phase based method

The phase-based method is insensitive to changes in speed and contrast is utilized to compute the phase response of the band-pass filter. Fleet-Jepson [48] defines the velocity component as the output of bandpass velocity tuned Gabor filters. The image is decomposed into band-pass channels, in a similar way as that of quadrature-pair filters in steerable pyramids [49]. Phase-based methods have sub-pixel accuracy and the velocity resolution can be further improved by taking responses from neighbouring filters.

2.2.5 CNN based method

The CNN based methods outperform most of image processing tasks including OF [50] by replacing the hand-crafted features with learned features [51]. The learned features are integrated into a common optimization framework to compute OF [52, 53]. CNN's are commonly trained in a supervised way requiring a large amount of ground-truth to achieve reasonable accuracy. However, with limited data set in the case of OF, it is difficult to train the algorithm to make it adaptable to the respective scenarios. The recent work also addresses the unsupervised training of CNN for computing OF [51].

2.3 Algorithm formulation of Variational OF

With the introduction of variational OF algorithm by HS, many extensions and modifications have been suggested to improve the performance of the HS method on estimating OF [14, 54]. According to the most recent survey [3], most of the top performing OF algorithms are based on variational HS technique, where increased accuracy and the fill-in effect are of major concern [55, 56]. The three main advantages of variational OF [31] are, a) it can combine merits of different assumptions into one single minimization framework, b) it has the filling-in effect which yields a dense flow field, whereas most other methods perform post-processing to interpolate the sparse flow field and c) the cost functional can be made rotational invariant.

The HS algorithm formulates the OF as a variational problem, where the desired vector field h is defined as the minimizer of a certain energy functional $E(h)$ as given in equation (2.4). This functional has two terms: a data term, given by the OF constraint, and a smoothness term that is based on the gradient of flow:

$$E(h) = (I_x u + I_y v + I_t)^2 + \alpha^2(|\nabla u|^2 + |\nabla v|^2) \tag{2.4}$$

where the parameter α controls the weight of smoothness term in comparison with the OF constraint. The minimization of cost functional given in the equation (2.4) by Euler-Lagrange formulation produces a system of Partial Differential Equations (PDE) given by equation (2.5).

$$\begin{aligned} {I_x}^2 u + I_x I_y v &= \alpha^2 div(\nabla u) - I_x I_t \\ I_x I_y u + {I_y}^2 v &= \alpha^2 div(\nabla v) - I_y I_t \end{aligned} \tag{2.5}$$

The equations (2.5) is discretized by approximating Laplacian according to $div(\nabla u) = (\bar{u} - u)$, where ($\bar{u}$, $\bar{v}$) represents the local averages of velocity components. The local averages $(\bar{u}, \bar{v})$ are estimated from the eight neighbours of (u, v) given in equation (2.6).

$$\begin{aligned} \bar{u} = \frac{1}{6}(u^n_{(i-1,j)} + u^n_{(i+1,j)} + u^n_{(i,j-1)} + u^n_{(i,j+1)}) + \frac{1}{12}(u^n_{(i-1,j-1)} + \\ u^n_{(i+1,j-1)} + u^n_{(i-1,j+1)} + u^n_{(i+1,j+1)}); \\ \bar{v} = \frac{1}{6}(v^n_{(i-1,j)} + v^n_{(i+1,j)} + v^n_{(i,j-1)} + v^n_{(i,j+1)}) + \frac{1}{12}(v^n_{(i-1,j-1)} + \\ v^n_{(i+1,j-1)} + v^n_{(i-1,j+1)} + v^n_{(i+1,j+1)}); \end{aligned} \tag{2.6}$$

After substitution and rearranging the following system of equations (2.7) is obtained.

$$\begin{aligned} ({I_x}^2 u + {I_y}^2 u + \alpha^2)(u - \bar{u}) &= -I_x(I_x \bar{u} + I_y \bar{v} + I_t) \\ ({I_x}^2 u + {I_y}^2 u + \alpha^2)(v - \bar{v}) &= -I_y(I_x \bar{u} + I_y \bar{v} + I_t) \end{aligned} \tag{2.7}$$

Writing these equations (2.7) for each pixel of the input images form a sparse system of linear equations. This equation (2.8) can be solved efficiently with an iterative scheme.

$$\begin{aligned} u^{n+1} &= \bar{u}^n - I_x \frac{(I_x \bar{u}^n + I_y \bar{v}^n + I_t)}{(\alpha^2 + {I_x}^2 + {I_y}^2)} \\ v^{n+1} &= \bar{v}^n - I_y \frac{(I_x \bar{u}^n + I_y \bar{v}^n + I_t)}{(\alpha^2 + {I_x}^2 + {I_y}^2)} \end{aligned} \tag{2.8}$$

The iterative process can be stopped before a fixed number of iterations, if a stopping criterion in equation (2.9) based on two consecutive values of h is met.

$$\frac{1}{N} \sum_{i,j} (u^{n+1}_{i,j} - u^n_{i,j})^2 + (v^{n+1}_{i,j} - v^n_{i,j})^2 < \epsilon \tag{2.9}$$

2.4 Variational Multi-scale OF algorithm

The main shortcoming of variational OF methods is that they are extremely sensitive to large pixel displacement, as the OF constraint is based on the first order Taylor approximation. This leads to undesired results as the computed flow values can easily get trapped in local minima especially in the cases where objects tend to have large displacements. To estimate large pixel motion, the OF constraint is replaced with non-linear image differencing model given in the equation (2.10) to get a non-linear HS cost functional [14, 4].

$$I_1(X) - I_2(X+h) = 0 \tag{2.10}$$

Substituting the equation (2.10) in the original cost functional in the equation (2.4) yield a modified non-linear HS cost functional given in the equation (2.11).

$$E(h) = (I_1(X) - I_2(X+h))^2 + \alpha^2(|\nabla u|^2 + |\nabla v|^2) \tag{2.11}$$

The minimization of energy functional in the equation (2.11) yields the following Euler-Lagrange solution given in the equation (2.12).

$$\begin{aligned}(I_2(X+h) - I_1(X))I_{2x}(X+h) - \alpha^2 div(\nabla u) = 0 \\ (I_2(X+h) - I_1(X))I_{2y}(X+h) - \alpha^2 div(\nabla v) = 0\end{aligned} \tag{2.12}$$

The equations (2.12) are non-linear in h due to the $I_2(X+h)$ term. Hence a first order Taylor series linearisation is performed on $I_2(X+h)$ as given in the equation (2.13).

$$I_2(X+h^{n+1}) = I_2(X+h^n) + \nabla I_2(X+h^n)\cdot(h^{n+1} - h^n) \tag{2.13}$$

Further the divergence of the gradient of flow can be approximated by the equation (2.14).

$$\begin{aligned}div(\nabla u) = \overline{u} - u \\ div(\nabla v) = \overline{v} - v\end{aligned} \tag{2.14}$$

Substituting the equation (2.13) and (2.14) in the system of equations (2.12) to get the new system of equations (2.15) which is solved using Jacobi numerical iterative scheme with the iteration index l.

$$I_x(I_z + I_x du^{l+1} + I_y dv^{l+1}) - \alpha^2\big((\overline{u}^l + \overline{du}^l) - (u^l + du^l)\big) = 0 \qquad (2.15)$$
$$I_y(I_z + I_x du^{l+1} + I_y dv^{l+1}) - \alpha^2\big((\overline{v}^l + \overline{dv}^l) - (v^l + dv^l)\big) = 0$$

where the given notation corresponds to the equation (2.16).

$$\begin{aligned} I_1(x) &= I_1 \qquad (2.16)\\ I_2(x+h) &= I_2 \\ I_{2x}(x+h) &= I_x \\ I_{2y}(x+h) &= I_y \\ I_z &= I_2(x+h) - I_1(x) \end{aligned}$$

The system of equations (2.15) are rearranged to compute flow increments du and dv of the original flow u and v respectively as given in the equations (2.17).

$$du^{k,l+1}((I_x^k)^2 + \alpha^2) + dv^{k,l+1}(I_x^k I_y^k) = \alpha^2(\overline{u}^{k,l} + \overline{du}^{k,l} - u^{k,l}) - I_x^k I_t^k \qquad (2.17)$$
$$dv^{k,l+1}((I_y^k)^2 + \alpha^2) + du^{k,l+1}(I_x^k I_y^k) = \alpha^2(\overline{v}^{k,l} + \overline{dv}^{k,l} - v^{k,l}) - I_y^k I_t^k$$

The solution of equations (2.17) at fine resolutions is complex, as it involves a higher risk of converging to a local minimum. The numerical approximation is achieved using a coarse-to-fine warping strategy [14] or multi-grid approaches [55], providing a better initialization to improve the convergence. The multi-grid methods are numerical algorithms used to solve differential equations [57] and offer a fast numerical scheme for solving linear equations on CPUs for some low and medium resolution images. Bruhn et al. [58] proposed the use of a full multi-grid method to speed up the computation of variational OF. Another multi-grid computation algorithm was proposed in [55], which implements the variational algorithm that was proposed by Brox et al.[14]. However multi-grid methods are known to be problem specific and require very complicated implementation, hence multi-scale methods are commonly used for solving the equations (2.17). The over smoothing of fine structures in coarse-to-fine strategy can be handled by integrating an additional descriptor matching in the HS cost functional [59].

The Gaussian smoothing followed by the sub-sampling is applied to the input image

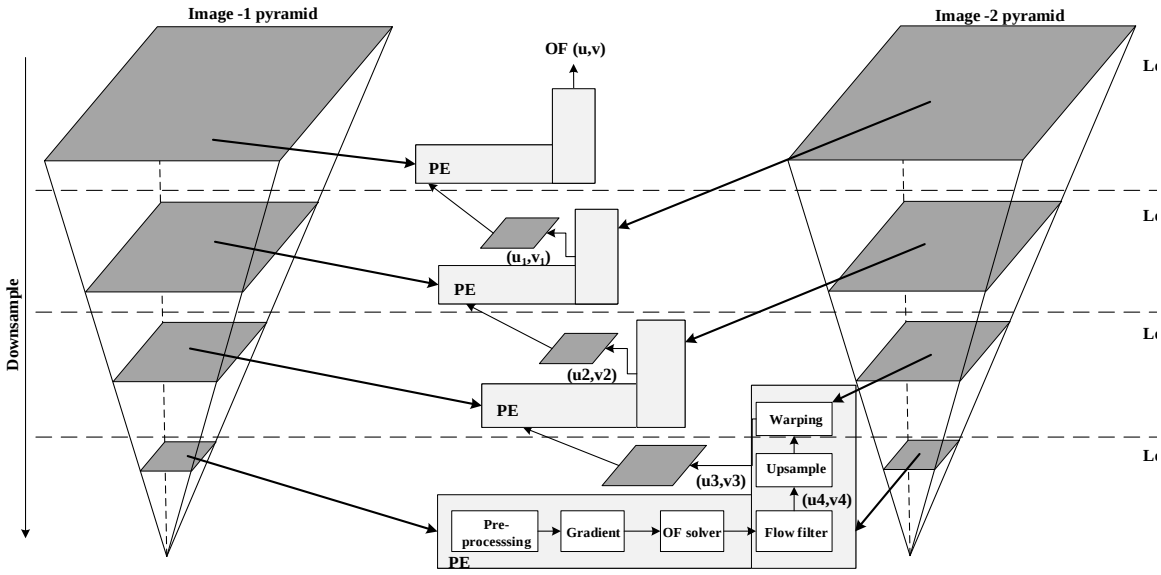


Figure 2.1: Block diagram of a variational multi-scale OF algorithm.

[60] to build the image pyramid as shown in Fig. 2.1. The sub-sampling at pyramid level k reduces the image resolution by a factor of η as compared to the previous level k-1, which also involves the reduction of the maximum pixel displacement in the available filter range applied at that level. The solution is first looked for in the coarsest layer of this pyramid, where the problem is usually easier to be solved. The flow computation stage evaluates the equations (2.17) using iterative solvers. The computed flow increment at each level is merged with the previous flow and filtered to remove the random flow values. A two-dimensional median filter is used to regularize the results. The flow field is upsampled and scaled by a factor of η to compute next fine level flow as given in the equation (2.18).

$$h^k = Upsamp(\frac{h^{(k-1)}}{\eta}) \tag{2.18}$$
$$h^k = h^k + < du^k, dv^k >$$

The images are warped with the flow estimate from the previous coarse level to compensate for Taylor approximation of the solver equations. The process of image warping is fundamental to many image processing application and is given in the equation (2.19). Warping adds the motion field obtained from the previous scale in order to reduce the

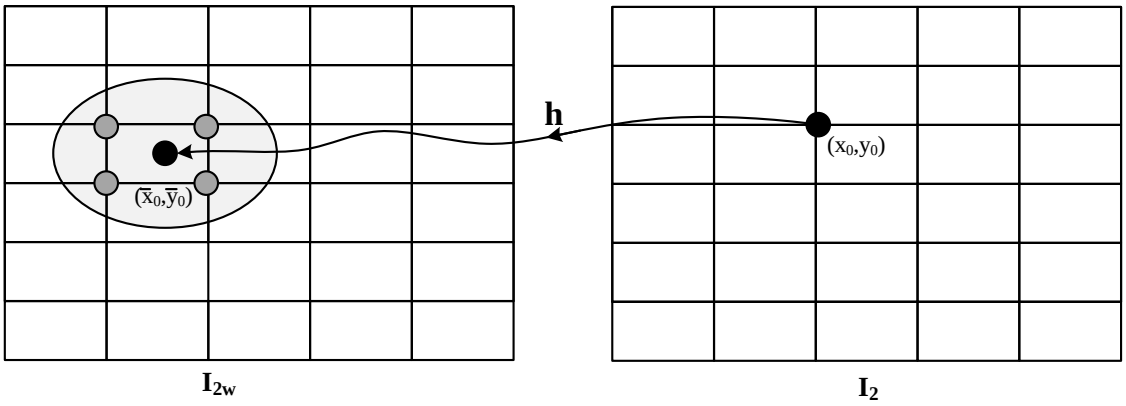


Figure 2.2: Backward image warping.

movement range in the input frames and adapt it to the filter size of the OF.

$$I_{2w}^{k} = I_{2}^{k} + \nabla I_2 \cdot h^{k} \tag{2.19}$$

Since forward warping results in holes in the destination image, the backward image warping transforms the coordinates of the destination image to the coordinates of the source image using the estimated flow as shown in Fig. 2.2. Although image holes are avoided during the transformation in backward warping, the transformed coordinates may be non-integers. The non-integer coordinates thus obtained are then corrected using interpolation techniques to get the nearest integer coordinates. The interpolation can be bilinear or bi-cubic, their performance differs especially when comparing the speed, sharpness of the interpolated image and how well they can preserve details and edges. In the bilinear interpolation, the non-integer coordinate is approximated to an integer coordinate based on a weighted average of 4 nearest neighbours.

It is having a better approximation value in comparison to nearest-neighbour interpolation, and the transition between intensity values is smoother. The design utilizes a simple 2×2 bilinear interpolation scheme which consumes lesser resources than bi-cubic interpolation for smoothing the interleaved pixels whose address is generated from the warped co-ordinates as given in equation (3.2) and (2.20).

$$\begin{aligned} I(x', y') = I(l, k)t_x t_y + I(l+1, k)(1 - t_x)t_y + \\ I(l, k+1)t_x(1 - t_y) + I(l+1, k+1)(1 - t_x)(1 - t_y) \end{aligned} \tag{2.20}$$

The warping utilize computed flow values to generate a motion compensated image at

every level, except the fine level. This process is repeated for each pyramid layer until the fine level is reached. The number of pyramid levels is dependent on the image resolution and expected motion range presented in the scene.

2.5 Implementation platforms

The software implementation of variational multi-scale OF algorithm on CPU cannot satisfy the real-time computation, size or power requirements due to the complex mathematical operations, iterative loops, high computation requirement and low available memory bandwidth. In recent years, many new alternatives including ASICs (Application Specific Integrated Circuits), FPGAs, GPUs, DSPs, MPPAs, appear to fill the gap between the conventional CPU and high-performance requirements.

Table 2.1: Performance of different available platforms.

Chara.[1] / Platform	Performance	Programmability	Cost	Power
CPU	Low	High	Low	High
ASIC	High	Low	Low	Low
FPGA	Medium	Low-Medium	Medium	Medium
GPGPU	High	High-Medium	Low	High
DSP	Medium	High-Medium	Low	Low

[1] Characteristics of the platform.
Source : Zhaoyi Wei [61]

Table 2.1 shows the comparison of the performance of the available platforms for implementing the variational multi-scale OF architecture. The flexibility of architecture comes at the cost of difficulty in programmability. The development of an algorithm in CPUs is easier but the serial nature of execution leads to low throughput. An ASIC implementation can handle complex and compute-intensive processing tasks, but the design is very complex and time consuming due to the timing, layout and routing in the hardware. Even though the ASICs can achieve maximum performance, its high Non-Recurring Engineering cost (NRE) and large design cycle lead to the development of alternatives like FPGAs, GPGPUs and DSPs to fill the gap between ASICs and CPUs.

2.5.1 GPGPU

With the release of Compute Unified Device Architecture (CUDA) by NVIDIA in 2007, it became a popular platform for implementing the compute-intensive tasks. The CUDA Application Program Interface (API) released by NVIDIA allows programming the CUDA-compatible GPGPUs in the custom CUDA C language. The CUDA architecture is based on a hierarchical structure with the top level having an array of Streaming Multiprocessors (SMs). For each streaming multiprocessor, there are multiple Scalar Processor (SP) cores, shared memory and other modules. The basic functional units are called threads which are grouped to form warps, which are executed in SP cores inside the multiprocessor. The GPGPU also have a hierarchical memory structure which includes global memory, constant memory, texture memory, local memory and shared memory. The local memory is owned by the SP core whereas the shared memory is accessible to all the SPs in a multiprocessor and all multiprocessors have access to the global memory. The design of a GPGPU algorithm with high performance needs critical memory management, as different memories have different latency.

2.5.2 FPGA

Nowadays, FPGAs with good cost and performance traits is found to be a better alternative for most of video and image processing applications [62]. The FPGA architectures improve the throughput by ordering the data in a hardware pipeline and processing them in parallel. An FPGA design is coded in Verilog/VHDL Register Transfer Logic (RTL) and synthesized to corresponding circuits. The place and route tool will map the hardware circuits onto the physical resources on the FPGA. The higher level languages lead to low quality synthesized codes than the manually written RTL. The basic elements of the FPGAs include Block RAMs (BRAM), Ultra RAMs (URAM), Look-Up Tables (LUTs), DSP48, registers, interconnection wires, etc. The FPGA designer needs to specify the behaviour of hardware circuitry, the interconnection between modules, timing and other details at each clock cycle. Hence FPGA design process consumes significantly more time than CPU implementations. A post place and route simulation of the entire system and the independent modules need to be performed separately before validating in the FPGA board. The design-verification cycle of FPGA is much shorter than that of ASICs. The configurable structure of FPGAs helps to design scalable architectures for multi-FPGAs systems. It provides an efficient solution for implementing large resource-consuming algorithms. The tradeoffs between FPGA and software implementation need to be analysed before mapping an al-

gorithm into FPGA. It includes the feasibility of using the fixed-point operations instead of floating-point operations, feasibility to fully pipelined the existing computer vision algorithms, presence of complex arithmetic operations like division, trigonometric functions and so on. With the floating point support available in the current FPGA, it is not still an efficient platform for implementing complex floating-point algorithms.

2.5.3 Others

Besides FPGAs and GPGPUs, some of the other popular hardware architectures are DSPs and Massively Parallel Processor Arrays (MPPAs). The DSP is a dedicated microprocessor with an optimal architecture for improving real-time signal processing performance. It usually supports optimized arithmetic operations like geometric transforms, Multiply-Accumulates (MACs) and even special memory addressing modes to improve efficiency. Hence the DSP with a microprocessor architecture is more suitable for embedded, less computationally intensive and lower power applications. An MPPA architecture is a single-chip with a massively parallel processing array with hundreds or thousands of processor cores with memories [63]. The array is based on a Reduced Instruction Set Computer (RISC) processor. These processor arrays can be programmed using software languages and are compiled with custom compilers. The complex algorithms are partitioned across different processor arrays and there processing and communication are controlled by the custom scheduler and dedicated bus interfaces. An Ambric [64] is an MPPA architecture implementing OF algorithm.

2.6 Applications of Variational Multi-scale OF algorithm

Even though the OF is an approximate projection of the true motion of the scene, it provides valuable information about the spatial arrangement of the viewed objects and the change rate. To extract information at a much finer granularity, OF becomes a necessity for a wide range of applications such as activity recognition [5] through improved tracking of limbs and balls in sports videos, video surveillance [6], vehicle speed estimation [7], analysing the variation of long term cloud patterns [9] and so on.

2.6.1 Video surveillance

One of the important application of OF is visual surveillance. The surveillance system contain various functional modules such as motion detection, depth estimation, segmentation

[65], object tracking[66], and object behavioural analysis [67]. The OF aids in effectively separating the foreground objects from the background, and identifying the moving objects [68]. This helps to detect and tracking of objects accurately across the image sequence.

2.6.2 Autonomous robot navigation

The behaviour of flying bees [69] have inspired the development of OF for achieving robot navigation. The autonomous navigation system helps the robots in determining the best suitable path or track between the start location and destination [70]. OF is widely used for obstacle detection and collision avoidance [71]. The OF helps to extract the information about the unknown environment and aids in determining the speed and direction at which the robot can navigate [72].

2.6.3 Metrological applications

Since the cloud has a significant role in controlling tropical circulation, the cloud analysis based on OF helps to study the interaction between cloud systems, modification of radiation by absorption and reflection. The cloud systems can move, grow and decay or could be embedded in larger systems such as a Tropical Cyclones (TC). A larger cloud system contains various hierarchy of cloud segments with different temporal and spatial scales. The smaller segments can move at different speed or direction as compared to the large cloud segment.

The tracking of cloud system interactions using near-real-time satellite data aids in now-casting [73] to predict high-intensity precipitation. The tracking of TC centres from satellite images aids in forecasting the cyclone track by integrating into a deep learning framework [74]. It helps in predicting disastrous winds and heavy landfall on coastal areas. Also, the analysis of cloud lifetime characteristics helps to study the variation of weather condition [75] over a given region for a specific period of time.

2.7 Evaluation benchmarks

The public datasets with ground truth allow researchers to compare their novel algorithms to existing work, and understand relative strengths and weaknesses. In [20], a quantitative comparison of nine classical OF algorithms is performed with five synthetic and quasi-synthetic testing sequences, but are too simple to check the performance of modern OF algorithms. The image database is mainly divided into two types, synthetic and real-world

images. It is relatively easy to find a precise ground truth for synthetic images, as these images are generated by computer graphics. Conversely, obtaining ground truths for images with real objects are more challenging. This section introduces effective benchmarks recently generated with modern techniques to evaluate OF algorithms throughout the research work.

(a) Frame 10 **(b)** Frame 11

(c) Frame 10 **(d)** Frame 11

Figure 2.3: Backyard and Grove sequence from the Middlebury database.

2.7.1 Middlebury dataset

The Middlebury dataset [76] is divided into four types of test images, each encompassing different challenge goals, such as non-rigid motion, real-world scenarios, and dense ground truths with sub-pixel accuracy. The dataset includes both synthetic and real-world images as shown in Fig. 2.3. The synthetic images have the ability to compute a precise ground truth, in addition to having control of the texture and the scenario of the scenes. The training contains images with less than 3% of the pixels have a displacement of over 20 pixels, and none goes over 25 pixels.

(a) (b)

Figure 2.4: Image sequence from MPI-Sintel database.

2.7.2 MPI-Sintel dataset

The MPI-Sintel dataset [77] is a long photo-realistic sequence with extremely difficult cases taken from an animated 3D short film as shown in Fig. 2.4, it contains large motion, specular reflections, motion blur, defocus blur, and atmospheric effects. The dataset is divided into two categories. The first is the training image category, which includes images with open-access ground truth. The second category is the test image category, with withheld ground truth. MPI-Sintel provides 1064 frames for training and 564 for testing. The frames were taken from 35 clips selected from the film. Images used in this dataset are rendered at different levels. The first level is *albedo* which is the simplest rendering which does not contain illumination effect and has a piecewise constant colour. This means that the OF constraint holds across the whole image. The second level is the *clean* rendering level which includes illumination effects (e.g. shading, specular reflections). The final level is the one that matches the final version of the film, which includes more complex effects and adds motion blur, atmospheric effect, colour correction, etc. The training set contains more than 17.5% of the pixels with motion over 20 pixels and approximately 10% over 40 pixels.

(a) (b)

Figure 2.5: Image sequence from KITTI database.

2.7.3 The KITTI dataset

The KITTI [78] is a dataset taken from a driving platform in an uncontrolled environment. It contains real sequences captured from challenging conditions like non-Lambertian sur-

faces, different illumination conditions and large motion. The database contains 50% density ground truth but offers a real-world benchmarking sequence rather than images taken in a controlled environment as shown in Fig. 2.5. The KITTI benchmark evaluation is based on a specially developed protocol. It uses Average Endpoint Error (AEE) threshold of τ pixels ($\tau \in$ (2,.. 5)), and computes the percentage of pixels above the threshold.

2.7.4 Satellite Images dataset

The database contains Thermal Infra-red (TIR) channel images from various Geostationary satellites like INSAT-3D and Kalpana-1 from Indian Space Research Organization, Meteosat-$5/7$ from European Space Agency and so on, as shown in the Table. 2.2.

Table 2.2: Satellite dataset with their characteristics.

Chara. / Satellite	Country	Location (deg)	Sensor	Launch	Channel (μm)	Spat.[1] (km)	Temp.[2] (min)	Radio.[3] (bits)
KALPANA-1	India	74 E	VHRR	2002	$10.5 - 12.5$	8×8	30	10
INSAT-3D	India	82 E	VHRR	2013	$10.3 - 11.3$	4×4	30	10
METEOSAT-7	Europe	57.5 E	MVIRI	1997	$10.5 - 12.5$	5×5	30	8
METEOSAT-5	Europe	0 E	MVIRI	1991	$10.5 - 12.5$	5×5	30	8
GMS	Japan	140 E	VISSR	1995	$10.5 - 12.5$	5×5	30	8
GEOS-8	US	75 W	VISSR	1994	$10.5 - 12.5$	4×4	15	10
GEOS-10	US	135 W	VISSR	1997	$10.5 - 12.5$	4×4	15	10

[1] Spatial resolution is represented in kilometres.
[2] Temporal resolution is denoted in minutes.
[3] Radiometric resolution is described in number of bits.

The Kalapana-1 and INSAT-$3D$ (Indian satellites) datasets are available from the operational website [79] the Meteosat-$5/7$ images are obtained from EUMETSAT [80] and the NCEP/CPC 4km Global (60N-60S) IR Dataset is accessible from NOAA climate prediction centre [81]. The Indian meteorological satellite Kalpana-1 was launched in the year 2002. It features a Very High-Resolution Scanning Radiometer (VHRR) for visible (VIS), thermal infrared (TIR) and water vapour (WV) band images, and a Data Relay Transponder (DRT) payload. The ground resolution of data is 2 km for VIS $(0.55\text{-}0.75\mu m)$ and 8 km

for TIR $(10.5\text{-}12.5\mu m)$ and WV $(5.7\text{-}7.1\mu m)$. It is a geostationary satellite with a temporal frequency of 30 min.

Table 2.3: Different TCs with their characteristics.

Chara. / Cyclone	Landfall	Lifetime	Max.Speed (km/h)	Min.Pressure (mbar)	Maturity
Hudhud	Visakhapatnam 12th Oct-0700 UTC	Oct:7→14, 2014	185	950	11th Oct-1200 UTC
Phet	Oman 3rd June-0200 UTC	May:31→Jun:7, 2010	155	964	2nd Jun-1200 UTC
Aila	Kolkata 25th May-0900 UTC	May:23→26, 2009	110	968	25th May-0600 UTC
Laila	Andhra Pradesh 20th May-1200 UTC	May:17→21, 2010	100	986	19th May-0000 UTC
Amara-Bruce	N.A	Dec:14→28, 2013	205	933	21st Dec-0600 UTC
Katrina	Florida 25th Aug-2230 UTC	Aug:23→31, 2005	280	902	28th Aug-1800 UTC
George	Port Hedland 8th Mar-1400 UTC	Feb:26→Mar:13, 2007	205	902	8th Mar-1200 UTC

2.7.5 Tropical cyclone dataset

Table. 2.3 shows the database of several cyclones including their characteristics like formation time, maximum speed and lifetime. The accuracy of the proposed cyclone tracking framework is tested with several TCs to name a few Aila, Hudhud, Laila and Phet that hit the Indian coastline formed during $2009\text{-}2014$ in the Northern Indian Ocean, the twin TC Amara and Bruce formed during $2013\text{-}2014$ in South-West Indian Ocean, the Atlantic hurricane Katrina formed during 2005 and so on.

2.7.6 Numerical Weather Model

The Numerical Weather Prediction (NWP) model provides the prediction of the climatic conditions for varying timescales with high accuracy for shorter predictions. The operational forecast by numerical models such as Global Forecast System (GFS) by the National Centers for Environmental Prediction (NCEP), Weather Research and Forecasting Model (WRF) supported by the National Center for Atmospheric Research (NCAR) Mesoscale

and Micro-scale Meteorology Division provides better representations of the environment and Tropical Cyclone (TC) structure. The Indian Meteorological Department (IMD) make use of WRF regional models for short-range operational forecasts. The WRF model has a horizontal resolution of 9km$\times 9$ km and has 38 vertical levels with the boundary conditions being updated every six hours. The GFS model data is archived in the website [82], whereas the WRF model data is accessible from [83]. The near real-time forecast is commonly used for the height assignment to the cloud tracer for accurate Atmospheric Motion Vector (AMV) computation [84]. The model solves a set of nonlinear partial differential equations about each grid point, does not perform any spatial analysis like segmentation or morphological operation on the model response to study cyclogenesis.

2.8 FPGA Platforms

The different FPGA platforms utilized for hardware emulation in this research work is briefly introduced in the Table 2.4. The selection of FPGA is an important factor when comparing different architectures.

Table 2.4: Characteristics of various utilized FPGA prototyping platforms.

Platform / **Chara.**[1]	**XUPV5**	**VC707**	**VC709**	**VCU118**
Device	Virtex-5	Virtex-7	Virtex-7	Virtex Ultrascale+
FPGA	XC5VLX110T	XC7VX485T	XC7VX690T	XCVU9P
Manufacturer	Digilent	Xilinx	Xilinx	Xilinx
Logic cells	17280	485760	693120	2586 (K)
DSP slices	64	2800	3600	6840
Internal Memory	5328 (Kb)	37080 (Kb)	52920 (Kb)	345.9 (Mb)
PCIe	Gen1x1	Gen2x8	Gen3x8	Gen3 x16
External Memory	256MB DDR2	1 GB DDR3	4 GB DDR3	4 GB DDR4

[1] Characteristics of the FPGA device.

2.9 Summary

After a comparative study between different existing methods, it can be stated that the variational multi-scale OF algorithm is best suitable for computing dense and accurate large displacement OF. But the real-time implementation of the variational multi-scale OF

algorithm is restricted due to the high algorithm complexity and sequential order of execution. Based on the comparative study across different implementation platforms, FPGA is found to be the best candidate for prototyping the variational multi-scale OF algorithm while meeting the requirements of embedded vision applications. The chapter also discussed the application of variational multi-scale OF algorithm along with other supporting computer vision algorithms to perform cloud/cyclone analysis and tracking on available benchmarking datasets.

Chapter 3

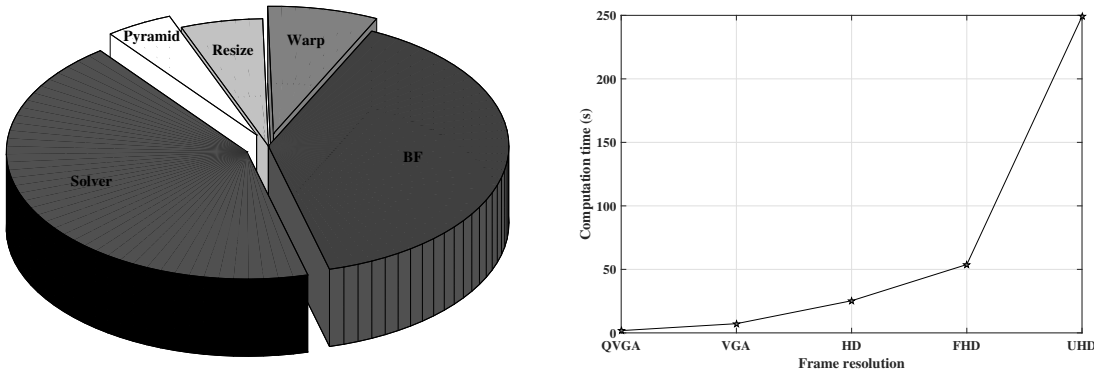
High Throughput Variational Multi-scale OF Architecture

This chapter deals with the design of a high throughput time sharing VLSI architecture for variational multi-scale OF algorithm to compute large pixel displacement in real-time by exploiting dedicated memory banking schemes.

3.1 Introduction

Dense and accurate computation of OF for fast moving objects is part of many embedded vision sensors. The sensor must be portable and need to meet the high computation requirement and low power consumption. It utilizes variational multi-scale OF algorithm discussed in the previous Chapter 2.4 to compute large displacement OF. An unoptimized version of the variational multi-scale OF algorithm with 4 pyramid levels, 40 SOR solver iteration and 1 flow refinement iteration modelled on Matlab 2018 running on a host PC with single-core Intel CPU $i5$-$M460$ operating at 2.53 GHz with 4 GB RAM takes 25.3 sec for computing dense and accurate OF from an HD image sequence.

The software profiling of the variational multi-scale OF algorithm identifies that 47% and 42% of the total processing time accounts for the SOR solver computation and BF denoising respectively as shown in the Fig. 3.1 (a). Also, the computation time of the variational multi-scale OF algorithm for different image resolutions is shown in Fig. 3.1 (b). This shows that a direct implementation of variational multi-scale OF algorithm for high-resolution images in real-time is limited due to a large number of arithmetic operations involved in the computation of every flow value, the presence of various feedback loops corresponding to the number of pyramid levels, flow refinements and solver iterations, the high memory bandwidth required for buffering and retrieving intermediate flow



(a) **(b)**

Figure 3.1: Block diagram of (a) performance results and (b) computation cost for different image resolutions.

vectors and the huge amount of storage need for buffering the image pyramids as well as the intermediate flow vectors.

Hence this work identifies the bottlenecks of the variational multi-scale OF algorithm and devises a high throughput VLSI architecture while reducing the area utilization and storage. This is achieved by making various hardware adaptations to the variational multi-scale OF algorithm which involves the restriction of the complex and resource-intensive flow refinement loop to a single iteration without much loss in accuracy, utilizing a Jacobi solver whose current pixel value depends only on the neighbourhood pixels of the previous iteration instead of a Successive Over Relaxation (SOR) solver, eliminating some of the complex repetitive arithmetic logic present in each solver iterations to a single arithmetic logic before the solver stage to reduce the resource utilization. The work proposes a variable fixed point time-sharing architecture for the adapted algorithm to reduce resource utilization, utilize parallel and unfolding architectures of solver, gradient, denoising and interpolation modules to maximize the throughput. It also introduces three different memory banking schemes with customized access pattern for the pyramid, warping and flow resizing stage to improve the system throughput while minimizing the storage requirement.

3.2 Related work

This section discusses the existing hardware implementation of the original and multi-scale architectures of variational and non-variational methods along with their characteristics and limitations. Diaz et.al [85] proposes a superpipelined and fully parallelized architecture of

Lucas Kanade (LK) OF on a Xilinx Virtex 2 device. It utilizes 70 pipeline stages to process 170 fps at a resolution of 800×600 pixels. [41] propose an efficient VLSI architecture LK OF. The architecture is based on a fixed-point version utilizing an optimal bit width without much loss in accuracy. It can process 640×480 frames at 30 fps on Xilinx Virtex 2 Pro FPGA device. In the work [86], a novel customizable architecture based on multichannel gradient model is proposed. Based on inspiration from the properties of cortical motion pathway, complex bioinspired real-time systems with high computational complexity is introduced. It has a higher complexity than other local differential methods. It is appropriate for scenarios containing varying luminance, noisy environments. The implementation of the architecture on Virtex $2000E$ FPGA can process 16 frames of resolution 128×96 in a second.

Tomasi et al. [87] proposed a phase-based OF in FPGA device as the phase-information provide more stability as well as sub-pixel accuracy. The architecture is extended to implement a multi-resolution and multi-orientation to enhance the accuracy and coverage over a wide range of detected velocities. The implementation utilizes 1750 parallel processing units for the single scale OF and consumes more than 2000 basic elements for multi-scale implementation. The multi-scale architecture can process VGA frames up to 31 fps with 4 scales whereas the original phase-based approach can compute OF for up to 81 fps. Barranco et al [88] presented an efficient architecture for LK approach along with its multi-scale implementation on a Virtex 4 FPGA device. The LK implementation is used as a valid alternative when using high frame-rate cameras is able to achieve a throughput of 270 frames per second. Instead, a more accurate multi-scale version is able to capture 32 frames per second for a frame resolution of 640×480. The LK implementation corresponds to 10%-15% of the available resources while the multi-scale takes about 60%. The warping stage consumes 23% of the total resources and reduce the maximum working frequency of 83 MHz for the LK core to 44 MHz for the multi-scale system.

The work [89] introduces an energy-efficient time-sharing pyramid pipeline architecture for multi-resolution LK. The work compares segment and linear pyramid architecture, in which segment pyramid [90] architecture utilizes a single Processing Element (PE) in a sequential manner to compute OF at every pyramid levels. Instead, a linear pyramid [91] uses as many processing elements as the number of pyramid levels to operate simultaneously and the intermediate data is buffered internally at each pyramid level to improve the throughput. The linear pyramid architecture is inefficient due to the unbalanced workload in different pyramid levels and has high resource utilization while a segment pipeline takes large computation time. Hence time-sharing pyramid pipeline achieves about 50% of hard-

ware savings in terms of area and power consumption compared to the traditional linear pyramid implementation. Also, the time-sharing pyramid efficiently reduces the off-chip memory traffic by re-organizing the data storage and processing order of an image pyramid. Detailed analysis on the number of scales, frame resolutions and frame rate is missing in the work.

An efficient hardware implementation of the variational HS algorithm is presented in [92]. It achieves a throughput of 175 MPixels/s and process Full HD (1920×1080) frames at 60 fps on Virtex 7 FPGA device. The fixed-point architecture achieves a performance of 418 GOPS with power efficiency of 34 GOPS/W whereas floating-point module achieves 103 GFLOPS, with power efficiency of 24 GFLOPS/W. The proposed module does not require external RAM memory in order to store intermediate flow vectors between the iterations. It utilizes a separate hardware submodule for each iteration leading to large resource usage.

Syed et.al [93] proposed a hierarchical block matching (HBM) based OF algorithm for real-time hardware implementation. It utilizes block matching to generate initial optical flow and refines the flow vectors at each level using local smoothness constraints. The proposed system utilizes a two-dimensional reconfigurable systolic array for real-time implementation of the Sum of Absolute Difference (SAD) based block matching algorithm. The architecture is capable of computing OF with half-pixel precision utilizing a block size of 4×4. It can process 640×480 resolution frames at 39 fps on Virtex 7 FPGA device. In [94], a dense multi-scale implementation of LK is implemented on Xilinx Zynq Ultrascale+ FPGA device. The design can compute OF of UHD (3840×2160) resolution with 60 frames per second.

The existing literature lacks the design of high throughput and low power variational multi-scale architecture for computing dense and accurate OF for fast moving objects. This serves as the major motivation behind the adaptation of multi-scale variational OF algorithm to implement a dedicated high throughput time-sharing architecture with efficient memory banking schemes.

3.3 Proposed algorithm adaptation

This section discusses various hardware adaptations present in variational multi-scale OF algorithm along with pseudo code in the Algorithm 3.1. The OF computation starts from the coarsest layer of the pyramid, where the problem is usually easier to be solved. For each pyramid level L, the algorithm sub-samples the input image by a factor η and apply

Gaussian smoothing to create a stack of downsampled images given in line 4 of the Algorithm 3.1. The flow values are initialized to zero at the starting of pyramid operation. The second iterative loop corresponds to the flow refinement present in each pyramid level which involves warping, gradient computation, solver, flow merging and filtering stages. The second image is warped with initial flow values to compensate for Taylor approximation of the solver equations 2.17, except at the fine level. The spatiotemporal gradient is implemented based on the work [95].

Algorithm 3.1: Non-linear Multi-scale variational OF

1 **Multiscale_OF** (inputs (I_1, I_2,), levels (L), warping (N_{warp}) & solver iterations (N_{solv}))
2 **for** $k \leftarrow L$ *to* 1 `// Pyramid Iteration Step`
3 **do**
4 $I_1(I_2) = Downsamp(\eta \cdot I_1(I_2))$ `// Image Decimation`
5 Initialize $h^L \leftarrow 0$
6 **for** $i \leftarrow 1$ *to* N_{warp} `// Flow Refinement Step`
7 **do**
8 $I^k_{2w} = I^k_2 + \nabla I_2 \cdot h^k$ `// Image Warping Step`
9 $I^k_z = (I^k_{2w} - I^k_1)$ `// Gradient Compute`
10 $I^k_x(I^k_y) = [I^k_{2wx}(I^k_{2wy}) + I^k_{1x}(I^k_{1y})]/2$
11 $a = 1/((I^k_x)^2 + \alpha^2)$ `// Compute Reduction`
12 $b = I^k_x I^k_y$
13 $d = 1/((I^k_y)^2 + \alpha^2)$
14 Initialize $du^k, dv^k \leftarrow 0$
15 **for** $l \leftarrow 1$ ***to*** N_{solv} `// Solver Iteration`
16 **do**
17 $c = \alpha^2(\overline{u}^{k,l} - u^{k,l}) - I^k_x \cdot I^k_z$
18 $e = \alpha^2(\overline{v}^{k,l} - v^{k,l}) - I^k_y \cdot I^k_z$
19 Compute Flow Increment:
20 $du^{k,l} = a(c + \alpha^2 \cdot \overline{du}^{k,l} - b \cdot dv^{k,l})$
21 $dv^{k,l} = d(e + \alpha^2 \cdot \overline{dv}^{k,l} - b \cdot du^{k,l})$
22 **end**
23 $h^k = h^k + < du^k, dv^k >$ `// Flow Merging`
24 $MedFilt(h^k)$ `// Flow Filtering Step`
25 **end**
26 **if** *k > 1* **then**
27 $h^k = Upsamp(\frac{h^{(k-1)}}{\eta})$ `// Flow Resizing`
28 **end**
29 **end**

The flow refinement loop is the major time-consuming part of the software implementation as it involves complex operations like warping, solver, flow merging and so on. The parallel implementation of the flow refinement loop scales the resource utilization by a factor of $O(N_{warp} \times N_{solv})$. Also, the serial data dependency restricts parallel implementation. Hence the first hardware adaptation provides a direct inclusion of the flow increments at every pyramid level which is same as restricting the flow refinement loop (given in the lines 5-22 of the Algorithm 3.1) to a single iteration (N_{warp}=1). This avoids the use of large data caches and eliminates the multi-cycle execution, resulting in improved throughput performance. This leads to a slight reduction in the convergence rate, which is compensated by increasing the number of solver iterations. Another hardware adaptation is related to the introduction of a compute reduction stage. This helps to move many of the redundant compute-intensive operations in the solver iterations to before the solver stage as given in line number 11 to 13 in the Algorithm 3.1. The computed results are then forwarded to all the solver iterations using suitable line buffers, thus effectively reducing the number of hardware operations involved in the solver stage.

The third adaptation is replacement of the SOR solver architecture with a hardware-friendly Jacobi solver, as the SOR solver provides faster convergence at the cost of increased processing time and memory requirement. Since the computation of the current iteration in the Jacobi solver depends only on the previous iteration values, it leads to a simple and regular design with low cache requirement trading off the increased number of solver iterations. Whereas the SOR solver nearly requires only half the number of iterations as compared to Jacobi solver for achieving the same accuracy [96]. Hence the flow computation stage evaluates the line numbers 20-21 in the Algorithm 3.1 using Jacobi solvers. The computed flow increment at each level is merged with the previous flow and filtered to remove the random flow values. The flow field is upsampled and scaled by a factor of η to compute next fine level flow. This process is repeated for each pyramid layer until the fine level is reached. The number of pyramid levels is dependent on the image resolution and expected motion range presented in the scene.

3.4 Proposed time-sharing architecture

The work proposes a time-sharing architecture combining the advantages of the segment and linear pyramid approaches for computing variational multi-scale OF. In the Linear Pyramid (LP) architecture, for each pyramid level there exists a dedicated Processing element (PE) for flow computation. The basic stages of the PE include pre-processing, warp-

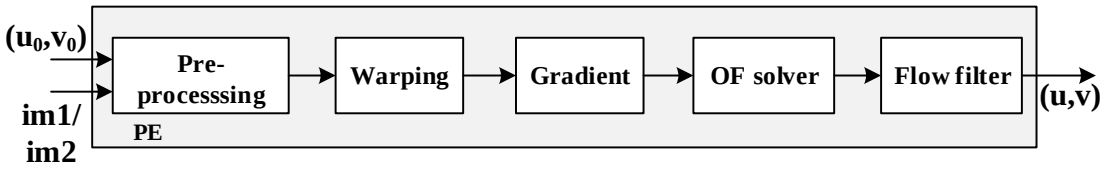


Figure 3.2: Processing element with different modules.

ing, gradient computation, OF solver and flow filter as shown in Fig. 3.2.

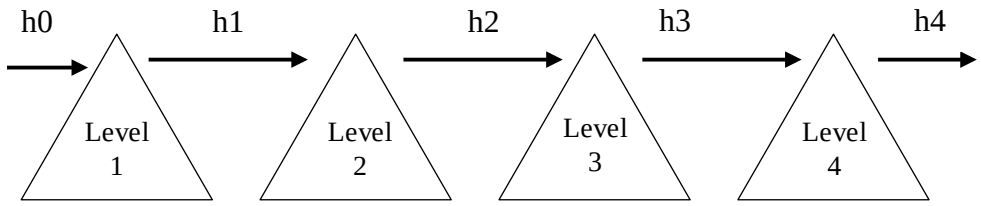


Figure 3.3: Linear pyramid architecture for 4 pyramid level.

A 4-level LP architecture has 4 different PEs to estimate flow. From the Fig. 3.3, it can be observed that the level-1 PE computes flow at pyramid level-1 using the initial flow vector h0=0, level-2 PE computes flow at pyramid level-2 using the previously computed flow h1 and so on.

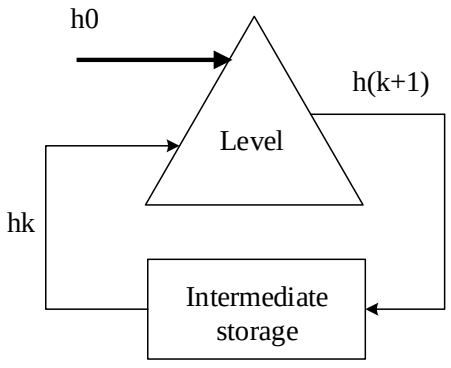


Figure 3.4: Segment pyramid architecture for 4 pyramid level.

Here, the input image is read from external memory, the intermediate flow values (h1, h2 and h3) are buffered internally and the final computed flow h4 is sent to the external

memory. The LP architecture is inefficient due to the unequal number of computations in different pyramid levels and has high resource utilization. In contrary to LP architecture, in the Segment Pyramid (SP) architecture, a single set of PE is shared among each pyramid level for flow computation. So for the multi-scale variational method with 4 pyramid level have a single set of PE, used to compute flow in all the levels as shown in Fig. 3.4. Since the PEs are shared among the pyramid levels, the computation of flow at every pyramid level begins only after the completion of computation in the previous level. This leads to the storage of intermediate flow vectors into the external memory until the beginning of the next computation level, which results in high computational cost due to the large amount of slow external memory accesses.

Since the PE are not shared among the pyramid levels in case of LP architecture, it can be fully pipelined to maximize the performance. Hence the LP architecture involves less processing time as compared to the SP architecture. Also, the power dissipation in LP architecture is high as compared to SP, as every component is operational at any instant of time due to fully pipelined operation. Hence LP architecture is better in terms of computational cost and SP architecture is better in terms of power dissipation and resource utilization.

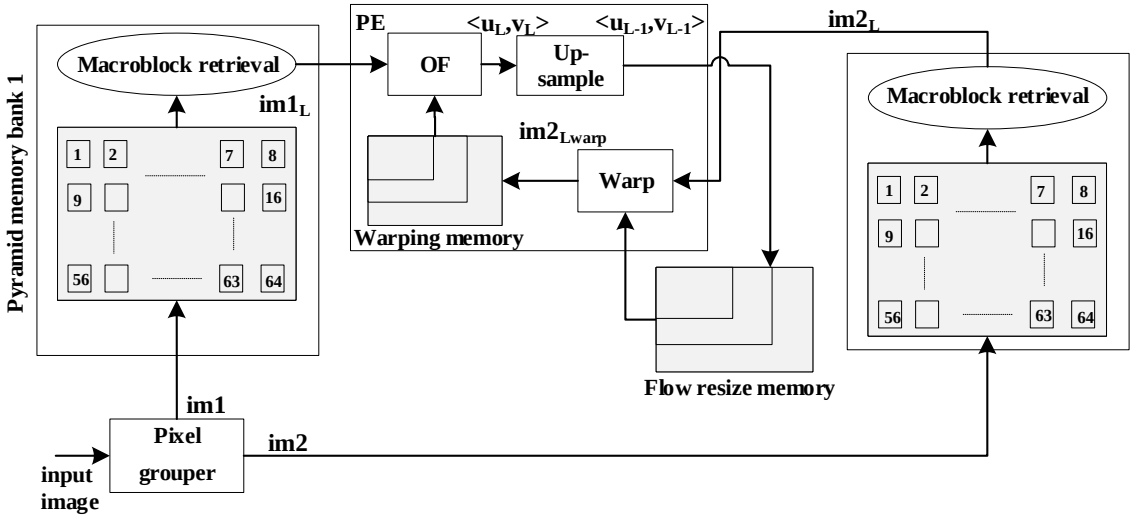


Figure 3.5: Proposed time sharing architecture.

The proposed work integrates the advantages of LP and SP architectures to introduce a time-sharing approach as shown in Fig. 3.5. It takes advantage of SP architecture with single resource sharing PE performing the computation of different image pyramids to minimize resource utilization with a negligible time overhead. Also, it achieves parallelism in

the pyramid construction and PE computation for different pyramid levels as in the case LP architecture. This is made possible with the design of dedicated pyramid memory banking scheme consisting of a single large memory composed of various dual port memories. A scheduler and pixel grouper are in charge of generating downsample images according to the pyramid level. The simultaneous availability of downsampled images helps to parallelize the execution of PE in the different pyramid levels. The modules in PE's are also computed in a parallel fashion. The architecture details of the pyramid generation are explained in the below Section 3.5.1. The proposed architecture further improves the memory bandwidth by buffering the intermediate flow vectors instead of external memory. The details of the parallel execution of the modules and PE's are discussed in the below Section 3.6.5.

3.5 Proposed Hardware Architecture

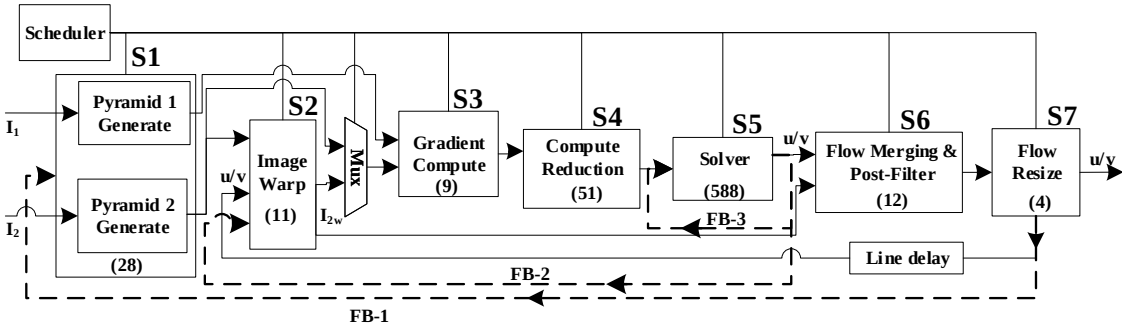


Figure 3.6: Block diagram of variational multi-scale OF architecture.

The block diagram of the proposed Time Sharing Multi-scale variational OF (TSMOF) architecture with all subsystems is shown in Fig. 3.6. The design is based on stream processing, hence the operation on a single pixel depends on the neighbouring pixels which are cached using line buffers. It consists of six modules including pyramid creation, image warping, gradient computation, solver, flow resize and scheduler module. The pyramid creation module uses a single large memory and simultaneously downsamples input image to each pyramid level, the warping module transforms the second image at the current pyramid level using the estimated flow at the previous level, the Gradient module computes the spatial and temporal gradients of the input images. In the coarsest level, image warping is not performed as the initial flow value is assumed to be zero. Hence a multiplexer is utilized by the Gradient module to choose between the downsampled or warped image.

Further, the Solver module solves the system of equations using Jacobi solver and the Flow resize module upsamples the estimated flow at the current level to the next finer level based on the nearest neighbour interpolation scheme. Since the architecture is time-sharing, the scheduler logic plays an important role in sharing data appropriately among these modules at different pyramid levels. The scheduler logic is formed of several counters and conditional statements for coordinating the time-sharing control signals among all pyramid levels. The number of clock cycles required for each stage and level is precomputed and stored in registers for controlling start and stop of each module. It utilizes multiple counters and multiplexers to control the start and end of each pyramid level and the modules in it. For each pyramid level, the scheduler also controls the storage of the incoming pixels into a memory bank based on bank selection and memory address.

The block diagram contains three Feed Back (FB) loops and the count on each of the box corresponds to the number of pipeline stages. The FB-1 is based on the number of pyramid levels L, FB-2 represents the flow refinement operation and FB-3 implements the solver iterations. The design considers hardware optimization like deep data path pipelining and unfolding of the solver iteration, data parallelism in gradient, filtering and interpolation modules to maximize the throughput. The pipelining helps the designer to divide a complex task into smaller and more manageable pieces and makes the design occupied for all the cycle. The fine and coarse-grained pipelines are considered for maximizing throughput performance.

3.5.1 Pyramid Generation Stage: S1

Pyramid generation stage parallelizes the image stack creation, by down-sampling fine level images I_1^1 (I_2^1) to coarse level images I_1^{1+k} (I_2^{1+k}) based on η as shown in Fig. 3.7. The computation of the coarse level I_1^L (I_2^L) starts either after $W_1 \times H_1 - W_L \times H_L$ cycles or after internally caching macro-block of size $P_1 \times Q_1$ pixels of the fine level. A pixel grouper logic generates the address of memory bank to store the incoming pixel stream in the macro-block. It provides a two-way pixel grouping, i.e. memory bank selection followed by memory element identification.

The macro-block retriever block enables simultaneous access to multiple pixels in the pyramid memory bank to compute each pixel in the coarse level I_1^L (I_2^L) by averaging the macro-block of size $P_1 \times Q_1$ of the fine level I_1^1 (I_2^1). An anti-aliasing filter based on 3×3 Gaussian mask is applied before the downsampling to suppress the high-frequency components to improve the accuracy of the first order image derivatives. The details of

the Gaussian filter implementation is discussed in the previous work [97]. The pyramid controller co-ordinates the pixel generation by identifying the pyramid level and generating control signals to read and write macro-blocks from the pyramid memory bank.

The pyramid generation stage is the first step in the variational multi-scale OF architecture to down-sample the fine level images I_1^1 (I_2^1) to coarse level images I_1^{1+k} (I_2^{1+k}) based on η as shown in Fig. 3.7. It utilizes a pixel grouper logic to generate the address of memory bank to store the incoming pixel stream in the macro-block. It provides a two-way pixel grouping, i.e. memory bank selection followed by memory element identification. The computation of the coarse level I_1^L (I_2^L) starts after internally caching macro-block of size $P_1 \times Q_1$ pixels of the fine level. The macro-block retriever block enables simultaneous access to multiple pixels in the pyramid memory bank to compute each pixel in the coarse level I_1^L (I_2^L). It is achieved by downsampling the macro-block of size $P_1 \times Q_1$ of the fine level I_1^1 (I_2^1). The downsampling is preceded by an antialiasing filter based on 3×3 Gaussian low pass kernel to suppress the high-frequency components [98]. The scheduler coordinates the pixel generation by identifying the pyramid level and generating control signals to read and write macro-blocks from the pyramid memory bank.

Consider a 4 stage pyramid with a downsampling factor of η =0.5, a single pixel generation at the coarsest level pyramid (level-4) needs at least 8×8 pixels (η =0.125) from the finest level (level-1). For instance, with a 8×8 macro-block the simultaneous access to 64 pixels is achieved by storing these 64 pixels in an interleaved pattern into 64 different internal dual port RAMs such that every pixel of the original image is written only once to one of these memories. Therefore each memory will be storing only $\frac{1}{64}^{th}$ of the entire image. Thus the entire image is stored efficiently into 64 RAMs to achieve simultaneous read from different locations. Similarly, for the creation of a single pixel in the pyramid level-3, a patch of 4×4 pixels is needed which in turn requires 16 pixels from the finest level.

Pyramid Memory Bank

A conventional 4 level pyramid architecture, either utilizes separate memory elements to store images at different pyramid levels ($W_1 \times H_1$ + $W_2 \times H_2$ + $W_3 \times H_3$ +$W_4 \times H_4$) or buffers the entire image into a single large memory ($W_1 \times H_1$) and uses separate line buffers for each pyramid levels leading to large memory consumption. Instead, this work devises a dedicated pyramid memory bank and access scheme to store the incoming pixel stream without pixel duplication while allowing simultaneous pixel transactions using a pixel grouper. The dedicated pyramid memory banking scheme uses P_1 memory banks

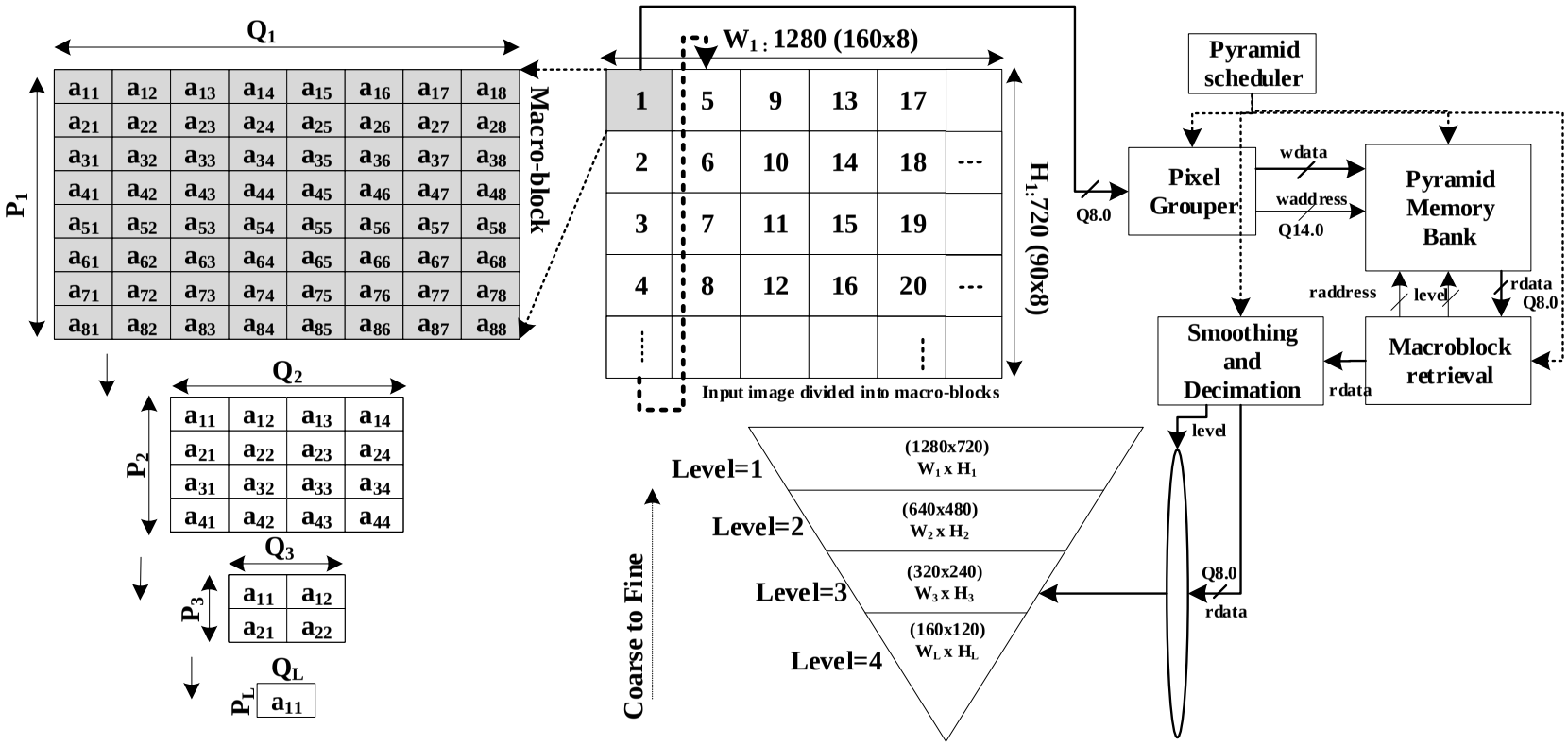


Figure 3.7: Pyramid Generation Stage: The input image organized into macro-blocks at each of the pyramid level.

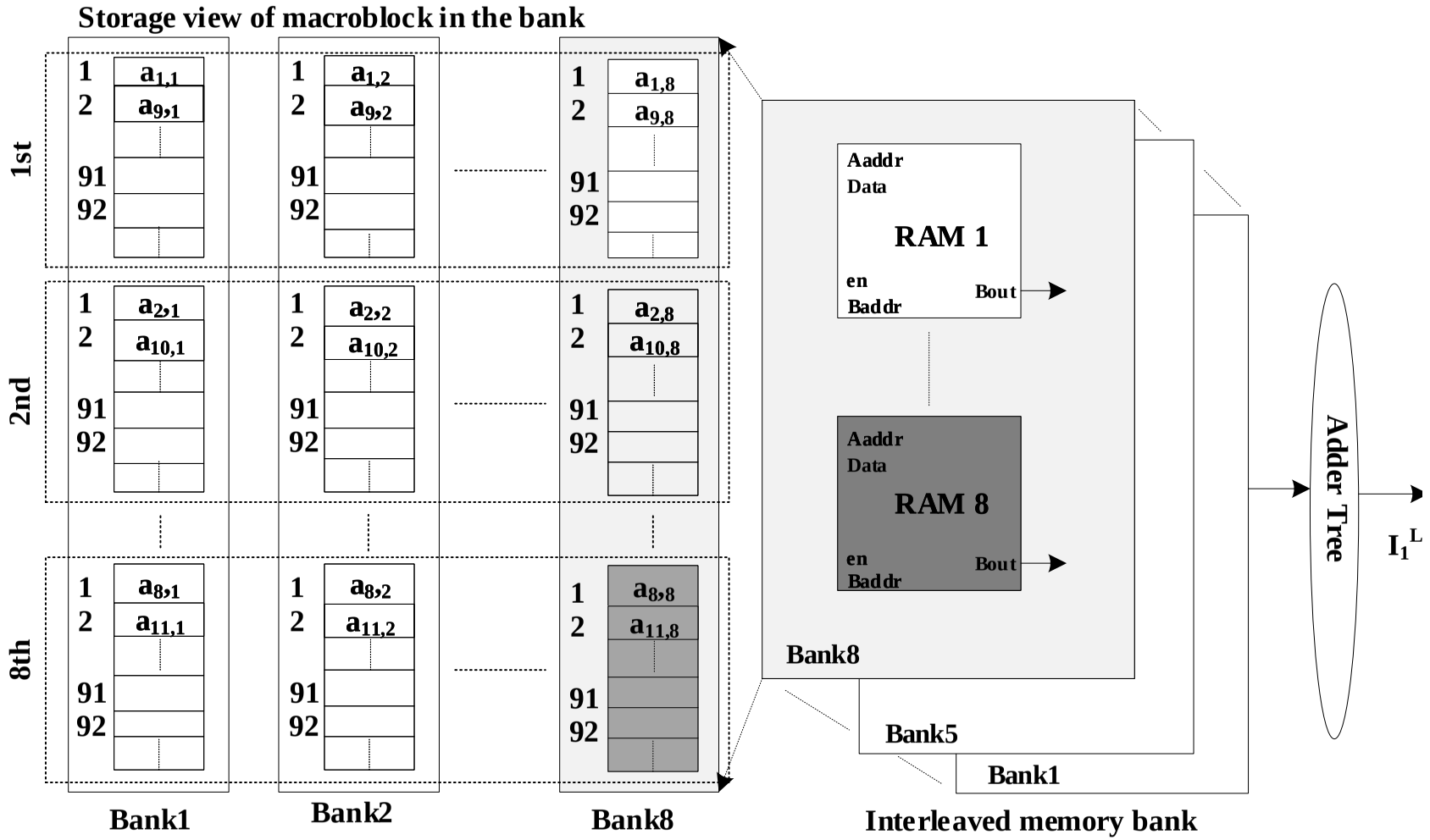


Figure 3.8: Pyramid generation stage: The macro-blocks are stored into pyramid memory bank using a pixel grouper, with all consecutive pixels in a macro-block fall into different memories in the bank.

with Q_1 dual-port RAM allowing simultaneous pixel read, without any pixel duplication. So instead of a typical pyramid implementation, the proposed banking scheme only utilize $W_1 \times H_1$ memory.

A special access pattern provides simultaneous access to the whole macro-blocks in the images. It is achieved by storing $P_1 \times Q_1$ pixels into different internal dual port RAMs in a specific pattern as illustrated in Fig. 3.8. This is done with the help of pixel grouper which selects the memory address to store the incoming stream of pixels of the original image. The pixels in the first image column is stored into the first memory bank, second column pixels into the second memory bank and so on up to the eighth image column. This pattern of storing the pixels is repeated for rest of the columns, i.e. pixels in the 9th column is stored again in the first memory bank, 10th column to the second memory bank and so on. Similarly, other memories banks are filled, with a period of 8. The 8 consecutive pixels in a column are written into 8 different memory locations in the same bank with a similar address.

For instance, the pixel $a_{1,1}$ (macroblock-1) is written to the first location of the first memory present in the bank 1, second pixel to the first location of the second memory in the same bank and so on till the 8th pixel. Further, the pixel $a_{9,1}$ (macroblock-2) is written into the second location of the first memory in the bank 1, $a_{10,1}$ pixel to the second location of the second memory in the bank 1, which continues till the end of the column. Since the image pixels are written in a specific pattern, the address of memory location has to be generated accordingly i.e. each pair of row and column indices are converted to a one-dimensional index. Thus the pyramid memory bank stores the two incoming frames I_1 and I_2 while achieving a memory saving of more than 24% in case of a non-banking scheme.

3.5.2 Image Warping Stage: S2

The internal architecture of the image warping stage is shown in Fig. 3.9. The two major challenges limiting the high throughput architecture for warping stage are the large memory consumption and the high computational delay. The large memory consumption is optimized by utilizing a warping memory bank discussed in Section 3.5.2 to improve the throughput by providing parallel data access. The scheduler coordinates the interleaved memory accesses. The image warping transforms the coordinates of the destination image to the coordinates of the source image using the computed flow. This avoids the image holes during the transformation but leads to the generation of non-integer co-ordinates or

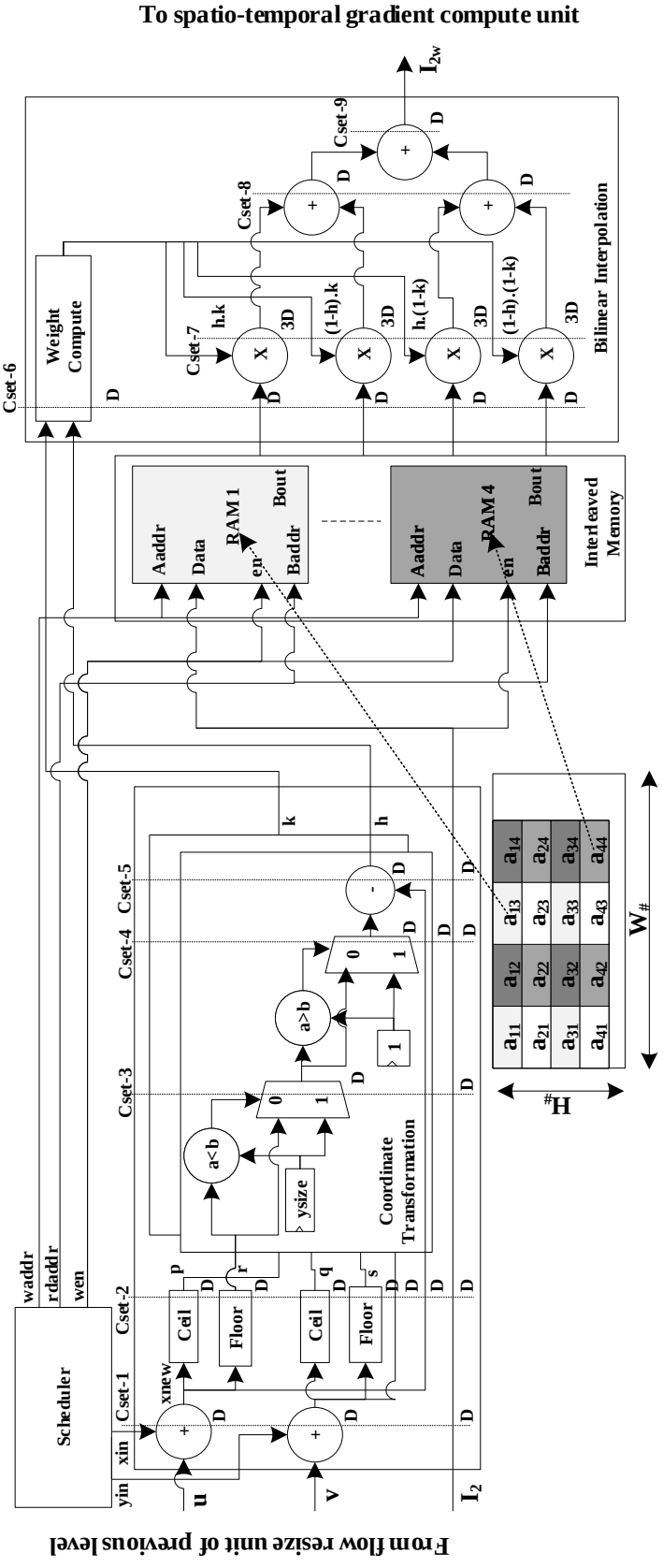


Figure 3.9: Image warping stage: After co-ordinate transformation with computed flow, the second image frame is stored in warping memory bank in an interleaved fashion which is interpolated to get the warped image.

addresses given by equation (3.1).

$$(l, k) = (x + u, y + v) \tag{3.1}$$

The non-integer addresses are discretized using bilinear interpolation techniques to get the nearest integer addresses as given in equation (3.2).

$$(t_x, t_y) = (l - \lfloor l \rfloor, k - \lfloor k \rfloor) \tag{3.2}$$

For a high throughput implementation, the bilinear interpolation needs to access four neighbouring pixels simultaneously from random locations in the image. This lead to the unwanted waiting delay (4 cycle latency) associated with reading the image from random locations. A conventional method for pixel interpolation uses two methods a) storing the entire image pixels of the second image in 4 different memories and access 4 neighbouring pixels for bilinear interpolation from these memories b)run the memory access at 4 times faster than rest of modules. However, in the first method, the image has to be written into 4 memories making the architecture less efficient in terms of memory usage. In the second method, the interpolation has to run 4 times faster than rest of the system, thus leading to an overall reduction in the system frequency by 4 times which significantly impact the performance of the architecture.

To overcome these limitations, an efficient pixel interpolation architecture is developed. It can be observed that all the pixels in an image can be characterized into 4 different groups such that every adjacent pixel to a pixel falls into different groups. Hence in the proposed architecture, the second image frame is written into 4 different dual port RAMs in the above-interleaved pattern such that the neighbouring pixels of any non-integer coordinate can be assessed simultaneously from these 4 dual port RAMs. So in summary, each of the 4 dual port RAM is storing only $\frac{1}{4}^{th}$ of the entire image. This method reduces the memory usage by 4 times by writing a pixel into only a single memory without creating any duplicates. Moreover, the computational frequency of the system does not reduce as the system is running at a single rate. Thus the warping memory consumes $W_1 \times H_1$ memory to buffer the second frame thereby minimizing the memory utilization to $1/4th$ of the conventional design.

The second major challenge in the warping architecture is associated with the computation delay involved in the warping stage. In a direct implementation, the interpolation operation is started once the entire image is available in the internal memory. But this approach has the disadvantage of waiting to start interpolation until the entire image is written

into the memory. For instance, if the resolution of the image is $W_1 \times H_1$, then the waiting delay involved for interpolation is $W_1 \times H_1$ clock cycles which significantly reduces the performance of the system in terms of computation time. In the proposed work the buffering latency associated with each pyramid level is overcome by providing two novel memory access mechanisms. First is to fix the maximum possible pixel displacement as D based on which the number of pyramid levels L is decided. For a displacement of D, L is calculated as L = (1 + $log2D$). Hence only $D \times H_{L-1}$ pixels need to be written into the interleaved memory before the start of the warping stage. This modification incorporates the buffering latency associated with each pyramid level. Hence the interpolation can be started once $10 \times W_1$ (assuming D=10) pixels are written into the warping memory, thereby reducing the waiting delay from $W_1 \times H_1$ clock cycles to $10 \times W_1$ clock cycles.

The second modification hides the latency for caching $D \times H_{L-1}$ pixels, by writing I_2^{L-1} pixels of the pyramid into the interleaved memory much before the start of the creation of pyramid I_2^L. This is possible, as port B of the dual port RAM remains idle during I_2^L creation. Since $D \times H_{L-1}$ pixels of I_2^{L-1} is written into the warping memory, the I_2^{L-1} warping can start immediately after I_2^L completion. The remaining pixels of pyramid I_2^{L-1} is written using port A, without affecting the simultaneous pixel read from port B. To elaborate, the pyramid 4 creation begins only when $\frac{63}{64}^{th}$ portion of the image has arrived. Till that time, the second port of the dual port RAM used for warping remains idle. Making use of this fact, the pyramid 3 creation can be started when the required pixels for generating $D \times W_1$ patch in the pyramid 3 have arrived. So before completing the pyramid 4 level computation, the $10 \times W_1$ neighbourhood pixels needed for pyramid 3 operations are written into the warping memory. Hence, after the completion of pyramid 4 computation, the system need not wait to start reading the pyramid 3 pixels from the memory. Subsequently, the remaining pixels of pyramid 3 is written into the warping memory using port A and reading of pixels continues using port B simultaneously. Thus the unwanted waiting delay for reading the image from the warping memory can be avoided, improving the performance of the system.

Warping Memory Bank

A conventional image warping operation can only be started once the entire image is buffered internally or involves large overhead in case of external memory access [89]. Also, the bilinear interpolation in the warping stage needs 4 simultaneous pixel accesses. The proposed warping memory bank scheme utilizes 4 internal dual port RAMs each of size $0.25 \times (W_1 \times H_1)$. In order to achieve simultaneous access of 4 pixels, the image pixels

are stored in a special pattern based on the pixel indices such that every adjacent pixel falls into different RAMs. The address for writing the flow into the memory is generated by using the row and column indices at each of the pyramid levels. The pixels with both row and column indices as an odd number are stored in RAM 1, the pixels with odd row index and even column index in RAM 2, the pixels with even row index and odd column index in RAM 3 and those with both even indices are stored in RAM 4. This scheme avoids creating multiple copies of the pixel data while providing simultaneous pixel access.

3.5.3 Gradient Compute Stage: S3

The gradient stage computes the spatio-temporal gradient based on the work [95]. The spatial gradient is computed as the average of warped second image gradient and first image gradient, where the individual gradient is derived using a 5-point derivative filter $\frac{1}{12}[-1\ 8\ 0\ -8\ 1]$ as given in equation (3.3). The temporal derivative is computed by subtracting the first image from the warped second image.

$$I_x^k = [I_{2wx}^k + I_{1x}^k]/2 \tag{3.3}$$

$$I_y^k = [I_{2wy}^k + I_{1y}^k]/2 I_t^k = (I_{2w}^k - I_1^k) \tag{3.4}$$

The 5 point gradient filter computation in X-direction utilize 4 line buffers of size $W_{\#}$ (W_1, W_2, W_3 or W_4 depending on image width at each pyramid level) to convert one-dimensional pixel stream into a two-dimensional patch. The streams are multiplied with the saved coefficients, accumulated and normalized to compute the 5-point image derivative which is averaged with the warped second image derivative to get resultant gradient in X direction.

A direct implementation of the gradient computation for all 4 pyramid levels needs $4 \times (W_1 + W_2 + W_3 + W_4)$ line buffers, where W_1, W_2, W_3, W_4 represents the width of image size at every pyramid levels. The number of delay caches utilized for line buffers is optimized by sharing the resources across parallel data-paths. This is implemented by proposing a line buffer sharing approach containing 10 line buffers, of which four corresponds to W_1 and two for W_3, W_2 and W_1 levels as shown in Fig. 3.10. The channel selection signal ($sel\#$) is controlled by the scheduler to select the incoming channels to datapath multiplexer based on the pyramid level. For level $L4$, the channels 1, 2, 3, 4, and 5 are selected similarly other details regarding the channel selection are described in the Fig. 3.10. Similar is the case for y direction since there are three independent circuits for computing spatiotemporal gradients which helps to operate in parallel.

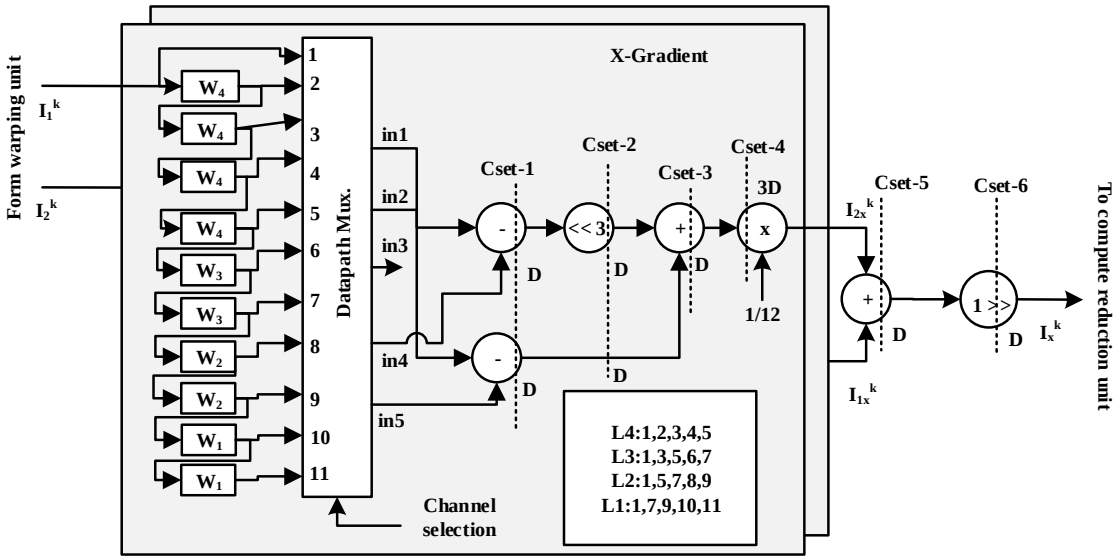


Figure 3.10: Internal architecture of X-direction gradient compute module.

3.5.4 Compute Reduction Stage: S4

The spatial (I_x, I_y) gradients from gradient module are sent to the solver stage to compute equations (3.5) in the Algorithm. 3.1.

$$\begin{aligned} du^{k,l} &= a(c + \alpha^2 \cdot \overline{du}^{k,l} - b \cdot dv^{k,l}) \\ dv^{k,l} &= d(e + \alpha^2 \cdot \overline{dv}^{k,l} - b \cdot du^{k,l}) \end{aligned} \tag{3.5}$$

$$\begin{aligned} a &= \frac{1}{\left((I_x^k)^2 + \alpha^2\right)} \\ d &= \frac{1}{\left((I_y^k)^2 + \alpha^2\right)} \end{aligned} \tag{3.6}$$

It involves the computation of reciprocal operation given in equation (3.6), which results in large resource consumption of solver iteration stage. The proposed architecture mitigates this by moving the reciprocal computation ($N_{solv} \times X_{res}$) from each solver iteration to before the solver iteration ($1 \times X_{res}$) where X_{res} represent the resource utilization of reciprocal stage. The computed results are transmitted to the solver downstream using separate line buffers as shown in Fig. 3.11.

The Averaging (AVG) module computes the average of flow values utilizing a 3×3 kernel with a normalizing factor of $1/12$ given in equation (2.6). There exist two independent

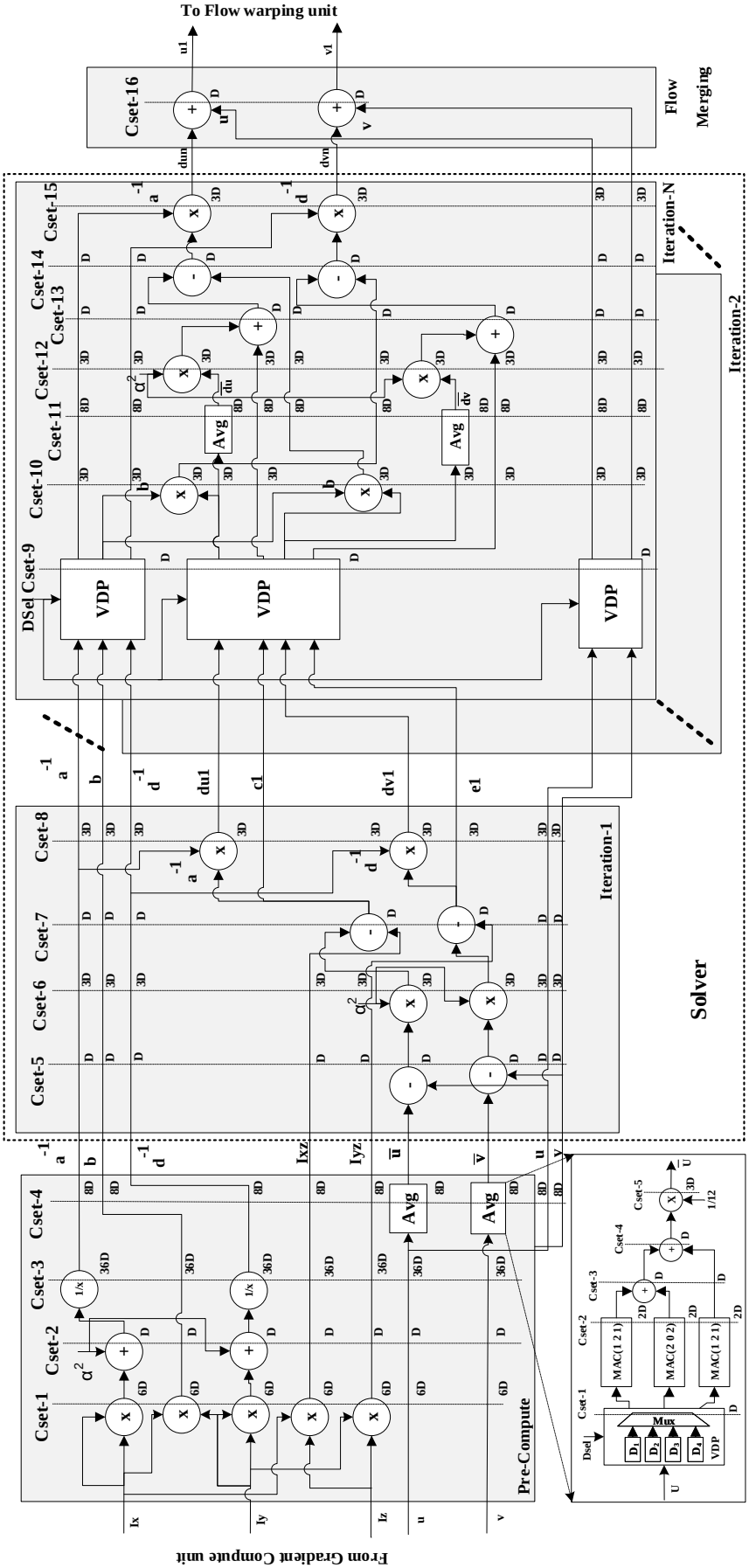


Figure 3.11: Flow computation architecture with compute reduction and solver stage.

averaging modules for u and v. It contains three one dimensional Multiply-Accumulate (MAC) units, which multiplies the pixel with given coefficients and adds resultant values using the adder tree. Since the coefficients are the power of 2, the multipliers are replaced by shift operations. It utilizes 4 left shifters, 7 adders and 6 data registers. The pre-compute module involves 51 pipeline stages. The averaging module utilizes a one-dimensional flow vector to two-dimensional patch converter with Variable Delay Path (VDP) in accordance with pyramid level. The amount of delay is controlled by a select signal $dsel$, from the scheduler.

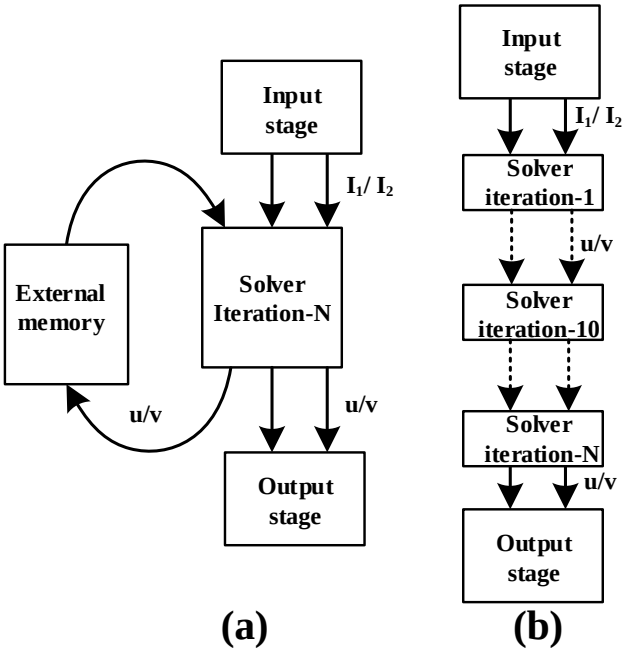


Figure 3.12: Block diagram of, a) Iterative solver and b) N_{solv}-Unfolded solver implementation.

3.5.5 Flow Compute Stage: S5

The precomputed values and the flow values are fed to flow compute stage to estimate non-linear variational OF by solving the system of equations described in the Algorithm 3.1. Since the direct solvers are inefficient for solving the sparse system of equations, an iterative Jacobi solver is chosen. In Jacobi solver the computation of current pixel values depends only on the values of the previous iteration of the neighbouring pixel rather than the latest neighbourhood pixels, thereby eliminating data dependency and makes it suitable for parallel implementation. But eliminating the use of updated neighbourhood degrades

the convergence rate of Jacobi solver, hence the solver implementation needs a large number of iterations. The number of iterations is kept fixed, based on experimental analysis across different datasets.

The internal architecture of the flow compute stages is shown in Fig. 3.11. The proposed scheme improves the architecture performance by unfolding the solver iteration in Fig. 3.12 (a) followed by pipelining, which increases the throughput and energy efficiency by reducing memory access as illustrated in Fig. 3.12 (b). The unfolding is implemented by unrolling the solver iteration so that the data from the current iteration is directly taken to the next iterations in a pipelined fashion. The unfolding by N_{solv} creates N_{solv} independent solver iteration. The number of clock cycles per solver iteration gets divided by N_{solv}, thereby improving the throughput. The computed flow increments are fed back to the rest of solver iterations to improve the flow accuracy. The $1st$ iteration consumes less resource compared to other $N_{solv} - 1$ iterations, as it does not involve flow averaging and utilizes an 18 stage pipeline. The solver utilizes a VDP to adjust the path delay associated with each pyramid level.

3.5.6 Flow Merging and Post-Filter Stage: S6

The flow merging stage accumulates the flow increment from the current level h^k with previous flow increment h^{k-1} as shown in Fig. 3.11. It helps to reduce the flow deviation by the incremental motion compensation towards the actual pixel location. It utilizes two addition operations to accumulate the flow from the previous level and flow increment. The accumulated flow field of the current level is filtered before resizing to the next fine level. A post-filter based on the two-dimensional median filter of kernel size $K \times K$ is used to remove uncertainties in flow estimate caused by illumination artefacts and occlusions.

The median of flow vectors u and v is computed parallelly using two independent filter modules as shown in Fig. 3.13. It utilizes $K-1$ line buffers for converting one-dimensional stream into two-dimensional patch, $K \times K - 1$ DFFs and $(K^2 \times (K^2 - 1))/2$ number of comparators. The three input comparators sort the incoming flow values into a minimum (Min), maximum (Max) and middle (Mid) values using conditional operators and multiplexers. This module utilizes 7 stage pipelines.

3.5.7 Flow Resize Stage: S7

The upsampling of the flow field is done on moving from the coarse pyramid level to the fine level. On the transition from one pyramid level to the next, the flow has to be

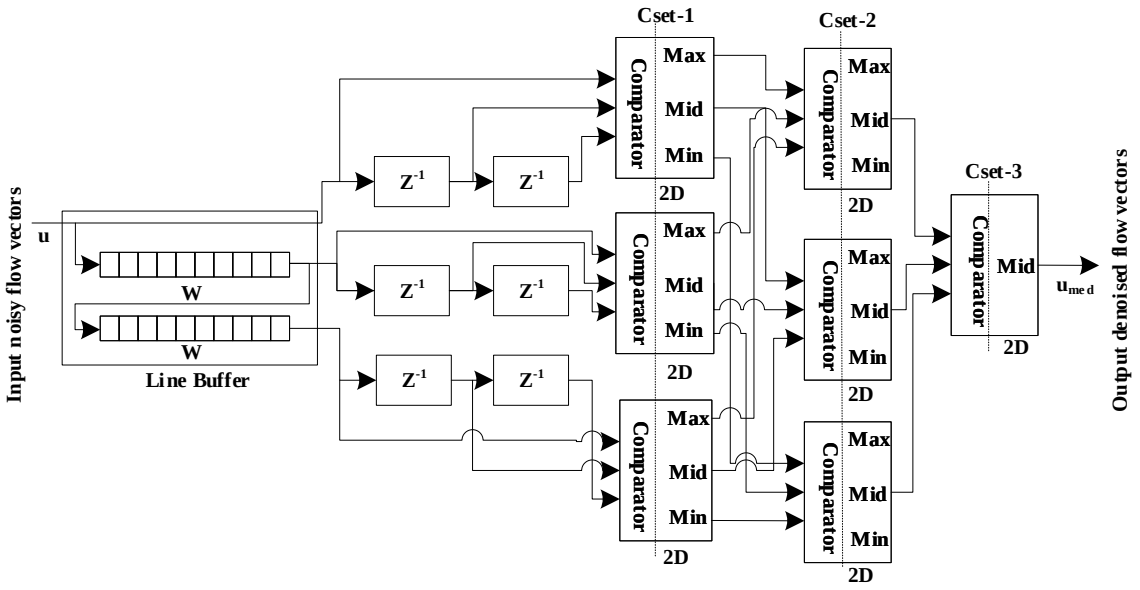


Figure 3.13: Internal architecture of the post filter stage.

quadrupled for a scaling factor of 0.5 (η), i.e 20×20 size of the flow has to be increased to 40×40. The up-sampling is implemented using an interpolation scheme, the selection of the interpolation kernel plays an important role in controlling the accuracy and throughput of the multi-scale OF architecture. Some of the commonly used interpolation kernels include the nearest neighbour, bilinear interpolation, bicubic interpolation, sinc and so on. The experimental analysis shows that the nearest neighbour interpolation method is less complex approach and produces satisfactory results.

The flow resizing architecture up-samples the filtered flow estimate from the coarser level h^k to the next finer level h^{k-1} using nearest neighbour average method as shown in Fig. 3.14. The scheduler generates write or read addresses with corresponding enable signals to perform resizing at each pyramid level. It employs resizing memory bank to parallelize the pixel read and thereby minimize the storage requirement. The interpolated flow is scaled by a factor of $1/\eta$ to compensate for the level switching.

Resize Memory Bank

The flow resize stage perform nearest neighbour interpolation, which needs simultaneous access of a maximum of 4 nearest neighbours. This is achieved by using an efficient resize memory banking scheme which stores the flow values without duplication for all levels except the finest level (level=1). The upsampling stores the flow values of pixels into 4

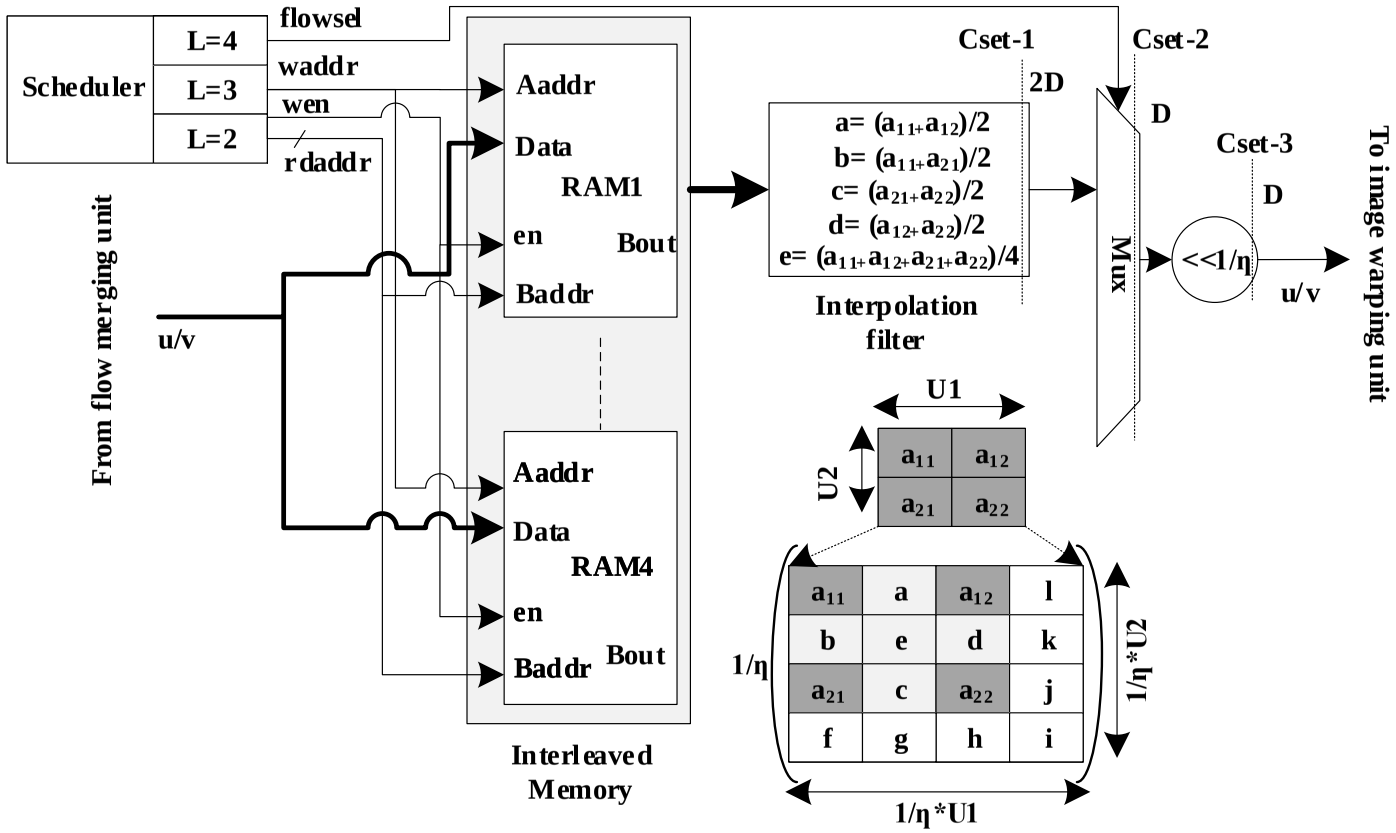


Figure 3.14: Flow resize stage: The scheduler co-ordinates interleaved storage of the computed flow vectors at each level in the resize memory bank which is read simultaneously to perform interpolation using custom filter.

different internal dual port RAMs in an interleaved pattern in the same way as that of the warping stage described in the previous Section 3.5.2, such that no flow value is written more than once into these memories.

The resizing memory bank utilizes 4 internal dual-port RAMs each of size $0.25 \times W_2 \times H_2$. In terms of memory usage, to upsample from 2^{nd} level to the 1^{st} level, only the flow of 2^{nd} level has to be stored which is $\frac{1}{4}^{th}$ size of the 1^{st} level image. The flow resizing is not required in the finest pyramid level. Hence, a maximum of $0.25 \times W_1 \times H_1$ memory is needed to store a single frame which is $\frac{1}{4}^{th}$ of the conventional resizing architecture.

3.5.8 Hardware Design Variants

The modular design makes it easier to develop fixed and floating point variants. The work proposes three different hardware variants of the time sharing variational multi-scale OF (TSMOF) architecture to study the tradeoff between area, accuracy and throughput. The characteristics of these variants are described in Table. 3.1.

Table 3.1: Functional specification of various hardware variants.

Variant / Chara.[1]	HEOF	TSMOF	HTMOF	HPMOF
Scale	Singe	Multi	Multi	Multi
Accuracy	Fixed	Fixed	Fixed	Floating
Motion	Small	Fast	Fast	Fast
Resource	Minimum	Medium	Large	Very Large
Power	Minimum	Medium	Large	Very Large
PE	1	1	4	1

[1] Characteristics of the architecture.

The High Throughput Multi-scale Optical flow (HTMOF) is a fully parallel version of the TSMOF architecture having L number of parallel PEs with separate memory banks for each pyramid levels. This architecture achieves high throughput, but compromises on resource and memory utilization. The coarse level OF computation of the second image pair (I_2/I_3) can start after a very small latency on completion of flow computation of the first image pair (I_1/I_2), in the parallel-pipelined HTMOF architecture. The second hardware variant is the High Precision Multi-scale Optical flow (HPMOF) architecture, which is a floating point single precision variant of TSMOF architecture trading off resource utilization for increased accuracy. The third variant of TSMOF architecture is High-Efficiency

Optical Flow (HEOF), which is a single scale implementation of non-linear OF algorithm achieving the lowest resource and power consumption but at the cost of poor flow accuracy. The fixed-point HEOF architecture does not contain $S1$, $S2$ and $S7$ stages.

3.6 Hardware Implementation Results

The architecture and its variants are synthesized, placed and routed on Xilinx $VC709$ board using Vivado design tools (2017.2). The board is selected as the design needs high throughput and have higher Configurable Logic Blocks (CLB) and memory utilization. Each module is implemented separately to identify the critical path that impacts the maximum throughput. All arithmetic operations like multiplication, addition and division are highly pipelined to maximize the throughput. The variable operand multiplier is implemented using DSP48 slices whereas BRAM is used for constant multipliers. Performance and Resource utilization of each submodule are computed individually and as a whole. The power consumption of each module is analysed by Xilinx power estimator after place and route considering the maximum implementation frequency.

3.6.1 Evaluation Metric

In this section, a general discussion on the different type of error metrics to assess the estimation accuracy is introduced. Most common approaches for quantitative performance measurements calculate error in relation to a precomputed ground truth displacement field. A qualitative measure, on the other hand, relies on visually inspecting a colour-coded version of the computed flow field. While it may be more efficient to quantitatively compare the performance of different algorithms, it is not always possible to compute the ground truth of test images. The most popular metric used for quantitative evaluation is the Average Angular Error (AAE) [20] and the Average Endpoint Error (AEE) [99].

Average Angular Error

The AAE estimates the deviation of the computed OF in relation to the ground truth. This is done by calculating the angle between the vector of the computed optical field w = $(u, v, 1)$ and the given ground truth flow field w_{gt} = $(u_{gt}, v_{gt}, 1)$. This is done by taking the ratio of the dot product of the vectors by their lengths, and then taking the inverse cosine

as given in the equation (3.7).

$$AAE = cos^{-1}\Big(\frac{1 + u \times u_{gt} + v \times v_{gt}}{\sqrt{1 + u^2 + v^2}\sqrt{1 + u_{gt}^2 + v_{gt}^2}}\Big) \tag{3.7}$$

Average Endpoint Error

AEE is another error measurement which calculates the error as the square root of the sum of squared differenced between the computed and ground truth displacement fields as given in the equation (3.8).

$$AEE = \sqrt{(u - u_{gt})^2 + (v - v_{gt})^2} \tag{3.8}$$

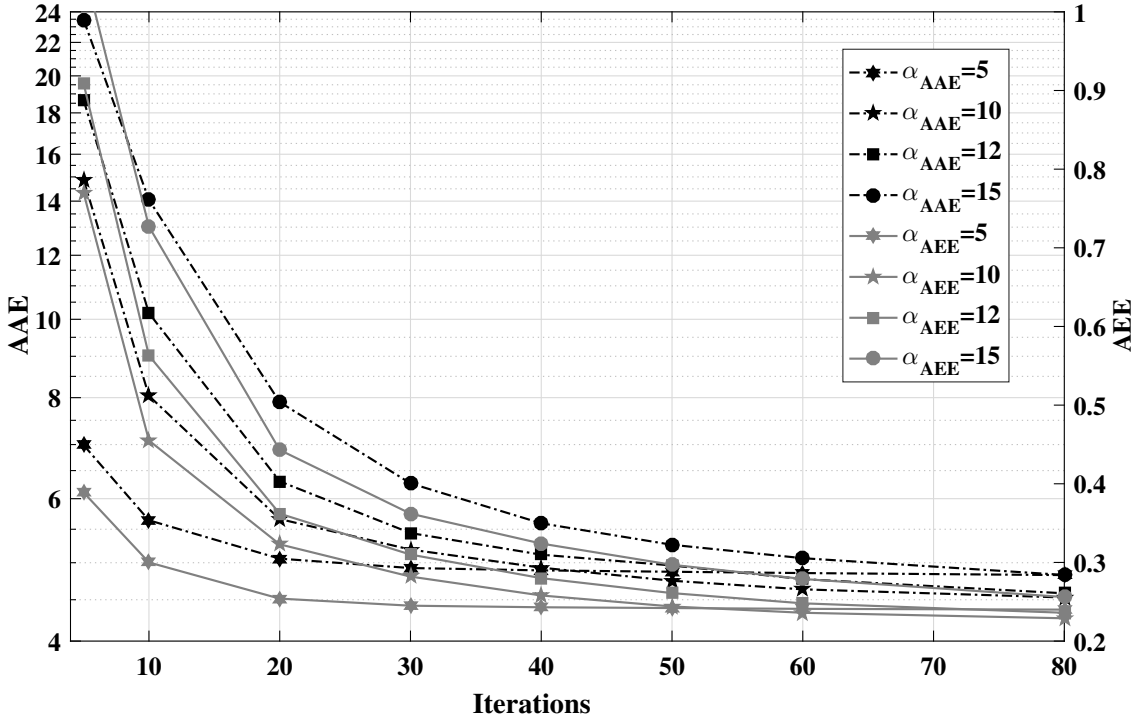


Figure 3.15: Variation of AAE and AEE for different α values.

3.6.2 Parameter Selection

Performance of the proposed architecture is computed as a function of several design parameters. The Middlebury benchmark [76] is utilized for performing the analysis. Fig. 3.15 shows the variation of AAE and AEE for different values of α and solver iterations (N_{solv}),

where α controls the smoothness of flow. The graph is captured for each value of α varying from 5 to 15 averaging over a large number of datasets. The AAE for most of the α values are high for 20 or lesser number of iterations, shows a fast decreasing trend till 35 iterations followed by a very slow converging trend for rest of the iterations. It can be inferred that for $\alpha = 5$ the AAE and AEE gets settled within 20 iterations.

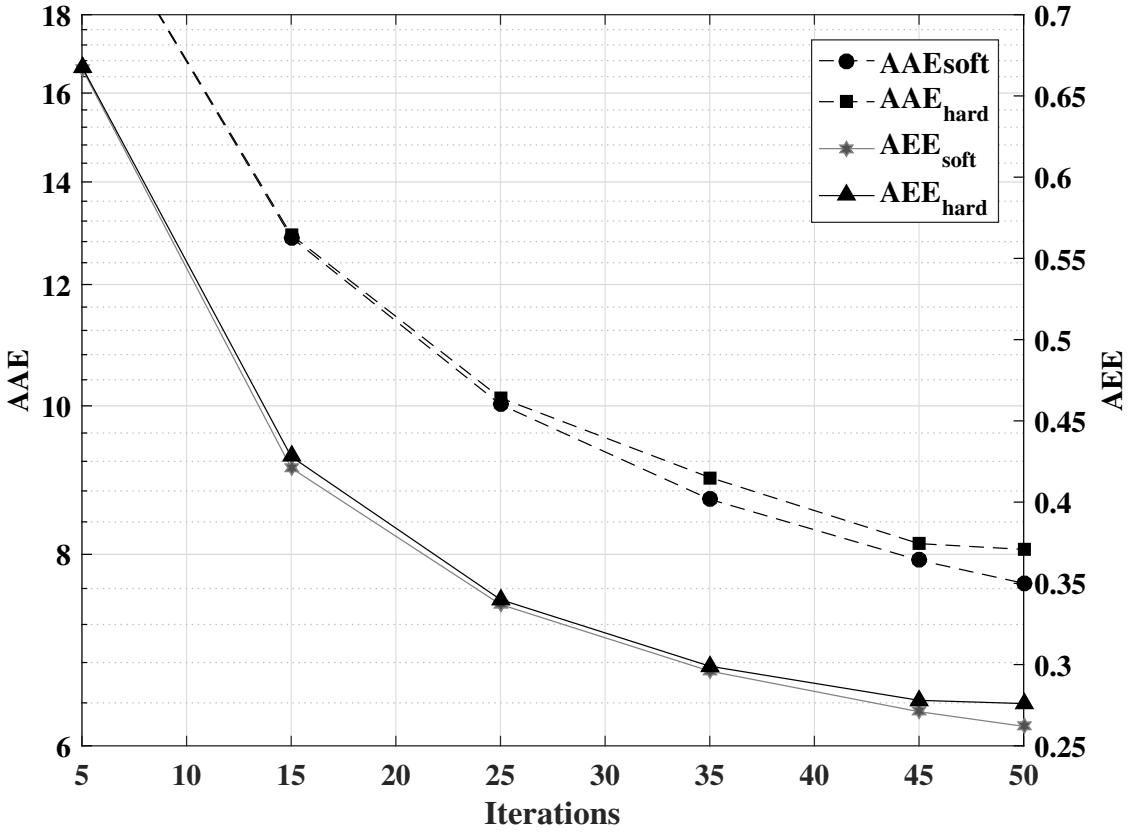


Figure 3.16: Variation of AAE and AEE for fixed and floating point implementations.

3.6.3 Numerical Representation

The selection of fixed or floating point implementation of variational multi-scale OF algorithm plays an important role in the trade-off between accuracy and power consumption. The fixed point representation opens a large space for the exploration of the suitable bit width requirement for each of the stages. The conversion from floating point to fixed point results in quantization effects. A comparison between the fixed-point Verilog and floating point implementations in terms of AAE and AEE for Middlebury dataset [100] is shown in the Fig. 3.16.

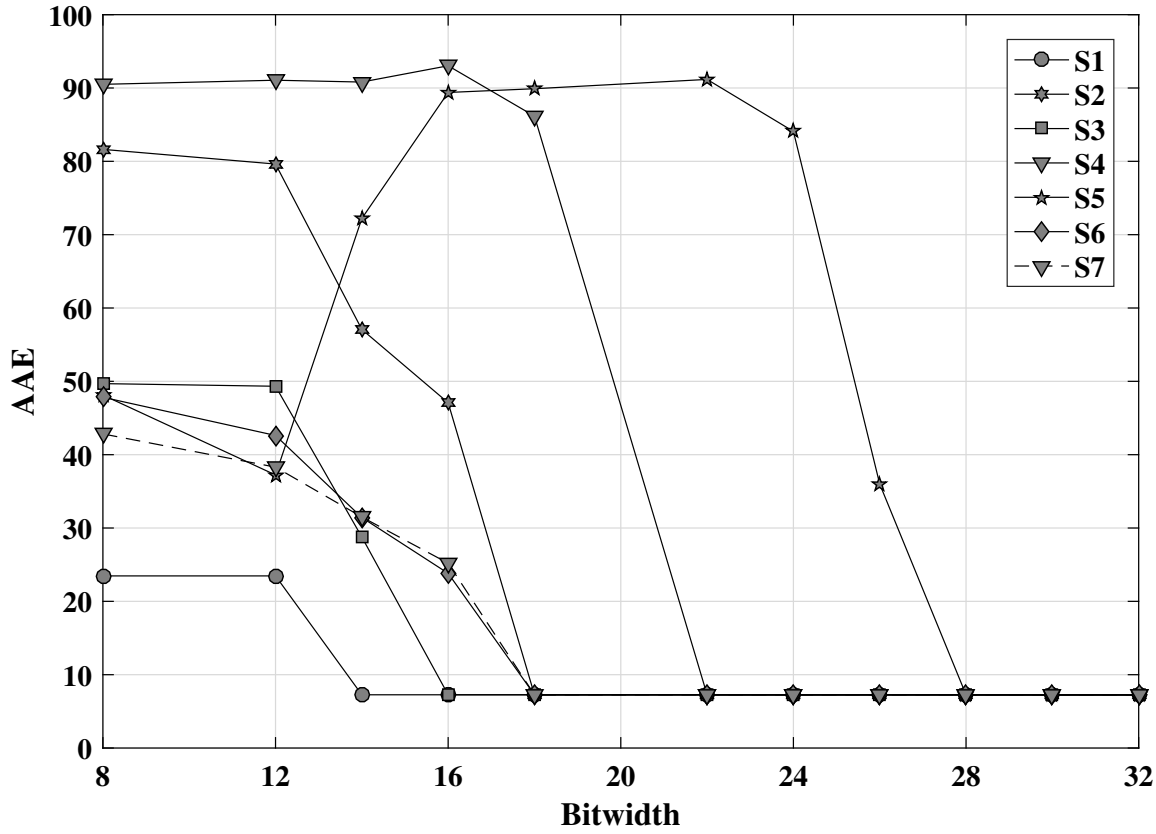


Figure 3.17: Variation of AAE and AEE for different bit-width selection.

Table 3.2: Selected bit-width for different stages.

	S1	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**
BW$_{min}$[1]	14	18	16	22	28	18	18

[1] represent the maximum number of bits utilized for each stage.

The floating point implementation shows lesser error for a fixed iteration but consumes large resource utilization. The bit-width for each module is computed by feeding reference images to the design and measuring the degradation in terms of AAE and AEE, by varying the bit-width from 8 to 32 while keeping the rest of the circuit in full precision as shown in Fig. 3.17. Here the $S5$ graph shows an initial reduction in the AAE (which is not significant as the error is significantly large), attributed to the fact that there is an insufficient number available for integer part. The design follows a variable precision signed fixed point $Qm.n$ representation with m and n corresponds to the integer and fractional part. The highest bit-width used by each module or stage of the architecture is given in Table. 3.2.

3.6.4 Performance and Resource Analysis

The hardware implementations make use of LUT for logic functions, DFF for pipelining the critical path, BRAM for intermediate data storage and DSP48 slices for complex signal processing tasks. The TSMOF architecture is configured to process a macro-block of size 8×8, the number of solver iteration N_{solv} is set to 30 and α is tuned for each image sequence to get minimum AEE.

Table 3.3: Performance comparison between software (soft) and hardware (hard) implementations of variational multi-scale OF with different pyramid levels.

Level / Database	1				2				3				4			
	AAE		AEE		AAE		AEE		AAE		AEE		AAE		AEE	
	soft[1]	hard[2]	soft	hard	soft	hard	soft	hard	soft	hard	soft	hard	soft	hard	soft	hard
Marble	20.98	21.29	0.74	0.75	18.91	18.99	0.71	0.71	7.63	7.65	0.25	0.25	7.28	7.31	0.23	0.23
Dimetrodon	39.72	40.69	1.59	1.61	28.54	29.40	1.31	1.33	12.71	13.69	0.62	0.66	9.33	10.40	0.48	0.52
Rubberwhale	33.45	33.69	0.94	0.94	27.10	27.23	0.81	0.82	10.41	10.57	0.33	0.33	6.64	6.79	0.21	0.21
Grove2	34.51	35.74	2.19	2.22	21.79	22.51	1.76	1.79	7.67	8.10	0.56	0.59	4.73	5.13	0.34	0.37
Venus	41.82	42.19	3.01	3.02	26.59	26.92	2.45	2.46	13.70	14.03	1.03	1.07	12.27	12.63	0.91	0.94

[1] refers to the software implementation of variational multi-scale OF.
[2] represents the hardware implementation of variational multi-scale OF.

Table. 3.3 shows the performance comparison of the TSMOF architecture with the software implementation considering different pyramid levels using Middlebury [100] database. From the table, it is understood that the performance of the multi-scale variational OF architecture with a single pyramid level is the same as that of the single scale HEOF architecture. It is noticed that the AAE and AEE reduce significantly from pyramid level two to three, which implies that the pixel displacements of the datasets are within the filter motion range (2^3). It can be inferred that the error gets settled with each additional layer, till motion is within the considered filter range. From the table, it can be observed that the fixed-point Verilog implementation shows a negligible loss in flow accuracy compared to the floating-point design.

Table. 3.4 shows the performance evaluation of the TSMOF architecture on KITTI and MIPI SINTEL dataset. Since the KITTI dataset has significant illumination artefacts, the TSMOF architecture shows higher AEE.

Table. 3.5 shows the resource requirement and power consumption of the architecture

Table 3.4: Performance of variational multi-scale OF architecture on the KITTI and Sintel dataset in nonoccluded (Nocc) and all (Occ) areas.

Dataset / Error	MIPI Sintel		KITTI	
	Final	clean	Nocc	Occ
AEE	7.62	7.04	18.04	28.01

Table 3.5: Performance of TSMOF architecture stages.

Module / Resource	S1	S2	S3	S4	S5	S6	S7
Slice	2,931 (2.7%)	1,395 (1.28%)	439 (0.4%)	841 (0.78%)	9,307 (8.6%)	2,467 (2.28%)	638 (0.59%)
BRAM36	256 (17.4%)	256 (17.4%)	15.5 (1.05%)	0	218 (14.83%)	128(8.7%)	2(0.13%)
DSP48	0	33 (0.92%)	0	5 (0.14%)	148 (4.11%)	0	0
Fmax (MHz)	420	420	410	440	420	430	440
Power (W)	1.14	1.30	0.475	0.394	9.572	0.634	0.258

for QVGA (320×240) resolution. It can be inferred from the Table. 3.5 that $S3$ stage uses a minimum number of slices but consumes higher power than $S7$ due to the large number of BRAMs utilized for implementing line buffers. The $S5$ stage with 20 solver iteration is the highest resource utilization stage, as it makes use of a large number of DSP48 slices for implementing the complex arithmetic. Unlike other stages, $S1$ and $S2$ has higher BRAMs utilization due to large buffers required for pyramid generation and intermediate data caching which results in consuming significantly more power than other stages excluding $S5$. Individual stages of TSMOF architecture reaches up to a maximum frequency of 410 MHz but this gets reduced to 378 MHz for the full system due to the routing congestion and resource utilization.

Table. 3.6 shows the resource consumption of the TSMOF architecture for the different number of solver iterations. It shows that for every 10 additional number of solver iterations, the BRAMs utilization shows a slight increase as the image resolution is kept fixed. But DSP48 usage increases by more than $2\times$ due to more number of arithmetic computations involved in the $S5$ solver stage. It can be observed from Table. 3.6 that the maximum number of solver iterations is limited by the availability of BRAMs in the current FPGA implementation.

The design has 169 super-scalar units of which, stage $S1$ has $2 \times 8 \times 8$ parallel dual port

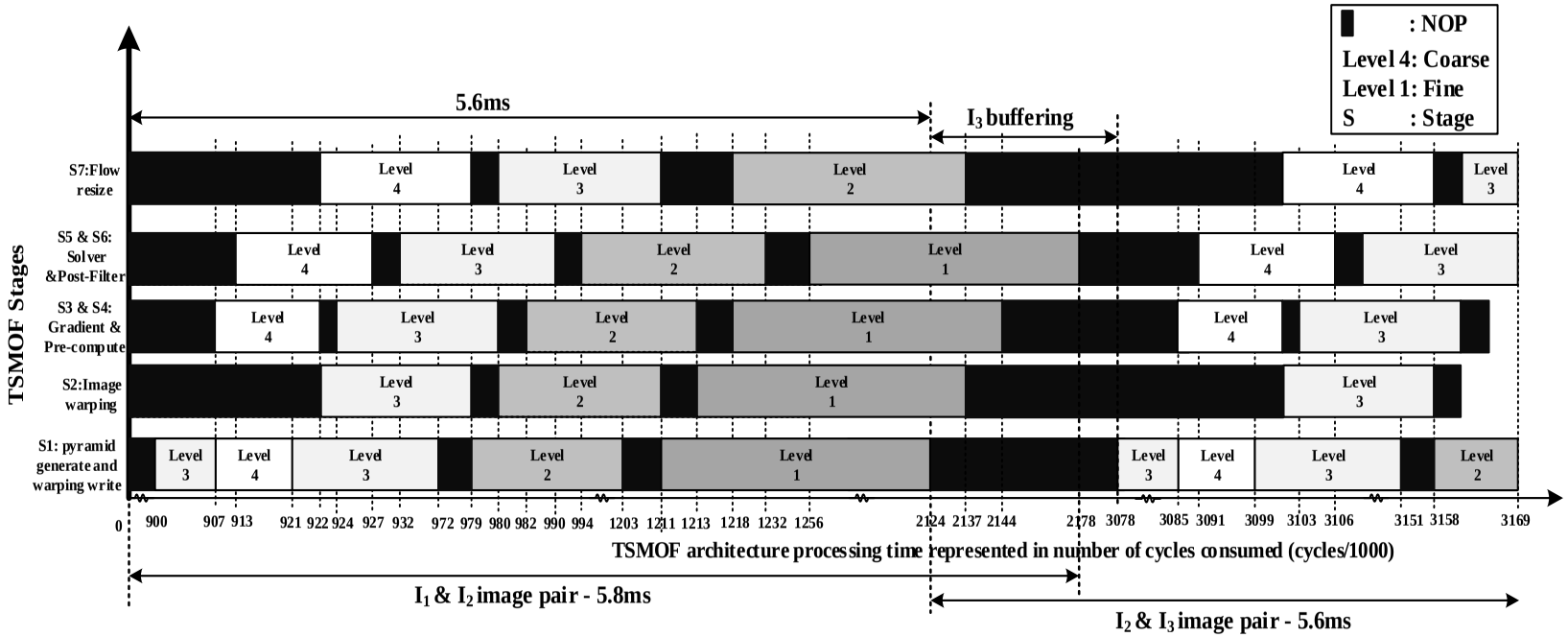


Figure 3.18: Timing diagram of the time-sharing variational multi-scale OF architecture.

Table 3.6: Resource utilization and power dissipation of TSMOF architecture for different Jacobi solver iterations.

Iteration / Resource	5	10	25	50
Slice	16,370 (15.1%)	17,519 (16.2%)	23,462 (21.6%)	36,715 (33.9%)
BRAM36	1,085 (73.8%)	1,130 (76.9%)	1,265 (86.1%)	1,452 (98.8%)
DSP48	66 (1.83%)	96 (2.7%)	186 (5.1%)	335 (9.3%)
Fmax (MHz)	432	430	410	378
Power (W)	8.44	10.1	16.1	28.5

memories for pyramid generation, $S2$ has 4 independent warping memories, $S3$ utilizes 9 parallel units for gradient computation, $S4$ has 9 units for computing solver coefficients, $S5$ has 9 units for the solver, $S6$ has 6 units for filtering and $S7$ has 4 parallel resizing memories. This accounts for a total of 1404 fixed-point operations of which 344 is for sub-sampling and filtering, 19 for gradient compute stage, 791 for 30 iterations of flow computation and 182 for flow merging and resizing stage. This leads to a maximum throughput of 395 GOPS while consuming the power of 18.42 W at 378MHz.

3.6.5 Timing Analysis

The Fig. 3.18 shows the OF computation time required by 30 iteration of TSMOF architecture for processing an HD image pair while running at 378 MHz. The time taken by each of the modules in different pyramid levels are described in Fig. 3.18. The x-axis of the timing diagram shows the number of cycles (scaled by $1/1000$) taken for computing different stages of the TSMOF architecture. The timing diagram depicts the parallel operation of stages at every pyramid level (i.e. $S1$, $S3$, and $S4$ stage of level 4) as well as between different pyramid levels (i.e. $S1$ of level 2 and $S2$-$S6$ of level 3). An initial latency of 2.3 ms ($\approx 900,000$ cycles) is required for buffering the first image pair. From the figure, it is understood that level=4 and level=1 does not require the warping stage $S2$ and flow resizing stage $S7$ respectively. As discussed in Sect. 3.5.1 and can be observed from the diagram that the pyramid generation ($S1$) for level=3 starts before the pyramid generation ($S1$) for the coarsest level (level=4). Also, it can be observed that the finest level (level=1) computation takes $16\times$ more time than the coarse level (level=4).

3.6.6 Comparison among Hardware variants

In order to compare the performance of the proposed hardware variants, QVGA (320×240) resolution images were considered, so that all architectures with 30 iterations fit on the existing evaluation board as shown in Table. 3.7. This reduces the resource congestion but leads to a lower frequency of operation for HPMOF as device utilization is high. The proposed hardware variants are implemented separately using Vivado. Each PE consists of pyramid creation, warping, gradient compute, compute reduction, solver, flow merging, post filter and flow resizing stage. Table. 3.7 shows the implementation details of the proposed architecture.

Table 3.7: Performance comparison with proposed variants for QVGA resolution.

Variant / Resource	HEOF	TSMOF	HTMOF	HPMOF
Slice	14,583 (13.5%)	23,408 (21.6%)	61,637 (56.9%)	94,292(87.1%)
BRAM36	261 (17.8%)	374 (25.4%)	1137 (77.3%)	595 (40.5%)
DSP48	183 (5.1%)	216 (6%)	711 (19.8%)	3527 (97.7%)
Fmax (MHz)	410	384	360	220
FPS[1]	5338	2147	4571	2793
Power (W)	8.37	11.58	47.33	34.2

[1] refers to frames per second.

The coarsest level does not have a warping stage and the finest level does not have a resizing stage in the HTMOF architecture. The HTMOF architecture consumes $3.5\times$ more resource and power as compared to TSMOF architecture. Since HTMOF architecture has four independent PE, it consumes $4\times$ more resources and power than TSMOF variant with single PE. The size of memory banks in the pyramid creation, warping and flow resizing are different for every level in a HTMOF architecture. The level-1 pyramid creation stage stores $2 \times W_1 \times H_1$ image, whereas level-2 stage only needs to store $2 \times W_2 \times H_2$ and so on. Similar case for the warping and flow resizing banks. From the table, it is understood that HTMOF and HPMOF variant cannot be deployed for images having resolutions higher than QVGA as most of the BRAMs and DSP48 are utilized. The HEOF variant consumes almost $5.6\times$ lesser power than the HTMOF.

Table 3.8: Comparison of proposed variational multi-scale architecture with state of the art methods.

Resource / Method	Model	Device	Slices	BRAM36	DSP48	Resol.	Iter.[2]	F_{max}[3]	FPS[4]	Power (W)	Speedup	Area	FFD	S_{ADN}	AAE
HEOF[1]	HS	XC7VX690T	14,843 (13.7%)	276 (18.7%)	183 (5.1%)	1280x720	30	402	436	8.43	0.94	3.29	1	0.28	22.46
Barranco[88]	LK	XC4VFX100	4,128 (9.7%)	48 (12.7%)	30 (18.7%)	640x480	1	44	270	-	8.59	17.92	0.72	0.65	4.55
komo[92]	HS	XC7VX980T	137,049 (89.5%)	1346 (89.7%)	-	1920x1080	128	129	62	12.22	2.92	0.83	1	3.52	5.35
Diaz[85]	LK	V2C6000	9,123 (26.9%)	20 (13.8%)	12 (8.3%)	800x600	1	82	170	-	4.61	24	0.10	1.92	7.86
Pauwels[101]	Phase	GPGPU	-	-	-	640x512	1	-	20.6	-	-	-	-	-	2.09
TSMOF[1]	HS-mul	XC7VX690T	25,887 (23.9%)	1,310 (89.1%)	216 (6%)	1280x720	30	378	176	18.42	1	1	1	1	4.49
HTMOF[1]	HS-mul	XC7VX690T	61,637 (56.9%)	1137 (77.3%)	711 (19.7%)	320x240	30	360	4571	47.33	1.05	0.81	1	1.29	4.49
HPMOF[1]	HS-mul	XC7VX690T	94,292 (87.1%)	595 (40.4%)	3527 (97.9%)	320x240	30	220	2116	34.2	1.71	0.42	1	4.07	4.37
Tomasi[87]	LK-mul	XC4VFX100	36,603 (86.7%)	106 (28.1%)	132 (82.5%)	640x480	1	45	31.5	4.35	8.4	4.43	0.72	2.62	7.91
Barranco[88]	LK-mul	XC4VFX100	26,036 (61.7%)	112 (29.7%)	62 (38.7%)	640x480	1	83	32	-	4.55	5.96	0.72	1.05	5.97
Seyid[93]	HBM	XC7VX485T	-	65 (6.3%)	48 (1.7%)	640x480	-	200	39	-	-	-	-	-	4.57

[1] refers to the proposed methods.
[2] denotes the number of solver iterations.
[3] frequency is expressed in MHz.
[4] represents the frames per second.

3.6.7 Comparison with state-of-the-art methods

Table. 3.8 shows the comparison of TSMOF with state of the art OF architectures and variants. The FPS is calculated as the reciprocal of the time taken for flow computation with the inter-frame latency. The TSMOF architecture achieves a frame rate of 176 fps for HD images which is the highest among other existing multi-scale OF architectures. The proposed architecture also achieves the highest Compute Density per Joule of 21.5 GOPS/Watt among other state-of-the-art architectures. It also offers the smallest AAE of 4.49 without discarding flow components for Yosemite sequence without clouds. The single scale LK [88] consumes a minimum number of resources including slices, BRAM and DSP48 and hence it has $\approx 17\times$ better area efficiency than the TSMOF architecture.

A metric defined in the work [102] is utilized to compare the performance of the proposed architecture among different FPGA families, vendors and operating parameters. To simplify the comparison the number of equivalent slices used for implementing BRAM36 and DSP48 are assumed to be 250 and 200 respectively. The term speedup (S) is computed as the ratio of the execution time (T_{per}) of the proposed design to the reference design, the area is computed as the ratio of the resource utilization (DSP48, BRAM and Slices) of reference design to that of the proposed design. Area normalized speedup (S_{AN}) is the ratio of the speed up to the area of the proposed design. Area-technology normalized speedup (S_{ADN}) is defined as the ratio of speed up to area and delay (LUT/FF) characteristics of a particular device family.

On observing S_{ADN} values from Table. 3.8 it can be inferred that the proposed TSMOF architecture achieves $2.6\times$ performance improvement over the state of art multi-scale OF architectures. The S_{ADN} value is less than one for the single scale OF designs due to less resource utilization and power consumption but leads to sparse flow fields. It shows that the S_{ADN} value of TSMOF is $5.1\times$ higher than the HPMOF variant. It can also be noticed that the HPMOF variant provides accurate flow vectors than other proposed variants, but have a low frequency of operation and higher DSP48 utilization.

3.6.8 Design Scalability

Unlike other multi-scale architectures, the TSMOF is directly scalable to handle a multitude of image resolutions as shown in Table. 3.9.

It is understood from the table that, as the resolution increases DSP48 utilization remains constant, but there is a significant increase in the number of BRAMs due to high usage of line buffers and image memories. It is also noted that for a small increase in

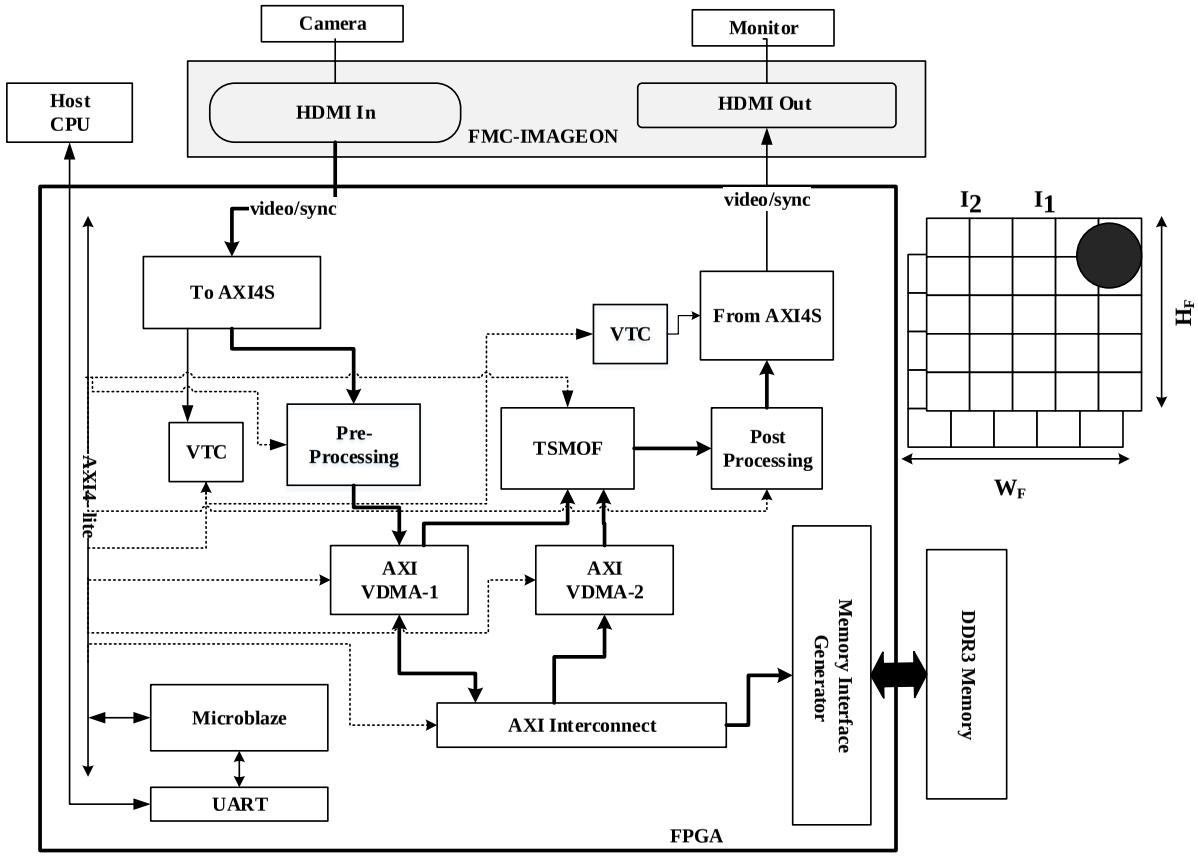


Figure 3.19: Block diagram of a real-time system for large displacement OF computation.

Table 3.9: Design scalability of TSMOF architecture.

Resource \ Resol.	QVGA	VGA	SVGA	XGA
Slice	21663 (20.0%)	23896 (22.1%)	25299 (23.4%)	27024 (24.9%)
BRAM36	381.5 (25.9%)	717.5 (48.8%)	797 (54.2%)	1310 (89.1%)
DSP48	216 (6%)	216 (6%)	216 (6%)	216 (6%)
FPS [1]	3896	946	590	360

[1] represents the number of frames per second.

image resolution (VGA to SVGA), the amount of BRAMs shows only a slight increase due to the presence of unused regions in the memory bank. Since the design uses memory banks constructed from the BRAMs, the image will not fit perfectly into the memory. Similarly, the resource optimized implementation of the TSMOF architecture for HD image consumes 1310 BRAMs and fewer slices compared to XGA architecture. The maximum supported resolution of TSMOF architecture can be improved by utilizing the unused distributed memories ($SLICEM$) for buffering the pixel data or can switch to a high-end FPGA with more available memory.

3.7 Real-time Implementation

Real-time implementation of the proposed TSMOF architecture on Virtex-$VC709$ FPGA board is shown in Fig. 3.19. The image sensor interfaced to the FPGA development board captures high-resolution images and streams as one-dimensional raw data to the FPGA. The HDMI camera is interfaced to the FPGA using an FMC-IMAGEON daughter card producing HD frames at 30 fps. The proposed system utilizes an HDMI TX/RX, UART, Microblaze and DDR controller from Xilinx LogiCORE IP.

The architecture needs two images, the previous and current frame. The previous frame N_{solv}-1 is stored in the external memory and read back from the memory and the current frame N_{solv} is streamed to internal memory. The image is preprocessed and stored in the external DDR3 memory which is read by Video Direct Memory Access (VDMA) and fed into the flow computation engine. Finally, the dense flow computation is displayed on an HDMI monitor based on a simple pixel colouring scheme as in Fig. 3.20. The computed flow vectors are displayed using a simple pixel colouring scheme, $R = \lfloor 128 + u \rfloor; G = \lfloor 128 + v \rfloor; B = \lfloor 255 - u \rfloor$. The UART interface allows configuring architecture parameters

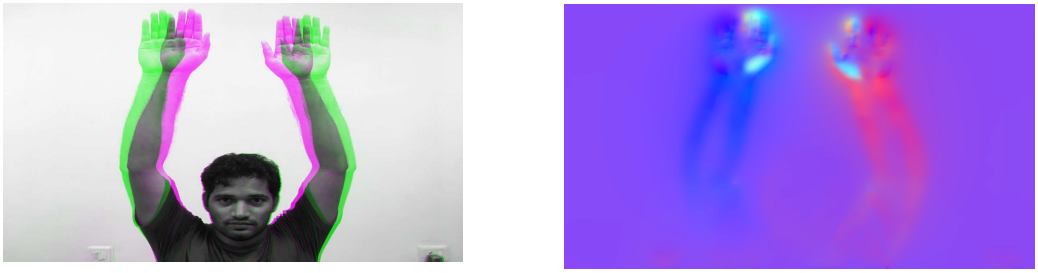


Figure 3.20: Colour coded optical flow

from the Host PC.

Table 3.10: Resource utilization of real-time variational multi-scale OF system.

Util.[1] / Resource	Used	Available	Percentage
Slice	44,147	108,300	40.7%
BRAM36	1412	1470	96.1%
DSP48	222	3600	6.2%

[1] Resource utilization of the design.

Table. 3.10 shows the resource utilization of real-time implementation of the proposed TSMOF architecture utilizing Xilinx image processing pipeline. Even though the TSMOF architecture alone achieves 378 MHz, a real-time implementation of TSMOF architecture utilizing the existing Xilinx image processing pipeline runs at a lower operating frequency of 160 MHz.

3.8 Summary

This chapter discussed the hardware adaptation and design of the high throughput time-sharing architecture of variational multi-scale OF algorithm. The dedicated memory banks and the special access schemes helps to achieve superior area and energy efficiency while attaining massive parallelism and accuracy. The design is scalable to fit in an embedded device with different image resolution in real-time while consuming low-power. This is the first work on deeply pipelined time-sharing architecture for variational multi-scale OF to capture dense and accurate OF of fast-moving objects from HD frames in real-time (176 fps). The architecture makes use of 169 super-scalar units with 702 deep pipelines

to achieve a throughput of 395 Giga Operations Per Second (GOPS) with a computation density of 21.5 GOPS/Watt. This architecture achieves an effective speed-up of $2.6\times$ compared to state of the art multi-scale OF implementations. The work also analyses three architecture variants to evaluate the trade-off between accuracy, resource utilization and throughput.

Table 3.5 shows the resource utilization of $S5$ stage with 20 solver iteration, consuming $3\times$ higher number of slices than other stages in the variational multi-scale OF architecture. This lead to a higher resource consumption which needs to be analysed for improved area efficiency. The selection of edge-preserving flow filter at each pyramid level also plays an important role in improving the flow accuracy of the variational multi-scale OF architecture. Since the direct implementation of BF is complex and have low throughput, the proposed architecture utilizes a two-dimensional median filter. Hence the design of a high throughput architecture for BF is also analysed to improve the flow accuracy.

Chapter 4

High Throughput Architecture for OF Subsystems

This chapter focuses on the design of a high throughput and resource optimized architecture for the internal subsystems of the proposed variational multi-scale OF architecture discussed in the previous chapter 3. The work starts with the design of a high throughput RBSOR solver architecture and its integration into the variational High Throughput Optical Flow (HTOF) architecture to analyse the reduction in resource utilization. The chapter concludes with design of a high throughput BF architecture to denoise the flow field while preserving edges.

4.1 Proposed RBSOR Solver architecture

4.1.1 Introduction

The selection of solver is a critical design consideration made during the implementation of variational HSOF architecture. Since the linear system of equations for all input pixels forms a sparse system, the direct methods are not efficient to solve it. The numerical or iterative solvers are commonly used for solving such type of sparse systems. The iterative solver starts with an approximation of the true solution and finds a closer approximation in each iteration until the required accuracy is obtained. This means that for an iterative solver the number of iteration depends on the accuracy required.

A conventional variational HSOF architecture utilize some form of an iterative solver to compute the OF with the intermediate flow vectors buffered in external memory. Each solver iteration is evaluated by retrieving the input image and intermediate flow values using two independent data ports leading to low throughput performance. Even with the

use of high-performance data port, this scheme leads to a significant delay in the flow computation. This can be improved by unfolding the solver iteration and buffering the intermediate data. But this results in the solver stage consuming the highest resource among other modules in the OF architecture.

Considering these facts, the research work analyses different types of iterative solvers such as Jacobi, Gauss-Siedel, SOR, Conjugate Gradients and Multi-grid approaches [103, 104] in terms of the number of iterations and convergence rate to achieve faster convergence and less resource utilization. For instance, consider a system of n linear equations in n unknowns represented by the given equation (4.1).

$$\begin{aligned} a_{11} \cdot q_1 + a_{12} \cdot q_2 + ...a_{1n} \cdot q_n &= b_1 \\ a_{21} \cdot q_1 + a_{22} \cdot q_2 + ...a_{2n} \cdot q_n &= b_2 \\ ..... & \\ a_{n1} \cdot q_1 + a_{n2} \cdot q_2 + ...a_{nn} \cdot q_n &= b_n \end{aligned} \tag{4.1}$$

The matrix notation of the equation (4.1) is given in equation (4.2),

$$A \times Q = B \tag{4.2}$$

The substitution of initial guess q_1^0, q_2^0, q_3^0 $\ldots$ q_n^0 for all the unknowns into equations (4.2), leads to the computation of an updated guess q_1^1, q_2^1, $q_3^1 \ldots$ q_n^1.

Jacobi solver

The Jacobi solver forms a diagonally dominant system utilizing a linear system of equations. The diagonal elements are solved iteratively by plugging in an approximate value until it converges. With the Jacobi method, the values of q_i^k obtained in the k^{th} iteration remain unchanged until the entire $(k+1)^{th}$ iteration is completed. It starts with the initial guess q_1^0, q_2^0, q_3^0 $\ldots$ q_n^0 and compute the next approximation of the solution as given in the equation (4.3).

$${q_i}^{k+1} = \frac{1}{a_{ii}}(b_i - \sum_{i \neq j} a_{ij} \cdot {q_j}^k) \tag{4.3}$$

Gauss Seidel solver

In Gauss-Seidel method, the new values q_i^{k+1} will be used as soon as they are available by the equation (4.4). It can be observed that once q_1^{k+1} is computed from the first equation, this is used in the second equation to compute new value for q_2^{k+1} and so on.

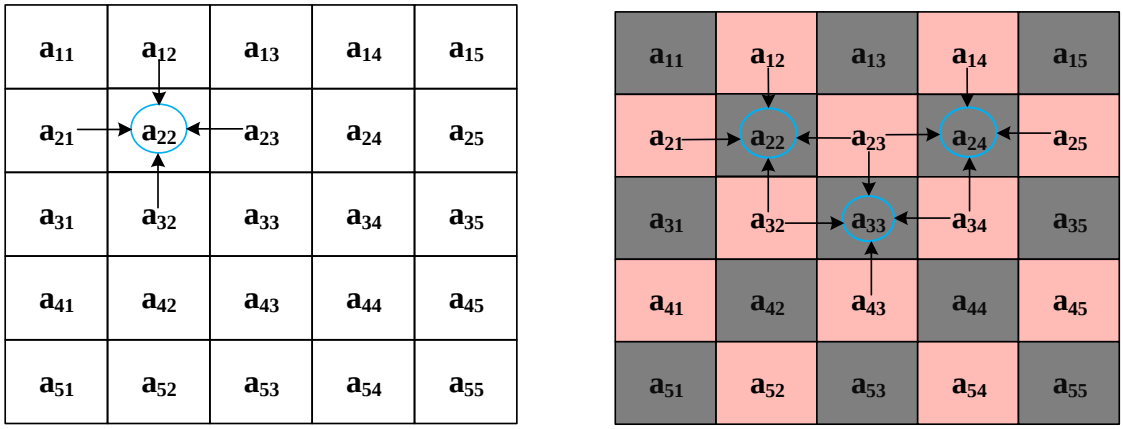
$$q_i{}^{k+1} = \frac{1}{a_{ii}}(b_i - \sum_{j<i} a_{ij} \cdot q_j{}^{k+1} - \sum_{j>i} a_{ij} \cdot q_j{}^{k}) \qquad with \qquad i \neq j \quad (4.4)$$

Successive over-relaxation solver

The Successive Over Relaxation (SOR) solver is computed as an extrapolation of the Gauss-Seidel method. It provides the highest convergence rate by introducing a weighted average (ω) of the current flow vector from n^{th} iteration in addition to the computed Gauss-Seidel flow as given in the equation (4.5).

$$q_i{}^{k+1} = (1-\omega) \cdot q_i{}^{k} + \frac{\omega}{a_{ii}}(b_i - \sum_{j<i} a_{ij} \cdot q_j{}^{k+1} - \sum_{j>i} a_{ij} \cdot q_j{}^{k}) \qquad (4.5)$$

where ω is the convergence factor. The value of ω varies in the range of $\omega \in (0, 2)$ and will control the rate of convergence. The SOR method simplifies to the Gauss-Seidel method, when ω =1. The variational OF solver equation (2.8) described in the Section 2.3 is evaluated using SOR iterative scheme given in equation (4.6).



(a) SOR solver **(b)** Red Black SOR solver

Figure 4.1: Data dependency in the pixel computation across different solver iterations.

$$u^{n+1} = w.u^n + [1-w].\left(\overline{u}^{n+1} - I_x \frac{(I_x.\overline{u}^{n+1} + I_y.\overline{v}^n + I_z)}{(\alpha^2 + I_x^2 + I_y^2)}\right) \tag{4.6}$$
$$v^{n+1} = w.v^n + [1-w].\left(\overline{v}^{n+1} - I_y \frac{(I_x.\overline{u}^{n+1} + I_y.\overline{v}^n + I_z)}{(\alpha^2 + I_x^2 + I_y^2)}\right)$$

A SOR solver needs approximately half the number of iterations as compared to Jacobi for achieving the same accuracy [96]. Fig.4.1 (a) shows the data dependency of the SOR solver in computing the given pixel value. For example, the computation of the pixel value a_{22} in the $(n+1)^{th}$ iteration requires a_{12}, a_{21}, a_{23} and a_{32} pixels from the same iteration. Among which the $(n+1)^{th}$ iteration of top pixel (a_{12}) is already computed and available. While the right (a_{23}) and the bottom (a_{32}) pixels are not available as they are not yet computed. The $(n+1)^{th}$ iteration value of the left pixel (a_{21}) is also required to compute a_{22} pixel. But the processing and pipelining latency involved in the computation of the left pixel (a_{21}) adds delay to the pixel computation. This further limits the parallelism and pipelining achieved with the hardware implementation.

The data dependency of the original SOR is eliminated using a two-pass Red-Black SOR scheme [104] by dividing the incoming pixel stream into red and black pixels; with the red being surrounded by black pixels and vice versa as illustrated in the Fig.4.1 (b). The red pixels values are computed in the first pass, the remaining black pixel values are evaluated in the second pass while keeping red pixels untouched. In the second pass, the black pixel (a_{22}) is computed using the neighbouring values of the red pixels (a_{12}, a_{21}, a_{23} and a_{32}) from the first pass. Thus the computation of black pixels and their adjacent red pixels in different passes leads to parallel operation. Also, since all the neighbouring pixels are available during the flow computation, all elements can be processed in parallel and the system can be pipelined to achieve better operational frequency. Hence this work focuses on the design of a high throughput hardware architecture of RBSOR solver to provide higher accuracy with a lesser number of solver iterations as compared to other solvers.

4.1.2 Related work

The RBSOR solver architecture can be used as a replacement for existing slow convergence and large area consuming solvers in numerous signal processing applications. Since the literature lacks information about the hardware implementation of RBSOR solver architecture, the work explores the OF architecture utilizing the RBSOR solver. Most of the existing variational HSOF architectures are based on parallel and slow converging Jacobi

solver. Martin et al.[105] implemented an HSOF algorithm on the Altera APEX20K device. It can process 256×256 images at 60 fps. In [106], a high-performance FPGA implementation of HSOF algorithm is proposed. It is implemented on the Cyclone II FPGA device and reaches a throughput of 734.3 Kpixel/s which is appropriate for real-time motion detection. It can process 240×320 frames at 1030 fps. An efficient hardware implementation of the variational HSOF algorithm is presented in [92]. It achieves a throughput of 175 MPixels/s and process Full HD (1920×1080) frames at 60 fps on Virtex 7 FPGA device. The fixed-point architecture achieves the performance of 418 GOPS with power efficiency of 34 GOPS/W whereas as floating-point module achieves 103 GFLOPS, with power efficiency of 24 GFLOPS/W. The proposed module does not require external memory to store intermediate flow vectors between the iterations. It utilizes a separate hardware submodule for each iteration leading to large resource usage. The HSOF implementation with Jacobi solver uses a large number of iterations to get sufficient accuracy. This leads to increased resource utilization and hence more power consumption, thereby limiting the applicability for embedded platforms.

Chen et.al [107] proposed the FPGA implementation of dense OF based on classical Combine-Brightness-Gradient (CBG) model with RBSOR solver. The design is implemented using C rather than HDLs for rapid prototyping. It takes two passes to complete the flow computation. The intermediate flow field from the first pass is buffered in external memory and retrieved during the second pass to complete the flow computation. The optimized implementation Xilinx ZYNQ FPGA-SoC can only process 640×480 images at 1.72 fps while consuming 1.84 watt power. It achieves a $32\times$ improvement in the computation speed as compared to CPU implementation which takes around 2.6s and consumes 35 watt power. This motivated to the design of a high throughput RBSOR solver architecture and its integration in variational HTOF architecture to compute accurate and dense OF with less number of iterations in real-time.

4.1.3 RBSOR Solver Architecture

This section focuses on the design of a novel two-pass hardware architecture of RBSOR solver with a pixel selection unit to maximize the throughput. The conventional RBSOR solver architecture [107] waits until the completion of the red pass to start the black pass, degrading the throughput performance. Instead, the red pass and black pass of the proposed architecture operates simultaneously after a small buffering latency.

The internal architecture of the proposed RBSOR solver for computing the system of

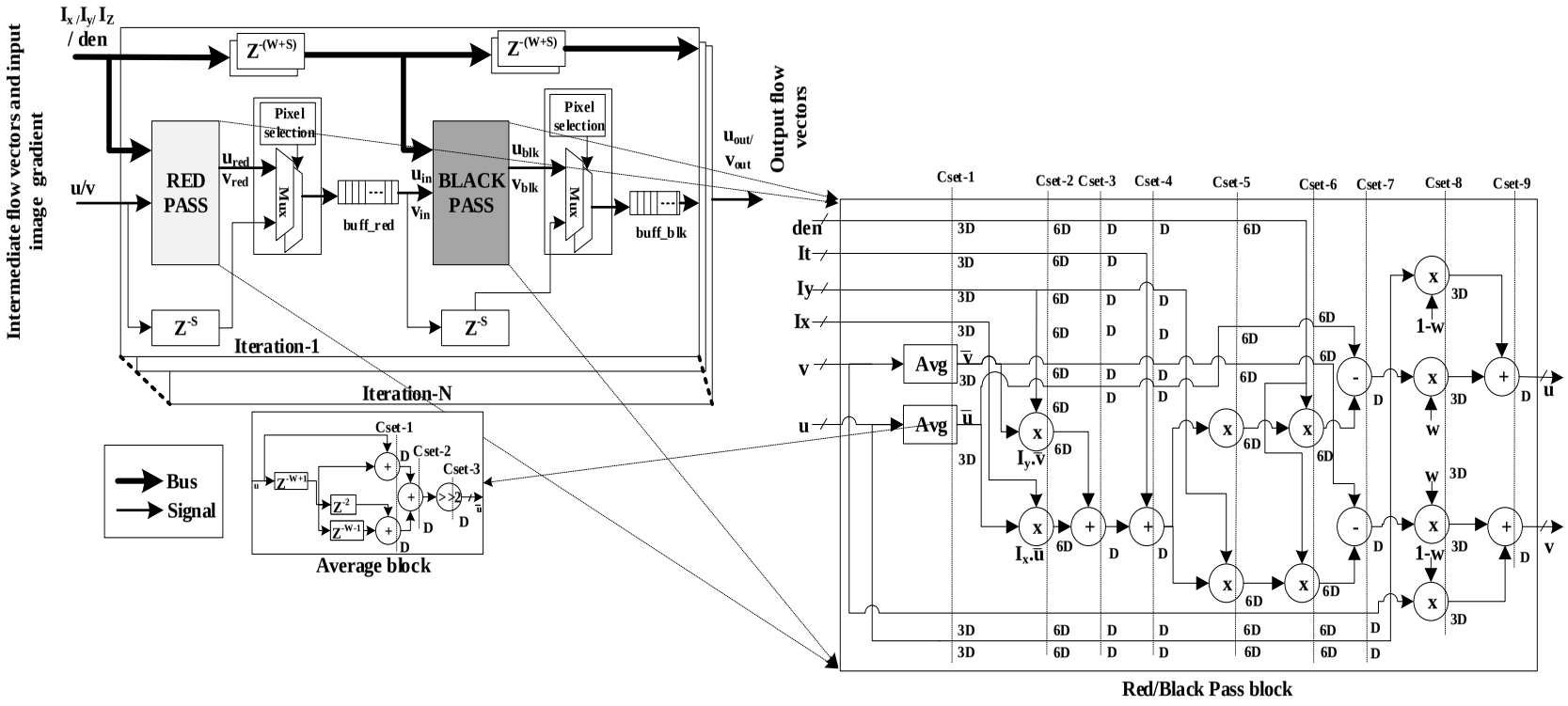


Figure 4.2: RBSOR Solver architecture.

equations (4.6) is shown in Fig. 4.2. The design utilizes superscalar and deep pipelined structures for implementing the most efficient architecture [108]. The RBSOR is a two-pass solver architecture with the red pass processing the odd set of pixels while the black pass processes the flow of even pixel as selected by pixel selection architecture. The Red and Black pass modules share the same circuitry. A pixel selection architecture is used to choose between odd and even pixels in the incoming stream, while the synchronization buffers help to cache intermediate flow values across the two passes to eliminate the need for external data buffering. The proposed scheme uses all four neighbouring pixels for the flow computation by internally caching the intermediate optical flow vectors. From the figure, it can be observed that the input gradients and denominator values are delayed by a line buffer of size $2 \times (W + S)$ ($W + S$ delay cycles for each pass) and fed to the next solver iteration, where S is the processing time of Red/Black pass block and W is the width of the input frame.

For the initial Red pass, the flow values are assumed to be zero. Each of the pass involves a flow averaging operation utilizing 3×3 kernel [1] with a normalizing factor of $1/12$. It uses three daisy-chained line buffers ($W - 1$, $W + 1$ and 2) and a register bank to provide simultaneous access to four adjacent flow values. Once the line buffers are full, the stored values are simultaneously made available at the output of register banks. The tuned value of ω is pre-loaded and used in the constant multiplier. The pixel selection block at the output of each pass decides whether to forward the incoming data or processed values from the pass using two multiplexers (Mux) for two different flow component. The output of the first pass needs to be stored in a red line buffer (buff$_{red}$) so that the next pass at least gets one full line (W) and a pixel to start the black pass computation. Once the buffers are full, the black pass computation is started and the stored values are sent through a pixel selection block and buffered into the black buffer (buff$_{blk}$). Thus intermediate line buffers (buff$_{red}$, buff$_{blk}$) help to provide data synchronization between the red and black passes.

4.1.3.1 Pixel selection architecture

The internal architecture of the pixel selection unit is shown in Fig. 4.3. It determines the incoming data to be processed in the red pass or black pass of the solver iteration. The start of the image frame is used to find the pixel indices of the input data in the streaming architecture. This is propagated to each of the solver iterations and is compared with the solver start value to enable the local counter (En[++]) in pixel selection architecture. The start value for each solver iteration is pre-computed and stored based on the processing time and pipelining latency of the previous modules.

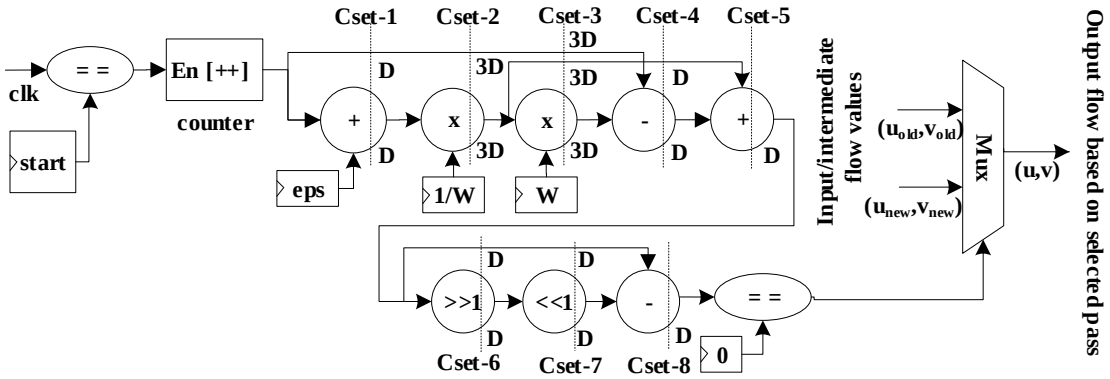


Figure 4.3: Pixel selection architecture.

The modulo operation on the local counter value using input data width (W) generates the row data index which is then fed to a modulo two operator to classify the index into odd or even. The eps is a small positive value accumulated with the local counter to avoid divide by zero condition. The output from the modulo two operation is sent as selection line to the multiplexer to select either the solver result or delayed input values from the previous iteration. If the input data index is even, then that data is labelled as red and during the red pass, this data is updated with the newly computed value. Otherwise, the data is labelled as black and during the red pass, it is kept as the previous value. The solver stage accounts for a total of 16 coarse pipelines, with the average block having a total of 3 coarse and 2 fine grain pipelines.

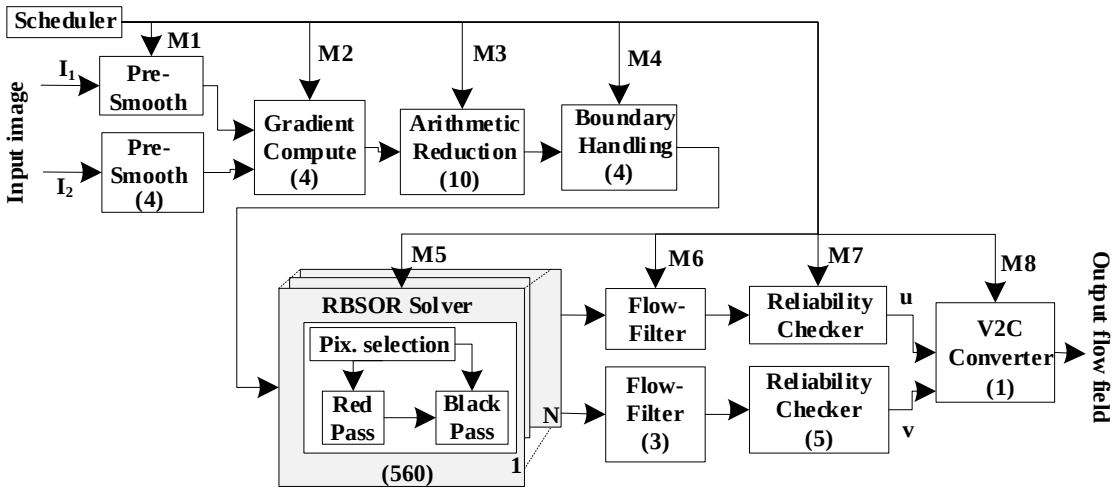


Figure 4.4: Block diagram of HTOF architecture based on RBSOR solver

4.1.4 Proposed High Throughput Optical Flow (HTOF) architecture using RBSOR solver

The block diagram of variational HTOF based on RBSOR solver is shown in Fig. 4.4. The design considers hardware optimization like deep data path pipelining and unfolding of the solver iteration to provide better accuracy with less number of iteration as compared to other iterative solvers. It implements data parallelism in gradient, filtering and reliability checker modules to maximize the throughput. The architecture preserves image boundaries by selectively computing the flow vectors at every location. It also minimizes the resource utilization of entire architecture by moving the complex arithmetic operation in the unfolded solver iterations to the preceding block before the solver iteration. The number in the box corresponds to the depth of pipeline applied to each module in the HTOF architecture.

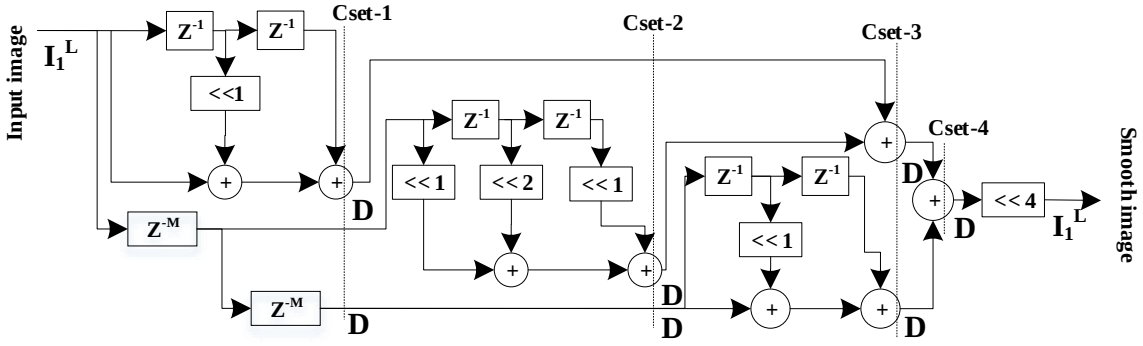


Figure 4.5: Gaussian smoothing architecture.

4.1.4.1 Pre-smoothing module: M1

The input images are pre-smoothed to suppress the high-frequency distortions caused by external environmental effects and internal thermal noise of the sensor. It also helps to reduce the noise sensitivity of the first order spatial gradients. Fig. 4.5 shows the image smoothing architecture of a 3×3 Gaussian filter [109].

It utilizes an image patch converter to convert the one-dimensional input stream into a two-dimensional patch using 2 line buffers of width (W). The two-dimensional streams are fed to averaging unit consisting of 8 adders, 6 shifters and 6 flipflops to get the smooth response. The filter architecture utilizes 4 feed forward cutsets to perform pipelining.

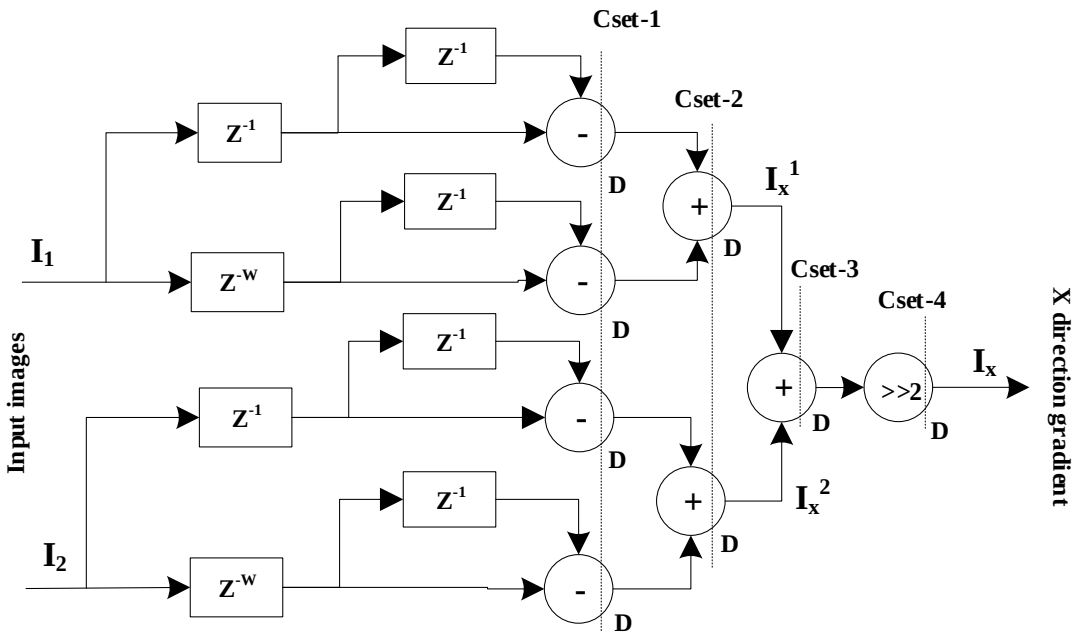


Figure 4.6: X-direction gradient computation architecture.

4.1.4.2 Gradient compute module: M2

The gradient module utilizes three parallel spatiotemporal gradient units to compute x, y and t gradients of the input data streams as shown in Fig. 4.6. It utilizes six line buffers (two for each gradient direction) of width W to perform gradient computation. The gradient architecture utilizes 21 addition/subtraction blocks along with 3 shift registers to implement gradient model based on Robert mask [1]. The architecture is deeply pipelined with 2 coarse and 2 fine cutsets.

4.1.4.3 Arithmetic reduction module: M3

A direct implementation of the unfolded solver architecture needs to compute reciprocal of the equation (4.6) in every solver iteration, which leads to high resource utilization. To mitigate this, the proposed design isolates the reciprocal computation present in every unfolded solver iteration to an arithmetic reduction module prior to solver computation stage. The reciprocal values are forwarded to all the solver iterations using dedicated line buffers as illustrated in Fig. 4.7. The architecture utilizes 4 coarse grain cutsets to perform 44 stage deep pipelines.

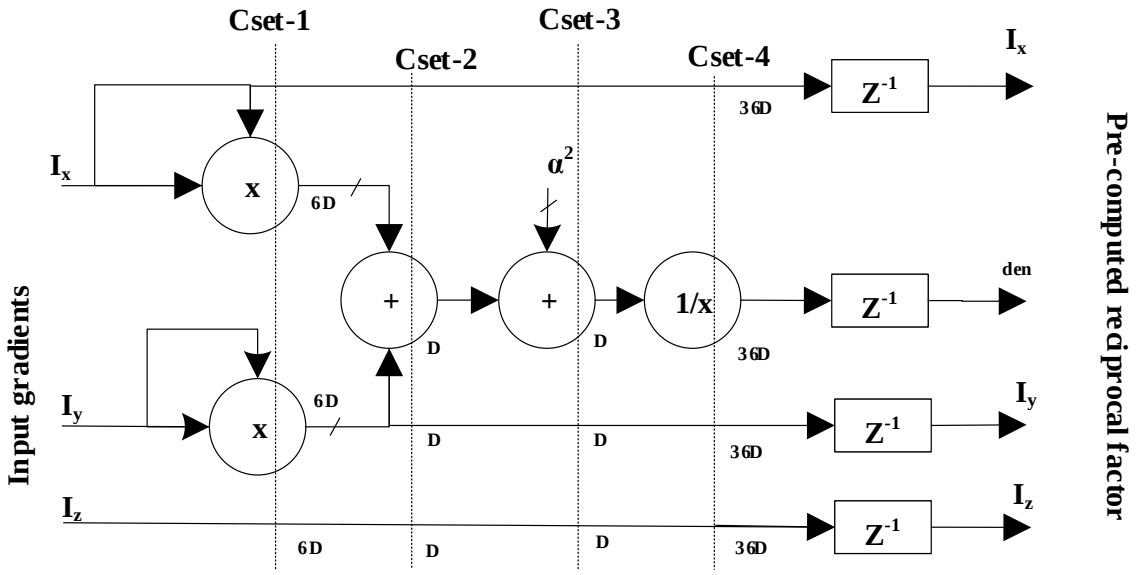


Figure 4.7: Arithmetic compute reduction architecture.

4.1.4.4 Boundary handling architecture: M4

The computation of the gradient in the HTOF architecture is corrupted in the two outer image boundary layers due to boundary effects. Since the corrupted gradient value is used in the N_{solv} iteration of the solver stage, this leads to corruption of N_{solv}-1 outer layers of the flow field. This can be prevented by padding the incoming stream based on the size of the gradient mask but leads to an additional processing latency. Instead, the proposed architecture utilizes an additional control logic to identify the image boundaries and employs a separate set of multiplexer arrays to either bypass the input stream or send a pre-defined value as shown in Fig. 4.8.

4.1.4.5 Solver module: M5

The RBSOR solver architecture described in the previous Section 4.1.3 is utilized for implementing the solver stage. The solver needs to compute OF by solving the equation (4.6) described in the Section 4.1.1. The $1st$ iteration does not involve flow averaging compared to remaining N_{solv}-1 iterations.

4.1.4.6 Flow filter module: M6

The computed flow values contain outliers due to illumination artefacts, occlusions and other discrepancies due to the approximation error in the cost functional 2.4. A two-

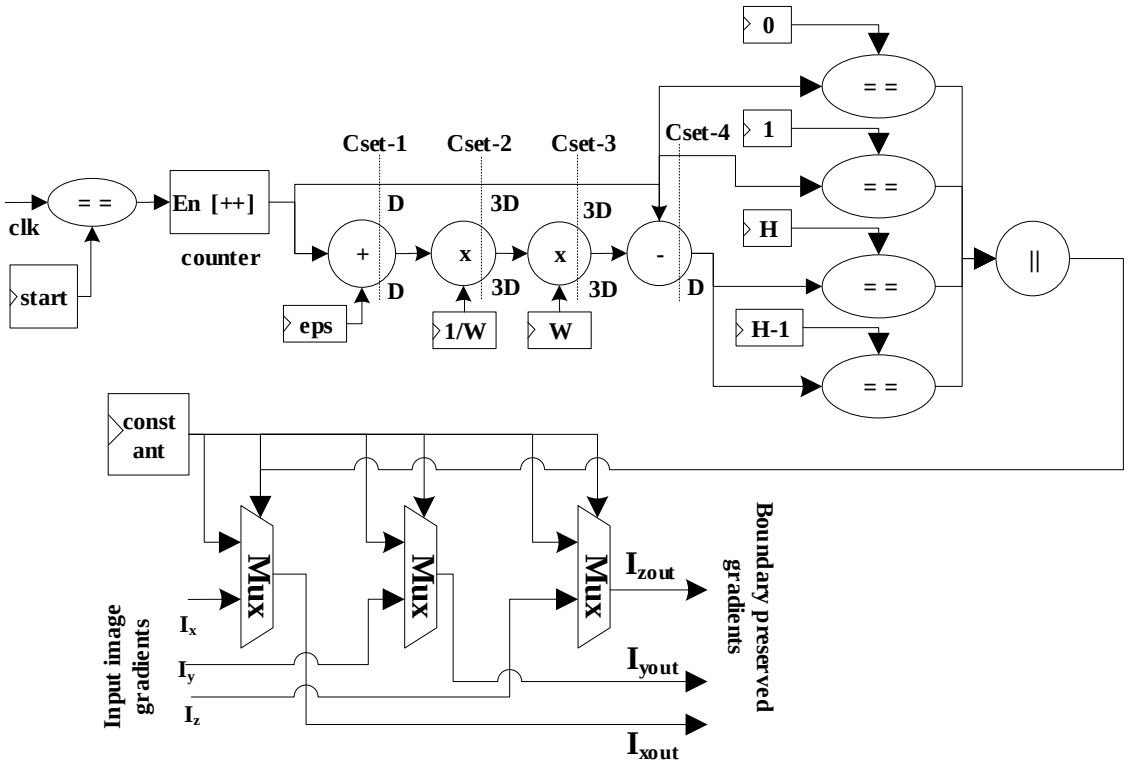


Figure 4.8: Boundary handling architecture.

dimensional median filter of kernel size $K \times K$ described in the previous Section 3.5.6 is utilized to eliminate the randomness in computed flow vectors.

4.1.4.7 Reliability checker module: M7

The reliability checker module performs an additional check on the local flow smoothness by computing the flow variance. If the flow variance of the input patch is higher than the threshold, then the corresponding flow value is truncated. The variance is computed as given in the equation (4.7),

$$\sigma^2 = \frac{1}{W \times H} \sum_{i=1}^{W} \sum_{j=1}^{H} (u(i,j) - \bar{u})^2 \tag{4.7}$$

where m, n corresponds to indices of flow values, $u(m,n)$ represents pixel values and $\bar{u}$ indicates mean of the pixel values over the entire image region. Since the direct computation of equation (4.7) is complex in the hardware, a hardware-friendly flow variance is implemented by subtracting the individual flow from the mean of neighbouring flow values.

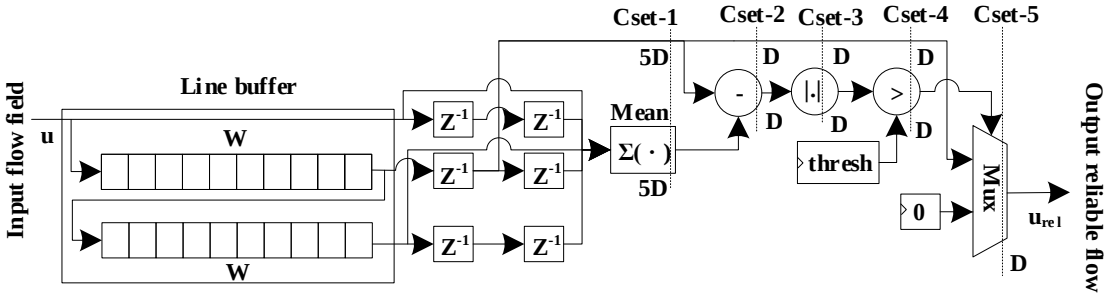


Figure 4.9: Reliability checker architecture.

It replaces the complex standard deviation operation requiring division and multiplication with simple addition and subtraction operation. Fig. 4.9 shows a hardware-friendly architecture of the reliability checker module. It utilizes a stream converter using two line buffers of width W. The mean block computes the average of the neighbours in the two-dimensional patch, subtract from the centre pixel value and is compared with a pre-saved threshold to generate the selection line for the multiplexer. The architecture uses 5 cutsets for performing the pipelining.

4.1.4.8 Vector to Colour (V2C) converter module: M8

The V2C module encodes the flow values into RGB data which is used to display in an HDMI monitor. The flow is converted to RGB components using a simpler formula, with the red (green) component is computed by adding $u(v)$ with 128 and the blue component is obtained by subtracting u values from 255. The computed values should lie in the considered range $[-128 : 127]$, for any values above or below the chosen threshold are saturated.

4.1.4.9 Hardware Design Variants

The work proposes a High Precision Optical Flow (HPOF) architecture of HTOF algorithm. This variant is single precision floating point architecture providing high accuracy but at the cost of increased resource utilization.

4.1.5 Hardware implementation Results

The proposed architecture is synthesised and implemented on Xilinx Virtex-$VC707$ board using Vivado design tools (2017.2). An unoptimized version of the variational HSOF algorithm with 10 RBSOR solver iteration on a single core Intel CPU $i5$-$M460$ running at 2.53

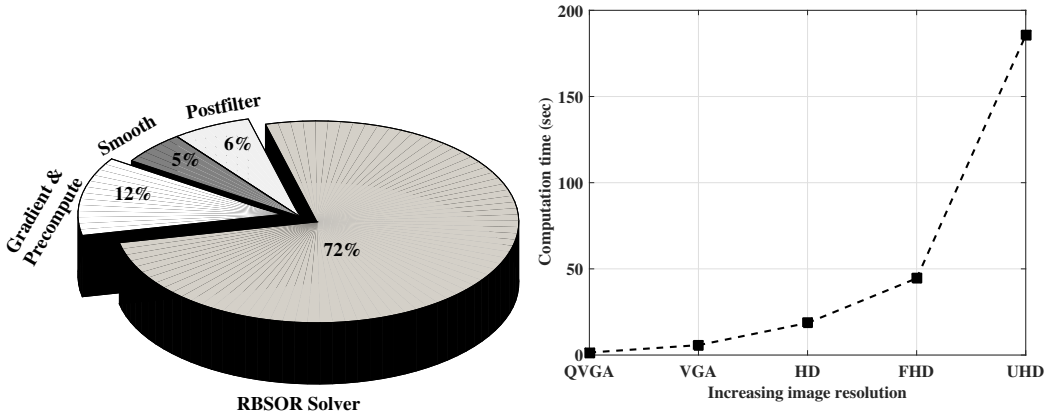


Figure 4.10: Block diagram of (a) performance results and (b) computation cost for different image resolutions.

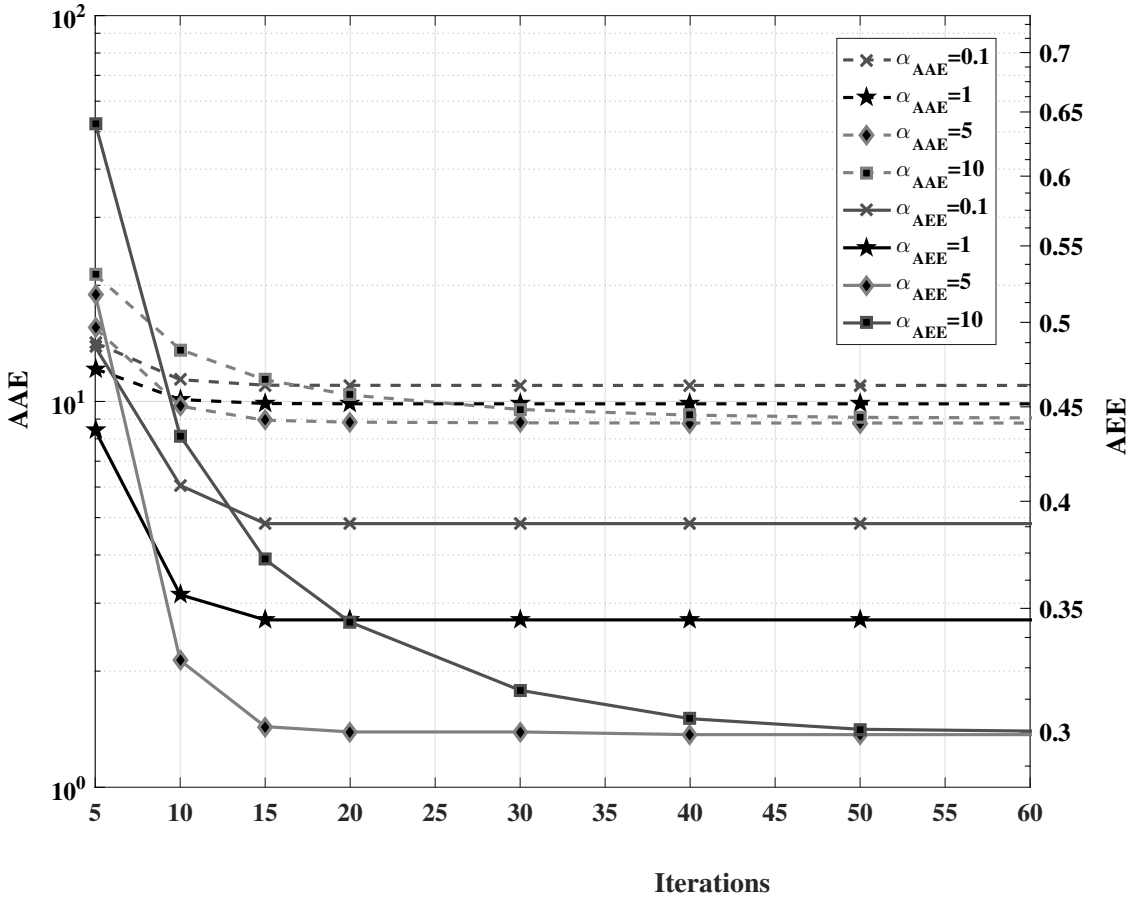


Figure 4.11: Variation of AAE for different α.

GHz with 4 GB RAM takes 3.26 sec for a 512×512 resolution image. Fig. 4.10 (a) illustrates the amount of time spent on each of the processing stage by profiling the algorithm, of which solver is the most time-consuming part. Fig. 4.10 (b) shows the computation time of the HSOF algorithm for different image resolutions.

4.1.5.1 Parameter analysis

The design is analysed to find the dependencies and effects of various parameters such as computation time, iterations and accuracy on throughput performance and quality of the HSOF architecture. The parameter N_{solv} controls the accuracy of the algorithm. Since N_{solv} is inversely proportional to the AAE or AEE, the computational accuracy improves with N_{solv}. Another parameter α controls the smoothness of flow vectors, whereas ω controls the rate of convergence. In order to analyze the effect of α on AAE or AEE and number of iterations, a graph is plotted between AAE and N_{solv}, with different alpha while keeping ω as constant depicted in Fig. 4.11. For different α values, the AAE/AEE reaches a constant minimum value within 10 to 15 iterations as observed from the graph.

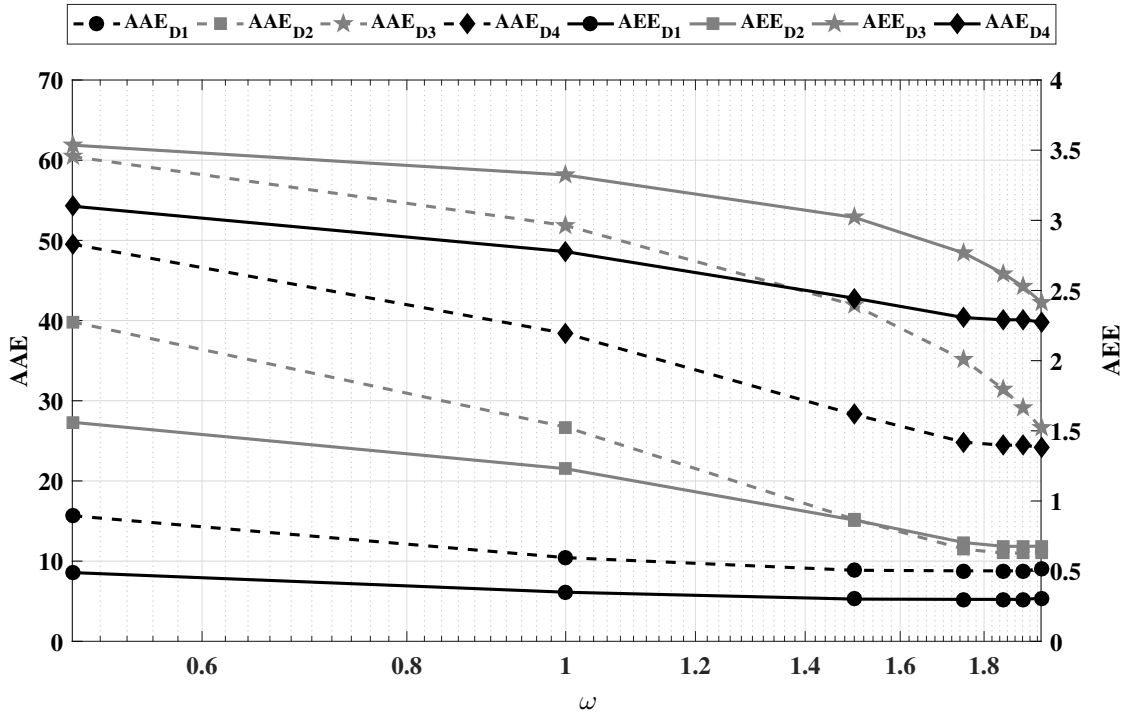


Figure 4.12: Variation of AAE for different convergence factor (ω).

Similarly Fig. 4.12 shows the rate of convergence of the proposed HTOF architecture for different ω values with tuned α value for each dataset. It can be observed that for most

cases the AAE/AEE settles to a minimum value for $\omega = 1.7$.

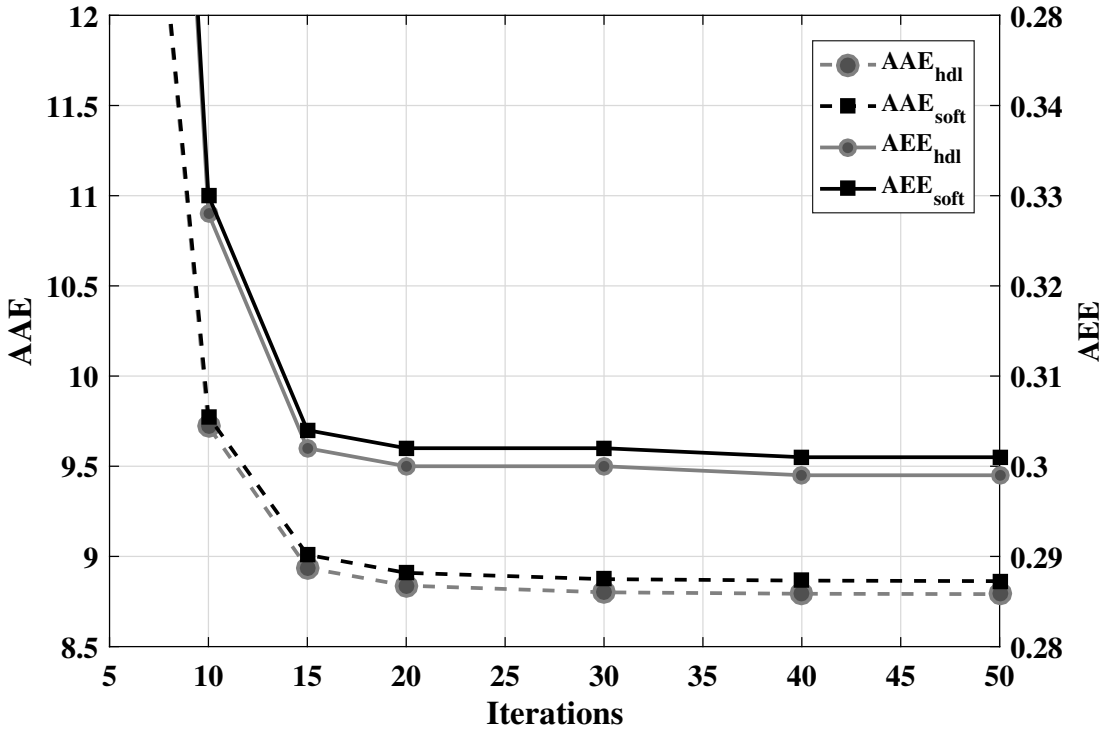


Figure 4.13: Performance comparison between software and hardware error performance.

4.1.5.2 Impact of Numerical representation

The selection of fixed or floating point implementation of HSOF algorithm plays an important role in the trade-off between accuracy and power consumption. The floating point algorithm shows lesser error for a fixed number of iteration but is high on resource utilization. The fixed point representation opens a large space for the exploration of the suitable bit width requirement for each of the stages. The conversion from floating point to fixed point results in quantization effects. The software algorithm with these inferred parameters is converted to a fixed point Verilog architecture with the quantization error as depicted in Fig. 4.13. The highest bit width required in each submodules of the architecture is given in the Table. 4.1.

4.1.5.3 Resource and performance analysis

Table. 4.2 shows the resource utilization, maximum operating frequency and power dissipation of architecture variants at the submodule level while processing FHD (1920×1080)

Table 4.1: Selected bit-width for different modules.

	M1	**M2**	**M3**	**M4**	**M5**	**M6**	**M7**
$\mathbf{BW}_{min}$[1]	8	14	22	14	28	18	18

[1] represent the maximum number of bits utilized for each stage.

sequence. It can be inferred from the table that, $M7$ stage uses a minimum number of slices but consumes higher power than $M3$ due to the more number of BRAMs utilized for implementing line buffers. In the HTOF variant, the $M5$ stage with 1 solver iteration has the highest resource utilization stage, as it makes use of a large number of DSP48 slices for implementing the complex arithmetic. Unlike other stages, $M5$ has high BRAMs utilization due to large buffers required for intermediate data caching which results in consuming significantly more power than other stages.

Table 4.2: Resource utilization of the proposed hardware variants.

Modules / **Resource**	**M1**		**M2**		**M3**		**M4**		**M5**[1]		**M6**		**M7**	
	HPOF	HTOF	HPOF	HTOF	HPOF	HTOF	HPOF	HTOF	HPOF	HTOF	HPOF	HTOF	HPOF	HTOF
FlipFlop	2123	255	1806	374	622	254	2061	2250	7953	2310	2260	880	1065	71
LUT	5448	68	8452	192	1887	178	3483	1024	22054	1415	2088	790	3714	19
Slice	1657	48	2914	104	563	67	1131	461	7055	639	642	256	1265	18
BRAM36	4	0.5	8	1	0	0	0	0	40	10.5	4	1	4	0.5
DSP48	0	0	0	0	0	0	6	2	36	16	0	0	0	0
Fmax (MHz)	102	480	110	470	120	480	110	480	105	460	310	490	102	490
Power (W)	0.300	0.308	0.428	0.459	0.229	0.270	0.290	0.558	0.691	0.948	0.284	0.356	0.295	0.314

[1] represents the resource utilization of M5 stage with one solver iteration.

Further, the resource utilization of proposed HTOF architecture with different iterations is shown in Table. 4.3, ignoring the HPOF due to the resource unavailability for a higher number of iterations. From the table 4.2 and 4.3, it can be observed that each stage of HTOF can operate to a maximum frequency of 490 MHz but gets reduced to 350 MHz due to large utilization and interconnection. The unavailability of DSP and BRAM limits the maximum possible iterations. The critical path in RBSOR solver iteration loop limits the overall performance. Even though a large number of iterations will slightly improve architecture accuracy, fewer numbers of iterations are sufficient for achieving reasonable

Table 4.3: Resource utilization of HTOF architecture for different number of iterations.

Iteration / Resource	10	15	25	35	50
FlipFlop	26184	37821	58959	80809	111398
LUT	18378	27197	42373	58932	81536
Slices	7529	11123	17259	23756	33338
BRAM36	209.5	314.5	524.5	734.5	1028.5
DSP48	157	237	394	557	781
Fmax (MHz)	430	426	417	400	350
Latency (μs)	95	142	238	346	560
Power (W)	6.471	9.45	15.15	20.99	30.24

accuracy.

Table 4.4: Performance comparison of variational OF architecture on different FPGA devices.

FPGA / Resource	V6-Util		Avail	V7-Util		Avail
	HPOF-5	HTOF-10	-	HPOF-10	HTOF-15	-
FlipFlop	50994	39803	301440	90764	37821	607200
LUT	125642	34343	150720	228385	27197	303600
Slice	34241	11977	37680	63558	11123	75900
BRAM36	212	304	416	412	314.5	1030
DSP48	177	237	768	357	237	2800
Fmax (MHz)	70	240	-	90	426	-
Latency (μs)	306	170	-	455	142	-
Power (W)	15.14	10.13	-	19.32	9.45	-

A Virtex-6 FPGA based implementation shown in Table. 4.4 helps to study variation in the resource, power and operating frequency when choosing a lower FPGA device family. The highly pipelined HTOF architecture achieves 426 MHz in Virtex-7 FPGA, this corresponds to a reduction of maximum operating frequency by 44% as compared to the Virtex-6 implementation. The HPOF implementation also shows a similar pattern as in the case of Virtex-7.

The performance of HTOF architecture is compared with HPOF in Table. 4.5. The implementation utilizes 10 iterations of RBSOR solver with each iteration containing a

Table 4.5: Accuracy of the variational OF architecture on Middlebury database.

	Rubberwhale (D1)		**Dimetrodon** (D2)		**Venus** (D3)		**Hydrangea** (D4)		**Grove 2** (D5)		**Grove 3** (D6)	
	HPOF	HTOF	HPOF	HTOF	HPOF	HTOF	HPOF	HTOF	HPOF	HTOF	HPOF	HTOF
AAE	8.54	8.79	10.67	10.99	26.12	26.68	25.23	25.56	26.88	27.08	26.64	26.89
AEE	0.28	0.29	0.63	0.65	2.34	2.43	2.23	2.34	1.56	1.66	2.53	2.62

red pass and a black pass. The parameters of the architecture are tuned to minimize the AAE/AEE errors. The smoothing parameter α was set to 3 and the convergence factor w was set to 1.75. It can also be inferred from the table that a fixed-point HTOF design shows negligible accuracy loss as compared to HPOF architecture having high resource utilization due to a large number of float/fixed conversions. As observed from Table 4.5, the HPOF variant provides slightly better flow accuracy than HTOF, it's high resource utilization and low throughput makes it less preferable for embedded and portable applications. Hence HTOF architecture is used for further analysis.

4.1.5.4 Power analysis

The architecture complexity is evaluated in terms of the number of fixed or floating point operations performed in unit time. The design has several super-scalar units across different stages to improve throughput of which, stage $M1$ has two parallel smoothing filters, $M2$ utilizes 9 parallel units for gradient computation, $M3$ has 9 units for computing solver coefficients, $M4$ has 9 units for the solver, $M5$ has 6 units for flow filtering and $M6$ has 3 parallel units for reliability computation. In addition, some of the internal operations and data paths are computed in parallel to improve the throughput.

The $M1$ stage reads two pixel information and performs 14 operation which include 8 addition and 6 shift operations, gradient computation stage $M2$ involves 24 operations, the arithmetic reduction stage $M3$ include 2 multiplication, one addition and a single reciprocal operation, boundary handling stage $M4$ involve 13 operations, 15 iterations of RBSOR solver $M5$ stage includes 1080 arithmetic operations, median filter stage $M6$ involves 30 logical operation and 55 multiplexers, and the reliability checker stage $M7$ constitutes 13 operations.

Thus the HTOF architecture accounts for a total of 1234 fixed-point operations. The processing of HTOF architecture on a UHD sequence reaches a maximum throughput of 491 Giga Operations Per Second (GOPS) while consuming 11.27 watts at 412 MHz. This

leads to a power efficiency of 43 GOPS/Watt. The major source of power dissipation is the solver module containing 15 iterative pipeline stages. The increased number of BRAM accounts for the major part of the dynamic power of HTOF architecture. The static power remains the same and is dependent on resource utilization and the operating frequency. Since the processing time of the HTOF is drastically reduced, net energy consumption also reduces significantly.

Table 4.6: Performance of the HTOF architecture for various image resolutions.

Resolution / Resource	QVGA	VGA	SVGA	HD	FHD	UHD
FlipFlop	35598	36395	36979	37114	37821	37823
LUT	25848	26278	26644	26409	27197	26891
Slices	9712	9860	9938	10825	11123	11384
BRAM36	166	180.5	180.5	314.5	314.5	467
DSP48	237	237	237	237	237	237
Fmax (MHz)	442	440	438	430	430	412
Latency (μs)	26	47	58	94	140	291
Power (W)	6.16	7.714	7.77	9.34	9.45	11.27

4.1.5.5 Design scalability

The design is directly scalable to support a multitude of resolutions as shown in Table. 4.6. As the resolution increases DSP48 utilization remains constant but leads to an increase in BRAMs for internal buffering. For a small increase in image resolution, the BRAMs show a slight increase due to the presence of unused regions in the memory bank.

4.1.6 Comparison with other state-of-the-art architectures

Table. 4.7 shows the performance improvement of the HTOF with RBSOR solver architecture over state of the art methods. The proposed architecture with several adders and multipliers using structural code, utilize deep pipelines (fine and coarse grain) to minimize the critical path to achieve a $F_{max} \gg 400$MHz. A high-speed interface like PCIe, HDMI or Display Port (DP) with a bandwidth of more than 6.5Gbps can feed data to the proposed architecture. Due to the streaming nature of the architecture, a similar high-speed interface is required in the output to transmit the computed flow to the external devices. Among the

Table 4.7: Comparison of variational OF architecture with state of the art methods.

Method / Resource	HTOF	Bahar[106]	Barranco[88]	Tomasi[87]	Botella[86]	Maha[41]	Diaz[85]	komo[92]	Chen[107]	Martin[105]
Device	Virtex-7	Cyclone-II	Virtex-4	Virtex-4	Virtex-V2	Virtex-V2P	Virtex-V2	Virtex-7	ZYNQ	APEX
Model	HSOF	HSOF	LK	LK	McGM	LK	LK	HSOF	HSOF	HSOF
Slices	11384	2708	4128	36603	19200	11086	9123	137049	51072	11520
BRAM36	467	28	48	106	102	20	20	1346	130	8
DSP48	237	12	50	132	109	23	12	-	114	72
Max.Res[1]	3840×2160	240×320	640×480	640×480	128×96	640×480	800×600	1920×1080	640×480	256×256
FPS[2]	48	1029	270	31.5	16	30	170	62	1.72	60
Fmax (MHz)	412	-	44	45	-	55	82	129	83	-
Latency (μs)	291	-	-	-	-	-	120	-	-	-
Power (W)	11.27	-	-	4.35	-	-	-	12.22	1.85	-
Speedup	1	-	9.36	9.3	-	7.5	5.01	3.2	4.96	-
Area	1	14.49	6.71	1.96	2.63	8.48	10.62	0.37	1.65	6.28
FFDelay	1	0.81	0.72	0.72	0.57	0.42	0.57	0.58	0.50	0.81
S_{AN}	1	-	1.39	4.77	-	0.88	0.47	8.64	3.06	-
S_{ADN}	1	-	1.01	3.43	-	0.37	0.27	5.1	1.50	-
Iteration	15	-	1	1	-	1	1	128	-	-
AAE	6.82	-	4.55	7.91	5.5	6.37	7.86	16.51	18.56	-
Density (%)	100	-	58.5	92.01	100	38.6	57.2	100	100	-

[1] represents the maximum supported resolution.
[2] denotes the frames per second.

existing methods, the proposed HTOF architecture achieves highest frame-rate of 48 fps for UHD frames with minimal inter-frame latency. Also, it shows the computation accuracy of different OF algorithms with flow density on standard Yosemite sequence without clouds. The proposed HTOF architecture achieve reasonable accuracy with 100% density while utilizing a lesser number of iterations in comparison with other architectures. The proposed HTOF architecture shows a peak improvement of $8.64\times$ for S_{AN} and $5.1\times$ for S_{ADN} as compared to HSOF implementation based on Jacobi solver [92]. Since the LK algorithm is sparse and non-iterative, the area utilization is quite less compared to the HSOF architectures. Hence the S_{AN} and S_{ADN} doesn't show the significance of HTOF over LK methods. In comparison with the existing HSOF architectures, the HTOF achieves at least $3.06\times$ and $1.5\times$ improvement in S_{AN} and S_{ADN} respectively. The real-time evaluation system based on the proposed architecture is implemented using a Xilinx sample video processing pipeline, which results in a much lower speed of operation. The implemented system operating at 110 MHz is capable of processing FHD frames at more than 45 fps meeting the real-time requirement.

4.2 Proposed Bilateral filter architecture

4.2.1 Introduction

Type of intermediate flow filter in the variational multi-scale OF algorithm with a less number of solver iteration is an important design consideration to improve the accuracy of the computed flow [110]. Since errors in the flow field are generated due to edges, outliers and occlusions, it is necessary to develop smoothing techniques that would preserve the motion boundaries in the calculated displacement field. Using a suitable filter such as the median filter to post-filter the intermediate flow field during incremental estimation and warping is an effective way to remove outliers [95]. The median filter is not good at handling occlusions, instead a BF [111] with the edge-preserving properties [112, 113] is effective in treating occlusions [114].

The integration of the BF in the variational multi-scale OF algorithm [115] helps to improve the accuracy of the computed flow field by denoising the displacements fields at each pyramid layer. It ensures the removal of outliers that may appear during the calculation of the flow field. The high computational complexity is a well-known limitation of the BF [116]. This motivates the design of a high throughput parallel-pipelined architecture of original BF.

4.2.2 Related Work

The existing literature for BF is classified mainly in two directions. The first set of approaches try to improve the performance of true or original BF by incorporating advanced optimization strategies. The second approach implements an approximated variant of BF which has higher performance but deviates from the original BF solution. Durand et al. [113] uses a piecewise linear approximated spatio-tonal BF, suitable for large kernel size but suffers from large memory requirement. A modified BF in [117] makes use of the separability property to fasten up the execution. The work in [118] proposes a Bilateral grid, which has a compact three-dimensional data structure that combines the two-dimensional position of the image spatial domain with the intensity of the reference image. The spatial sampling controls the amount of smoothing, while the range controls the degree of edge preservation. The CPU implementation of Bilateral grid using fewer input data points help to achieve performance in the order of one second, while modern graphics hardware can achieve in order of few milliseconds. In [119, 120], the BF is implemented using trigonometric range kernels which allows to linearise the BF. A recursive approximation of BF is proposed in [121] decomposes the range filter into recursive products and adopt any spatial filter that can be implemented recursively.

A Xilinx system generator implementation of BF with varying kernel size up to 15×15 with minimal reduction in picture sharpness is proposed in [122]. An FPGA implementation of BF for real-time stereo background subtraction processing 320×240 at 30 fps is proposed in the work [123]. A fully parallel modified BF architecture operating at 159 MHz based on photometric filter alone is proposed in [124]. The work in [125, 126] discusses the importance of choosing the optimal parameters for the standard deviation of the range filter which need to be in agreement with the noise in the input image. The work [127] propose a BF architecture using a kernel size of 5×5, operating four times faster than the input sampling rate but leads to poor performance for high-resolution images in real-time. In [128], a BF for real-world interactive medical application is implemented using high-level synthesis tools and shows the superiority of design over GPGPU implementation.

Tseng et al. [129] show an efficient and scalable design of BF using integral histogram-based and joint BF by memory reduction and architecture design techniques. This design can process FHD frames at 60 fps using 356 K gates and 23 KB memory. The work proposed in [130] implements a heavily parallel and deeply pipelined stereo computation co-processing system, stores the data locally in shift registers thereby achieving a faster throughput of 1024×768 at 15 fps. The work [131] discusses the importance of a software-

hardware co-design of the modified BF utilizing trigonometric range kernel. Another approximated BF architecture is proposed in [132], operates at a maximum frequency of 450 MHz with one pixel per clock cycle.

From the aforementioned work, it is observed that most of the high throughput implementation of BF architecture is based on some form of approximation to the original BF algorithm. Also, the performance of the existing BF architectures will get affected by varying noise level in the input data. This motivated to the design of a high throughput architecture for true BF which can adapt to varying noise density of the incoming data. Further, the work proposes several hardware architecture variants in accordance with the varying application requirements.

4.2.3 Algorithm formulation

Consider an input flow field $u(x, y)$, corrupted by additive white Gaussian noise of zero mean and standard deviation σ^2 to generate a noisy field $f(x, y)$ as given in equation (4.8).

$$f(x, y) = u(x, y) + AWGN(0, \sigma^2) \tag{4.8}$$

The bilateral filter denoises $f(m, n)$ while preserving edges to get the denoised flow field $\hat{g}(x, y)$, based on the equation (4.9).

$$\hat{g}(x, y) = \frac{1}{r(x, y)} \sum_{i \epsilon \Omega} \sum_{j \epsilon \Omega} [\phi_{rang}(x, y) \times \phi_{dom}(x, y) \times f(i, j)] \tag{4.9}$$

Where m, n represents the input data coordinates, i, j are the coordinates of the flow values in the filter neighbourhood Ω. σ_r and σ_d denotes the standard deviation of range and domain filters respectively. BF suppresses the uncorrelated noise in a uniform region by enforcing photometric and geometric variability of range $\phi_{rang}(x, y)$ and domain filter $\phi_{dom}(x, y)$ stages as given in equation (4.10) and (4.11) respectively.

$$\phi_{rang}(x, y) = exp(-\frac{(\|f(x, y) - f(i, j)\|)^2}{2\sigma_r^2}) \tag{4.10}$$

$$\phi_{dom}(x, y) = exp(-\frac{(\|x - i\|)^2 + (\|y - j\|)^2}{2\sigma_d^2}) \tag{4.11}$$

Weight estimation of the range filter is based on the amount of dissimilarity between the centre and its neighbouring values in the considered data patch. The domain filter performs

weighted averaging of nearby values based on a geometric separation between the central and neighbouring pixels. A regularizer $r(x, y)$ preserves the dynamic range of the output to be same as input by computing the sum of products of range and domain filter coefficients as in equation (4.12).

$$r(x,y) = \sum\sum[exp(-\frac{(\|x-i\|)^2 + (\|y-j\|)^2}{2\sigma_d^2}) \cdot exp(-\frac{(\|f(x,y) - f(i,j)\|)^2}{2\sigma_r^2})] \tag{4.12}$$

4.2.4 System Overview

The high-level view of the proposed true BF architecture is shown in Fig. 4.14. The architecture is modularized into five sub-blocks: a) Segment creator, b) Range filter, c) Domain filter, d) Regularizer and e) Running variance. The input stream buffer converts the one-dimensional flow vectors into two-dimensional patch of size $K \times K$. This is achieved by internally caching the flow values using line buffers of depth $2 \times W$. The number of stream buffers depends on the size of the kernel $(K-1)$. The segment creator groups the K input streams into P parallel segments $S_0, S_1..S_{P-1}$.

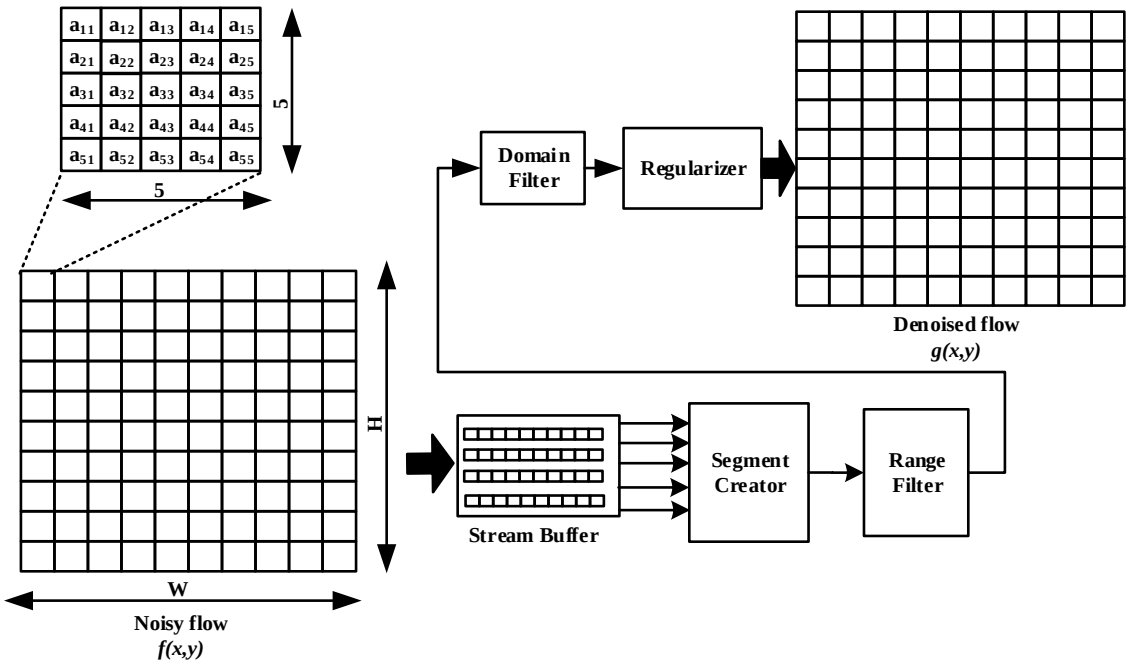


Figure 4.14: Block Diagram of BF denoising system

The P parallel segments are fed to the range filter in N_{cyc} clock cycles, which is required to denoise the complete patch. The range filter has P parallel path to compute the absolute

difference between flow values of the centre and neighbouring locations. The flow differences are Gaussian weighted and fed to a domain filter. It performs a geometric separation aware Gaussian smoothing on range read-outs. The separability and symmetric property of Gaussian coefficients provide a cost-effective implementation of the edge-aware smoothing. A regularization stage at the output of BF normalizes the filter response to be the same as the input dynamic range by finding the ratio between accumulated filter responses and sum of coefficients. The proposed system has very low latency in the order of few input data lines ($2 \times W$) along with latency due to pipelining.

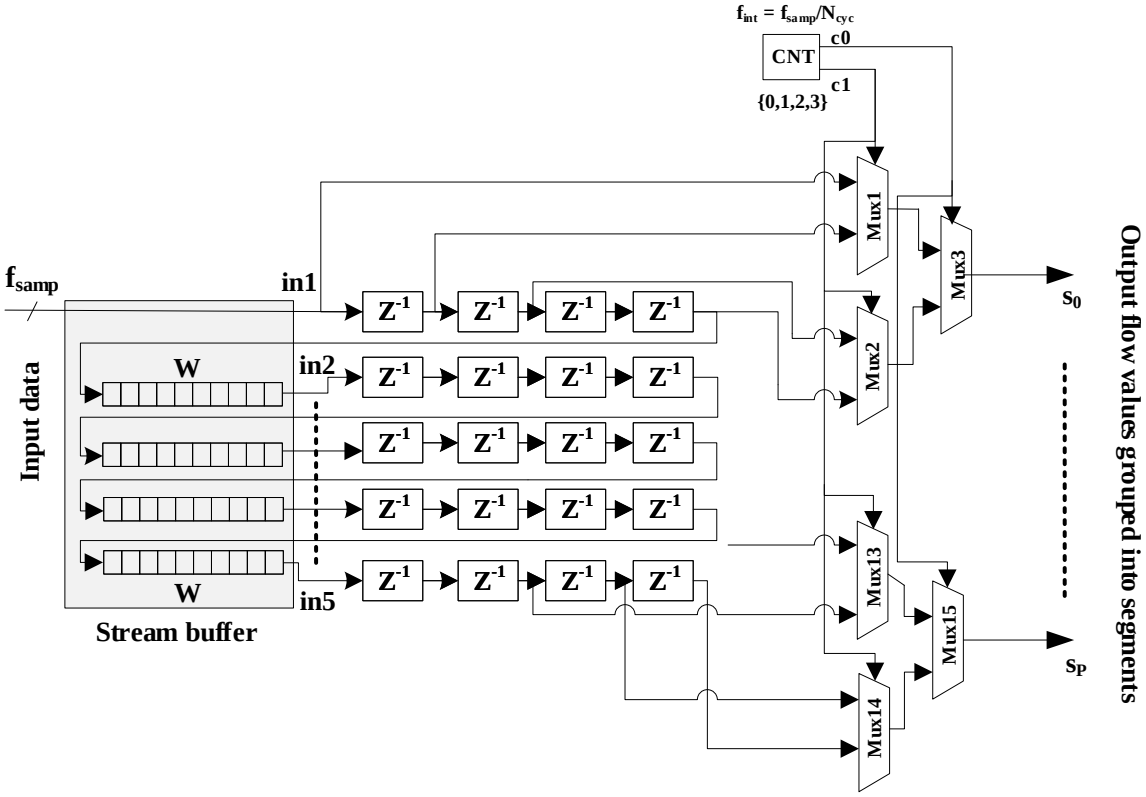


Figure 4.15: Segment Creator Architecture

4.2.4.1 Segment Creator

Fig. 4.15 shows the internal architecture of the segment creator module. For every input value, the segment creator generates $K \times K$ output values which is grouped into P ($S_0, S_1..S_{P-1}$) parallel segments by operating the multiplexer (Mux) at a lower internal clock frequency $f_{int} = \frac{f_{samp}}{N_{cyc}}$ than the input sampling clock frequency f_{samp}, where P is given by $\frac{K^2-1}{N_{cyc}}$. This results in reduction of throughput by a factor of $\frac{1}{N_{cyc}}$.

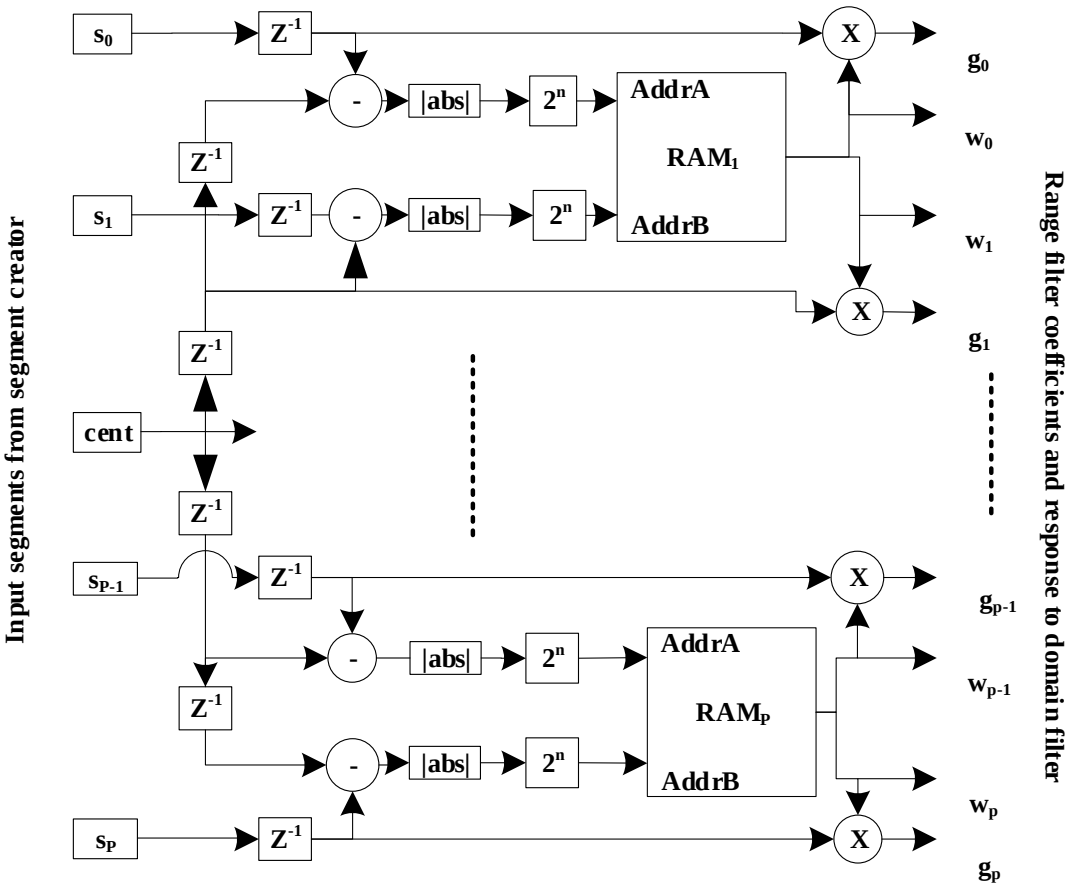


Figure 4.16: Range Filter Architecture

4.2.4.2 Range filter

The range filter computes the similarity between the centre and neighbouring flow values. The range filter coefficient is a function of input flow values and hence needs to be computed dynamically. Since the hardware cost for the coefficient computation of range filter is high, a look-up table initialized with all the possible range coefficients values is utilized for exponential and square operations. The range filter contains P parallel paths for computing average similarity as shown in Fig. 4.16.

The input flow values are subtracted from centre flow value and fed to an absolute operator block ($|abs|$) to compute the magnitude of difference. It is then upscaled by a post scaling factor (2^n) to map to the address range of filter coefficient memory. Since each path requires independent access to the coefficient lookup table, a single dual port memory is shared across two parallel paths to allow simultaneous access and thereby reducing the memory requirement. The depth of the lookup table is kept minimum, as most of the other

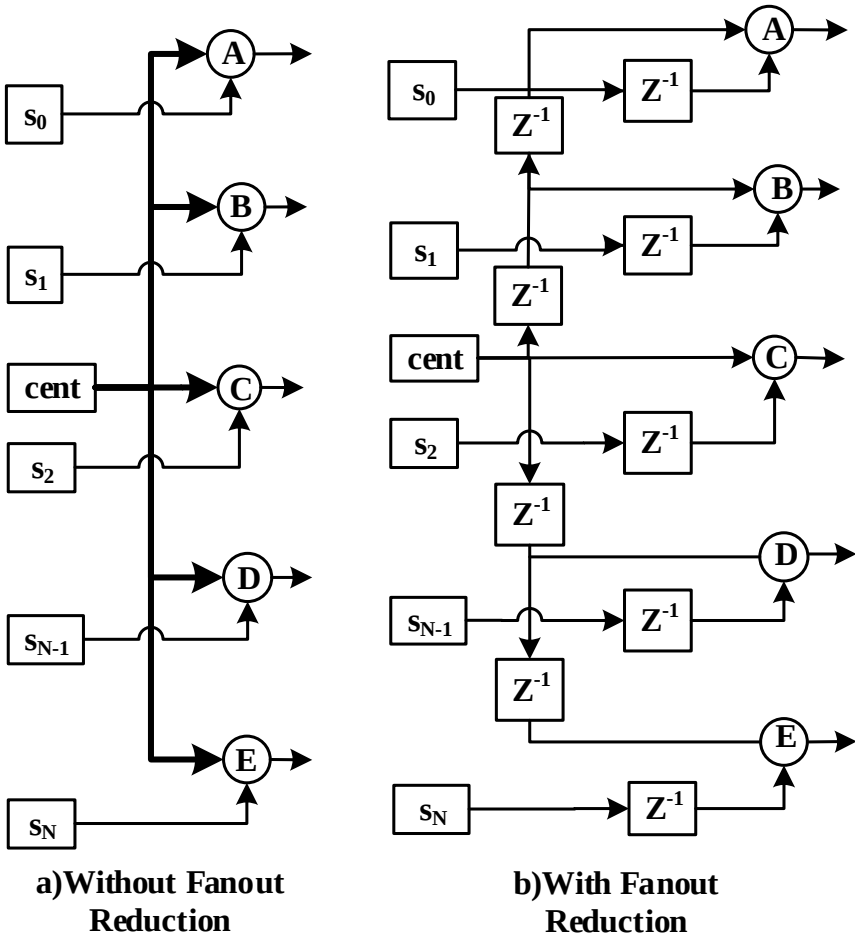


Figure 4.17: Fanout Reduction Technique

coefficients are zeros or negligible.

The centre value drives P parallel subtraction (SUB) loads to compute the absolute difference. This leads to high fanout as shown in Fig. 4.17 (a), resulting in the reduction of the maximum frequency of operation. Hence a delay tree structure is utilized to overcome this issue as shown in Fig. 4.17 (b). Delay blocks are placed extensively to ensure that fan out of each block is less than 2. The net delay of each path is balanced to ensure the proper working of the modules. These additional delay elements in each path are re-timed across the preceding blocks for better performance. The Fig. 4.18 shows the 10 stage data-path pipelining of the Range filter. The design utilizes five cut-sets (1 coarse and 5 fine-grained) to break the critical path and thereby improving the sampling frequency f_{samp} at cost of 9

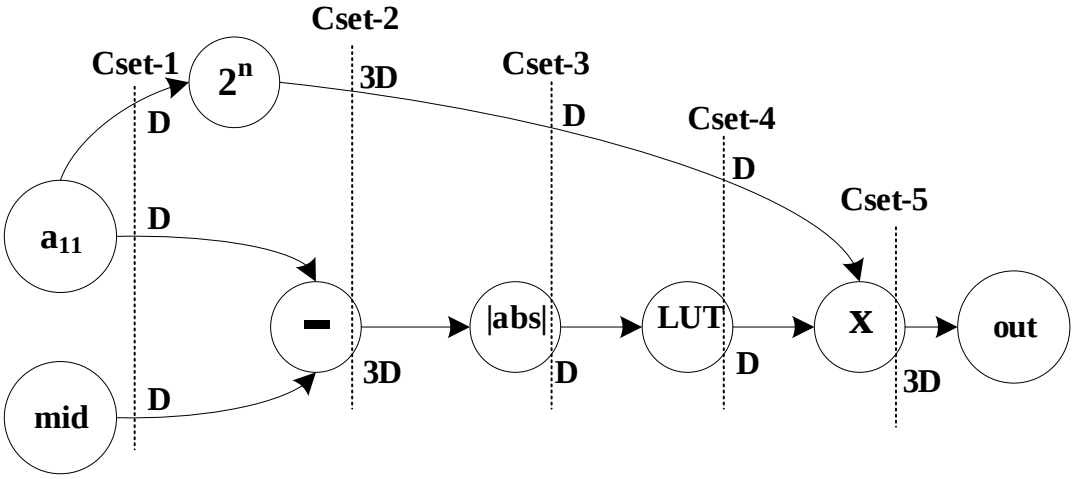


Figure 4.18: Data Flow Graph (DFG) of a single processing path of the Range Filter

additional latency.

4.2.4.3 Domain Filter

The domain filter denoises the flow values by selective averaging. The complexity of the two dimensional convolution with $K \times K$ kernel is reduced by utilizing two separable one dimensional convolutions, using $K \times 1$ and $1 \times K$ kernels. The separable coefficients of the domain filter are denoted by c_0, c_1, c_2, c_3, c_4 for a kernel size ($K = 5$) as shown in equation (4.13).

$$y = \begin{bmatrix} c_0 & c_1 & c_2 & c_3 & c_4 \end{bmatrix}_{1\times K} \tag{4.13}$$

The filter is symmetric with the centre value (c_2) as unity. The flow values which are symmetric can be added before multiplication. The segments from the middle column or middle row are multiplied only once because the coefficient c_2 will be 1 in the column and row section. Thus symmetry and separability help in reducing the total number of constant multiplication (implemented using ROM) in column and row sections from $2K$ to $K - 1$ but at an additional cost of two adder units as shown in the Fig. 4.19. All the parallel data-paths converges into a single path operating at higher clock frequency f_{int} = $4 \times f_{samp}$. The output rate is scaled to the input sampling rate by interleaving every $D = 4^{th}$ sample using down sampler block ($\downarrow$ D) which is effectively a DFF with enable signal. The Fig. 4.20 shows a 24 stage data-path pipelining of the Domain filter. The design utilizes 7 fine-grained cut-set to break the critical path at the cost of an additional 23 clock cycle

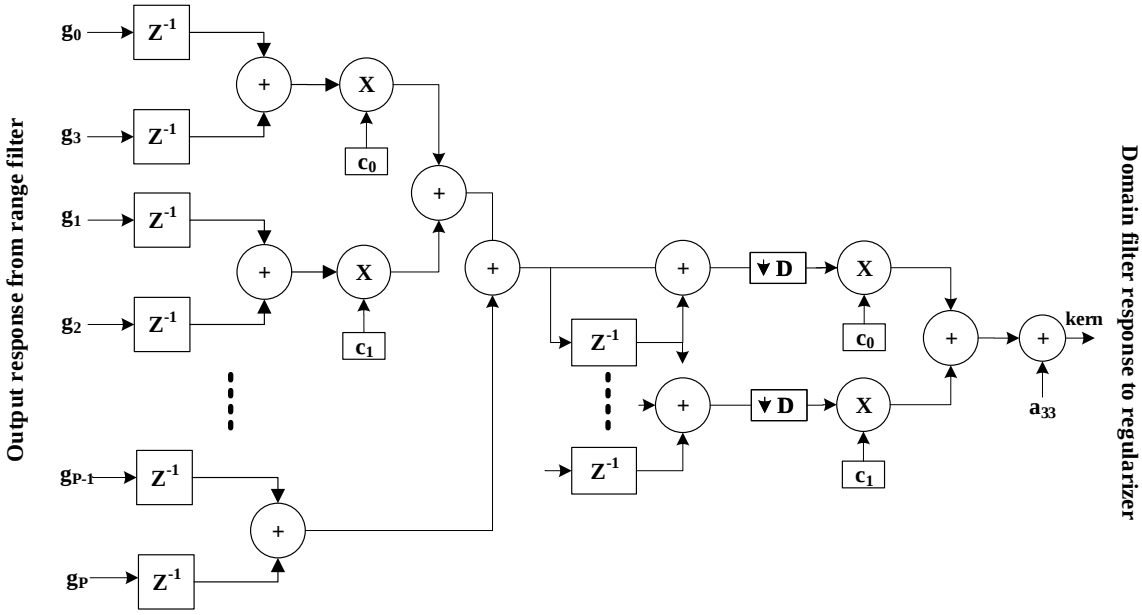


Figure 4.19: Domain Filter Architecture

initial latency.

4.2.4.4 Regularizer

A regularizer normalizes the dynamic range of the output response by finding the ratio between accumulated filter response ($kern$) to the sum of the product of domain and range filter coefficients ($coef$). The coefficients of the range and domain stages are read simultaneously utilizing parallel delay buffers. The division is implemented as a product of $kern$ values with the computed reciprocal $coef$ based on iterative Newton Raphson (LUT-NR) method. The Fig. 4.21 shows the implementation of the division unit using two iterations of the Newton Raphson method and additional multiplier.

$$reg_{i+1} = reg_i(2 - in * reg_i) \tag{4.14}$$

The initial value (reg_0) of the reciprocal function as given in the equation (4.14) is stored in the lookup table. It lies in the range of $0 < reg_0 < \frac{2}{in}$.

4.2.4.5 Running variance

A BF provide optimal denoising of the input data when the noise variance of the input data is known. The estimation of noise variance from the input flow field is not trivial. The running variance module estimates the noise variance of the input data in each sampling

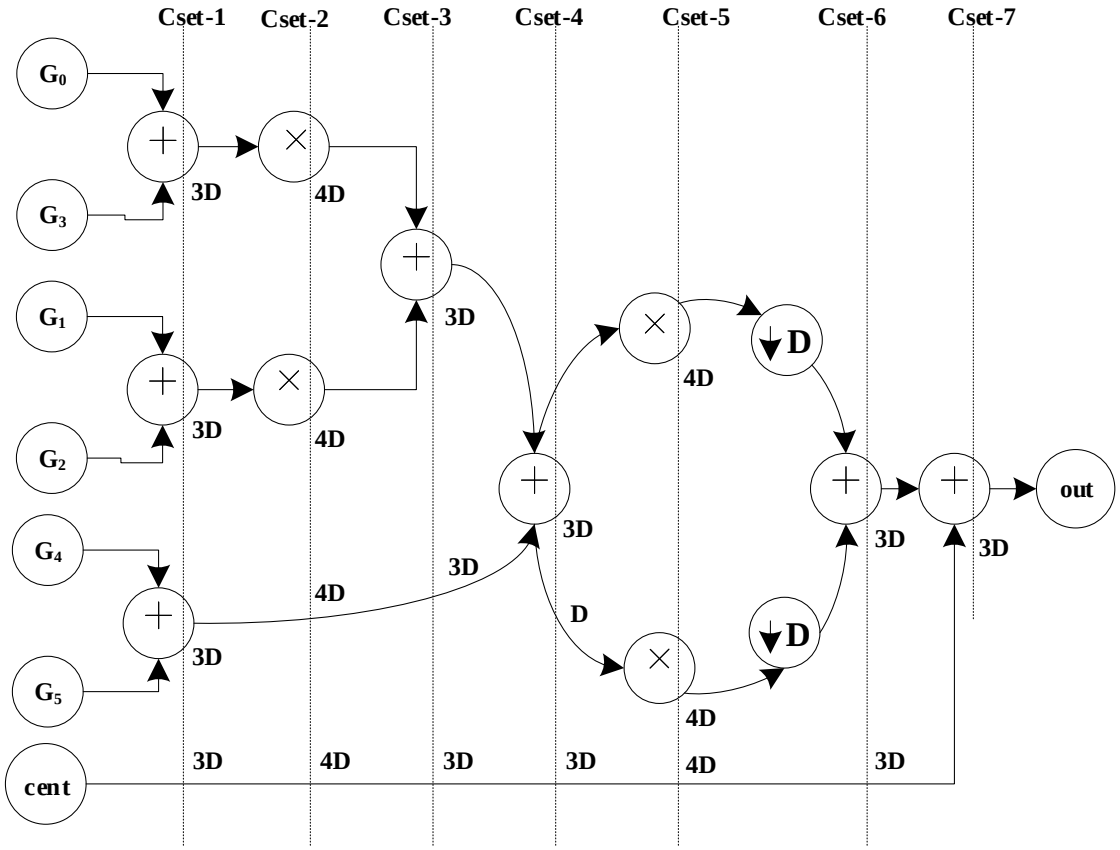


Figure 4.20: 24 Stage Pipelined Domain Filter

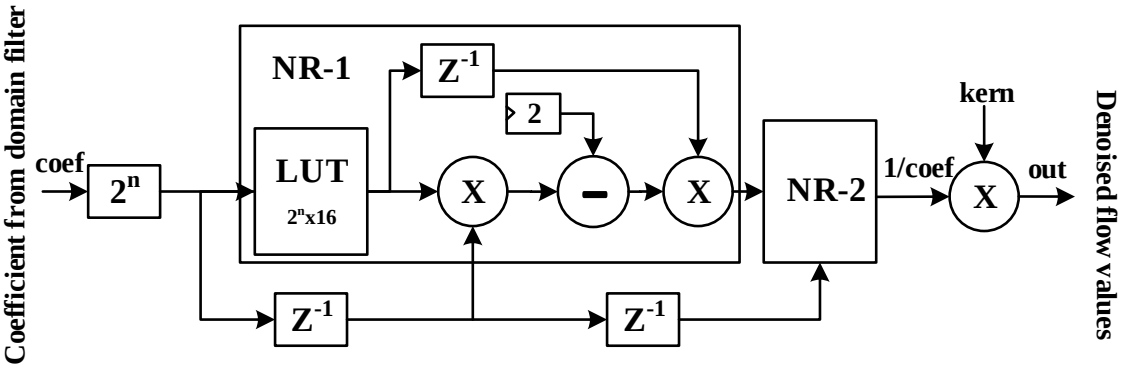


Figure 4.21: LUT-NR Regularizer Architecture

clock cycle f_{samp}. It helps in handling varying noise levels by updating the range filter coefficients at regular frame intervals. This is done by reloading new range coefficients (pre-computed and stored) in response to the amount of noise present in the input flow

field.

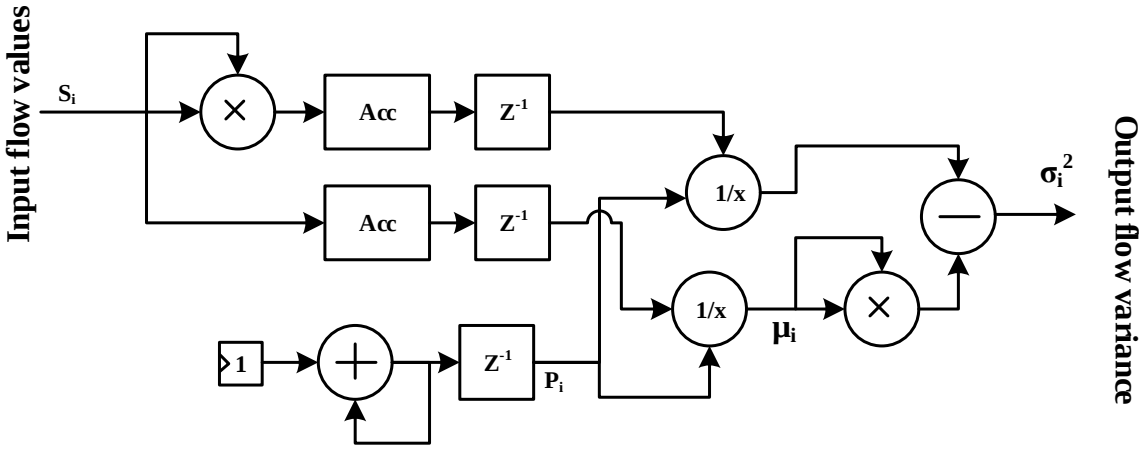


Figure 4.22: Running Variance Compute Block

The conventional formulation estimates mean and variance of the input data after a latency of one frame. This problem is solved using a running variance compute module [133] as approximated in equation (4.15).

$$\sigma_{ri}^2 = \frac{1}{O_i - 1} \sum_{i=1}^{O} (F_i)^2 - \mu_i^2 \tag{4.15}$$

Where F represents vectorized notation of two-dimensional noisy flow field $f(x, y)$, total number of values is denoted by O. The internal architecture of the running variance module is shown in Fig. 4.22. The estimated variance settles to valid value with 2% tolerance after $2 \times W$ cycles and gets updated in every clock cycle. The Acc in the running variance block performs the accumulation operation. It is a highly compute-intensive block containing two divider blocks and two 32 bit accumulators. The divider block is implemented using LUT-NR method.

4.2.5 Hardware Variants

The work proposes four different hardware variants of true BF architecture utilizing the above-discussed modules to study the tradeoff between area, throughput and power. The variants are 1) High Efficiency Bilateral Filter (HEBF), 2) High throughput Bilateral Filter (HTBF), 3) Mid Range Bilateral Filter (MRBF) and 4) Self adaptable High throughput Bilateral Filter (SAHTBF).

a) HEBF

	t_1	t_2	t_3	t_4
s_1	a_{11}	a_{12}	a_{14}	a_{15}
s_2	a_{21}	a_{22}	a_{24}	a_{25}
s_3	a_{31}	a_{32}	a_{34}	a_{35}
s_4	a_{41}	a_{42}	a_{44}	a_{45}
s_5	a_{51}	a_{52}	a_{54}	a_{55}
s_6	a_{13}	a_{23}	a_{43}	a_{53}

b) MRBF

	t_1	t_2
s_1	a_{11}	a_{12}
s_2	a_{21}	a_{22}
s_3	a_{31}	a_{32}
s_4	a_{41}	a_{42}
s_5	a_{51}	a_{52}
s_6	a_{13}	a_{53}
s_7	a_{23}	a_{43}
s_8	a_{14}	a_{15}
s_9	a_{24}	a_{25}
s_{10}	a_{34}	a_{35}
s_{11}	a_{44}	a_{45}
s_{12}	a_{54}	a_{55}

Figure 4.23: P-Parallel Segments of HEBF and MRBF.

Table 4.8: Performance characteristics of BF hardware variants.

Variant / Perf.[3]	HEBF	MRBF	HTBF (SAHTBF)
Segments[1]	6	12	24
Throughput[2]	0.25	0.5	1
f_{samp} **(cycles)**	1	1	1
f_{int} **(cycles)**	4	2	1

[1] denotes the number of parallel compute elements.
[2] refers to the maximum number flow values denoised per second by BF.
[3] represents the performance characteristics of the architecture.

The HEBF variant divides the input data stream into P=6 segments, which leads to the reduction of flow denoising to N_cyc =4 cycles. Whereas an MRBF variant divides the input stream into P=12 segments, results in the flow denoising at N_{cyc}=2 cycles. Instead, a HTBF architecture eliminates the grouping of flow values into segments to improve the denoising throughput to one flow value at every clock cycle. The number of parallel segments present in HEBF and MRBF variants is given in Fig. 4.23. Table 4.8 highlights the summary of the performance and throughput of the above-proposed architectures.

4.2.5.1 High Efficiency Bilateral Filter Architecture (HEBF)

Fig. 4.24 shows the internal architecture of the HEBF. It achieves the lowest resource utilization and highest power efficiency among other BF architecture variants by trading off throughput. The throughput is reduced by one fourth as the number of considered parallel segments is P=6 (N_{cyc} equals to 4). The rest of the modules are utilized directly to implement HEBF architecture.

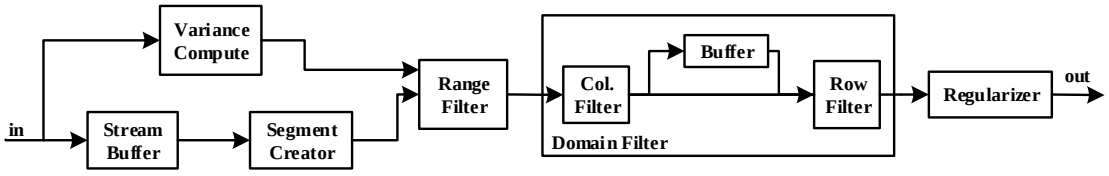


Figure 4.24: HEBF architecture.

4.2.5.2 High Throughput Bilateral Filter Architecture (HTBF)

The HTBF architecture aims at improving the throughput while trading off an increased resource and power consumption. The submodules of HTBF architecture operates at same frequency as that of the input sampling frequency ($f_{int} = f_{samp}$). The segment creator block is modified to eliminate the grouping of flow values into segments. The new segment creator block sends out P=24 flow values simultaneously to the range filter at the input sampling rate. The range filter is modified to contain P=24 parallel processing elements instead of P=6 to compute the similarity between the centre and neighbouring flow values. This results in quadrupling the resources of the range filter, which includes 18 MUL, 18 SUB and 12 RAM as compared to HEBF. The Combiner block is a new submodule considered in HTBF architecture which is basically an adder tree. The combiner module makes use of symmetry property to accumulate range filter response and coefficients, thus reducing the number of multipliers used in domain filtering. Another issue is with the fan out of the centre value (P= 24), which is solved as described in Sect. 4.2.4.2.

4.2.5.3 Mid Range Bilateral Filter Architecture (MRBF)

The goal of a MRBF architecture is to simultaneously optimize throughput (increase by at-least 100%) and power (reduction by at least 30%) compared to HEBF and HTBF architectures respectively. This is different from the two above proposed architecture variants,

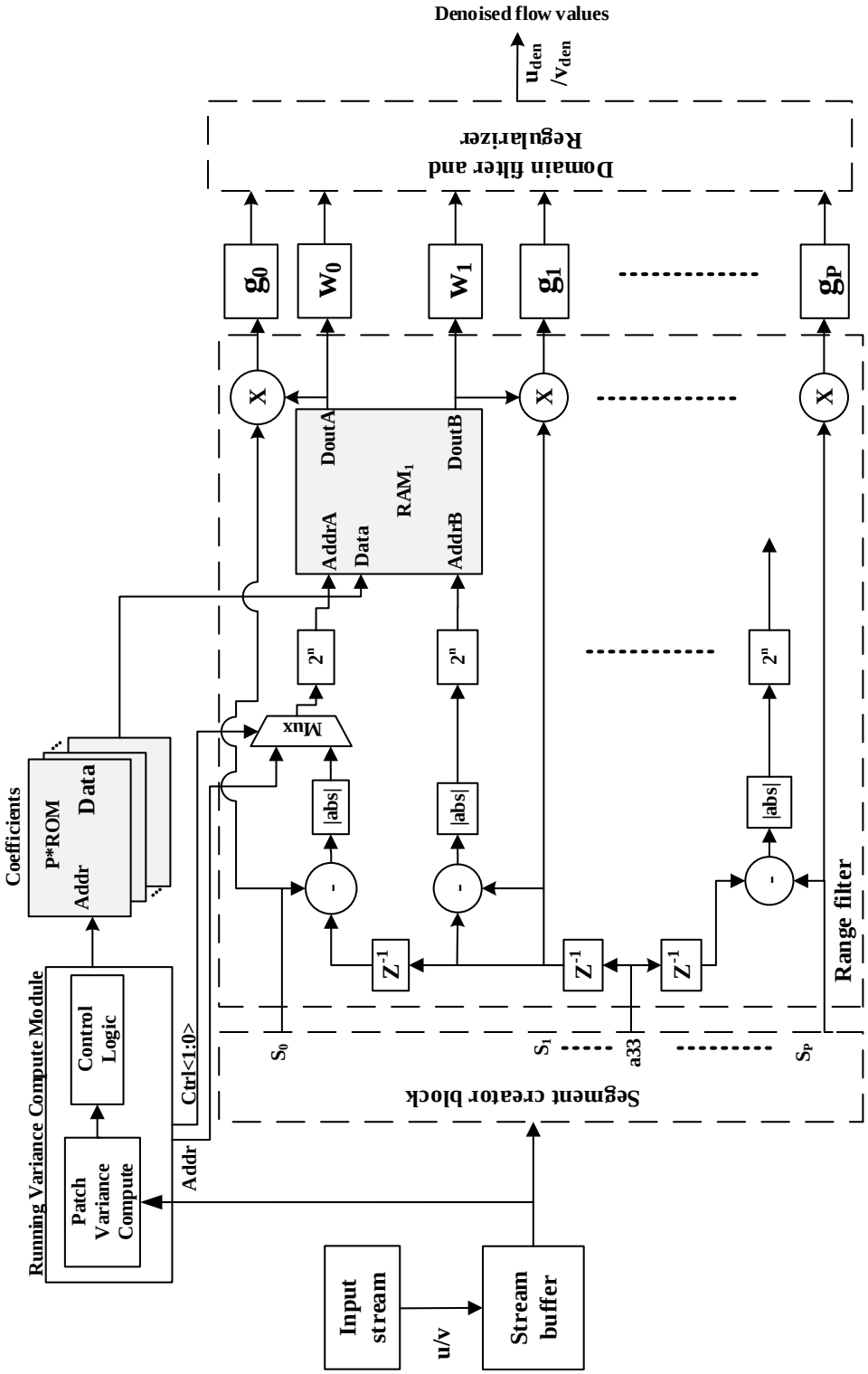


Figure 4.25: Self Adaptable Bilateral Filter Architecture

which either concentrates on maximizing power efficiency or throughput but not simultaneously. This section discusses only the modified and new submodules, the rest of which are the same as that of HEBF. The internal operating frequency of MRBF is half of the input sampling rate (f_{int} = $f_{samp}/2$). The modified segment creator module operating at f_{int} groups the input patch into P=12 parallel segments each containing 2 values. These values are fed simultaneously to a modified range filter containing P=12 parallel processing elements to compute flow similarity between the centre and neighbouring flow values. The range filter includes additional resources like 6 MUL, 6 SUB and 3 RAM as compared to HEBF. The combiner module containing 12 ADD blocks make use of symmetry to merge the range module outputs and feed it to the domain filter.

4.2.5.4 Self Adaptable High Throughput Bilateral filter Architecture (SAHTBF)

The internal architecture of the SAHTBF is shown in Fig. 4.25. The main difference from the HTBF variant is the presence of a running variance module. The minimum variance among all the input data patches in the current frame is compared with the variance of the previously stored frame, assuming there is no sudden change in the flow values. The deviation in the running variance is thresholded to five different ranges which are computed empirically. The five sets of coefficients are stored in 5 separate ROM blocks. An address selector block helps in reloading the range filter coefficients. The address of the memory (RAM) block is multiplexed between the absolute block output and the re-loadable counter data. Each ROM has a depth of 256 locations. When one of the ROM blocks is selected, the control circuitry waits for the end of the current frame being denoised and changes the multiplexer state from 1 to 0. The address port of all even RAMs is connected to the 8 bit counter synchronized with the filter coefficient port. The BF blocks the input stream until the reloading of the coefficients is finished. The address selection unit switches back to the incoming flow difference after the coefficient update. The rest of the blocks is the same as that of a HTBF architecture.

4.2.6 Hardware Implementation and Results

The proposed design is simulated, synthesized and implemented in Verilog on Xilinx $VC707$ evaluation kit and Xilinx $XUPV5$ board.

4.2.6.1 Evaluation Metric

There exists several metric to measure the performance of BF. The first among them is Peak Signal to Noise Ratio (PSNR) given in the equation (4.16), which evaluates the accuracy of the estimate rather than the quality of denoising.

$$PSNR = 20log(\frac{max(u(i,j))}{\frac{1}{m \times n}\sum_{i=1}^{m}\sum_{j=1}^{n}(g(i,j)-u(i,j))^2}) \tag{4.16}$$

Mean Structural Similarity (MSSIM) is another metric considered which is based on human perception rather than just mere accuracy. It measures correlation between the original and distorted field in-terms the object structure, contrast and luminance as given in the equation (4.18).

$$SSIM(u_i, g_i) = \frac{(2\mu_{u_i}\mu_{g_i} + C_1)(2\sigma_{u_i}\sigma_{g_i} + C_2)}{(\mu_{u_i}^2 + \mu_{g_i}^2 + C_1)(\sigma_{u_i}^2 + \sigma_{g_i}^2 + C_2)} \tag{4.17}$$

$$MSSIM = \frac{1}{n}\sum_{i=0}^{n-1} SSIM(u_i, g_i) \tag{4.18}$$

Here μ_u, σ_u and μ_g, σ_g represent the mean and standard deviation of the original and denoised flow values respectively. C_1 and C_2 are employed to eliminate instability and is computed according to $(K_1 \times L)^2$ and $(K_2 \times L)^2$, were $K1 << 1$ and $K_2 << 1$, L represents the dynamic range.

4.2.6.2 Impact of Numerical Representation

A variable fixed point representation is followed in the design of the BF architectures. The conversion from floating point to fixed point results in quantization effects. The input flow values are normalized to the range 0 to 1 before feeding to stream buffer. The highest bit-width required for each stage in the BF architecture is shown in Table. 4.9. The bit-width for each module is computed by feeding reference data to the design and measuring the degradation in terms of PSNR, by varying the bit-width from 10 to 30 while keeping the rest of the circuit in full precision as shown in Fig. 4.26.

4.2.6.3 Filter parameter analysis

The performance of the BF architecture depends on the selection of suitable design parameters which include kernel size and standard deviation of the domain and range filter

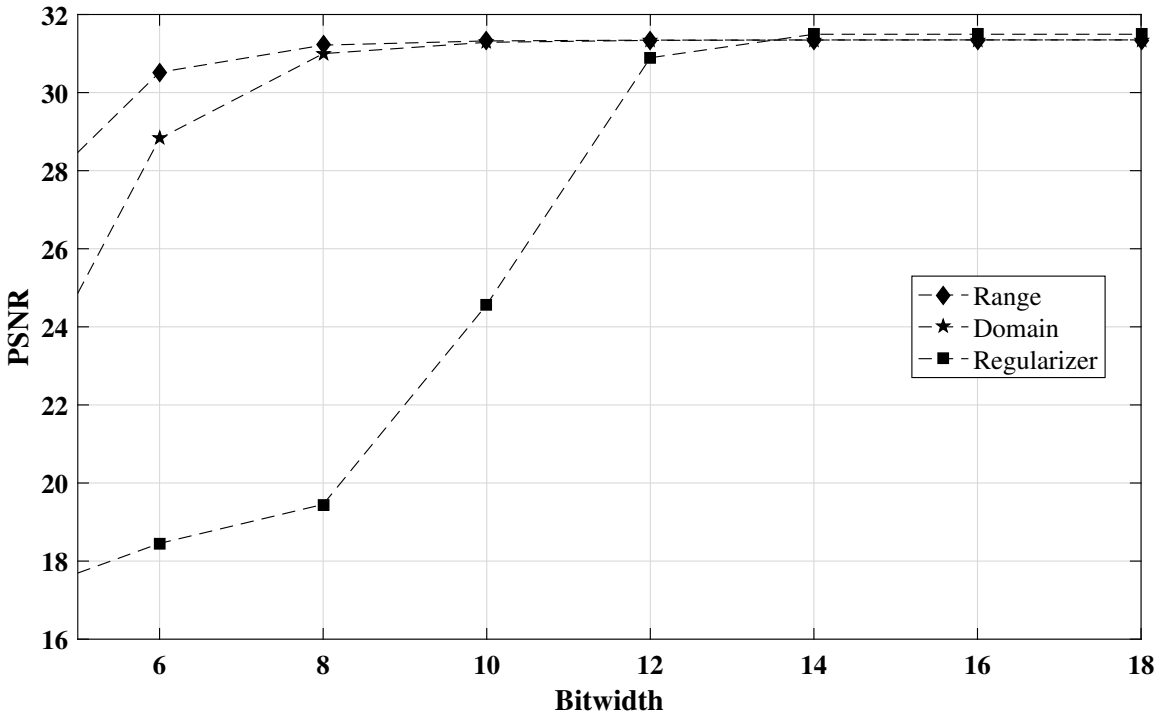


Figure 4.26: Bit-width chosen for BF modules.

Table 4.9: Selected bit-width for different modules.

	Segment	**Range**	**Domain**	**Regularize**
BW$_{min}$[1]	8	10	12	14

[1] represent the maximum number of bits utilized for each stage.

modules. A trade-off between the computational complexity and blurring effect imposes an optimal choice of 5×5 as the filter kernel size for our experiments. The scalability of architecture for different kernel sizes are discussed in section 4.2.6.5. A trade-off between noise reduction and fine detail preservation is made while choosing the final values. The standard deviation (σ_{range}) of the range filter depends on the amount of noise density present in the input flow field. The filter parameters are chosen by analysing the denoising performance by varying the amount of Additive White Gaussian Noise (AWGN) noise added to the input field. The value of σ_{range} is chosen to be $k \times \sigma_{noise}$. The Fig. 4.27 shows the selected value of k for a range of varying noise density satisfying the maximum PSNR.

It can be observed that the k varies in the range 0 to 10 for noise densities 0 to 60. Similarly, the standard deviation of the domain filter σ_{dom} is estimated as shown in Fig. 4.28.

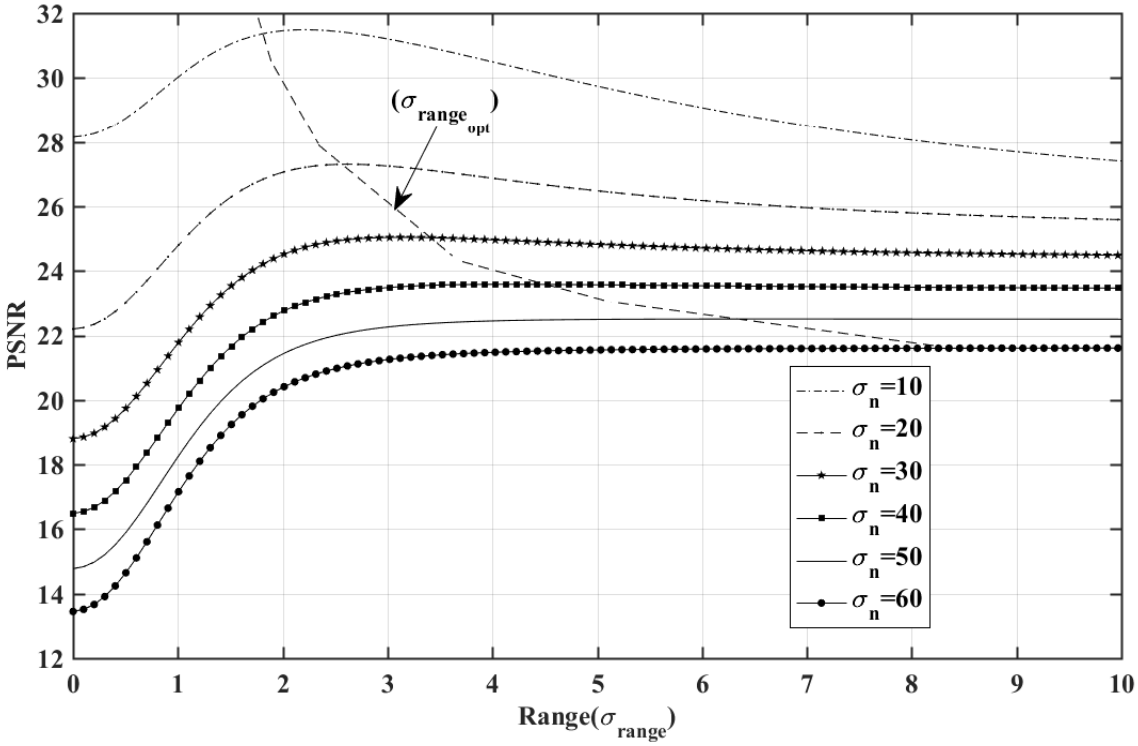


Figure 4.27: Selection of σ_{range} for Range filter

From the above graph for varying noise densities (has least dependency), the σ_{dom} value is chosen to be 1. The Fig. 4.29 (a, c) shows the noisy flow field generated by adding AWGN noise with noise density 50 dB and Fig. 4.29 (b, d) shows the denoised flow using BF.

4.2.6.4 FPGA Resource Analysis

The resource utilization is estimated for all the five proposed hardware designs as a whole as well as submodules. This helps in identifying the critical path that limits the maximum frequency of operation. A fully pipelined design divides the critical path to improve the performance trading off additional resources. The Tables 4.10, 4.11, 4.12 and 4.13 shows detailed breakdown of the resource utilization, maximum operating frequency and power dissipation of the various proposed architectures HEBF, MRBF, HTBF and SAHTBF in both fixed point and floating point representations respectively. The floating point implementation of exponential operator uses a 6^{th} order Taylor series to get higher accuracy trading off additional resources. This results in increased resource utilization for the range module, which impacts all the FLP architectures.

The unoptimized floating point implementation of HEBF shows $27.8\times$ increase in overall resource utilization, $5.4\times$ higher power consumption, and $1.8\times$ slower frequency of op-

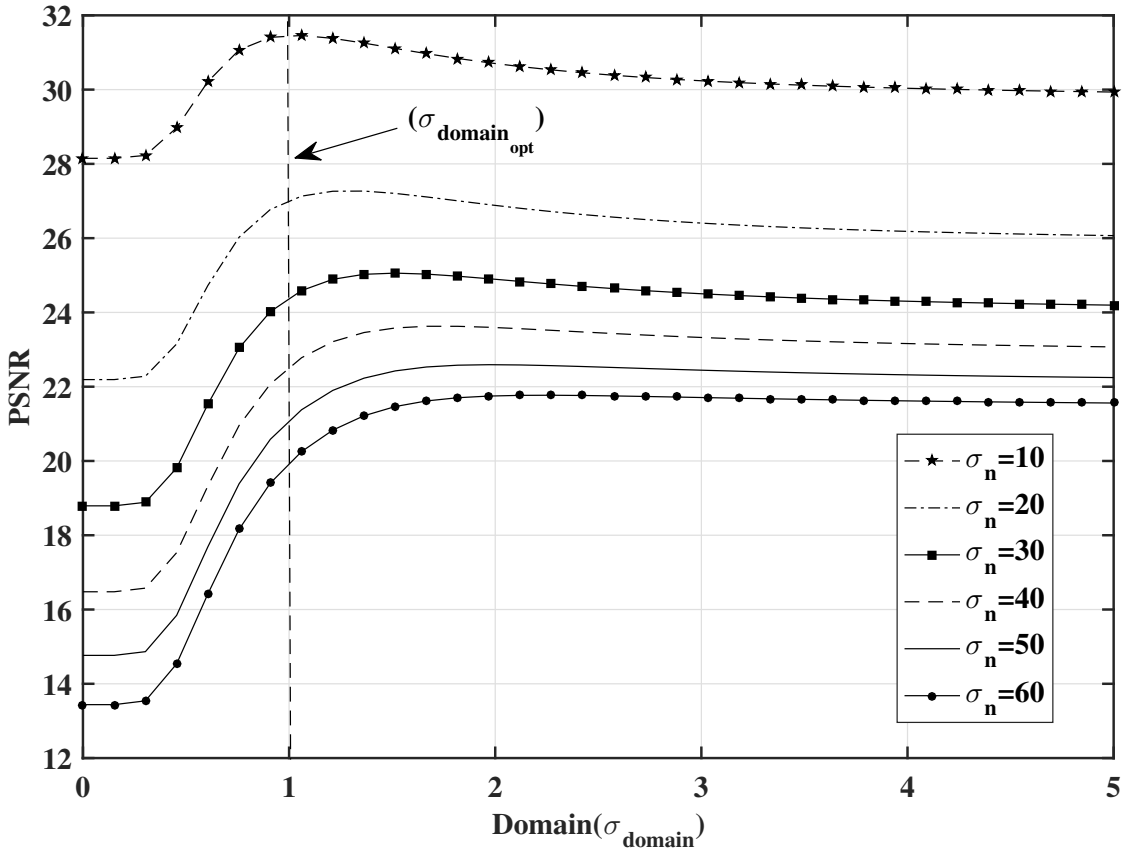


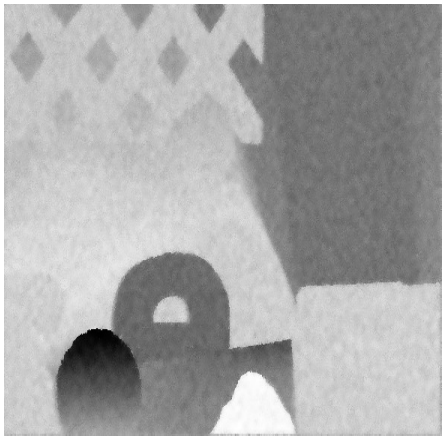
Figure 4.28: Selection of σ_{domian} for Domain filter

eration compared to fixed point implementation. The critical path of the design is identified in the range filtering stage. The number of parallel processing elements in the range filter module for MRBF and HTBF architecture is $2\times$ and $4\times$ as compared to HEBF. This results in increased resource utilization ($1.9\times$, $4.2\times$) and dynamic power consumption ($1.2\times$, $1.8\times$) for the range module which is shown in the Table. 4.11 and Table. 4.12 respectively.

The Table 4.13 demonstrates a higher resource requirement for SAHTBF architecture due to presence running variance compute module. The dividers are implemented using LUT-NR method as described in the Sect. 4.2.4.4.

A comparative analysis of the proposed architectures on a Virtex-5 FPGA is also performed as shown in Table 4.14. This helps in getting a clear understanding of the variation in resource, power and operating frequency when switching to a different FPGA family. Among the proposed architectures, HEBF shows the highest efficiency and is found suitable for low power embedded applications. The HEBF architecture shows a $1.9\times$ reduction

(a) Noisy



(b) Denoised

(c) Noisy

(d) Denoised

Figure 4.29: Noisy and denoised version of Rubberwhale and Alley 1 flow values from Middlebury and Sintel database.

in dynamic power and resource consumption as compared to HTBF. The proposed architectures are highly pipelined to achieve an average operating frequency of 340 MHz. The Virtex-5 architectures suffer from 27% reduction in the maximum frequency of operation due to the difference in FPGA technology. The single precision (FLP) implementation follows a similar trend as in the case of Virtex-7.

4.2.6.5 Scaling up the BF architecture

The kernel size of the proposed BF architecture be can extended from small (3×3) to large(15×15). For a $K \times K$ kernel, a total of K^2 samples need to denoised at one f_{samp} clock cycle. The number of BRAMs utilized in the design of streaming buffer (K) varies with the chosen kernel size. The streaming module reads out K number of flow

Table 4.10: Resource utilization of HEBF architecture.

Module / Resource	SEG		RANG		DOM		REG		HEBF	
	FLP[1]	**FXP**[2]	**FLP**	**FXP**	**FLP**	**FXP**	**FLP**	**FXP**	**FLP**	**FXP**
FF	2249	370	46027	479	7692	943	1370	50	68456	2254
LUT	802	163	49311	363	11596	654	933	68	66647	1337
Slice	583	174	16013	179	3625	261	342	31	22267	748
BRAM18	4	4	0	0	0	8	0	1	4	12
DSP48	0	0	144	6	0	0	0	4	144	10
Fmax (MHz)	472	477	260	445	305	514	487	508	254	457
Dynamic (W)	0.080	0.365	2.598	0.135	0.601	0.205	0.165	0.135	1.396	0.258
Static (W)	0.207	0.207	0.210	0.207	0.245	0.208	0.207	0.207	0.217	0.209

[1] represent the floating point BF architecture.
[2] denotes the fixed point BF architecture.

Table 4.11: Resource utilization of MRBF architecture.

Module / Resource	SEG		RANG		DOM		REG		MRBF	
	FLP	**FXP**	**FLP**	**FXP**	**FLP**	**FXP**	**FLP**	**FXP**	**FLP**	**FXP**
FF	2238	381	91218	770	15912	706	1370	125	129433	2609
LUT	1057	122	87117	634	15287	466	933	112	126502	1653
Slice	548	183	31145	354	5404	190	342	65	42129	903
BRAM18	4	4	0	0	0	11	0	14	4	25
DSP48	0	0	288	12	0	0	0	4	288	12
Fmax (MHz)	475	477	230	569	318	458	487	556	302	437
Dynamic (W)	0.103	0.376	2.782	0.165	1.214	0.226	0.165	0.145	3.391	0.362
Static(W)	0.207	0.207	0.211	0.207	0.215	0.208	0.207	0.207	0.233	0.209

values every clock cycle. The segment creator modules divides the incoming stream into $(K^2-1)/f_{int}$ number of segments with one additional centre value, K^2 flow values will be denoised in every f_{int}=4 clock cycles (f_{int}=1 for HTBF). The number of multiplexers in the segment creator module also gets increased to $(K^2-1)/f_{int}$. The ordering of input values

Table 4.12: Resource utilization of HTBF architecture.

Module / Resource	SEG		RANG		DOM		REG		HTBF	
	FLP	**FXP**	**FLP**	**FXP**	**FLP**	**FXP**	**FLP**	**FXP**	**FLP**	**FXP**
FF	1368	321	121983	2005	13719	1072	1370	125	198608	4048
LUT	784	72	111069	1879	18357	805	933	112	21136	3136
Slice	325	201	37145	760	6616	347	342	65	63520	1307
BRAM18	4	4	0	0	0	14	0	14	4	27
DSP48	0	0	488	24	70	0	4	0	694	24
Fmax (MHz)	437	446	210	445	240	467	487	502	190	467
Dynamic(W)	0.113	0.410	3.948	0.242	1.130	0.288	0.165	0.145	5.495	0.510
Static(W)	0.207	0.209	0.233	0.207	0.214	0.208	0.207	0.207	0.251	0.210

Table 4.13: Resource utilization of SAHTBF architecture.

Module / Resource	SEG		RANG		DOM		REG		VAR		SAHTBF	
	FLP	**FXP**	**FLP**	**FXP**	**FLP**	**FXP**	**FLP**	**FXP**	**FLP**	**FXP**	**FLP**	**FXP**
FF	1368	321	125212	8631	13719	1072	1370	125	86217	5617	248163	10198
LUT	784	72	111713	5528	18357	805	933	112	82072	2783	276141	7083
Slice	325	201	38211	2785	6616	347	342	65	22185	1320	65117	2304
BRAM18	4	4	0	9	0	14	0	14	0	2	4	32
DSP48	0	210	488	28	70	0	0	4	225	4	740	28
Fmax (MHz)	437	446	210	370	210	467	487	502	190	380	180	335
Dynamic (W)	0.113	0.410	3.978	0.715	1.130	0.288	0.165	0.145	2.543	0.429	5.982	0.860
Static (W)	0.207	0.209	0.238	0.211	0.214	0.208	0.207	0.207	0.210	0.209	0.268	0.211

remains the same and depends on the symmetry of the kernel. The photometric modules are replicated $(K^2 - 1)/f_{int}$ times without disturbing the functionality of internal blocks. Similar is the case with the geometric filter. Even though the number of pipeline stages (L) for a scaled up version remains the same, the pipelining registers got increased by a factor of $L * (K^2 - 1)/f_{int}$; depending on the available number of parallel channels. The size of the internal buffer stages (K) of the domain filter module increases based on the size of the kernel. Rest of the modules are similar to the original design.

The throughput of the extended architectures will remain the same as the original design

Table 4.14: Comparison of proposed BF architectures on Virtex-5 FPGA.

Variants / Resource	HTBF		MRBF		HEBF	
	Used	**Util.**	**Used**	**Util.**	**Used**	**Util.**
FF	4227	14%	2598	9%	2347	7%
LUT	3249	11%	1682	5%	1286	4%
Slices	1562	21%	927	12%	807	10%
BRAM18	22	45%	18	37%	8	25%
DSP48	24	50%	12	25%	6	12%
Fmax (MHz)	333		343		343	
Dynamic (W)	0.597		0.424		0.317	
Static (W)	0.425		0.424		0.423	

Table 4.15: Scalability of BF architecture for a variable kernel size.

Kernel / Resource	5×5		9×9		15×15	
	HEBF	**HTBF**	**HEBF**	**HTBF**	**HEBF**	**HTBF**
FF	2254	4048	7748	10544	18251	30241
LUT	1337	3136	5940	8552	14464	24892
Slices	748	1307	2123	3569	5961	10055
BRAM18	12	27	14	29	29	77
DSP48	10	24	18	72	72	216
Fmax (MHz)	450	467	251	310	254	277
Dynamic (W)	0.258	0.510	0.262	1.228	1.802	3.075
Static (W)	0.209	0.210	0.208	0.216	0.218	0.232

while trading off the increased resource utilization. The Table. 4.15 shows the comparison of the performance and resource utilization of the proposed BF architectures on Virtex-7 ($VC707$) FPGA, with varying kernel sizes 9×9 and 15×15. The resource utilization of the HEBF, as well as HTBF, shows an increasing trend for higher kernel size as compared to 5×5 implementation, a slight decrease in the throughput is also observed which accounts for the routing congestion and unoptimized architecture. A similar trend is observed in the case of other architectures. The modularity in the proposed architecture allows the implementation of BF with any larger kernel size. But the current implementation on a Virtex-7 ($VC707$) FPGA restricts the kernel size to 81×81, limited by the availability of DSP48 multipliers used in photometric module and the pipelining registers. The optimal value of σ_{range} and σ_{dom} will remain the same for different kernel sizes.

4.2.6.6 Power and Computation time Analysis

The analysis of static and dynamic power based on Power efficiency metric (Compute Density /Joule) is performed to estimate the advantage of FPGA's over other solutions. Table. 4.15 compares the power dissipation of a BF architecture with varying kernel size. The total power consumption of BF architecture varies from 0.72 W to 1.44 W to 3.31 W for kernels of size 5×5 to 9×9 to 15×15 respectively, showing a linearly increasing trend while operating at an average frequency of 250 MHz. A high throughput BF operating at high frequency constitutes, most of the dynamic power in the total power consumption. A large number of DSP48 and BRAM blocks accounts for the surge in the dynamic power. A larger kernel size implies an increased number of RAM blocks for buffering, additional multiplexers in the segment creator module, P number of parallel photometric modules with dedicated multiplier blocks, additional pipelining registers for P parallel paths and large internal buffers for the domain filter. A higher bit-width is required for a BF architecture with the scalable kernel, which constitutes for increased power consumption.

The complexity of proposed fixed-point architectures in terms of the number of required operations is expressed in terms of Giga Operations Per Second (GOPS) whereas for floating point architecture its represented in Giga Floating-point Operations Per Second (GFLOPS). Consider an input flow field of size 1920×1080 de-noised by BF on Virtex-7 FPGA. The I/O and buffer stage are in-charge of receiving and transmitting the I/O stream. It involves 1 noisy input read, 1 denoised output values and (4) parallel read operations. The next is a computationally intensive range filter, which has (P=24) parallel processing elements, each containing $1SUB$, 1 absolute, 1 Look-Up, 1 Shift and $1MUL$, which adds up to ($5P$=120) operations. The domain filter stage performs 48 additions and $10MUL$, which adds up to 58 operations. The last stage is a regularizer stage which consists of $5MUL$, 2 Look-Up and $2SUB$ operations, which corresponds to 9 operations. In total, the proposed $HTBF$ architecture contains a total of 193 operations. Since the HTBF$_{FXP}$ implementation is operating at 470 MHz, each frame needs ≈ 2.6 ms, which corresponds to a maximum frame rate of ≈ 373 frames. Similarly, a single precision HTBF implementation involves a total of 668 operations. It corresponds to 6^{th} order Taylor series approximation of exponential operator involving $7ADD$ and $14MUL$ operation in each of the (P=24) parallel elements. This adds to a total of 504 operations. Similarly, a HTBF$_{FLP}$ implementation operating at 190 MHz on an FHD frame needs ≈ 5.3 ms which corresponds to a maximum frame rate of ≈ 152 frames. The total compute performance for any architecture is given by,

$$Perf = N_{ops} * Resolution_{frame} * Rate_{fps} \tag{4.19}$$

Table 4.16: Comparison of energy efficiency across different implementation platforms.

Chara. / Type	Platform	FPS[1]	Perf[2]	TPW[3]
BF	CPU	0.04	0.018	0.005
BF	GPGPU	5	2.3	0.069
BF Grid[118]	CPU	2.5	1.16	0.018
BF Grid[118]	GPGPU	111	153	1.1
$HTBF_{FLP}$	FPGA	152	210	37
$HTBF_{FXP}$	FPGA	373	149	318

[1] represents the number of frames per second achieved for a fixed resolution 1920×1080.
[2] denotes the all values in GFLOPS/Sec except $HTBF_{FXP}$ which is represented in GOPS/Sec.
[3] refers to throughput obtained per watt.

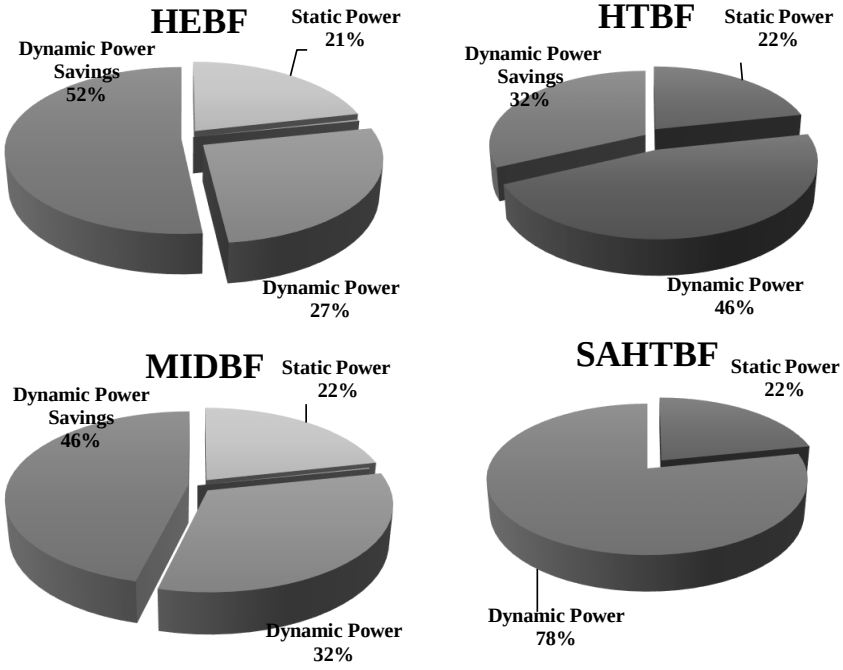


Figure 4.30: Comparison of power efficiency among proposed architectures

Table. 4.16 shows the comparison of proposed architecture with state of the art CPU/GPGPU solutions. Since the total number of operations performed in CPU/GPGPU platforms are not given, it is assumed that the same number of operation is present in the case of $HTBF_{FLP}$. The energy efficiency per Joule is much higher in FPGA designs than CPU or GPGPU platforms. It is clear from the table the Performance or Throughput Per Watt

Table 4.17: Comparison of BF architecture with state of the art methods.

Method / Resource	Proposed					Existing				
	HEBF	**MRBF**	**HTBF**	**HTBF**	**SAHTBF**	**Gabiger[134]**	**Han[123]**	**Vinh[124]**	**Dutta[135]**	**Villal[130]**
Filter	BF	BF	BF	BF	BF	BF	BF	ModBF	BF	BF
Kernel	5x5	5x5	5x5	5x5	5x5	5x5	11x11	3x3	3x3	9x9
Device	Virtex-7	Virtex-7	Virtex-7	Virtex-5	Virtex-7	Virtex-5	Virtex-4	Cyclone II	Virtex II	Virtex-4
FF	2254	2609	4048	4227	4229	-	28968	227	-	-
LUT	1337	1653	3136	3249	3128	-	20843	-	-	-
Slices	748	903	1307	1562	1374	1060	15110	567	1447	19893
BRAM	12	25	27	22	32	11	-	-	-	18
DSP48	6	12	24	24	24	29	242	32	9	
FPS[1]	109	208	445	317	432	52	-	-	151	83
Fmax (MHz)	457	437	467	333	454	220	80	159	87.65	124
Throughput[2]	0.25	0.5	1	1	1	0.25	-	-	-	-
Latency (Cycles)	2078	2079	2105	2105	2140	5156	-	-	-	-
Dynamic (mw)	258	362	510	597	760	-	-	-	-	-
Speedup	4.1	2.13	1	1.4	1.03	8.5	5.89	19.86	5.37	3.77
Area	1.75	1.45	1	0.84	0.95	1.23	0.087	2.25	0.93	0.068
FFDelay	1	1	1	1	1	0.74	0.93	0.81	0.58	0.94
S_{AN}	2.36	1.47	1	1.67	1.08	6.91	67.70	8.83	5.81	55.44
S_{ADN}	2.36	1.47	1	1.67	1.08	5.11	62.96	7.15	3.37	52.11

[1] represents the number of frames per second achieved for a fixed resolution 1024×1024.
[2] denotes the number of flow values processed per second.

(TPW) for the proposed fixed-point HTBF (HTBF$_{FXP}$) implementation on FPGA is $289\times$ higher than GPGPU [118] and $17000\times$ higher than CPU [118] implementation. Along the same line, an FPGA implementation of floating point HTBF (HTBF$_{FLP}$) shows $34\times$ improved efficiency than GPGPU [118]. Finally the Fig. 4.30 shows saving in the power consumption of a HEBF implementation compared to other proposed architectures. HEBF shows saving of 52% in dynamic power as compared to SAHTBF architecture.

4.2.7 Comparison with other state-of-the art architectures

Table. 4.17 shows a comparison of the proposed architectures with other states of the art BF architectures. The HTBF architecture provides highest frame-rate of 1024×1024p @ 445 fps, lowest system latency in the order of twice the width ($2 \times W$) and system pipelining latency is best among state of the art methods. The table shows that our proposed design achieves a minimum improvement of $5.8\times$, $3.4\times$ for S_{AN} and S_{ADN} respectively as compared to the prior art on true and modified BF architectures.

4.2.8 Summary

This chapter dealt with the modifications of the internal subsystems of the multi-scale variational OF architecture. The first part of the chapter described the design of high throughput RBSOR solver architecture and its integration into variational OF architecture. The proposed HTOF architecture is able to compute OF for UHD images at 48 fps. The design achieves the highest throughput of 491 GOPS with a power efficiency of 43 GOPS/W at 412 MHz, compared to the state of art architectures.

The second part of the chapter explained about the design of HTBF architecture. The proposed HTBF is tuned to achieve the highest throughput at the cost of a peak power envelope of 510 mW. Experimental results show the highest performance of HTBF architecture in terms of area normalized speedup ($5.8\times$) and area-technology normalized speedup ($3.4\times$) compared to other existing architectures. The proposed architecture can adapt to varying noise level (η) in the input flow field by updating the range filter coefficients. In order to validate the proposed variational multi-scale OF architecture with its improved subsystems, the next chapter 5 discusses the design of a high throughput accelerator framework for cloud/cyclone tracking from satellite images. It helps in taking necessary precaution to mitigate the impact on life and property.

Chapter 5

Hardware Accelerator for Cloud/Cyclone Tracking

This chapter focuses on the application of proposed variational multi-scale OF architecture with improved subsystems for cloud/cyclone analysis and tracking. The analysis framework is effective for studying the cloud systems interactions inside cyclones, nowcasting using near real-time satellite data to take necessary precaution to mitigate the impact on life and property. The software implementation of the cloud analysis framework is complex and needs huge computation time to process the large satellite image database. Hence the chapter focuses on the design of high throughput hardware accelerator for cloud/cyclone analysis and tracking based on a selective choice of various computer vision algorithms and the aforementioned OF architectures with the improved subsystems.

5.1 Introduction

Cloud analysis plays an important role in understanding extreme climatic events. Meteorologists use satellite images to investigate cloud patterns, clouds propagation characteristics to study their evolution process and life cycle. Recent advancements in the image capturing technologies, lead to increased availability of high-resolution satellite data in near real-time. The satellite images are commonly available in Visible (VIS), Infra Red (IR) and Water Vapour (WV) channel. An IR satellite image uses a channel recorded from infrared energy (10.5-12.5 um). IR images have the advantage that they can also detect clouds during the night as opposed to visible images. The continuous availability of high-resolution IR images helps to study the cloud system interactions which is effective for now-casting. These interactions are then used to estimate the wind speed, the area covered by the clouds, forecasting their path and development of storms, climate prediction and analysis.

The design of a cloud analysis framework involves a series of operations like pre-processing, Cloud Motion Computation (CMC), cloud segmentation, cloud labelling and tracking. Among them, the cloud motion computation based on variational multi-scale OF is the most computationally intensive task. The selection of each stage of the cloud analysis framework is based on a selective choice of different computer vision techniques, except the CMC stage to achieve high throughput while trading off accuracy. The cloud analysis framework can operate on high-resolution IR images with a temporal resolution less than or equal to 30 minutes. Hence the amount of data that is to be processed even when a short time period such as a day is considered is large. The software implementation of the framework on CPU and GPGPU platforms leads to poor performance per watt, this restricts the applicability of the cloud analysis on huge satellite database.

Hence this section focuses on the design of a high throughput hardware accelerator for speeding up the cloud/cyclone analysis from a large IR image database. The design of the hardware accelerator involves several hardware adaptations to the framework and utilizes the proposed variational multi-scale OF architecture and improved subsystems to achieve a high throughput performance.

5.2 Related work

Until 1980, the tracking of the cloud systems was a time-consuming process as it involves the manual intervention of a field expert [136]. This gives accurate tracks results, but the tracking performance is completely dependent on the user's expertise. In recent literature, there have been many works proposed for the extraction of cloud information and tracking from the satellite imagery. Many authors proposed automatized methods for tracking clouds starting with pioneer work by Woodley et.al in [137]. Cloud motion extraction from Meteosat images using cross-correlation and height assignment is presented in [138]. Cross correlation-based tracking is proposed in [139], [140], [141], [142]. Another work [143] fuses the multilevel thresholding, vector median regularization and block matching algorithm to perform cloud tracking.

In feature-based methods, the clouds are initially segmented based on thresholding or active contours or clustering techniques [144, 145]. Followed by the motion estimation across consecutive image sequence using cross-correlation or pattern matching techniques. The major drawback of the feature based methods is that they don't capture movement within the object. Also in [146], a hierarchical method using local analysis and global analysis is introduced to track non-rigid motion and structure from two-dimensional satel-

lite cloud images. In [147], the authors have proposed a novel strategy to track clouds by extracting cloud information using computer vision techniques. Similar and modified methods for cloud tracking are also been proposed in [148]. For airborne weather radar and ground primary surveillance radar, an advanced cloud tracking algorithm using greyscale skeleton model of the clouds is introduced in [149]. And in [150], a genetic algorithm is applied to track convective cloud images from Chinese FY-2C satellite.

A framework to explore and visualize the cloud system movements is proposed in the work [151]. It helps to analyse the cloud motion at various spatial and temporal scales and study the multi-scale interactions between clouds and cloud systems inside TC. The motion of clouds is computed based on OF technique. The user can provide various queries to the framework to analyse the cloud systems without knowing about the underlying framework. Further, in [8], the authors propose a strategy to track clouds using ground-based omni-directional camera images to ensure greater efficiency in the supply of renewable energy. The proposed tracking procedure is composed of segmentation, localization and detection of interest points of clouds, followed by tracking cloud motions using OF methods. This technique can track displacement of each interest point in different speeds and directions to follow the cloud path. The velocity computation of the cloud vectors helps to provide greater accuracy in the automatic control of solar power plants in any cloud conditions.

An OF algorithm based on polynomial expansion is utilized in the work [152], to compute atmospheric motion vectors (AMVs) from geostationary satellite images. In order to identify the semitransparent cloud pixels with their actual temperatures/ heights information, a two-dimensional histogram between infrared brightness temperatures formed from a long time series of cloud images. The OF computation keeps the spatial consistency of wind fields while deriving AMVs over the traditional cross-correlation method. An image-pyramid scheme based on multiscale iteration is used to avoid the loss of large velocities. The validation results show that deriving dense AMVs based on OF is better suitable for fast-tracking with rapid-scan imagery and the newest geostationary imagers in quasi-real-time applications.

A framework to analyse and explore the temporal evolution of cyclones is proposed in the work [9]. It improves the tracking robustness by combining the well-established topological approaches and the track information from OF analysis. These tracks are analysed in a specific time interval to preserve the significant coherent cyclone movements. Most of the existing implementations focus on the design of an accurate framework for analysing clouds/cyclones [151] but have not considered improving the throughput performance with less power consumption. This serves as a motivation behind the design of a high through-

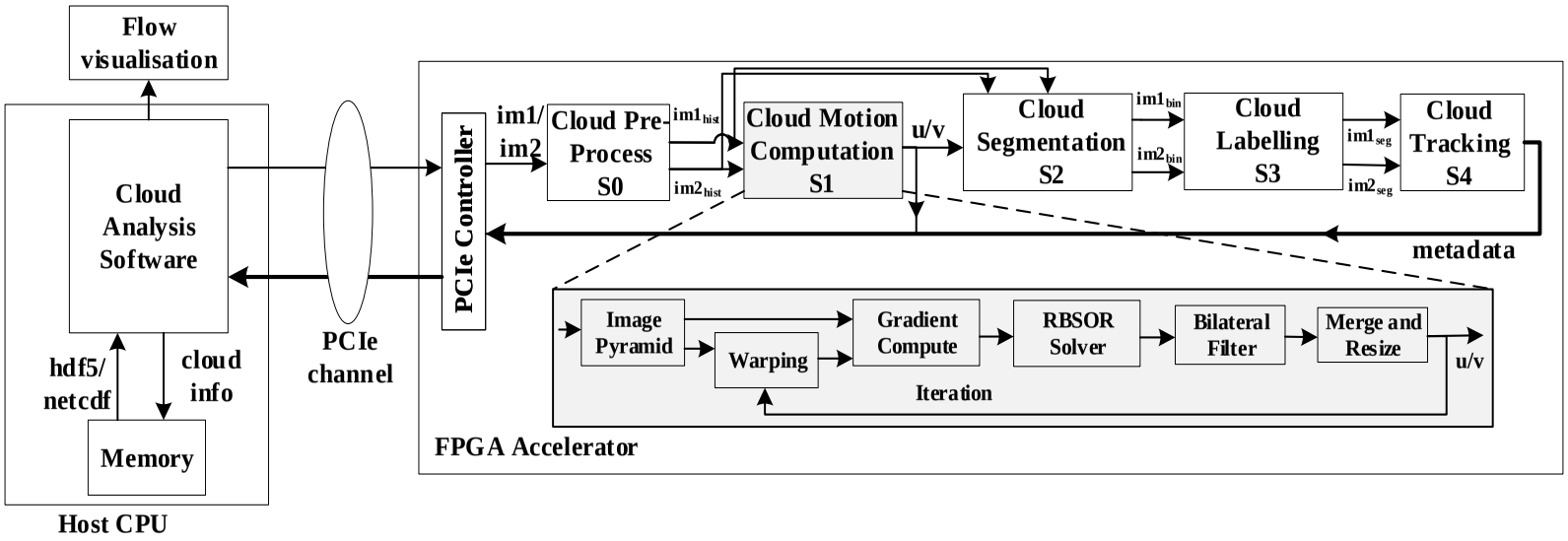


Figure 5.1: Block Diagram of High-Performance Cloud Analysis Framework

put accelerator for cloud/cyclone analysis from a large historical satellite image database utilizing proposed multi-scale variational OF architecture and improved subsystems.

5.3 Proposed hardware accelerator for cloud analysis framework

The work proposes a hardware accelerator for cloud analysis framework. An overview of the entire system is shown in Fig. 5.1. The satellite images are mostly available in a scientific format such as Hierarchical Data Format (HDF)/ Network Common Data Format (NetCDF), where the heterogeneity of underlying data formats is a significant hurdle. The cloud analysis software (CAS) utilizes the standard libraries provided by the vendor to convert raw satellite data into 10 bit image intensities. The images are sent out as a Transaction Layer Packet (TLP) over the Peripheral Component Interconnect Express (PCIe) interface to the FPGA accelerator. The hardware architecture involves several modules like a) Preprocessing, b) OF computation, c) Segmentation, d) Cloud labelling and e) Tracking. The CMC is implemented using a multi-scale variational OF architecture utilizing RBSOR Solver and BF architecture. The architecture exploits pipelining and parallel processing to accelerate throughput. The complex arithmetic operations are approximated to a simplified version without much accuracy loss. The computed flow values u,v and metadata are sent back to the host PC for monitoring and is stored in host memory for post analysis. The metadata contains the characteristics of all cloud segments and their track information, which is transferred at the end of every frame computation.

5.3.1 Cloud Pre-Processing:S0

The input image stream has illumination artefacts due to the variation in sun's reflectance from day to night, low contrast and contain random noise generated from external source or the operation of sensors and so on. Hence the preprocessing stage employs a two dimensional median filter of kernel size $K \times K$ as discussed in the previous Section 3.5.6.

The denoised image is fed to the histogram equalization module. It enhances the region of interests for improving the velocity estimation in the presence of noise and eliminate temporal aliasing and quantization effects. The internal architecture of the histogram equalization module is shown in Fig. 5.2. For an image of resolution $W \times H$ having G grey levels, the architecture utilizes G counters initialized with 0. It scans every pixel in the im-

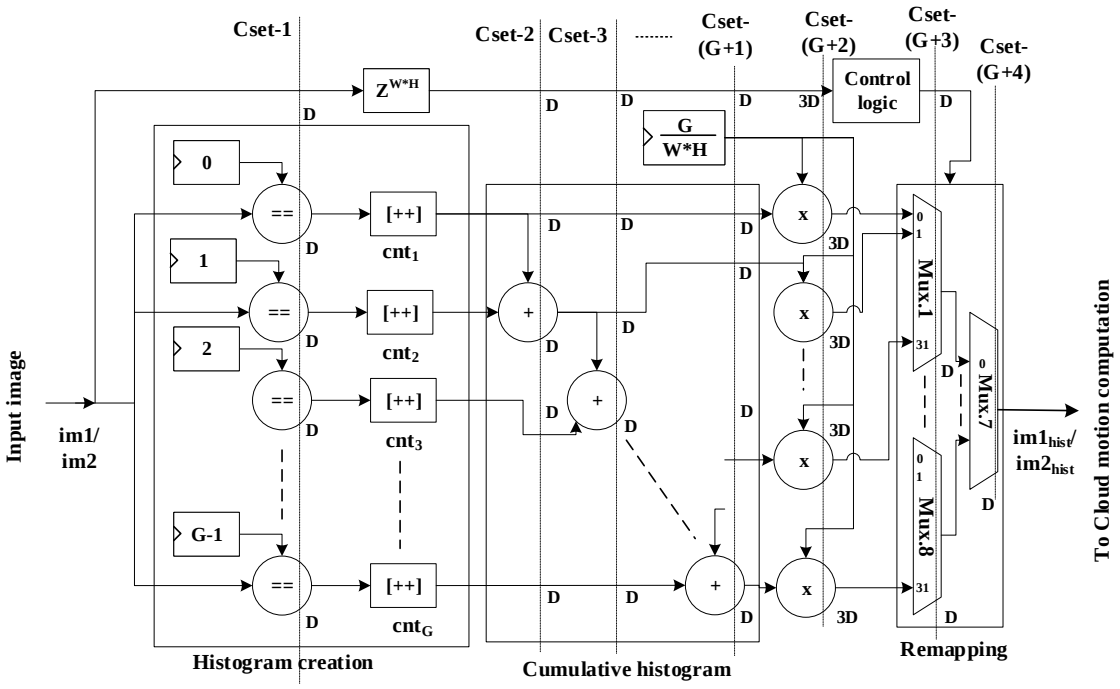


Figure 5.2: Histogram equalization architecture.

age stream and increments the counter (cnt_1,cnt_2...cnt_G) based on the number of pixel matches. The current counter (cnt_2) value is accumulated with the previous counter (cnt_1) value to create a cumulative histogram. The cumulative histogram values are normalized with a constant factor based on the maximum grey level to create a uniform histogram. The remapping stage utilizes conditional statements and multiplexers to transform the buffered input image to a new image based on the equalized histogram. The histogram equalization stage consumes G+4 coarse grain pipelines.

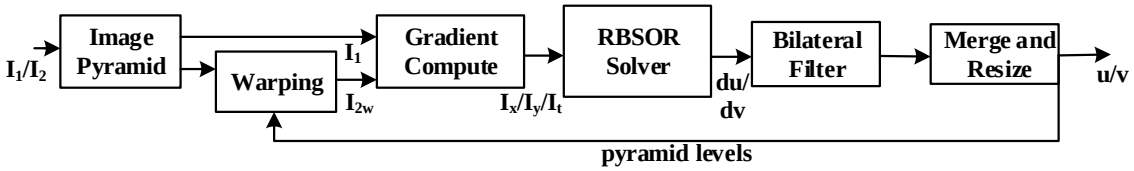


Figure 5.3: Block diagram of cloud motion computation module.

5.3.2 Cloud Motion Computation:S1

The pre-processed image is sent to the CMC module, which utilizes the proposed variational multi-scale OF architecture discussed in the previous Chapter 3 to accurately compute large displacement of the dense cloud motion field. The images are downsampled to create image pyramid, and the coarsest level image is fed to OF module to compute the flow field using 30 RBSOR Solver iteration and denoised using BF architecture as shown in the Fig. 5.3. The image rescaling block upsamples the computed flow at a coarser level to the next higher resolution and warps the first frame with the upsampled flow. The warped image is fed to OF computation engine till the flow is scaled to the original image dimension. The proposed architecture avoids the internal storage of input image and caches the intermediate flow vectors to remove the memory dependency and hence the memory bandwidth bottleneck. The internal architecture of the variational multi-scale OF, RBSOR solver and BF have been detailed in the previous Chapters.

5.3.3 Cloud Segmentation:S2

The pre-processed image and cloud motion fields from the CMC module are synchronized and fed simultaneously to the cloud segmentation module. The cloud is segmented using multi-level thresholding. A multilevel thresholding utilizes more than two thresholds ($thresh1$, $thresh2$) to segment the image into three classes. The empirical threshold obtained from histogram analysis helps in extracting the cyclone region given by the equation (5.1).

$$I_{thresh}(x,y) = \begin{cases} background & if(I(x,y) \leq thresh1) \\ cyclone & if(thresh1 < I(x,y) \leq thresh2) \\ others & if(thresh2 < I(x,y) \leq G) \end{cases} \tag{5.1}$$

Since the thresholding based on brightness temperature is inaccurate, the temporal information from the computed motion field is also used for distinguishing between background and moving pixel. The internal architecture of cloud segmentation module is shown in Fig. 5.4. The architecture utilizes a conditional statement to identify if the pixel intensity of the input stream is within the selected threshold range to create a binary image ($im1_{bin}/im2_{bin}$). The binary images are later compared with the velocity threshold ($thresh3$) to generate the final cloud segment mask. An important challenge in performing the segmentation based on thresholding is the choice of the threshold value. Instead

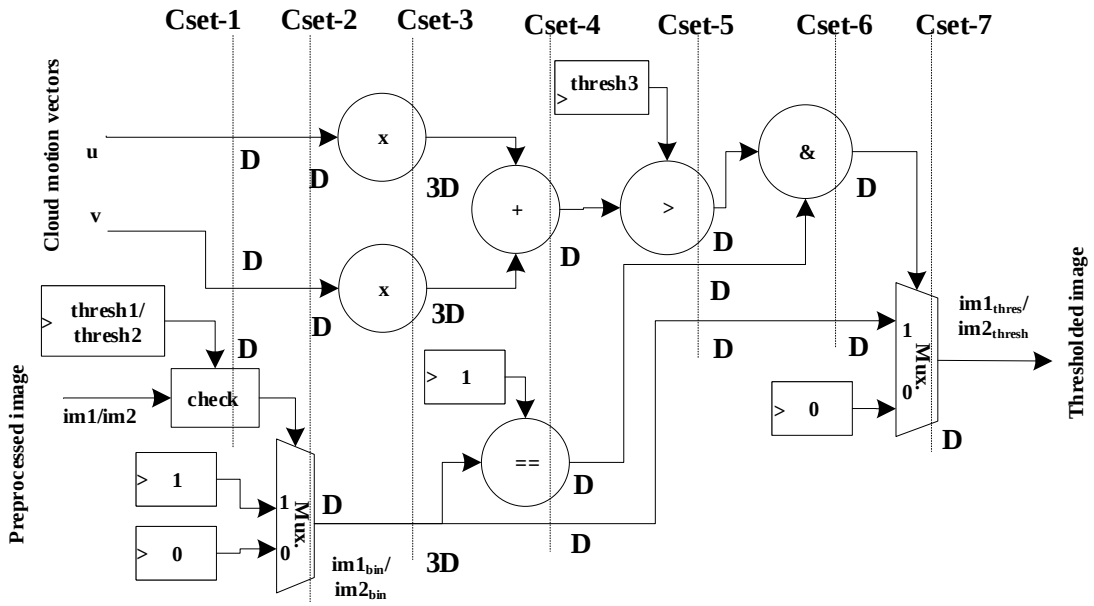


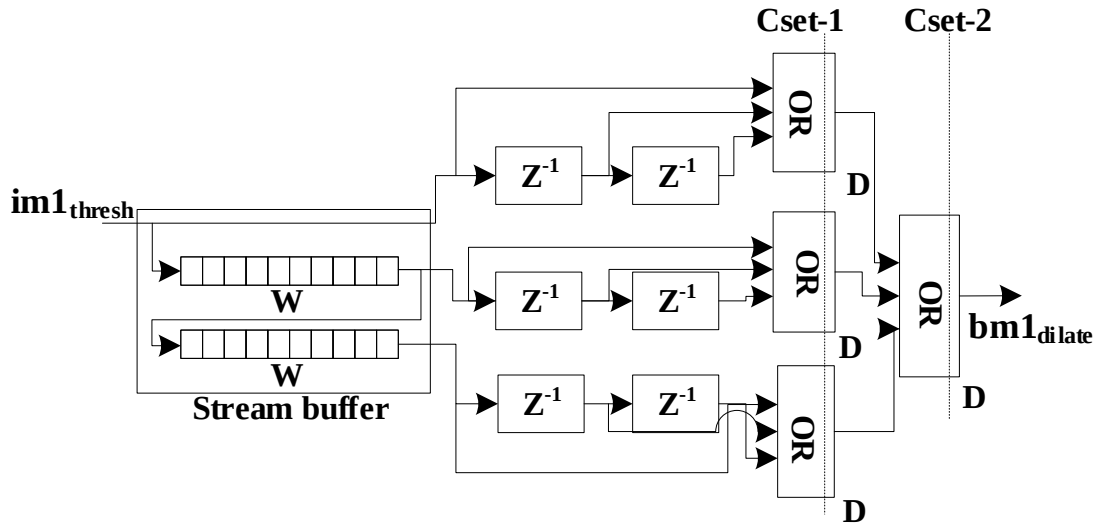
Figure 5.4: Cloud segmentation architecture.

of choosing a fixed value, a scale analysis [151] is performed with a large set of available images to find the optimum value of the threshold used for finding the maximum number of cloud segments.

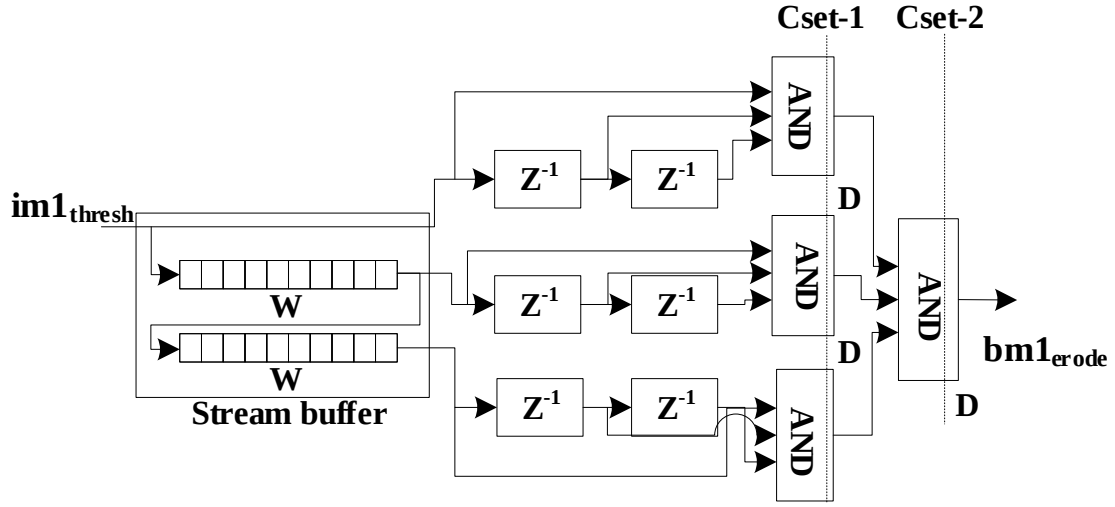
The segmented images are enhanced by morphological filtering [153] to eliminate small objects from the image foreground by placing them in the background and removing small holes in the foreground by changing small islands of background into the foreground. This helps to strengthen the segmented clouds with irregular and thin shapes. The structuring elements are chosen in a way to preserve the shape characteristics and eliminate irrelevances. This gives a better visual representation of cloud patterns. The hardware architecture for the morphological dilation and erosion operators is implemented using basic logic gates. For a structuring element of size 2×2, the dilation and erosion are implemented as shown in Fig. 5.5. The image opening is simply the direct combination of erosion and dilation operation. Similarly, image closing is a direct combination of dilation and erosion. Thus the morphological operation is performed by image opening followed by image closing operation to get the binary images.

5.3.4 Cloud labelling and Parameter extraction:S3

In the cloud labelling module, each of the cloud segments is given a unique number which is useful to separate it from other cloud segments. It helps in computing life cycle char-



(a) Dilation



(b) Erosion

Figure 5.5: Hardware architecture of morphological operation.

acteristics of each cloud segments including generation, dissipation, continuity, splitting and merging. The cloud labelling is performed on a streaming architecture based on the assumption that, if the pixel under consideration has an intensity 1, then all its $P \times Q$ neighbourhood pixels with intensity 1 should be given the same label as that of the considered pixel. Fig. 5.6 shows $P \times Q$ overlapping box inside the image, if the intensity of considered pixel A is 1, then all the pixels in the neighbourhood box with intensity 1 is given the same label as that of A. As the overlapping box scans the entire image stream, if it encounters a pixel with intensity 1 such that the previous pixels in the overlapping box have an intensity

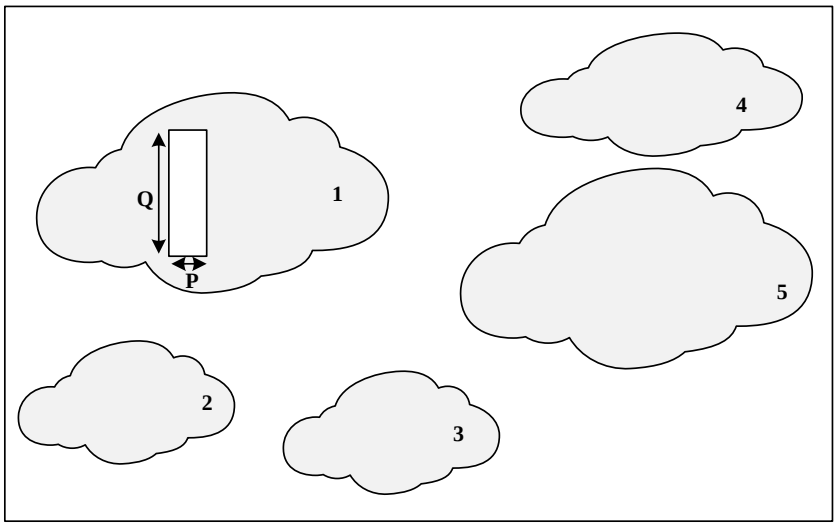


Figure 5.6: Labelling of cloud with the neighbourhood box (a) if the previous pixels inside the box is having intensity 1, then the same label is given to the new pixel (b) else a new label is given to the new pixel.

other than 1, then the selected pixel forms part of another cloud segment and a new label is given to that pixel.

The internal architecture of the cloud parameter extraction is given in Fig. 5.7. The design utilizes a Q-1 line buffers of width W to convert the input pixel stream into two-dimensional patch of size $P \times Q$ and sends it to cloud label retrieval block. It utilizes the overlapping box criteria, to identify if the incoming pixels are related to the same cloud segment or not. Based on which different labels are assigned, which are then fed to the segment boundary retrieval block (L_N).

This module utilizes separate counters to hold the number of non-zero pixels present in each cloud segment, which in turn helps in computing the cloud area in terms of the number of pixels. If the input pixel P has a label 2, then the counter corresponding to the second cloud segment is incremented. So once the entire pixels of the image is scanned, each counter holds the total number of pixels present in each cloud segment. The image index generation block generates the pixel indices for each incoming pixel stream. These indices are utilized to identify the boundaries of a rectangular box enclosing the cloud segment. It is compared with the previously stored indices to update the cloud segment boundaries.

Once the entire image is streamed, each segment block holds the information about the area and boundaries of each cloud segments with different labels. An area thresholding based on the recorded counter values helps to discard cloud segments which are not meet-

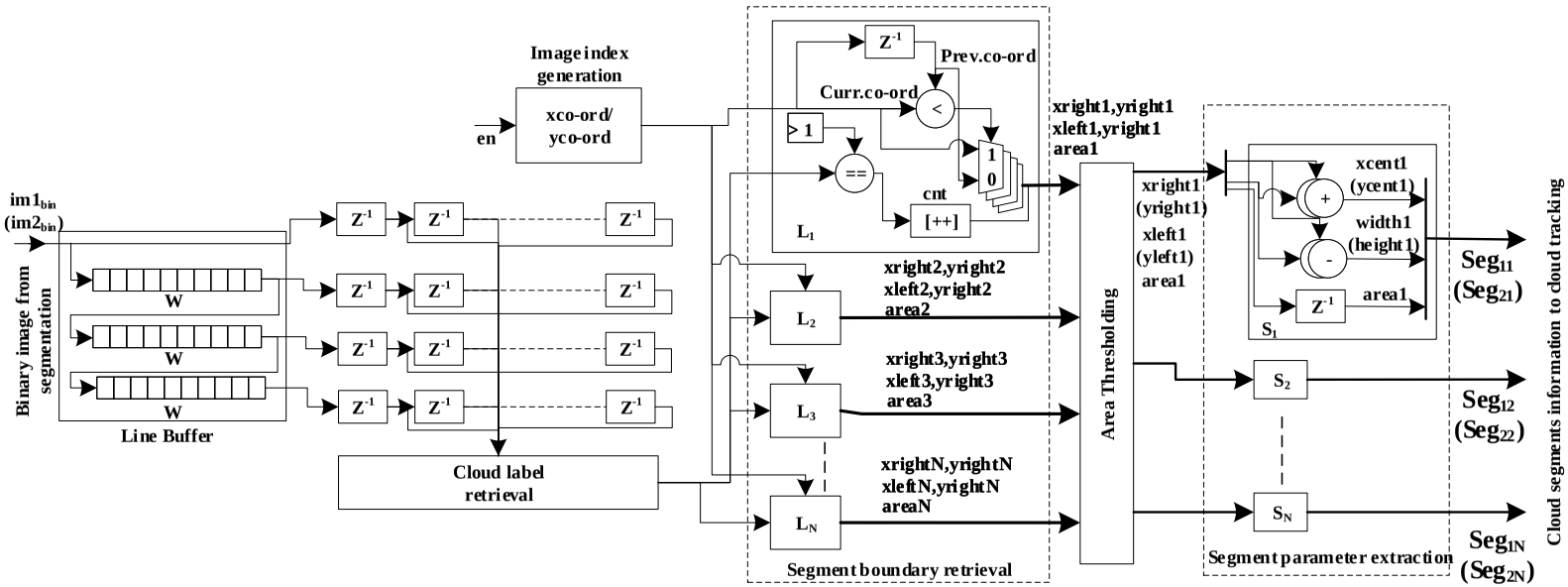


Figure 5.7: Cloud parameter extraction.

ing the minimum selected area. The number of cloud segment estimation blocks are chosen based on several experimental analysis. Similarly, the area threshold is also selected experimentally. It is implemented using conditional statements and multiplexers. The final step is to extract the relevant parameters of each cloud segment. The area and other parameters like width, height, aspect ratio and the centroid of the cloud segments are computed about the rectangular box which encloses the cloud segment as given in the Fig. 5.7.

5.3.5 Cloud Tracking: S4

After extracting the parameters of each cloud segments, all the cloud segments present in the first frame is matched with cloud segments in the second frame. If the parameters of the considered segment in the first frame show a close agreement with the parameter of the cloud segment in the second frame, then there is a high possibility that these two segments correspond to the same cloud. The work utilizes a matching cost based on region overlap based method introduced in the work [154].

The region overlap methods holds valid for satellite images having a high temporal resolution, and hence assume there is no drastic changes in cloud shape. It is based on a simple assumption that there are a set of common pixels in consecutive images with a particular size and temperature. It formulates a tracking cost given in equation (5.2) as a linear combination of the overlap, centroid and size term. The weights $w_{\#}$ represent the importance of each term in the computation of cost function and are chosen wisely based on domain knowledge.

$$\begin{aligned} J(c_i^t, c_j^{t+1}) = w_0 * O(c_i^t, c_j^{t+1}) + \\ w_1 * C(c_i^t, c_j^{t+1}) + w_2 * S(c_i^t, c_j^{t+1}) \end{aligned} \tag{5.2}$$

The Overlap term given in equation (5.3) determines the number of pixels in common for cloud segment in consecutive frames. If the cloud segments show slow movement, the chances of overlaps will be more which leads to the selection of higher weight to overlapping metric. If cloud overlap is above the minimum threshold then the tracker identifies there is a possibility of splitting. The new segment with maximum overlap will keep the same name as the parent segment whereas the less overlapping ones get a new segment name.

$$O(c_i^t, c_j^{t+1}) = 1 - \{\frac{N_{comm}(c_i^t, c_j^{t+1})}{2} * [\frac{1}{A_i^t} + \frac{1}{A_j^{t+1}}]\} \tag{5.3}$$

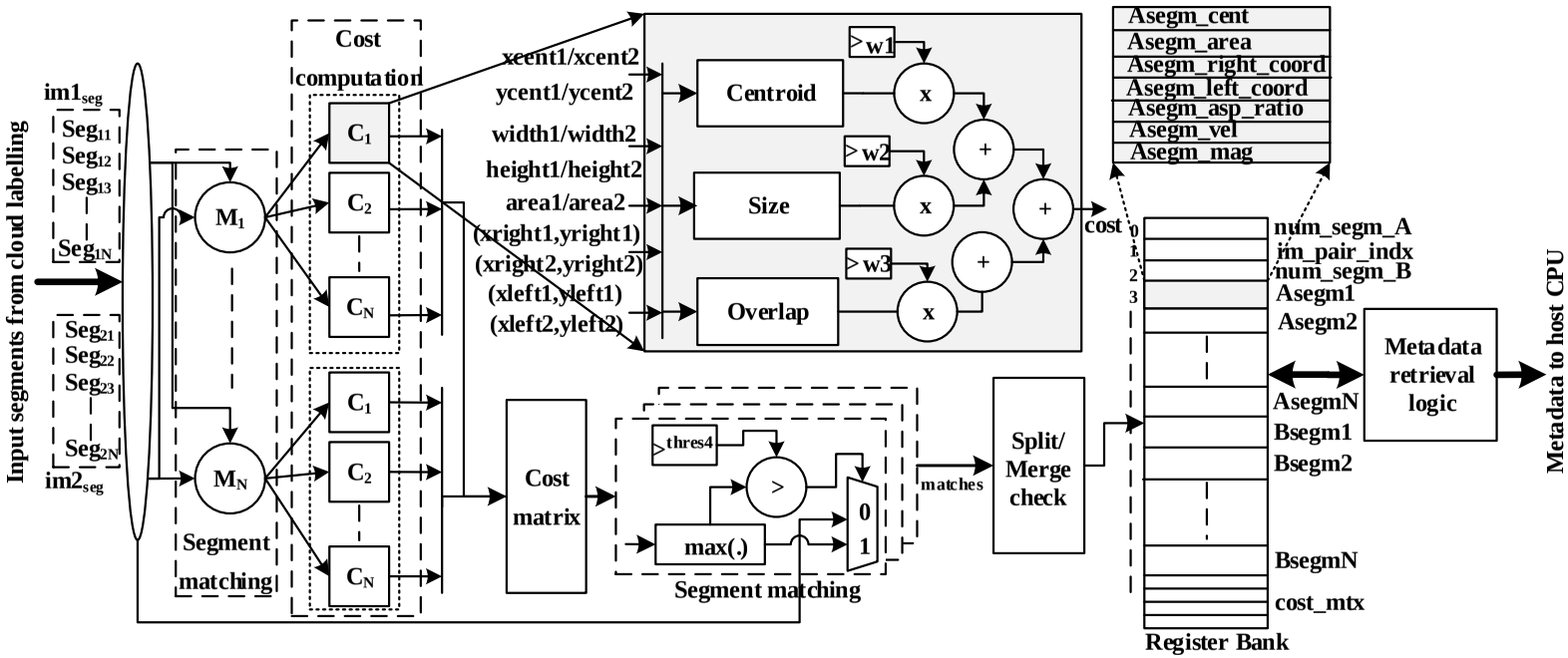


Figure 5.8: Cloud matching.

Here N_{comm} represents the number of common pixels between two consecutive frames, A_i^t and A_j^{t+1} represents the size/area of each cloud segments in consecutive frames. The Centroid Separation term in equation (5.4) gives the Euclidean distance between the centroids of the source and target cloud segments. This parameter is useful to when there is less overlap between the cloud segments due to splitting or merging conditions. In another situation when cloud segments do not overlap or their amount of overlap between different clusters are almost similar, the centroid distance metric will play a major role. The maximum centroid distance in pixels represents the farthest cloud segment which can be matched together.

$$C_(c_i^t, c_j^{t+1}) = \frac{\sqrt{(X_i^t - X_j^{t+1})^2 + (Y_i^t - Y_j^{t+1})^2}}{\sqrt{W^2 + H^2}} \tag{5.4}$$

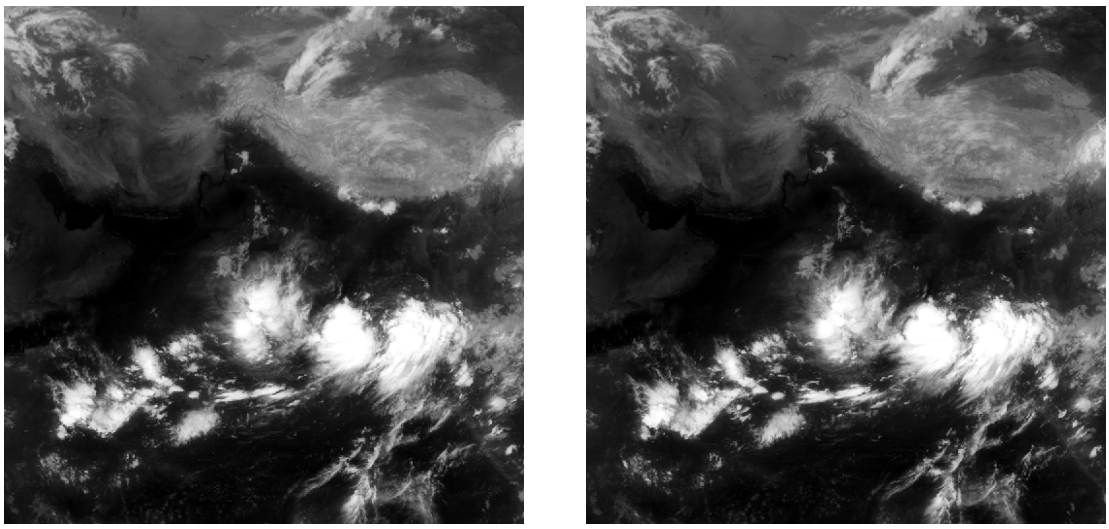
Where (X_i^t, Y_i^t) represents the centroids of each cloud segment in the current frame. W and H gives the cross-sectional information about the tracking images. The Size term in equation (5.5) defines the similarity in the cloud segments size between consecutive frames. Size metric will take low weight, if there exists more number of cloud splitting or merging cases.

$$S_(c_i^t, c_j^{t+1}) = \frac{|A_i^t - A_j^{t+1}|}{\max(A_i^t, A_j^{t+1})} \tag{5.5}$$

The internal architecture of the cloud tracking module is given in Fig. 5.8 to find the best match to generate a continuous track of the cloud segment. The input cloud segments from the two consecutive image frames are sent to the segment matching block. It utilizes M_1 to M_N number of cloud matching blocks. Each block $M_\#$ utilize N_{cost} number of cost computation block (C_1 to C_N) to match one cloud segment in the first frame with all cloud segments in the second frame. The cost is computed simultaneously utilizing a weighted average of the overlap, size and centroid metrics. The metrics computations involve complex division, square root operations which are implemented using second-order approximation by Newton-Raphson [155] method.

The computed cost for all the cloud matches is augmented in a cost matrix. The architecture sort the cost values along the row of the cost matrix to identify the highest value which corresponds to the best match. If the highest cost is less than the chosen threshold ($thresh4$), then the segment parameters are sent to split/merge block to further analyse the split, merge, creation and dissipation of cloud segments. The cloud segment lineage, the cost matrix and each segment information are stored in the register bank formed of a dual port memory. A scheduler inside the metadata retrieval block coordinates the read/write

access from the register bank and conversion into PCIe packets. This metadata is sent to the host PC after the completion of transmission of cloud motion vectors.



(a) Frame 1 **(b)** Frame 2

Figure 5.9: IR image pair taken on 22 May 2009 at (a) 0000 UTC and (b) 0030 UTC.

5.4 Results and Discussion

The hardware accelerator is designed in Verilog and implemented in Xilinx Virtex UltraScale $+$ $VCU118$ development board. The framework is tested on a half-hourly Near Infra Red (NIR) satellite images from Meteosat 7 having $5km \times 5km$ spatial resolution is shown in Fig. 5.9. The computation time of cloud analysis framework on HD image sequence in Matlab running on a single core on Intel Xenon $X5560$ 3.07 GHz PC takes around 1.5 minutes, whereas the proposed hardware accelerator running on the FPGA takes only 14 msec, excluding the time taken for exchanging data between PC and FPGA via PCIe interface.

The CAS utilizes PCIe driver to transmit two 10 bit wide input pixels between host PC and FPGA. After the flow computation the two 16 bit wide output flow values u,v are sent back to the host for further tracking procedures. The data transfer rate is within the available bandwidth of the PCIe $Gen3 \times 8$, assuming that cloud analysis software is the only process running in the host PC. The PCIe controller at the input side of FPGA transfer the received pixels to the pre-processing module using two separate Direct Memory Access (DMA) channels. The PCIe core provides split transmit (TX)/receive (RX) interface, solves the problems of high-speed Direct Memory Access (DMA) by providing an interface which

is generic and adaptable. The data is decoded considering the packet formation rules like addressing, packet sizing. The performance and resource utilization of the submodules and the entire hardware part of the cloud analysis framework are estimated using Xilinx Place and Route stage.

Table 5.1: Selected parameters for cloud analysis framework.

Chara.[1] / **Satellite**	**Resolution**		**Threshold**		
	Spatial	Radiometric	Intensity$_{thresh}$[2]	Area$_{thresh}$[3]	Motion$_{thresh}$[4]
Kalpana	8 km×8 km	10 bit	540 - 880	125	2
Meteosat-7	5 km×5 km	8 bit	130 - 220	200	2
INSAT-3D	4 km×4 km	10 bit	540 - 880	250	2

[1] represents the parameters used by the cloud analysis framework.
[2] represents the pixel intensities used for multi-level thresholding.
[3] denotes the minimum number of pixels present in a given cloud segment.
[4] refers to the minimum displacement required by a foreground pixel.

5.4.1 Performance Analysis

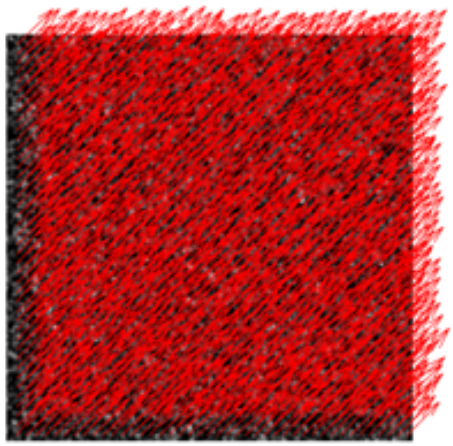
A generic set of parameters are discussed for providing a robust analysis of clouds from satellites with varying spatial and radiometric resolutions. The threshold and other parameters are statistically estimated from a set of 2000 IR images per satellite as stated in Table. 5.1.

Table 5.2: Selected bit-width for different modules.

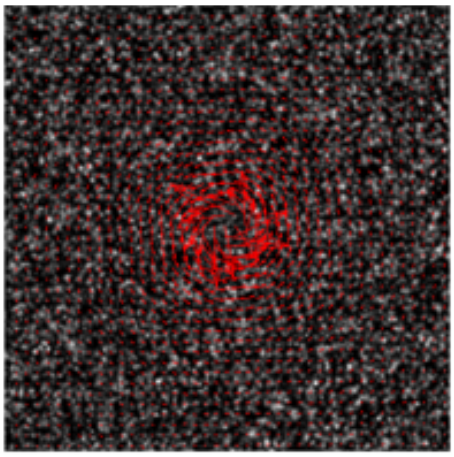
	S0	**S1**	**S2**	**S3**	**S4**
BW$_{min}$[1]	20	28	20	20	32

[1] represent the maximum number of bits utilized for each stage.

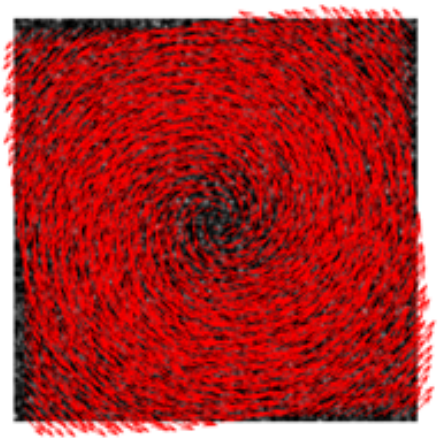
The median filter employs a 5×5 kernel to pre-process the IR imagery. The opening and closing operation employs a disc with 12 and 10 pixel diameter respectively. The variational multi-scale optical flow employs L=4 number of pyramid levels, with N_{solv}=30 number of iterations at each pyramid level. The maximum number of cloud segments that can be processed in each frame is limited to 15. This is achieved by selecting top 15 cloud segments with the highest area. This limits the number of matching unit utilized in the $S4$ stage. The cloud accelerator utilizes a variable precision design based on $Q_{m.n}$ representation of the fixed point number with integer and fractional part. The highest bit-width utilized for each stages is given in Table. 5.2.



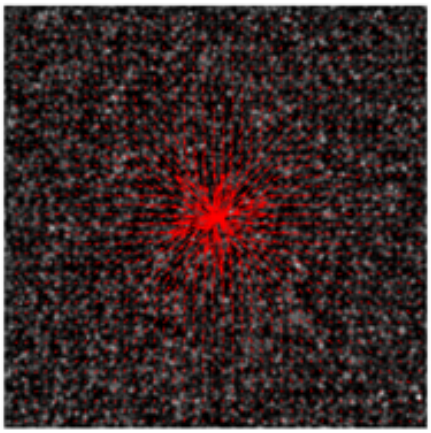
(a) Uniform



(b) Vortex



(c) Lamb-Oseen



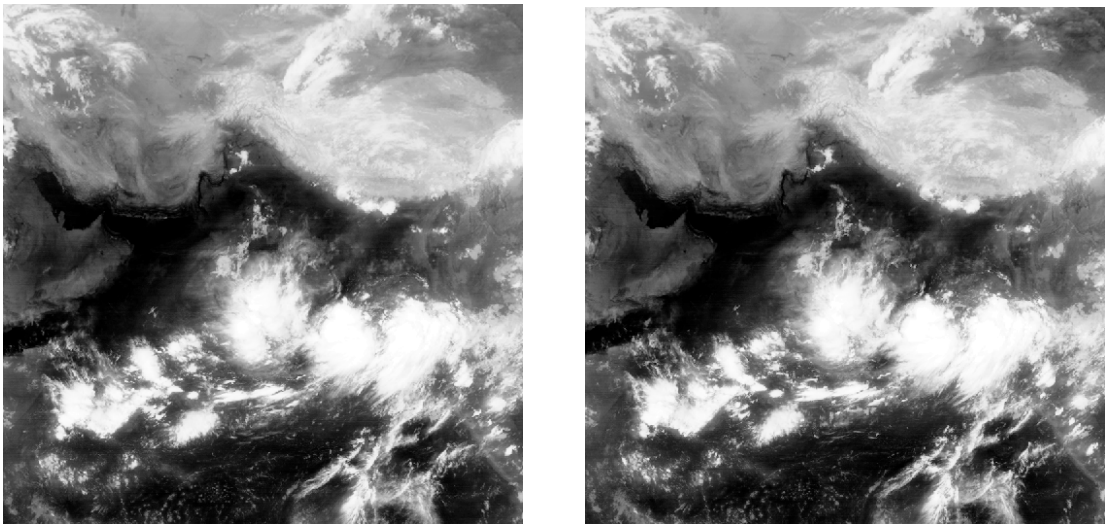
(d) Sink

Figure 5.10: Vector format representation of PIV images.

The performance of the CMC module of the cloud accelerator framework is tested using synthetic fluid Particle Image Velocimetry (PIV) images obtained from the FLUID Project [156] contains different sequences. Fig. 5.10 shows the computed flow vectors obtained from the fluid sequence, which shows close agreement to the ground truth. The red arrows show the magnitude and direction of the estimated motion.

Fig. 5.11 shows the input image after the histogram equalization whereas the Fig. 5.12 shows the computed cloud motion vectors.

The output of the segmentation module of the cloud analysis framework retrieves 5



(a) Frame 1 **(b)** Frame 2

Figure 5.11: Cloud images after histogram equalisation.

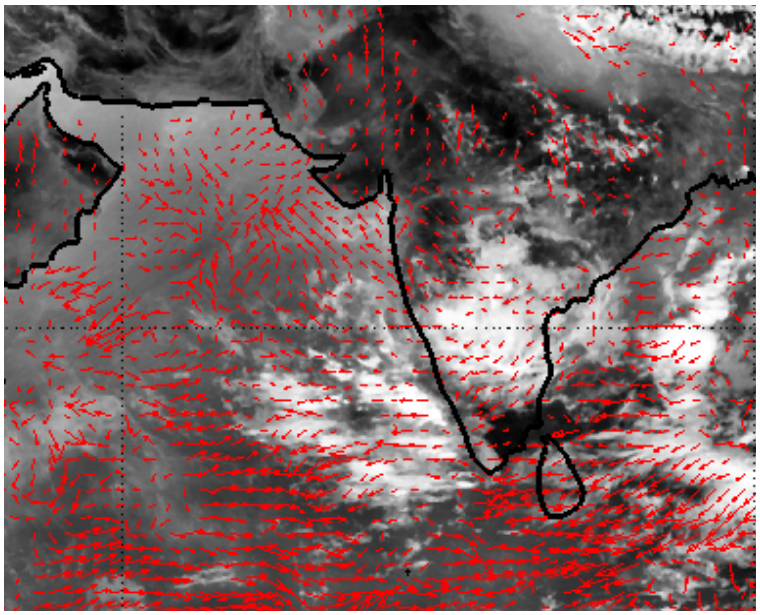


Figure 5.12: Vector format representation of cloud motion field.

cloud segments from the first image and 4 cloud segments from the subsequent image. Each segment is then labelled accordingly to get Fig. 5.13. The cloud segments are displayed with closed boundary lines and represented using a unique cloud number computed during the tracking stage.

Further, these labelled segments are matched with the segments in the subsequent image based on the matching cost given in equation (5.2) as depicted in Table 5.3. The cloud

(a) Frame 1 **(b)** Frame 2

Figure 5.13: Cloud image after labelling.

Table 5.3: Cloud tracking results.

Frame 1	A	B	C	D	E
Frame 2	A	B	C	C	D

parameters of each cloud segments in frame 1 is shown in Table. 5.4.

Table 5.4: Characteristics of the cloud segments in Frame 1.

Chara.[1] / **Segment**	**Height**[2]	**Width**[2]	**Centroid**	**Area**[2]
A	75	96	(147,624)	3402
B	126	98	(406,488)	7500
C	109	106	(527,517)	7708
D	144	137	(653,530)	10990
E	65	58	(780,220)	2624

[1] represents the cloud segment characteristics.
[2] denotes that the cloud segment parameters in terms of number of pixels.

5.4.2 Resource and Timing Analysis

The device utilization of the proposed hardware accelerator for cloud analysis is shown in Table. 5.5. It can be inferred from the Table that $S2$ stage uses a minimum number of slices,

Table 5.5: Resource utilization of cloud analysis framework with 30 RBSOR solver iteration.

Module / Resource	S0	S1	S2	S3	S4	System[1]
LUT	35544	1,37597	151	62741	1,01223	3,35285
FF	46882	1,66485	496	50864	1,13166	3,77241
BRAM36	451	1327	9	76	0	1863
URAM	0	960	0	0	0	960
DSP48	0	544	0	0	1620	2166

[1] represent the resource utilization of the entire cloud accelerator framework including the PCIe-DMA subsystem.

while the $S1$ stage with 30 solver iterations consumes the highest resource. The $S1$ and $S4$ stage consumes a large number of DSP48 slices for implementing the cost function for cloud motion computation and matching the cloud segments respectively. Most of the Ultra RAMs are utilized to construct Pyramid, Flow resize and Warping memory banks. It also consumes a large amount of BRAMs for implementing the line buffers for intermediate data storage in OF, RBSOR and BF stages.

Table 5.6: Performance of cloud accelerator framework for different RBSOR solver iterations.

Iteration / Resource	10	20	30
LUT	3,04489 (25.7%)	3,26884 (27.6%)	3,47130 (29.4%)
FF	3,24260 (13.7%)	3,59618 (15.2%)	3,91758 (16.5%)
BRAM36	1225 (56.7%)	1577 (73%)	1897 (87.8%)
URAM	960 (100%)	960 (100%)	960 (100%)
DSP48	1830 (26.8%)	2006 (29.3%)	2166 (31.7%)
CLB	63431 (42.9%)	68576 (46.4%)	73351 (49.6%)
Dynamic(W)	21.45	27.8	40.7
Fmax (MHz)	230	235	220

The resource utilization and performance of the cloud analysis framework for a varying number of RBSOR solver iteration is computed in Table. 5.6. From the table, it can be

observed that resource utilization is not constant for a different number of solver iterations. The cloud analysis framework with 30 RBSOR solver iterations shows 36% increase in the BRAMs utilization and 15% increase in the DSP48 usage. A slight reduction in the maximum frequency of operation of the accelerator can be observed with an increasing number of iteration due to the additional routing delay. The cloud accelerator with 30 solver iterations is running at a maximum frequency of 220 MHz, consuming a dynamic power of 40.7 W. The major part of the dynamic power is consumed by DSP48 and BRAM/URAM blocks. The implementation of the cloud accelerator framework achieves a real-time performance of 71 fps for HD resolution images. The proposed accelerator framework is more than $100\times$ faster than the existing CPU implementation.

Table 5.7: Performance of cloud analysis framework for various image resolution.

Format	Resolution	Latency (ms)	FPS[1]
nHD	640*360	3.5	286
qHD	960*540	7.8	127
HD	1280*720	13.9	71
HD+	1600*900	21.8	45

[1] represent the number of frames processed per second.

The cloud accelerator framework is directly scalable to handle a multitude of image resolutions as shown in Table 5.7. The latency of the cloud accelerator framework increase with the image resolution due to a large amount of pixel operation and degradation in the maximum frequency of operation. It also has a significant impact on the memory banks and intermediate data flow buffering. The number of RBSOR solver iterations and the maximum supported frame resolution is limited by the amount of DSP48s and BRAMs available in the current FPGA. Moving to another FPGA device with higher resource utilization or switching to a multi-FPGA platform helps to solve the issue.

5.5 Validation

Since the framework is novel, a direct comparison is not possible, instead, the performance of the framework is analysed based on several use cases.

5.5.1 Case 1: Tracking of long and short term cloud patterns

One of the important application of the cloud analysis framework is to study the variation of the cloud cover in a particular region by detecting and tracking of short and long term cloud pattern changes, the change in the direction of wind flow and surface area of the cloud mass by analysing large historical image database.

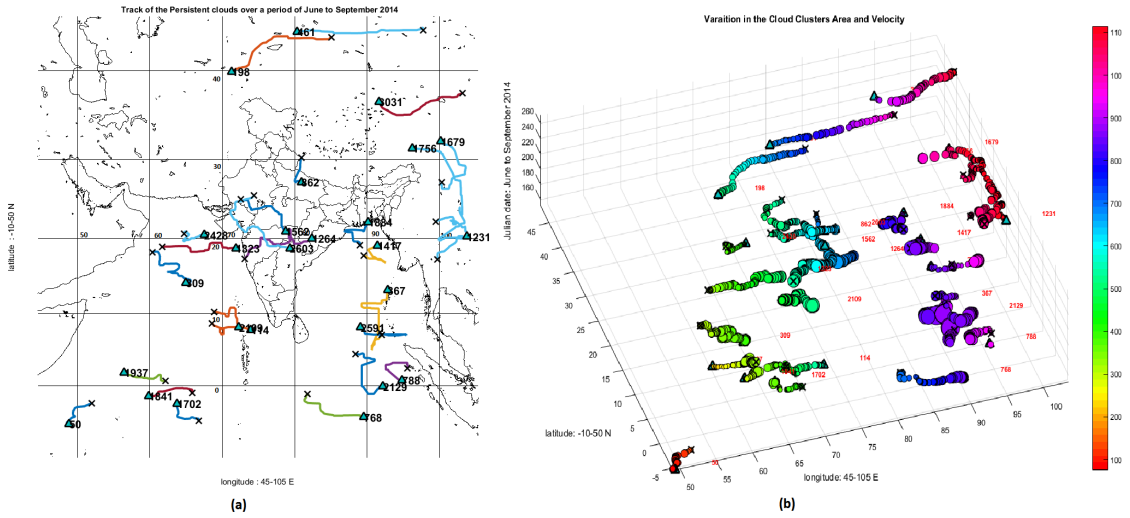


Figure 5.14: (a)Trajectories of cloud segments over a period of one month. The cloud clusters which persists for a minimum of 50 frames are only displayed. Green blob indicates the birth, the red cross represents the death and the small black dot shows each frame instance,(b) Variation of the cloud cluster's velocity at different locations(Interpolated using blue lines)

The cloud analysis framework is used to analyse cloud pattern formation from a large satellite database. The Fig.5.14 represents the track of the cloud system on top of the Indian region during the month of June to September 2014. The estimated track shows a close match to the true path on visual inspection. Similarly, other cloud parameters like size, speed, life-cycle etc are computed for making several scientific conclusions about the cloud propagation characteristics.

5.5.2 Case 2: Tropical Cyclone Tracking

The Tropical cyclones (TC) are an intense low-pressure area with organized circulating storms formed over warm tropical oceans [157], circulating either anti-clockwise or clockwise direction. It is considered as one of the major sources of disaster resulting in a huge catastrophe in the affected regions. The life cycle of a tropical cyclone can be characterized

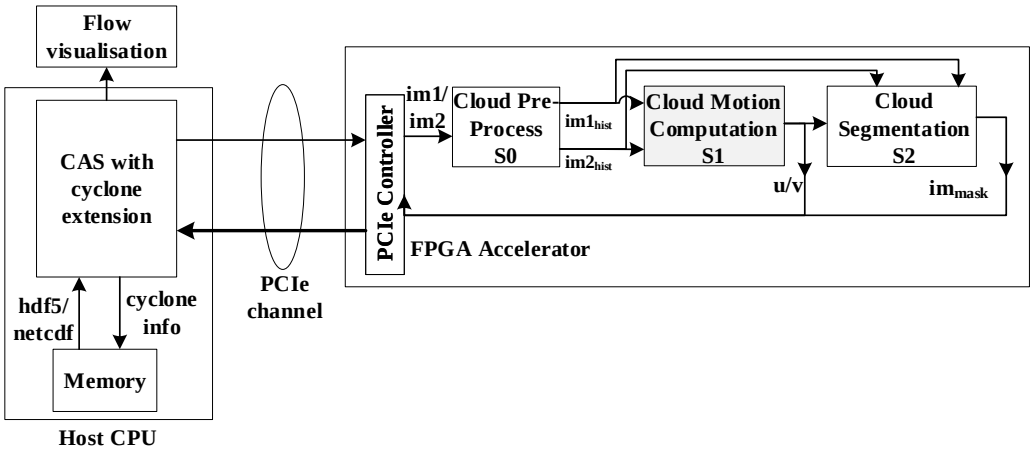


Figure 5.15: Proposed modified cloud analysis framework to track cyclones.

into formation, maturity and dissipation phases. An accurate analysis of the satellite data greatly helps the forecasters for a better prediction of the track and the potential landfall of the TC. The TC centre detection and tracking involve various computations stages such as pre-processing, CMC, segmentation, vortex detection and tracking. Since most of the processing stages are similar to the cloud accelerator framework, it is modified to add the support for TC centre detection and tracking. The modification involves an extension to cloud analysis software to incorporate spatiotemporal and spatial gradient scheme. The modified framework is shown in Fig. 5.15.

5.5.2.1 CAS extension

The CAS software retrieves the input pixel intensity from the raw image data. Due to the seasonal variability i.e. the time during which sun falls in the field of view of the satellite and infrequent glitches in the capturing or transmission system leads to missing or noisy frames. Hence the input frames are a) checked for highly uniform regions or subregions by computing the image variance, b) checked for the minimum threshold of the gradient magnitude in horizontal and vertical directions and c) checked for a minimum number of non zero pixel intensities. If the frames are consecutive and have less distortion, spatiotemporal gradient scheme is selected and in the case of missing or noisy frames, spatial gradient scheme is used as shown in the Fig. 5.16.

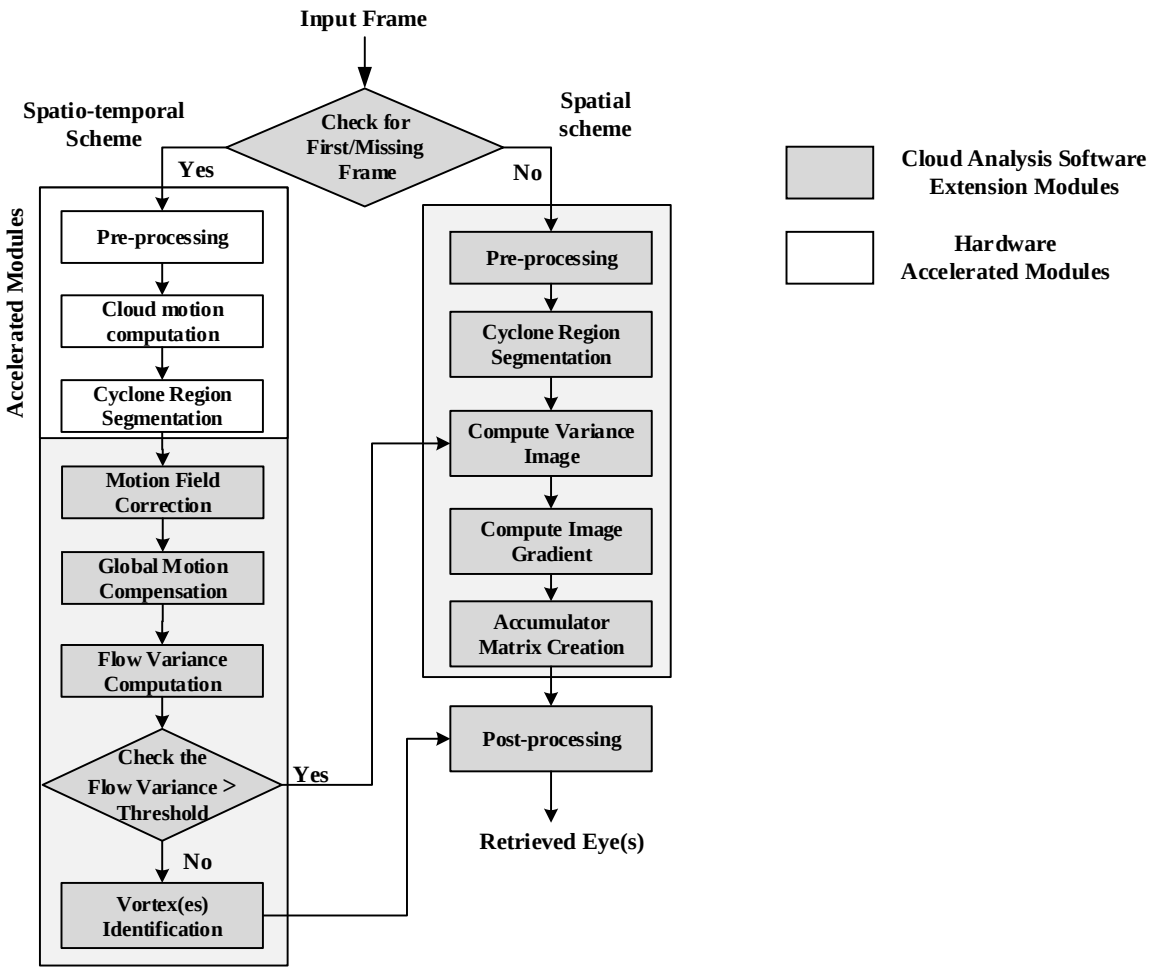


Figure 5.16: Proposed cloud analysis framework with CAS extension.

Spatio-temporal gradient scheme

If the spatiotemporal gradient scheme is selected, the input images are fed to the cloud analysis framework using a PCIe interface. The image is pre-processed and fed to the CMC module to extracts the motion vectors. The output flow field along with the intensity values are fed to the cloud segmentation unit to generate a threshold mask. The computed cloud mask along with the motion vectors is sent back to host PC.

The CAS extension software extracts the highest cloud segment in the frame to identify the cyclone region. The computed flow field contains spurious vectors due to the registration error and the inaccuracy in modelling the highly nonlinear dynamics of the cloud motion. A Navier-Stokes based smoothing [158] is adopted to impose fluid specific constraints on the flow field. It avoids the presence of noisy vectors and forces the flow vector to account for divergence and curl of the fluid motion, followed by smoothing in the fre-

quency domain. The histogram-based compensation method proposed in the work [159] is employed to remove the global motion in the retrieved field, caused by the limitation in the registration accuracy. It is based on an assumption that any IR image contains fewer cyclone-related pixels than background pixel and they have zero motion. But the motion field histograms have a non zero offset, signifying the presence of random motion in the background pixels. This offset is computed and utilized for flow compensation, to get a clearer and less noisy field.

The variance of cloud motion field in the cyclonic region is extracted from the vorticity of the rotated spatiotemporal flow structures estimated to find the phase of the cyclone. The vortex centre of the cyclonic region is determined by identifying the location where the vorticity component of the flow is dominating. The computed optical flow contains translational, divergence and vorticity components. Among them, the divergence and the vorticity are the two key parameters of the TC motion which is used for the identification of TC centre. The deformation tensor of the flow vector $\vec{U}$ is created as,

$$\nabla\vec{U} = \begin{bmatrix} \frac{\partial u}{\partial x} & \frac{\partial v}{\partial x} \\ \frac{\partial u}{\partial y} & \frac{\partial v}{\partial y} \end{bmatrix}$$

The deformation tensor is then decomposed into a symmetric part (S) and an asymmetric part (Ω) expressed as,

$$S = \frac{1}{2}(\nabla\vec{U} + \nabla\vec{U}^T) \tag{5.6}$$

$$\Omega = \frac{1}{2}(\nabla\vec{U} - \nabla\vec{U}^T) \tag{5.7}$$

Symmetric and asymmetric parts of the deformation tensor represents the strain and the rotation of the flow respectively. From the symmetric and the asymmetric parts, the vortex core is identified based on the Q-criterion.

$$Q = \frac{1}{2}(\|\Omega\|^2 - \|S\|^2) \tag{5.8}$$

where $\|.\|$ is the Hilbert-Schmidt norm of the matrix. The Q value measures the dominance of vorticity in the flow field. When $Q > 0$, the vorticity part dominates the flow and the corresponding location is considered as the vortex core or initial estimate of TC centre TC_{init}. The post-processing stage fuses the information from weather models and previous extrapolated locations to improve the initial TC centre estimate in case of noisy images. A small two-dimensional patch of size $R \times S$ surrounding the initial estimate of the TC centre

is extracted from the model output. The TC centre is characterized by 15% lower pressure compared to the nearby positions in the mean sea level (MSL) pressure profile. Also, the vortex centre estimated from 10 m wind field is utilized to compute TC_{mod}.

$$TC_{mod} = \alpha_1 * TC_{msl} + \alpha_2 * TC_{vel} \tag{5.9}$$

The parameters α_1, α_2 signifies the importance of pressure and velocity based center estimates. The MSL based TC center retrieval is given more weight as compared to the vortex estimate. A polynomial extrapolation based on the previously stored TC center locations are used for TC_{fit} predictions.

$$TC_{fit} = \frac{1}{N_{store}} \sum_{i=1}^{N_{store}} a_i * TC_{final}(i) \tag{5.10}$$

The predictor coefficients a_i are computed based on parallel discrete Kalman filter implemented in the work [160]. The model forecast TC_{mod} is given more weight than extrapolated data TC_{fit}.

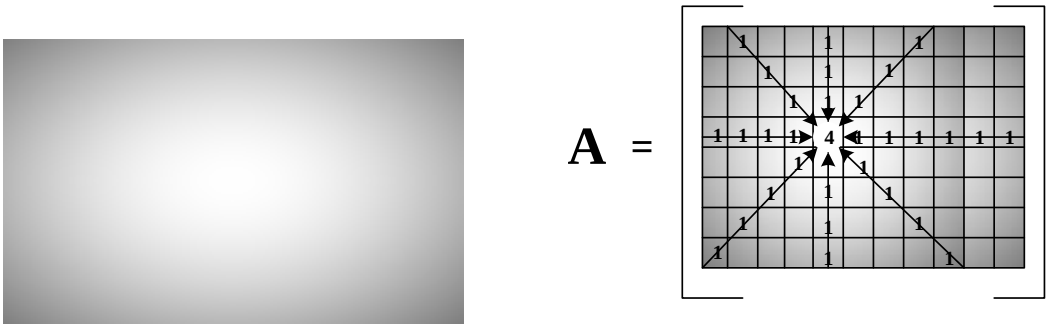
$$TC_{final} = w_{fit} * TC_{fit} + w_{mod} * TC_{mod} + w_{vort} * TC_{init} \tag{5.11}$$

where w_{fit}, w_{mod} represents the weights for the polynomial extrapolations and model forecasts. The final TC centre TC_{final} is estimated by the weighted average of TC centre estimate from the vortex, extrapolation and model forecasts.

Spatial gradient scheme

If the images are partial or missing and contain large noise a spatial gradient scheme is utilized. The cloud accelerator framework is not utilized in this case for CMC computation. The CAS software is modified to compute the TC from the noisy image. It computes the variance image V as the difference of the input pixel intensity from the mean intensity, for a small pixel neighbourhood Ω. A low value of variance is a characteristic feature near the vortex centre, corresponding to high axisymmetric about the neighbouring pixels.

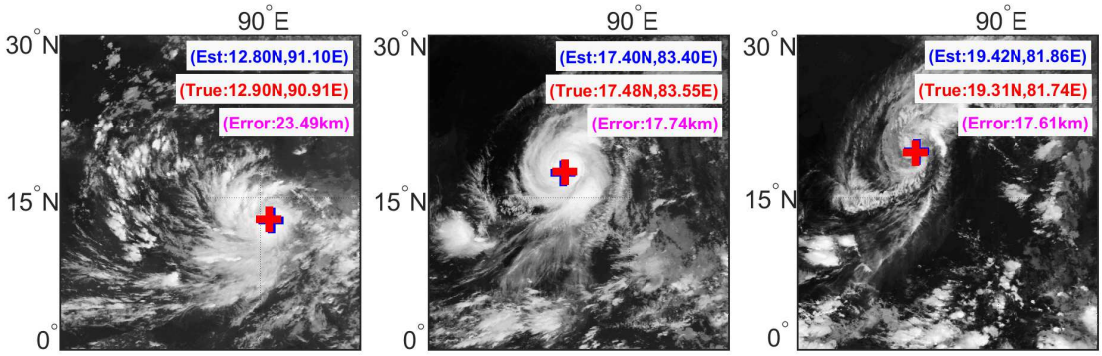
An accumulator matrix is constructed by augmenting the gradient vector orientations. It holds the number of times the gradient vectors have passed through a particular location in the image frame. For this, each gradient vector is extended and the score of locations where these lines pass is accumulated as shown in Fig. 5.17. The location with the highest score corresponds to the point where the maximum number of gradient vector intersects.



(a) Ideal TC **(b)** Accumulator matrix

Figure 5.17: TC centre corresponding to the point where four gradient vectors in the accumulator matrix intersects.

This location is identified from the accumulator matrix and is considered as the initial estimate TC_{init} of the TC centre, this is then fed to the post-processing stage to eliminate the ambiguity in the estimated TC centre.



(a) **(b)** **(c)**

Figure 5.18: The blue, red star pointer denotes the estimated and true center of the TC Hudhud during its entire life-cycle respectively (a) Formation phase (1030 UTC 8^{th} October 2014) (b) Maturity phase (0430 UTC 12^{th} October 2014) (c) Dissipation phase (2130 UTC 12^{th} October 2014)

5.5.2.2 Experimental Analysis

The variance of the image patch is computed for 350 km radius around the cyclonic region, with most of the circulation features concentrating near to TC centre. The post-processing extracts a small patch of 30 km radius from the model output based on the initial TC centre. The polynomial extrapolation needs 10 previously stored TC centres to obtain TC_{final}. The

TC considered being mainly concentrated on the Indian region to show the applicability of the framework for missing and noisy frames from Kalpana and INSAT satellites. The estimated and true centre of the TC Hudhud along with their deviation is shown in Fig. 5.18. An approximate of 200 images per satellite is employed for estimating the TC centre.

The accuracy of the proposed methodology is tested with several TCs that formed during the period of 2009 to 2014 in the Northern Indian oceans. Table. 5.8 shows the comparison of the mean track error of the proposed TC centre with other existing algorithms in terms of kilometres.

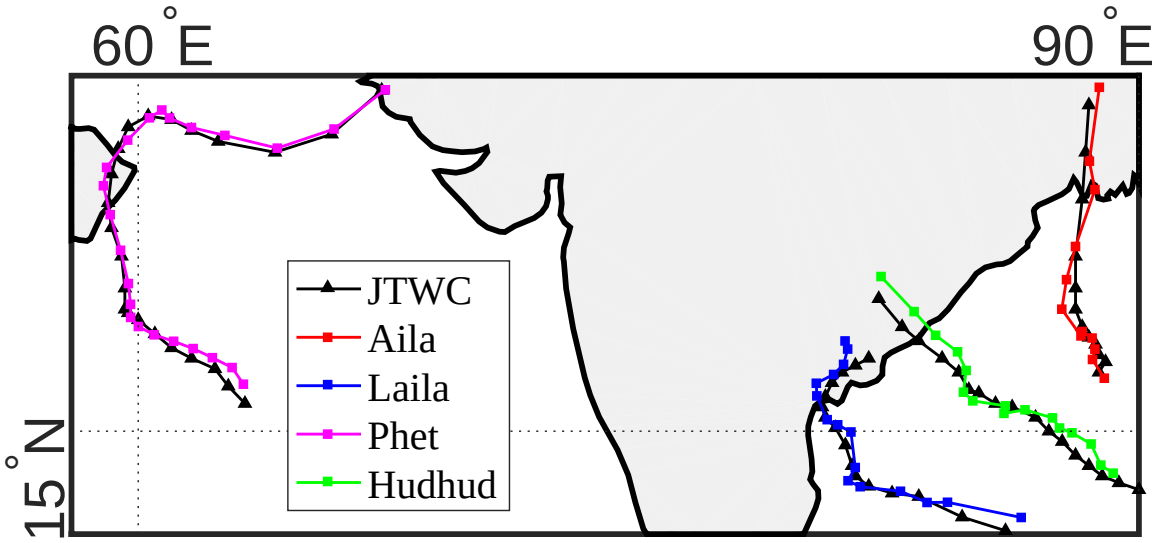


Figure 5.19: Comparison of the Original (JTWC) and Estimated Track for Different TCs

The mean track error for different algorithms in the presence of noisy image frames is shown in Table. 5.9. The table highlights the applicability of the proposed method in the presence of significant noisy density. Even though the mean track error increases with noise density, the proposed methodology is still able to handle images with significant noise without affecting the performance. The testing of the methodology in a real scenario with missing frames is not possible as the accuracy cannot be computed. Hence the testing is done with a continuous sequence of TC images with selectively discarding multiple frames. The Fig. 5.19 shows the computed cyclone centre track in comparison with Joint Typhoon Warning Centre (JTWC) tracks which are assumed to be true. It shows a reduction of mean track error by 11% as compared to the other state of the art methods.

Table 5.8: Performance of cloud analysis framework without noise.

Phases / Satellite		Formation			Maturity			Dissipation			Mean Track Error		
	TC	Liu[161]	Jaiswal[162]	Proposed	Liu[161]	Jaiswal[162]	Proposed	Liu[161]	Jaiswal[162]	Proposed	Liu[161]	Jaiswal[162]	Proposed
MET7	**Aila**	39.64	52.58	**39.42**	38.27	28.19	**27.94**	50.90	46.09	**45.41**	42.04	40.58	**35.53**
	Hudhud	47.99	75.65	**44.99**	45.80	37.31	**35.63**	73.15	65.81	**65.49**	54.63	58.15	**45.90**
	Laila	47.90	53.95	**45.84**	37.65	30.78	**29.83**	71.84	58.73	**55.43**	46.62	45.36	**40.95**
	Phet	39.92	46.51	**37.40**	31.12	28.92	**28.48**	43.83	35.28	**34.31**	36.11	36.73	**32.78**
	Twin	-	-	**44.51**	-	-	**32.37**	-	-	**41.34**	-	-	**37.41**
K1	**Aila**	44.56	57.89	**43.21**	43.89	35.23	**34.04**	53.35	50.31	**48.93**	47.63	46.17	**42.54**
	Hudhud	51.70	80.01	**48.17**	49.12	39.28	**36.26**	70.75	78.18	**66.02**	58.04	63.64	**50.45**
	Laila	61.62	67.11	**59.49**	48.02	45.50	**42.47**	69.19	71.21	**65.57**	57.41	59.03	**53.90**
	Phet	48.44	53.10	**48.25**	34.36	32.39	**31.06**	43.73	46.82	**40.98**	43.22	41.57	**38.64**
3D	**Hudhud**	47.43	71.49	**46.52**	36.37	33.41	**31.61**	73.58	63.38	**59.77**	49.31	57.64	**45.28**
Merged	**Katrina**	88.26	86.44	**79.07**	71.48	68.13	**63.65**	87.04	85.98	**78.38**	76.58	81.22	**72.18**
	George	75.98	75.43	**70.11**	66.76	63.34	**60.93**	75.40	78.18	**69.08**	69.63	73.55	**65.49**

Table 5.9: Performance of cloud analysis framework with various noise densities.

		Variance														
		0.001			0.02			0.04			0.06			0.08		
SAT	TC	Liu[161]	Jaiswal[162]	Proposed	Liu[161]	Jaiswal[162]	Proposed	Liu[161]	Jaiswal[162]	Proposed	Liu[161]	Jaiswal[162]	Proposed	Liu[161]	Jaiswal[162]	Proposed
Met7	**Aila**	43.11	42.97	**36.03**	45.22	44.89	**37.09**	46.40	45.68	**37.27**	47.01	47.15	**38.64**	50.33	49.59	**40.26**
	Hudhud	57.57	63.88	**48.62**	58.14	64.40	**48.98**	60.92	65.31	**49.35**	61.09	66.06	**51.47**	64.46	70.13	**54.72**
	Laila	48.28	49.75	**42.29**	49.46	50.91	**42.42**	51.03	52.12	**43.91**	52.82	52.89	**44.38**	54.48	55.61	**45.29**
	Phet	37.43	38.14	**33.27**	38.25	40.29	**34.76**	40.70	41.64	**35.12**	41.47	41.95	**36.79**	42.04	44.87	**37.66**
K1	**Aila**	48.31	48.87	**43.21**	50.79	50.96	**43.81**	51.31	51.78	**44.07**	52.69	53.02	**44.92**	55.83	56.17	**46.37**
	Hudhud	61.39	68.04	**51.78**	62.22	68.89	**52.32**	63.70	69.45	**53.84**	65.47	71.58	**55.38**	68.62	75.62	**58.35**
	Laila	59.92	63.31	**54.67**	60.61	64.17	**55.63**	62.09	66.31	**57.77**	63.54	66.84	**58.82**	65.85	68.59	**58.72**
	Phet	44.37	43.78	**39.49**	45.40	45.98	**40.11**	47.82	48.35	**42.83**	48.79	49.67	**43.20**	51.21	52.01	**44.14**
3D	**Hudhud**	52.26	62.85	**46.71**	53.05	63.23	**48.36**	55.37	63.88	**49.02**	57.51	65.09	**50.85**	59.80	69.24	**52.92**
Merged	**Katrina**	78.89	83.10	**74.06**	80.52	84.60	**75.37**	82.15	85.29	**76.51**	83.97	87.04	**76.92**	85.81	88.39	**78.69**
	George	72.33	78.42	**67.27**	74.98	79.60	**68.16**	75.56	81.04	**70.33**	77.58	82.14	**70.91**	79.59	84.11	**71.23**

5.6 Summary

The chapter described about the design of a high throughput hardware accelerator for cloud/cyclone analysis and tracking. The design involves several algorithm adaptations including the cost function and utilization of the aforementioned variational multi-scale OF and other supporting architectures. The proposed cloud accelerator framework is tested with real-world satellite images achieving a maximum throughput of 71 fps for HD images. The proposed framework is modified to detect and track TC centre during its entire life-cycle with a reduction in mean track error by 11% as compared to the other state of the art methods.

Chapter 6

Conclusion with Future scope

6.1 Conclusions

In recent years, high-throughput implementations become a major requirement in the field of VLSI design of embedded and mobile applications. The computation of dense OF of fast moving objects finds a wide range of applications ranging from vision aided-robots to unmanned areal vehicles. The work explores different OF implementations and comes up with a high throughput multi-scale non-linear HSOF architecture for tracking the fast-moving object in real-time without much loss in accuracy. To improve the area efficiency of the slow converging Gauss-Jacobi solver in the multi-scale non-linear HSOF implementations, the work proposes a high throughput architecture for RBSOR solver. The research work also focuses on the design of a cloud/cyclone tracking framework for tracking the fast moving cloud segments from infrared satellite images. The throughput performance of the tracking framework is improved by proposing a high throughput VLSI architecture utilizing proposed multi-scale non-linear HSOF, flow filtering architectures and supporting architectures.

6.1.1 Contributions

The contributions of this research work are summarized as follows.

The first part of the research work focuses on the hardware adaptation and design of the time-sharing architecture of variational multi-scale OF algorithm for real-time large-scale motion computation. The dedicated memory banks and the special access schemes achieve superior area and energy efficiency while attaining high parallelism and accuracy. The design is scalable to fit in an embedded device with different image resolution in real-time while consuming low-power. This is the first work on deeply pipelined time-sharing ar-

chitecture for variational multi-scale OF to capture dense and accurate OF of fast-moving objects from HD frames in real-time (176 fps). The architecture makes use of 169 super-scalar units with 702 deep pipelines to achieve a throughput of 395 Giga Operations Per Second (GOPS) with a computation density of 21.5 GOPS/Watt.

The work also proposes two major improvements in internal subsystems of the multi-scale variational OF architecture to improve the throughput and resource utilization. It involves the design of high throughput Red-Black SOR solver architecture for variational HTOF computation. The proposed variational HTOF based architecture is capable of computing OF for UHD images at 48 fps. The design achieves the highest throughput of 491 GOPS with a power efficiency of 43 GOPS/W at 412 MHz compared to the state of art architectures. The FPGA implementation consumes $2.5\times$ low power and 30% fewer resources in comparison with state of the art architectures and is scalable to process variable size flow values. Another improvement is the design of high throughput BF architecture for flow denoising. The proposed HTBF architecture is tuned to achieve the highest throughput at the cost of a peak power consumption of 510 mW while operating at 467 MHz. Experimental results show the highest performance of HTBF architecture in terms of area normalized speedup ($5.8\times$) and area-technology normalized speedup ($3.4\times$) compared to other existing architectures. It also modifies HTBF architecture, to adapt the range filter coefficients by varying noise level (η) in the input flow field.

The second part of the research focuses on the design of a general framework based on a selective choice and application of different computer vision techniques for cloud/cyclone analysis and tracking. The software implementation of the framework is complex and require huge computational time for processing high-resolution data of which variational multi-scale OF computation is one of the major contributors. Hence the proposed variational multi-scale OF architecture with its improved subsystems are integrated to design a high-speed accelerator for cloud analysis. The design involves several algorithm adaptations including the cost function and utilization of the aforementioned variational multi-scale OF and other supporting architectures. The proposed cloud analysis framework is tested with real-world satellite images achieving a maximum throughput of 71 fps for HD images. The accelerator framework is modified to track cyclones from near real-time satellite data. The modified framework is tested with different TCs from various Geostationary satellites such as the Meteosat-7, INSAT-$3D$, Kalpana-1 etc. The computed track is compared with the actual track data obtained from Joint Typhoon Warning Centre (JTWC), and it shows a reduction of mean track error by 11% as compared to the other state of the art methods.

6.2 Future work

There are several areas in which this work can be further explored: exploring the architecture design of complex cost functional to consider non-linear and higher order data and smoothness terms to handle complex non-linear motions, exploring the feasibility of a multi-FPGA platform or hybrid systems to implement more complex vision applications based on the proposed architectures for high resolution image sequence, modification of the proposed architecture for a low power on-board cyclone tracking in high frame rate satellites to provide online tracking of hazardous climatic situations.

The research work shows the feasibility of implementing the high throughput architectures for computing OF of fast moving objects. The motion model used in the algorithm might be violated in complex non-linear motions. With the availability of high computational resources, more general motion models can be developed to accommodate for more general environments. This can be analysed by modifying the cost functional to implement dense and accurate high throughput fluid flow / biological cell trackers in aerospace and biomedical industries respectively.

When more complex and computationally intensive non-linear model needs to be implemented, the current single FPGA platform may not be able to offer enough capacity. Hence the architectural modification required for enabling multi-FPGA operation can be explored, with the hardware pipelines divided into separate partitions that can be realized on different FPGAs. Data flow and control signals are communicated between FPGAs through normal or high speed I/Os. Data flow control and synchronization pose new challenges in algorithm development, FPGA implementation, simulation and verification. Another possibility is to design heterogeneous/hybrid systems and partition the compute intensive and control operation to appropriate platforms.

Bibliography

[1] B. K. Horn and B. G. Schunck, "Determining optical flow," in *1981 Technical symposium east*. International Society for Optics and Photonics, 1981, pp. 319–331.

[2] D. L. Russell and L. C. Young, *Calculus of variations and control theory: proceedings of a symposium conducted by the Mathematics Research Center, University of Wisconsin-Madison, September 22-24, 1975*. Academic Press, 1976, no. 36.

[3] Z. Tu, W. Xie, D. Zhang, R. Poppe, R. C. Veltkamp, B. Li, and J. Yuan, "A survey of variational and cnn-based optical flow techniques," *Signal Processing: Image Communication*, vol. 72, pp. 9 – 24, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0923596518302479

[4] E. Meinhardt-Llopis, J. S. Pérez, and D. Kondermann, "Horn-schunck optical flow with a multi-scale strategy," *Image Processing on line*, vol. 2013, pp. 151–172, 2013.

[5] K. Soomro and A. R. Zamir, *Action Recognition in Realistic Sports Videos*. Springer International Publishing, 2014.

[6] A. Wali and A. M. Alimi, "Event detection from video surveillance data based on optical flow histogram and high-level feature extraction," in *2009 20th International Workshop on Database and Expert Systems Application*, Aug 2009, pp. 221–225.

[7] N. Onkarappa and A. D. Sappa, "An empirical study on optical flow accuracy depending on vehicle speed," in *Intelligent Vehicles Symposium (IV), 2012 IEEE*. IEEE, 2012, pp. 1138–1143.

[8] Z. E. Jaouhari, Y. Zaz, and L. Masmoudi, "Cloud tracking from whole-sky ground-based images," in *2015 3rd International Renewable and Sustainable Energy Conference (IRSEC)*, Dec 2015, pp. 1–5.

[9] A. A. Valsangkar, J. M. Monteiro, V. Narayanan, I. Hotz, and V. Natarajan, "An exploratory framework for cyclone identification and tracking," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 3, pp. 1460–1473, March 2019.

[10] A. Bruhn, J. Weickert, T. Kohlberger, and C. Schnörr, "A multigrid platform for real-time motion computation with discontinuity-preserving variational methods," *International Journal of Computer Vision*, vol. 70, pp. 257–277, 2006.

[11] D. Fortun, P. Bouthemy, and C. Kervrann, "Optical flow modeling and computation: A survey," *Computer Vision and Image Understanding*, vol. 134, pp. 1–21, 2015.

[12] J. R. Bergen, P. Anandan, K. J. Hanna, and R. Hingorani, "Hierarchical model-based motion estimation," in *European conference on computer vision*. Springer, 1992, pp. 237–252.

[13] J.-Y. Bouguet, "Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm," *Intel Corporation*, vol. 5, no. 1-10, p. 4, 2001.

[14] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert, "High accuracy optical flow estimation based on a theory for warping," *Computer Vision-ECCV 2004*, pp. 25–36, 2004.

[15] M. Smirnov, "Optical flow estimation with cuda," 2012.

[16] K. Pauwels and M. M. Van Hulle, "Realtime phase-based optical flow on the gpu," in *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, June 2008, pp. 1–8.

[17] C. Nvidia, "Compute unified device architecture programming guide," 2007.

[18] D. C. M. Bilsby, R. L. Walke, and R. W. M. Smith, "Comparison of a programmable dsp and a fpga for real-time multiscale convolution," in *IEE Colloquium on High Performance Architectures for Real-Time Image Processing*, Feb 1998, pp. 4/1–4/6.

[19] E. H. Adelson and J. A. Movshon, "Phenomenal coherence of moving visual patterns," *Nature*, vol. 300, pp. 523–525, 1982.

[20] J. L. Barron, D. J. Fleet, and S. S. Beauchemin, "Performance of optical flow techniques," *International journal of computer vision*, vol. 12, no. 1, pp. 43–77, 1994.

[21] N. Papenberg, A. Bruhn, T. Brox, S. Didas, and J. Weickert, "Highly accurate optic flow computation with theoretically justified warping," *International Journal of Computer Vision*, vol. 67, no. 2, pp. 141–158, Apr 2006.

[22] H. Zimmer, A. Bruhn, J. Weickert, L. Valgaerts, A. Salgado, B. Rosenhahn, and H.-P. Seidel, "Complementary optic flow," in *Energy Minimization Methods in Computer Vision and Pattern Recognition*, D. Cremers, Y. Boykov, A. Blake, and F. R. Schmidt, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 207–220.

[23] L. Xu, J. Jia, and Y. Matsushita, "Motion detail preserving optical flow estimation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 9, pp. 1744–1757, Sep. 2012.

[24] L. Alvarez, J. Sánchez, and J. Weickert, "A scale-space approach to nonlocal optical flow calculations," in *Scale-Space Theories in Computer Vision*, M. Nielsen, P. Johansen, O. F. Olsen, and J. Weickert, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 235–246.

[25] L. Alvarez, J. Weickert, and J. Sánchez, "Reliable estimation of dense optical flow fields with large displacements," *International Journal of Computer Vision*, vol. 39, no. 1, pp. 41–56, Aug 2000.

[26] C. Schnörr, "Segmentation of visual motion by minimizing convex non-quadratic functionals," in *ICPR*, 1994.

[27] J. Weickert and C. Schnörr, "Variational optic flow computation with a spatio-temporal smoothness constraint," *Journal of Mathematical Imaging and Vision*, vol. 14, no. 3, pp. 245–255, May 2001.

[28] M. J. Black and P. Anandan, "Robust dynamic motion estimation over time," in *Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, June 1991, pp. 296–302.

[29] S. Volz, A. Bruhn, L. Valgaerts, and H. Zimmer, "Modeling temporal coherence for optical flow," in *2011 International Conference on Computer Vision*, Nov 2011, pp. 1116–1123.

[30] R. Garg, A. Roussos, and L. Agapito, "Robust trajectory-space tv-l1 optical flow for non-rigid sequences," in *Energy Minimization Methods in Computer Vision and*

Pattern Recognition, Y. Boykov, F. Kahl, V. Lempitsky, and F. R. Schmidt, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 300–314.

[31] A. Bruhn, "Variational optic flow computation: Accurate modelling and efficient numerics. awarded the 2006 outstanding dissertation award by the german computer society(gi). awarded the 2006 dr. eduard martin prize by the saarland university," Ph.D. dissertation, Department of Mathematics and Computer Science, Saarland University, 2006.

[32] G. Aubert, R. Deriche, and P. Kornprobst, "Computing optical flow via variational techniques," *SIAM Journal on Applied Mathematics*, vol. 60, no. 1, pp. 156–182, 1999. [Online]. Available: https://doi.org/10.1137/S0036139998340170

[33] M. G. Mozerov, "Constrained optical flow estimation as a matching problem," *IEEE Transactions on Image Processing*, vol. 22, no. 5, pp. 2044–2055, May 2013.

[34] W. Li, D. Cosker, M. Brown, and R. Tang, "Optical flow estimation using laplacian mesh energy," in *2013 IEEE Conference on Computer Vision and Pattern Recognition*, June 2013, pp. 2435–2442.

[35] S. Roth and M. J. Black, "On the spatial statistics of optical flow," *International Journal of Computer Vision*, vol. 74, no. 1, pp. 33–50, Aug 2007. [Online]. Available: https://doi.org/10.1007/s11263-006-0016-x

[36] S. Roth, V. Lempitsky, and C. Rother, "Discrete-continuous optimization for optical flow estimation," in *Statistical and Geometrical Approaches to Visual Motion Analysis*, D. Cremers, B. Rosenhahn, A. L. Yuille, and F. R. Schmidt, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–22.

[37] B. D. Lucas, T. Kanade *et al.*, "An iterative image registration technique with an application to stereo vision," 1981.

[38] S. Baker and I. Matthews, "Lucas-kanade 20 years on: A unifying framework," *International journal of computer vision*, vol. 56, no. 3, pp. 221–255, 2004.

[39] J. Marzat, Y. Dumortier, and A. Ducrot, "Real-time Dense and Accurate Parallel Optical Flow using CUDA," in *International Workshop on Computer Vision and Its Application to Image Media Processing*, Tokyo, Japan, Jan. 2009. [Online]. Available: https://hal.inria.fr/inria-00346710

[40] A. Plyer, G. Le Besnerais, and F. Champagnat, "Massively parallel lucas kanade optical flow for real-time video processing applications," *Journal of Real-Time Image Processing*, vol. 11, no. 4, pp. 713–730, Apr 2016. [Online]. Available: https://doi.org/10.1007/s11554-014-0423-0

[41] V. Mahalingam, K. Bhattacharya, N. Ranganathan, H. Chakravarthula, R. R. Murphy, and K. S. Pratt, "A vlsi architecture and algorithm for lucas-kanade-based optical flow computation." *IEEE Trans. VLSI Syst.*, vol. 18, no. 1, pp. 29–38, 2010.

[42] J. Diaz, E. Ros, F. Pelayo, E. M. Ortigosa, and S. Mota, "Fpga-based real-time optical-flow system," *IEEE transactions on circuits and systems for video technology*, vol. 16, no. 2, pp. 274–279, 2006.

[43] P. Anandan, "A computational framework and an algorithm for the measurement of visual motion," *International Journal of Computer Vision*, vol. 2, no. 3, pp. 283–310, Jan 1989. [Online]. Available: https://doi.org/10.1007/BF00158167

[44] J. Wills, S. Agarwal, and S. Belongie, "A feature-based approach for dense segmentation and estimation of large disparity motion," *International Journal of Computer Vision*, vol. 68, no. 2, pp. 125–143, Jun 2006. [Online]. Available: https://doi.org/10.1007/s11263-006-6660-3

[45] A. B. Watson and A. J. Ahumada, "A look at motion in the frequency domain," 1983.

[46] E. H. Adelson and J. R. Bergen, "Spatiotemporal energy models for the perception of motion," *J. Opt. Soc. Am. A*, vol. 2, no. 2, pp. 284–299, Feb 1985. [Online]. Available: http://josaa.osa.org/abstract.cfm?URI=josaa-2-2-284

[47] D. J. Heeger, "Optical flow using spatiotemporal filters," *International Journal of Computer Vision*, vol. 1, no. 4, pp. 279–302, Jan 1988.

[48] D. J. Fleet and A. D. Jepson, "Computation of component image velocity from local phase information," *International Journal of Computer Vision*, vol. 5, no. 1, pp. 77–104, Aug 1990. [Online]. Available: https://doi.org/10.1007/BF00056772

[49] W. T. Freeman and E. H. Adelson, "The design and use of steerable filters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 9, pp. 891–906, Sep. 1991.

[50] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "Flownet 2.0: Evolution of optical flow estimation with deep networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1647–1655, 2017.

[51] Z. Ren, J. Yan, B. Ni, B. Liu, X. Yang, and H. Zha, "Unsupervised deep learning for optical flow estimation," in *AAAI*, 2017.

[52] S. Zweig and L. Wolf, "Interponet, a brain inspired neural network for optical flow dense interpolation," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6363–6372, 2017.

[53] F. Güney and A. Geiger, "Deep discrete flow," in *ACCV*, 2016.

[54] D. Sun, S. Roth, and M. J. Black, "A quantitative analysis of current practices in optical flow estimation and the principles behind them," *International Journal of Computer Vision*, vol. 106, no. 2, pp. 115–137, 2014.

[55] A. Bruhn, J. Weickert, C. Feddern, T. Kohlberger, and C. Schnörr, "Variational optical flow computation in real time," *IEEE Transactions on Image Processing*, vol. 14, pp. 608–615, 2005.

[56] M. Drulea and S. Nedevschi, "Total variation regularization of local-global optical flow," in *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, Oct 2011, pp. 318–323.

[57] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A Multigrid Tutorial: Second Edition*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2000.

[58] A. Bruhn, J. Weickert, C. Feddern, T. Kohlberger, and C. Schnörr, "Real-time optic flow computation with variational methods," in *Computer Analysis of Images and Patterns*, N. Petkov and M. A. Westenberg, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 222–229.

[59] T. Brox and J. Malik, "Large displacement optical flow: Descriptor matching in variational motion estimation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 3, pp. 500–513, Mar. 2011. [Online]. Available: http://dx.doi.org/10.1109/TPAMI.2010.143

[60] P. Burt and E. Adelson, "The laplacian pyramid as a compact image code," *IEEE Transactions on Communications*, vol. 31, no. 4, pp. 532–540, April 1983.

[61] Z. Wei, "Real-time optical flow sensor design and its application on obstacle detection," 2009.

[62] W. J. MacLean, "An evaluation of the suitability of fpgas for embedded vision systems," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Workshops*, Sep. 2005, pp. 131–131.

[63] M. Butts, "Synchronization through communication in a massively parallel processor array," *IEEE Micro*, vol. 27, no. 5, pp. 32–40, Sep. 2007.

[64] B. Hutchings, B. Nelson, S. West, and R. Curtis, "Optical flow on the ambric massively parallel processor array (mppa)," in *2009 17th IEEE Symposium on Field Programmable Custom Computing Machines*, April 2009, pp. 141–148.

[65] Y. Tsai, M. Yang, and M. J. Black, "Video segmentation via object flow," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 3899–3908.

[66] F. Xiao and Y. J. Lee, "Track and segment: An iterative unsupervised approach for video object proposals," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 933–942.

[67] I. Kajo, A. S. Malik, and N. Kamel, "An evaluation of optical flow algorithms for crowd analytics in surveillance system," in *2016 6th International Conference on Intelligent and Advanced Systems (ICIAS)*, Aug 2016, pp. 1–6.

[68] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," *ACM Comput. Surv.*, vol. 38, p. 13, 2006.

[69] J. Santos-Victor, G. Sandini, F. Curotto, and S. Garibaldi, "Divergent stereo for robot navigation: learning from bees," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, June 1993, pp. 434–439.

[70] F. Bonin-Font, A. Ortiz, and G. Oliver, "Visual navigation for mobile robots: A survey," *Journal of Intelligent and Robotic Systems*, vol. 53, pp. 263–296, 2008.

[71] H. W. Ho, C. De Wagter, B. D. W. Remes, and G. C. H. E. de Croon, "Optical flow for self-supervised learning of obstacle appearance," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2015, pp. 3098–3104.

[72] G. N. Desouza and A. C. Kak, "Vision for mobile robot navigation: a survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 2, pp. 237–267, Feb 2002.

[73] J. W. Wilson, N. A. Crook, C. K. Mueller, J. Sun, and M. Dixon, "Nowcasting thunderstorms: A status report," *Bulletin of the American Meteorological Society*, vol. 79, no. 10, pp. 2079–2100, 1998.

[74] R. Kovordányi and C. Roy, "Cyclone track forecasting based on satellite images using artificial neural networks," 2008.

[75] C. Marzban and S. Sandgathe, "Optical flow for verification," *Weather and Forecasting*, vol. 25, no. 5, pp. 1479–1494, 2010.

[76] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. J. Black, and R. Szeliski, "A database and evaluation methodology for optical flow," *International Journal of Computer Vision*, vol. 92, no. 1, pp. 1–31, 2011.

[77] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black, "A naturalistic open source movie for optical flow evaluation," in *ECCV*, 2012.

[78] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *I. J. Robotics Res.*, vol. 32, pp. 1231–1237, 2013.

[79] "Meteorological & oceanographic satellite data archival centre | space applications centre, isro," https://www.mosdac.gov.in/, (Accessed on 05/25/2019).

[80] "Data registration âĂŤ eumetsat," https://www.eumetsat.int/website/home/Data/DataDelivery/DataRegistration/index.html, (Accessed on 05/24/2019).

[81] "Cpc: Monitoring and data - full-resolution ir data," https://www.cpc.ncep.noaa.gov/products/global_precip/html/wpage.full_res.html, (Accessed on 05/24/2019).

[82] "Ecmwf | public datasets," https://apps.ecmwf.int/datasets/, (Accessed on 05/25/2019).

[83] "Numerical weather prediction," http://nwp.imd.gov.in/index.php, (Accessed on 05/25/2019).

[84] C. M. Kishtawal, S. K. Deb, P. K. Pal, and P. C. Joshi, "Estimation of atmospheric motion vectors from kalpana-1 imagers," *Journal of Applied Meteorology and Climatology*, vol. 48, no. 11, pp. 2410–2421, 2009.

[85] J. Díaz, E. Ros, R. Agís, and J. L. Bernier, "Superpipelined high-performance optical-flow computation architecture," *Computer Vision and Image Understanding*, vol. 112, no. 3, pp. 262–273, 2008.

[86] G. Botella, A. García, M. Rodríguez-Álvarez, E. Ros, U. Meyer-Baese, and M. C. Molina, "Robust bioinspired architecture for optical-flow computation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 4, pp. 616–629, 2010.

[87] M. Tomasi, M. Vanegas, F. Barranco, J. Diaz, and E. Ros, "High-performance optical-flow architecture based on a multi-scale, multi-orientation phase-based model," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, no. 12, pp. 1797–1807, 2010.

[88] F. Barranco, M. Tomasi, J. Diaz, M. Vanegas, and E. Ros, "Parallel architecture for hierarchical optical flow estimation based on fpga," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 6, pp. 1058–1067, 2012.

[89] Q. Zhu, N. Garg, Y.-T. Tsai, and K. Pulli, "An energy efficient time-sharing pyramid pipeline for multi-resolution computer vision," in *2013 IFIP/IEEE 21st International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE, 2013, pp. 278–281.

[90] P. J. Burt and G. van der Wal, "An architecture for multiresolution, focal, image analysis," in *[1990] Proceedings. 10th International Conference on Pattern Recognition*, vol. ii, June 1990, pp. 305–311 vol.2.

[91] G. S. Van der Wal and P. J. Burt, "A vlsi pyramid chip for multiresolution image analysis," *International Journal of Computer Vision*, vol. 8, no. 3, pp. 177–189, Sep 1992. [Online]. Available: https://doi.org/10.1007/BF00055150

[92] M. Komorkiewicz, T. Kryjak, and M. Gorgon, "Efficient hardware implementation of the horn-schunck algorithm for high-resolution real-time dense optical flow sensor," *Sensors*, vol. 14, no. 2, pp. 2860–2891, 2014.

[93] K. Seyid, A. Richaud, R. Capoccia, and Y. Leblebici, "Fpga-based hardware implementation of real-time optical flow calculation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 1, pp. 206–216, Jan 2018.

[94] "4k60 dense optical flow using the revision stack," https://www.xilinx.com/video/application/4k60-dense-optical-flow.html, (Accessed on 02/16/2019).

[95] D. Sun, S. Roth, and M. Black, "Secrets of optical flow estimation and their principles," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010, pp. 2432–2439.

[96] G. E. Karniadakis and R. M. Kirby II, *Parallel scientific computing in C++ and MPI: a seamless approach to parallel algorithms and their implementation*. Cambridge University Press, 2003.

[97] B. J. and, "A high throughput fully parallel-pipelined fpga accelerator for dense cloud motion analysis," in *2016 IEEE Region 10 Conference (TENCON)*, Nov 2016, pp. 2589–2592.

[98] R. Szeliski, "Computer vision - algorithms and applications," in *Texts in Computer Science*, 2010.

[99] M. Otte and H. H. Nagel, "Optical flow estimation: Advances and comparisons," in *Computer Vision — ECCV '94*, J.-O. Eklundh, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 49–60.

[100] S. Baker, S. Roth, D. Scharstein, M. J. Black, J. Lewis, and R. Szeliski, "A database and evaluation methodology for optical flow," in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*. IEEE, 2007, pp. 1–8.

[101] K. Pauwels and M. M. Van Hulle, "Realtime phase-based optical flow on the gpu," in *2008. CVPRW'08. IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. IEEE, 2008, pp. 1–8.

[102] M. Isa, K. Benkrid, and T. Clayton, "Efficient architecture and scheduling technique for pairwise sequence alignment," *ACM SIGARCH Computer Architecture News*, vol. 40, no. 4, pp. 26–31, 2012.

[103] H. A. van der Vorst, "An iterative solution method for linear systems of which the coefficient matrix is a symmetric m-matrix," 2010.

[104] D. J. Evans, "Parallel sor iterative methods," *Parallel computing*, vol. 1, no. 1, pp. 3–18, 1984.

[105] J. L. Martín, A. Zuloaga, C. Cuadrado, J. Lázaro, and U. Bidarte, "Hardware implementation of optical flow constraint equation using fpgas," *Computer Vision and Image Understanding*, vol. 98, no. 3, pp. 462–490, 2005.

[106] M. R. B. Bahar and G. Karimian, "High performance implementation of the horn and schunck optical flow algorithm on fpga," *20th Iranian Conference on Electrical Engineering (ICEE2012)*, pp. 736–741, 2012.

[107] W. Chen, Z. Wang, Q. Wu, J. Liang, and Z. Chai, "Implementing dense optical flow computation on a heterogeneous fpga soc in c," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 13, no. 3, p. 25, 2016.

[108] M. J. Forsell, "Architectural differences of efficient sequential and parallel computers," *Journal of Systems Architecture*, vol. 47, no. 13, pp. 1017–1041, 2002.

[109] B. Johnson *et al.*, "A high throughput fully parallel-pipelined fpga accelerator for dense cloud motion analysis," in *Region 10 Conference (TENCON), 2016 IEEE*. IEEE, 2016, pp. 2589–2592.

[110] L. Hoeltgen, S. Setzer, and M. Breuß, "Intermediate flow field filtering in energy based optic flow computations," in *Energy Minimization Methods in Computer Vision and Pattern Recognition*, Y. Boykov, F. Kahl, V. Lempitsky, and F. R. Schmidt, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 315–328.

[111] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, Jan 1998, pp. 839–846.

[112] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Computer Vision, 1998. Sixth International Conference on*. IEEE, 1998, pp. 839–846.

[113] F. Durand and J. Dorsey, "Fast bilateral filtering for the display of high-dynamic-range images," in *ACM transactions on graphics (TOG)*, vol. 21, no. 3. ACM, 2002, pp. 257–266.

[114] J. Xiao, H. Cheng, H. Sawhney, C. Rao, and M. Isnardi, "Bilateral filtering-based optical flow estimation with occlusion detection," in *Computer Vision – ECCV 2006*,

A. Leonardis, H. Bischof, and A. Pinz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 211–224.

[115] A. Wedel, T. Pock, C. Zach, H. Bischof, and D. Cremers, "An improved algorithm for tv-l 1 optical flow," in *Statistical and Geometrical Approaches to Visual Motion Analysis*. Springer, 2009, pp. 23–45.

[116] S. Paris, P. Kornprobst, J. Tumblin, F. Durand *et al.*, "Bilateral filtering: Theory and applications," *Foundations and Trends® in Computer Graphics and Vision*, vol. 4, no. 1, pp. 1–73, 2009.

[117] T. Q. Pham and L. J. Van Vliet, "Separable bilateral filtering for fast video preprocessing," in *Multimedia and Expo, 2005. ICME 2005. IEEE International Conference on*. IEEE, 2005, pp. 4–pp.

[118] J. Chen, S. Paris, and F. Durand, "Real-time edge-aware image processing with the bilateral grid," in *ACM Transactions on Graphics (TOG)*, vol. 26, no. 3. ACM, 2007, p. 103.

[119] F. Porikli, "Constant time o (1) bilateral filtering," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE, 2008, pp. 1–8.

[120] K. N. Chaudhury, D. Sage, and M. Unser, "Fast bilateral filtering using trigonometric range kernels," *IEEE Transactions on Image Processing*, vol. 20, no. 12, pp. 3376–3382, 2011.

[121] Q. Yang, "Recursive approximation of the bilateral filter." *IEEE transactions on image processing*, vol. 24, no. 6, pp. 1919–1927, 2015.

[122] C. Charoensak and F. Sattar, "Fpga design of a real-time implementation of dynamic range compression for improving television picture," in *Information, Communications & Signal Processing, 2007 6th International Conference on*. IEEE, 2007, pp. 1–5.

[123] S.-K. Han, M.-H. Jeong, S. Woo, and B.-J. You, "Architecture and implementation of real-time stereo vision with bilateral background subtraction," in *Advanced Intelligent Computing Theories and Applications. With Aspects of Theoretical and Methodological Issues*. Springer, 2007, pp. 906–912.

[124] T. Q. Vinh, J. H. Park, Y.-C. Kim, and S. H. Hong, "Fpga implementation of real-time edge-preserving filter for video noise reduction," in *Computer and Electrical Engineering, 2008. ICCEE 2008. International Conference on*. IEEE, 2008, pp. 611–614.

[125] A. Gabiger-Rose, M. Kube, P. Schmitt, R. Weigel, and R. Rose, "Image denoising using bilateral filter with noise-adaptive parameter tuning," in *IECON 2011-37th Annual Conference on IEEE Industrial Electronics Society*. IEEE, 2011, pp. 4515–4520.

[126] B. Zhang and J. P. Allebach, "Adaptive bilateral filter for sharpness enhancement and noise removal," *Image Processing, IEEE Transactions on*, vol. 17, no. 5, pp. 664–678, 2008.

[127] A. Gabiger, M. Kube, and R. Weigel, "A synchronous fpga design of a bilateral filter for image processing," in *Industrial Electronics, 2009. IECON'09. 35th Annual Conference of IEEE*. IEEE, 2009, pp. 1990–1995.

[128] F. Hannig, M. Schmid, J. Teich, and H. Hornegger, "A deeply pipelined and parallel architecture for denoising medical images," in *Field-Programmable Technology (FPT), 2010 International Conference on*. IEEE, 2010, pp. 485–490.

[129] Y.-C. Tseng, P.-H. Hsu, and T.-S. Chang, "A 124 mpixels/s vlsi design for histogram-based joint bilateral filtering," *Image Processing, IEEE Transactions on*, vol. 20, no. 11, pp. 3231–3241, 2011.

[130] C. Y. Villalpando, A. Morfopolous, L. Matthies, and S. Goldberg, "Fpga implementation of stereo disparity with high throughput for mobility applications," in *Aerospace Conference, 2011 IEEE*. IEEE, 2011, pp. 1–10.

[131] C. Pal, K. N. Chaudhury, A. Samanta, A. Chakrabarti, and R. Ghosh, "Hardware software co-design of a fast bilateral filter in fpga," in *2013 Annual IEEE India Conference (INDICON)*. IEEE, 2013, pp. 1–6.

[132] J. S. S. Kutty, F. Boussaid, and A. Amira, "A high speed configurable fpga architecture for bilateral filtering," in *Image Processing (ICIP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 1248–1252.

[133] D. G. Bailey and M. J. Klaiber, "Efficient hardware calculation of running statistics," in *Image and Vision Computing New Zealand (IVCNZ), 2013 28th International Conference of*. IEEE, 2013, pp. 196–201.

[134] A. Gabiger-Rose, M. Kube, R. Weigel, and R. Rose, "An fpga-based fully synchronized design of a bilateral filter for real-time image denoising," *Industrial Electronics, IEEE Transactions on*, vol. 61, no. 8, pp. 4093–4104, 2014.

[135] H. Dutta, F. Hannig, J. Teich, B. Heigl, and H. Hornegger, "A design methodology for hardware acceleration of adaptive filter algorithms in image processing," in *Application-specific Systems, Architectures and Processors, 2006. ASAP'06. International Conference on*. IEEE, 2006, pp. 331–340.

[136] Y. Arnaud, M. Desbois, and J. Maizi, "Automatic tracking and characterization of african convective systems on meteosat pictures," *Journal of Applied Meteorology*, vol. 31, no. 5, pp. 443–453, 1992.

[137] W. L. Woodley, C. G. Griffith, J. S. Griffin, and S. C. Stromatt, "The inference of gate convective rainfall from sms-1 imagery," *Journal of Applied Meteorology*, vol. 19, no. 4, pp. 388–408, 1980.

[138] J. Schmetz, K. Holmlund, J. Hoffman, B. Strauss, B. Mason, V. Gaertner, A. Koch, and L. Van De Berg, "Operational cloud-motion winds from meteosat infrared images," *Journal of Applied Meteorology*, vol. 32, no. 7, pp. 1206–1225, 1993.

[139] J. A. Leese, C. S. Novak, and B. B. Clark, "An automated technique for obtaining cloud motion from geosynchronous satellite data using cross correlation," *Journal of applied meteorology*, vol. 10, no. 1, pp. 118–132, 1971.

[140] E. Smith, D. R. Phillips *et al.*, "Automated cloud tracking using precisely aligned digital ats pictures," *Computers, IEEE Transactions on*, vol. 100, no. 7, pp. 715–729, 1972.

[141] L. M. Carvalho and C. Jones, "A satellite method to identify structural properties of mesoscale convective systems based on the maximum spatial correlation tracking technique (mascotte)," *Journal of Applied Meteorology*, vol. 40, no. 10, pp. 1683–1701, 2001.

[142] R. A. Scofield, R. Kuligowski, and C. Davenport, "The use of the hydro-nowcaster for mesoscale convective systems and the tropical rainfall nowcaster (tran) for land-falling tropical systems. preprints," in *Symp. on Planning, Nowcasting, and Forecasting in the Urban Zone*, 2004.

[143] R. Brad and L. A. LETIA, "Extracting cloud motion from satellite image sequences," in *Control, Automation, Robotics and Vision, 2002. ICARCV 2002. 7th International Conference on*, vol. 3. IEEE, 2002, pp. 1303–1307.

[144] D. Laney, P.-T. Bremer, A. Mascarenhas, P. Miller, and V. Pascucci, "Understanding the structure of the turbulent mixing layer in hydrodynamic instabilities," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 12, no. 5, pp. 1053–1060, 2006.

[145] P.-T. Bremer, G. Weber, J. Tierny, V. Pascucci, M. Day, and J. Bell, "Interactive exploration and analysis of large-scale simulations using topology-based data segmentation," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 17, no. 9, pp. 1307–1324, 2011.

[146] and C. Kambhamettu, D. B. Goldgof, K. Palaniappan, and A. F. Hasler, "Tracking nonrigid motion and structure from 2d satellite cloud images without correspondences," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 11, pp. 1330–1336, Nov 2001.

[147] and, , and and, "Automatic tracking and characterization of multiple moving clouds in satellite images," in *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583)*, vol. 4, Oct 2004, pp. 3088–3093 vol.4.

[148] D. A. Vila, L. A. T. Machado, H. Laurent, and I. Velasco, "Forecast and tracking the evolution of cloud clusters (fortracc) using satellite infrared imagery: Methodology and validation," *Weather and Forecasting*, vol. 23, no. 2, pp. 233–245, 2008.

[149] C. Costes, J. . Artis, and F. Barbaresco, "Advanced clouds tracking for airborne weather radar amp;amp; ground primary surveillance radar," in *The 7th European Radar Conference*, Sep. 2010, pp. 141–144.

[150] and and, "Application of genetic algorithm in tracking convective cloud images from chinese fy-2c satellite," in *The 2nd International Conference on Information Science and Engineering*, Dec 2010, pp. 1–4.

[151] H. Doraiswamy, V. Natarajan, and R. S. Nanjundiah, "An exploration framework to identify and track movement of cloud systems," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 12, pp. 2896–2905, Dec 2013.

[152] Q. Wu, H.-Q. Wang, Y.-J. Lin, Y.-Z. Zhuang, and Y. Zhang, "Deriving amvs from geostationary satellite images using optical flow algorithm based on polynomial expansion," *Journal of Atmospheric and Oceanic Technology*, vol. 33, no. 8, pp. 1727–1747, 2016.

[153] J. P. F. Serra, "Image analysis and mathematical morphology," 1983.

[154] J. Chalfoun, A. Cardone, A. A. Dima, D. P. Allen, and M. W. Halter, "Overlap-based cell tracker," in *Journal of research of the National Institute of Standards and Technology*, 2009.

[155] G. R. Prabhu, B. Johnson, and J. S. Rani, "Scalable fixed point qrd core using dynamic partial reconfiguration," *International Journal of Reconfigurable Computing*, vol. 2014, p. 15, 2014.

[156] J. Carlier, "Second set of fluid mechanics image sequences," *European ProjectâĂŹFluid image analysis and descriptionâĂŹ(FLUID) http://www. fluid. irisa. fr*, 2005.

[157] M. A. Saunders and A. S. R. Lea, "Large contribution of sea surface warming to recent increase in atlantic hurricane activity," *Nature*, vol. 451, pp. 557–560, 2008.

[158] J. Stam, "Stable fluids," in *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '99. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1999, pp. 121–128. [Online]. Available: http://dx.doi.org/10.1145/311535.311548

[159] B. Johnson, J. S. Rani, and G. R. Manyam, "A novel visualization and tracking framework for analyzing the inter/intra cloud pattern formation to study their impact on climate," in *Proceedings of International Conference on Computer Vision and Image Processing*. Springer, 2017, pp. 499–509.

[160] B. Johnson, N. Thomas, and J. S. Rani, "An fpga based high throughput discrete kalman filter architecture for real-time image denoising," *2017 30th International Conference on VLSI Design and 2017 16th International Conference on Embedded Systems (VLSID)*, pp. 55–60, 2017.

[161] J. Liu, C. Liu, B. Wang, and D. Qin, "A novel algorithm for detecting center of tropical cyclone in satellite infrared images," in *2015 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, July 2015, pp. 917–920.

[162] N. Jaiswal and C. M. Kishtawal, "Objective detection of center of tropical cyclone in remotely sensed infrared images," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 6, no. 2, pp. 1031–1035, April 2013.

List of Publications

Papers in Refereed Journals

1. Johnson, B., Moncy, J.K. and Rani, Self adaptable high throughput reconfigurable bilateral filter architectures for real-time image de-noising. Journal of Real-Time Image Processing, pp.1-20

2. Johnson, B., Thomas, S. and Rani, J.S., 2018. A novel framework for objective detection and tracking of TC center from noisy satellite imagery. Advances in Space Research, 62(1), pp.44-54

3. B. Johnson, S. Thomas, and R. J. Sheeba, A high-performance dense optical flow architecture based on red-black sor solver", Journal of Signal Processing Systems, 2019, issn: 1939-8115. doi: 10.1007/s11265-019-01490-5.

Papers to be Communicated and Under Review

1. Bibin Johnson, Sheeba Rani J, *"A Novel High Performance Pyramidal Architecture for Real Time Flow Computation"*, Manuscript under major revision in IEEE Transactions on Circuits and Systems for Video Technology (TCSVT).

2. Bibin Johnson, Sheeba Rani J, Gorthi R.K.S.S, *"A Novel High Performance Framework for for Analysing the Variation of Cloud Patterns"*, Manuscript under preparation.

Papers in Refereed Conferences

1. Bibin Johnson, J. Sheeba Rani, Nimin Thomas, *"An FPGA Based High Throughput Discrete Kalman Filter Architecture For Real-time Image Denoising"*, Proceedings

of the IEEE VLSI Design Conference 2017, 55-60

2. Bibin Johnson, J.Sheeba Rani, *"A high Throughput fully parallel-pipelined FPGA accelerator for dense cloud motion analysis"*, Proceedings of IEEE Region 10 TENCON 2016 conference, Singapore, Dec 2016

3. Bibin Johnson, J. Sheeba Rani, G.R.K.Sai Subrahmanyam, *"A Novel Visualization and Tracking Framework for analyzing Inter/Intra cloud pattern formation to study their impact on climate"*, International conference on Computer Vision and Image Processing, CVIP 2016, IIT Roorkee, Feb 2016.

Others

1. Gayathri P, Bibin Johnson, Sheeba Rani J , *âĂİScalable Fixed Point QRD core using Dynamic Partial Reconfiguration,âĂİ* Int. J. Recon- fig. Comp. 2014: 243835:1-243835:9 (2014).

2. Gayathri R. Prabhu, Bibin Johnson, J. Sheeba Rani, *"FPGA Based Scalable Fixed Point QRD Core Using Dynamic Partial Reconfiguration"*, Proceedings of the IEEE VLSI Design Conference 2015, 345-350

Appendix A

Appendix A

A.1 Histogram equalization

For an image $I(m, n)$ having intensity ranging from 0 to $(L - 1)$. The input histogram is approximated by probability distribution function (PDF) $p_r(r)$ with pixel value r, $p_z(r)$ is the PDF of the specified histogram with z being the resultant pixel value given by,

$$T(r_k) = (L-1)\sum_{i=0}^{k} p_r(r_i) \tag{A.1}$$

$$G(z_q) = (L-1)\sum_{i=0}^{q} p_z(r_i) \tag{A.2}$$

$$z_q = G^{-1}(T(r_k)) \tag{A.3}$$

Since the histogram of the image is concentrated on a particular region, in order to distribute histogram across entire intensity and thereby enhancing the hidden details. For a $N \times M$ image of G grey levels, create an array H of length G initialized with 0 values. Form the image histogram, scan every pixel and increment the relevant number of H if pixel p has intensity gp, i.e perform

$$H[gp] = H[gp] + 1 \tag{A.4}$$

Form the cumulative histogram H_c :

$$H_c[0] = H[0] \tag{A.5}$$

$$H_c[p] = H_c[p-1] + H[p] \tag{A.6}$$

Substituting the values:

$$T[p] = round\left(\frac{G-1}{N \times M} \times H_c[p]\right) \tag{A.7}$$

Rescan the image and write an output image with grey-levels gq:

$$gq = T[gp] \tag{A.8}$$

A.2 Morphological Operation

Dilation adds pixels to the boundaries of objects while erosion removes pixels on object boundaries. It depends on the size and shape of the SE. Dilation operation is represented as the maximum value of all the pixels in the input whereas erosion the output is minimum value of all the pixels. The dilation and erosion operators can be implemented in the hardware using basic logic gates. For a structuring element of size 2×2, the dilation and erosion are implemented. Mathematically, the opening of set A by structuring element B, denoted by $A \circ B$ is defined as where $\ominus$ is the erosion operator and $\oplus$ is the dilation operator. Thus, the opening A by B is the erosion of A by B, followed by dilation of the result by B.Similarly, the closing of set A by structuring element B, denoted by $A \bullet B$.So closing of A by B is simply the dilation of A by B followed by the erosion of the result by B.

$$A \circ B = (A \ominus B) \oplus B \tag{A.9}$$

$$A \bullet B = (A \oplus B) \ominus B \tag{A.10}$$

The irregular and thin shapes in the segmented region is enhanced by morphological operations employing a structuring element $S1$. The images are opened by a disk-shaped STREL $(S1)$ in the equ. A.11 to remove small objects and break weak connections between the structures. It is followed by closing operation with a disc $S2$ in the equ. A.12 to fill the holes, while preserving small cloud segments other than the background.

$$I \circ S1 = (I \ominus S1) \oplus S1 \tag{A.11}$$

$$I \bullet S2 = (I \oplus S2) \ominus S2 \tag{A.12}$$

where $\oplus$ and $\ominus$ represents the dilation and erosion operation respectively and given by,

$$(I \oplus S1)(m,n) = max\{I(m+x, n+y) + S1(x,y)\}, \quad \forall (x,y) \in S \tag{A.13}$$

$$(I \ominus S1)(m,n) = min\{I(m+x, n+y) + S1(x,y)\}, \quad \forall (x,y) \in S \tag{A.14}$$