Design space exploration of image processing algorithms on FPGAs

Ilias Bournias

THÈSE DE DOCTORAT
DE SORBONNE UNIVERSITÉ

DESIGN SPACE EXPLORATION OF IMAGE PROCESSING ALGORITHMS ON FPGAS

présentée par

ILIAS BOURNIAS

École Doctorale Informatique, Télécommunications et Électronique

réalisée au
LIP6

soutenue le

devant le jury composé de :

M.	Florent De Dinechin, Professeur, INSA, Lyon, France	Rapporteur
M.	Steven Derrien, Professeur, Université Rennes 1, Rennes, France	Rapporteur
Mme.	Fabienne Jézéquiel, Maître de conférences, HDR, LIP6, Paris, France	Examinateur
M.	François Berry, Professeur, Université Cl. Auvergne, Clermont-Ferrand, France	Examinateur
M.	Nicolas Rambaux, Maître de Conférences, Sorbonne Université, Paris, France	Invité
M.	Lionel Lacassagne, Professeur, LIP6, Paris, France	Co-directeur de Thèse
Mme.	Roselyne Chotin, Maître de conférences, HDR, LIP6, Paris, France	Directrice de Thèse

To my Family that always supported me

ABSTRACT

Implementing image processing algorithms for embedded systems is a scientific topic of great importance and many researchers focus their work on this domain. Many trade-offs have to be made in order to fit these algorithms in a specific embedded system and at the same time achieve real time computation and acceptable precision.

In this thesis, we focus on the design space exploration of an optical flow algorithm called Multi-scale Horn and Schunck algorithm in an Arria 10 FPGA. Although we focus on a specific algorithm and a specific device, the exploration we perform and the propositions of this thesis can also be applied to other algorithms and FPGA devices too.

The first thing we explore is the accuracy of the algorithm. We use different floating point formats and we tune different parameters of the algorithm in order to increase the accuracy and at the same time provide an implementation which achieves real time computation. As this algorithm is a multi-rate image processing algorithm, we propose solutions in order to tackle this nature of the algorithm and increase computation throughput. We use pipeline and vectorized architectures in order to further increase the computation speed and we introduce trans-floating computation which enables us to fit more processing elements in our FPGA. We explore how all these solutions affect the resources usage of the FPGA, such as the LUTs, DSPs and Block RAMs utilization. Furthermore, we propose approaches in order to overcome the bottleneck of the external memory bandwidth. Following that, and by taking into account all our propositions, we perform a design space exploration of the algorithm, which helps the optical flow designer to choose among different configurations according to his constraints.

We compare our designs with other state of the art works on optical flow in FPGAs and we show that to the best of our knowledge our fastest design achieves the highest throughput compared to all the rest optical flow designs on single FPGA. At the same time our implementations achieve comparable accuracy of detection.

RÉSUMÉ

La mise en œuvre d'algorithmes de traitement d'image pour les systèmes embarqués est un sujet scientifique de grande importance et de nombreux chercheurs concentrent leurs travaux sur ce domaine. De nombreux compromis doivent être réalisés afin d'adapter ces algorithmes au système ciblé et obtenir en même temps un calcul en temps réel et une précision acceptable.

Dans cette thèse, nous nous concentrons sur l'exploration de l'espace de conception d'un algorithme de flot optique appelé algorithme de Horn et Schunck multi-échelles dans un FPGA Arria 10. Bien que nous nous concentrions sur un algorithme et une cible spécifiques, l'exploration que nous effectuons et les propositions de cette thèse peuvent élargies à d'autres algorithmes et FPGAs.

La première chose que nous explorons est la précision des calculs. Nous utilisons différents formats de virgule flottante et nous ajustons différents paramètres de l'algorithme afin d'augmenter la précision et en même temps fournir une implémentation qui réalise le calcul en temps réel. Comme cet algorithme est un algorithme de traitement d'images multi-débits, nous proposons des solutions pour augmenter le débit de calcul. Nous utilisons des architectures pipeline et vectorisées afin d'augmenter encore la vitesse de calcul et nous introduisons le calcul trans-flottant qui nous permet d'intégrer plus d'éléments de traitement sur le FPGA. Nous explorons comment toutes ces solutions affectent l'utilisation des ressources du FPGA, telles que l'utilisation des LUTs, des DSPs et des blocs de RAM. De plus, nous proposons des approches afin de surmonter le goulot d'étranglement lié à la bande passante de la mémoire externe. Enfin, en tenant compte de toutes nos propositions, nous effectuons une exploration de l'espace de conception, qui aide le concepteur à implanter l'algorithme de flot optique sur FPGA en choisissant parmi différentes configurations en fonction de ses contraintes.

Nous comparons nos résultats d'implémentations sur FPGA de l'algorithme de flot optique avec d'autres travaux à la pointe du domaine. Ainsi, nous montrons que notre implémentation la plus rapide atteint le débit le plus élevé par rapport à tous les autres travaux portés à notre connaissance et implantés sur un unique FPGA. En même temps, nos implémentations permettent d'obtenir une précision de détection comparable à ces travaux et une utilisation moindre des ressources du FPGA.

PUBLICATIONS

The following body of work has been published in the course of the thesis at hand:

[1] I. Bournias, R. Chotin, and L. Lacassagne, "Fpga acceleration of the horn and schunck hierarchical algorithm," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5. DOI: 10.1109/ISCAS51556.2021.9401068.

[2] I. Bournias, R. Chotin, and L. Lacassagne, "Using hls for designing a parametric optical flow hierarchical algorithm in fpgas," in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2022, pp. 1600–1604. DOI: 10.1109/ISCAS48785.2022.9937732.

scientia potentia est

— Sir Francis Bacon, 1668

ACKNOWLEDGMENTS

Three and a half wonderful years finally come to an end. It has been a very interesting journey, during which i met some amazing people.

First of all i would like to thank Roselyne. Thanks for guiding me through all these years and supporting me in all the decisions i took. I am really lucky that you were my supervisor as you are a truly wonderful person with endless will for life. The second person I would like to thank is my co-supervisor, Lionel. Lionel transmitted me the passion for research and was always there to help me with any difficulty i faced.

I would like to thank all my friends in the Lab with whom i spent many tremendous moments which i am already missing. Thank you Fedia, Julian, Gabriel, Sarah, Mohamed, Jonathan, Nathan, Thomas, Alan, Maxime, Rieul, Ning, Spyros, Theofilos and Antonis. I would also like to thank my friends, Dimitris, John, George, George and Kyrgiakos from the Fondation Hellenique with whom i spent many hours playing cards and walking around Paris. I will never forget these moments and the time we spent together.

I am very lucky that my sister, my mother and Maria were very patient with me and supported me through all these years. Finally, this thesis is dedicated to my father who led me to the world of engineering. I hope you can see this.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 CONTEXT

This thesis was conducted in the LIP6 Lab in Sorbonne University of Paris and is a part of the Meteorix project [1]. Meteorix is a university CubeSat mission and its aim is to integrate inside the CubeSat all the materials necessary (such as the cameras, the embedded devices, the algorithms, the memories etc.) for the detection and characterization of meteors. Meteor tracking and characterization requires fast meteor detection algorithms which in the case of the Meteorix project have to be able to fit in embedded systems. Within the LIP6 lab, this thesis was a collaboration between the Analog and Digital Integrated Circuit team (CIAN) and the Hardware and Software for Embedded System team (ALSOC). The activities of the ALSOC team concern methods, algorithms and tools for multiprocessors system on chip design and cots. Such highly integrated multiprocessors architectures are used in embedded applications such as automotive, nomad, audio and video, and telecommunications. The design of these systems requires the development of hardware and software co-design methods. They focus on advanced hardware architecture, communication protocols, embedded operating system, real-time constraints, formal methods for verification systems and optimization of code generation. The research areas of CIAN team focus on designing methods for analog and digital components which are integrated on the same chip (AMS SOC). The main aim is to design generic components that can be adapted for a wide range of applications. Components differ depending upon the nature of the signal to process : signal processors (DSP), digital ASIC, programmable logic component (FPGA), analog circuits, multi-standard RF components and SOC Clocks. The LIP6 laboratory is in charge of the embedded meteor detection and the expertise of the CIAN and ALSOC teams is employed to integrate the algorithms in the CubeSat. To this direction, CPUs [2] and GPUs [3] have been considered so far by the ALSOC team for the optimization of the algorithms . In this thesis, we will focus in FPGAs to make the implementation of the algorithms (specifically the optical flow) that detect the meteors, sustainable in terms of resources utilization and able to compute the required tasks in real time as the CPUs and GPUs implementations are not always able to achieve that. This is the main contribution of the CIAN team.

An essential algorithm which is part of the meteor detection is the optical flow which is used to estimated the pixel movement of two consecutive images [4],[2]. This algorithm is the one which demands the most computation time (over 80% of the total computation time[5]) and it demands special attention in order to be accelerated. Nowadays, neural

networks are used widely for optical flow and they achieve the best accuracy compared to all the other algorithms [6],[7]. However, the computation power they demand is way higher than the one embedded devices can offer [8]. Consequently, it is unrealistic to use neural networks for the detection of meteors in small CubeSats as they do not usually carry powerful processing units which require a lot of energy and space to function. By taking that into consideration, in this thesis we turn to a classical optical flow algorithm which with the appropriate handling and optimizations in an FPGA, offers real time computation, competitive accuracy and limited hardware resources consumption.

Nevertheless, optical flow is a vital part of many other computer vision applications in real life. Autonomous driving and vehicle navigation is the kind of algorithms which make a good use of optical flow [9],[10],[11],[12],[13],[14]. For example, when the car is moving along a flat road and the optical axis of the camera is parallel to the ground, the motion field is expected to be almost quadratic and have a specific structure and consequently information about the angular velocity and the speed can be extracted from the flow vectors [10]. Optical flow can be employed to provide strong information for hypothesis generation about the possible presence of cars. Approaching vehicles at an opposite direction produce a diverging flow, which can be quantitatively distinguished from the flow caused by the car ego-motion. In robotics navigation, it is used with the aim of helping solving the obstacle avoidance problem in the ground and in the air. With the vectors, the rotational velocity, translational velocity and terrain information can be estimated separately through the integration of optical flow with inertial, GPS, and range data as thoroughly explained in [13]. Optical flow is a vital algorithm in the biomedical domain too. It has been used to measure the functionality of many body organs with remarkable precision [15],[16],[17][18],[19]. The flow is used for the measurement of organs velocity in centimeter per second in each studied pixel and after that it is visualized as colored vectors superimposed on Magnetic resonance imaging (MRI) images. The measured flow is also used to extract useful biological information for some part detection in the organ which otherwise can not be extracted or even to measure some anomalies in the blood pressure[20],[21],[22],[23] . Moreover, optical flow is used widely in the Defence domain. In video surveillance finding the velocity vectors of the video is of great importance in order to extract useful information [24],[25],[26],[27],[28],[29], [30],[31],[32]. In [25] these vectors are used in the Histograms of Oriented Gradient (HOG) descriptors for captioning the motion information for gesture recognition. In [26] it was used for assessing crowd behaviors from videos in terms of motion pattern and dominant paths. To do that, they rely on the estimation of a collection of sub-affine motion models in the image. In [27] they propose to use optical flow as an igniting point for video recording. As the amount of data when recording for a long time is huge, the velocity vectors can be used to store on the memory data when only movement is happening. In [29], [30] and [28] they use optical flow for video denoising where the flow vectors are used for the detection of the noise

in the image. Finally, optical flow can be used for video indexing [33],[34],[32], restoration of aged videos [35] and aerodynamics or fluid mechanics [36],[37].

A special mention has to be done in the LIP6 lab in Sorbonne University where optical flow is developed for the detection of meteors in the sky and which is also part of the Meteorix Project. Millet [4],[2] in his work is implementing a large image processing pipeline where the dominant part is the optical flow [3], for the flow vectors estimation and in these vectors a component labeling algorithm is applied for the detection of the meteors [38],[39]. With this pipeline which is designed in CPU and GPUs the meteor detection is accomplished with remarkable accuracy.

As in the case of the CubeSat mission project, installing huge computation units in cars, biomedical devices and surveillance cameras for neural networks optical flow computations is not usually feasible. Moreover, the training they require is complex and a massive amount of data which must be also relevant has to be found in order to achieve an effective one. Conventional optical flow algorithms does not suffer from these bottlenecks. They achieve comparable accuracy and they are ideal for embedded devices as they can be massively accelerated. This is why still the conventional optical flow algorithms take the largest piece of the pie in embedded systems and the research done in this domains remains one of the most hot research topics.

The aforementioned applications demand specific constraints in terms of throughput and accuracy [2],[1]. Designing computer vision application for embedded devices is a challenging task and it requires a lot of trade off to be made in order to achieve real time computation and in the same time the hardware to remain low. In FPGAs, the trade off can be made in the arithmetic format used for the representation of the data, in the pipeline and the parallelism with respect to the resources utilization. However, increasing the paralellism might lead to cases where the external memory can not deliver the requested throughput. Thus, we have to take into account that different FPGAs offer different possibilities. It is essential for a design to be scalable in order to fit in different devices and meet certain constraints. In this thesis we offer this possibility which as far as we know is something that is missing from the state of the art.

To sum up, the first challenge that shows up when designing an optical flow algorithm for FPGAs is the accuracy of the detected flow. Increasing the accuracy too much potentially leads to excessive hardware resources utilization. Real time computation also demands careful handling in order to avoid a huge hardware resources utilization. As a result the design has to respect certain constraints, regarding the accuracy of the algorithm, the resources utilization and the computation throughput.

1.2 CONTRIBUTIONS

In this thesis we concentrate on the design space exploration of the multi-scale Horn and Schunck algorithm on FPGAs. We will use techniques which are widely used in FPGA

designing as well as new ones to overcome several bottlenecks. Although we focus on a specific algorithm, the propositions can be used in other image processing algorithms to deal with existing bottlenecks

Our main contributions are:

- One of the most important parts of the algorithm is the accuracy it achieves. Thus, we will explore the pyramid levels and iterations number of the algorithm with the aim of improving the accuracy of the flow detection. We implement two interpolation algorithms and we show the impact they have on the accuracy. Since we talk about embedded designs, we use less bit to decrease the resources utilization to represent the data and keep the accuracy as high as possible. We also use different arithmetic formats in each pyramid level and we observe the drop in accuracy. Thus, we show how all these parameters impact the accuracy.

- Optical flow in real world problems require real time and sometimes higher than real time throughput. Consequently, we explore different techniques to accelerate every part of the algorithm. We make use of previous state of the art techniques such as vectorization and deep pipeline to accelerate the mono-scale kernel. We treat the multi-rate nature of the algorithm more effectively than previous state of the art works by proposing architectures which are friendly to the external memory bus and bandwidth. We use trans-floating point arithmetic and lower precision arithmetic to fit more cores in the design which increases accuracy. We scale up our designs and we show that our fastest FPGA design outperforms in terms of throughput all the previous single FPGA State of The Art designs.

- Although throughput is very important, it usually comes with an excessive cost in hardware. This is not the case in our work as we propose solutions to keep the resource utilization low. Such solutions are the bi-linear interpolation and more important of all, the computation reuse. We focus on inter-iteration computation reuse which to the best of our knowledge, has never been done before in optical flow. We offer a way to use properly the Block RAMs and by using smaller arithmetic formats, the bus of the external memory can be used for increased vectorization.

- After accuracy, throughput and resources usage has been explored, we are performing a design space exploration by taking these three factors into account. We provide the FPGA designer with the ability to choose between different configurations according to his needs and his applications constraints. These configurations vary from small to large architectures which are either accurate or flexible in terms of resources. As far as we know, contrary to all the previous state of the art works in FPGAs, we offer a design which is scalable in terms of throughput, accuracy and resource utilization.

1.2.1 *Outline of The Thesis*

This thesis is composed from 7 chapters including this one.

The second chapter is about an introduction of the optical flow algorithm. A definition is given in the beginning. Following that, several optical flow algorithms are discussed and how they are already implemented by other state of the art works. Their designing advantages and disadvantages are pointed out and it is mentioned why a new optical flow implementation is needed. The multi-scale Horn and Schunck optical flow algorithm, which is used in this thesis, is described and we make an introduction of how a scalable design of this algorithm can solve many existing bottlenecks of other designs. A summary of the strong points of each state of the art work in FPGAs concludes this chapter.

Accuracy is essential in optical flow. Chapter three is devoted in the accuracy. In the beginning, the error metrics that are used for the evaluation are presented. The multi-scale Horn and Schunck is a pyramid design. Thus, an exploration is done regarding the number of the levels of the pyramid and the iterations' numbers in each pyramid level. Bi-linear and bi-cubic interpolation is included in the exploration too. With the aim of reducing the hardware usage, several floating point formats are explored to show their impact in the flow vectors. Trans-floating arithmetic is proposed as an alternative and in the end we do a comparison with the state of the art regarding the accuracy.

Achieving real time computation is very challenging in embedded systems. During the fourth chapter we deal with accelerating the algorithm in terms of throughput. Classic iterative stencil loops techniques are employed to accelerate the mono-scale algorithm. Warping is treated in a way in order not to stall the whole pipeline. The multi-rate nature of the algorithm is mentioned and solutions to overcome the bottlenecks of this kind of algorithms are proposed. Trans-floating point arithmetic format is used to further speed up the computation and finally a comparison with other state of the art works is done regarding the throughput.

Throughput comes with a cost in hardware resources. The resources utilization of the FPGA is a major part in this thesis and chapter five is solely related to this. The resources usage needs of the different components of the algorithm are discussed. Several trade-off are explored concerning the DSPs, logic blocks utilization, Blocks RAMs and the external memory bandwidth. The impact of the different floating point formats in the hardware resources is shown and with the help of trans-floating arithmetic the design is scaled up.

In chapter six all the information are combined in order to perform a design space exploration. This exploration gives the designer the opportunity to tune the design according to his needs. By inserting certain parameters on our model he can choose among different options. Then, we show the results of the exploration.

Finally, in chapter seven a conclusion of our work is done. We point out the strong contributions that this thesis is offering in the state of the art. We discuss the perspectives

and how our contributions can be employed in other algorithms too. We refer to the future work that can follow as this is an ongoing research.

2 OPTICAL FLOW AND EMBEDDED DEVICES

In the introduction it was pointed out, why image processing and especially optical flow computation in real time speed is very important in real life. Furthermore, the context of the thesis and how this thesis contributes in the state of the art was analyzed. In this thesis, optical flow is the kind of algorithm we will concentrate on. As so, we will present the different application domains where optical flow has been used by other state of the art works and why it is considered one of the most important image processing algorithms. The applications vary from the medical world to the autonomous driving and the surveillance systems. Then, we will describe the algorithms that are widely used now. Depending on the algorithm, better accuracy can be accomplished for example with convolution neural networks or in another case reduced computation time and effort with gradient based algorithms. Furthermore, since embedded devices require sophisticated designs we will mention to other recent state of the art implementation in embedded systems. We will refer to their strong parts, their disadvantages and why a new optical flow design in embedded systems is needed. The new algorithm will be presented, its different parts will be analyzed and other state of the art works will be presented which address the same problems. We will mention why our work is needed and where the other works lack in terms of design. Finally, we will sum up the chapter.

2.1 OPTICAL FLOW

Image processing is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it. There is a vast variety of image processing algorithms, one of the most hot topics in science. In this thesis we will focus in a family of image processing algorithms called optical flow algorithms.

Optical flow is defined as the motion of each pixel on the whole surface of an image [40]. It often serves as a good approximation of the true physical motion projected onto the image plane. This motion is usually ascribed to the motion of objects in the scene [41]. The notion of optical flow actually refers to the displacement of intensity patterns. It is caused by relative motion between the observer and the objects of the observed scene and it represents motion of intensities in the image plane. In computer vision, optical flow is usually used to detect the displacement of objects. Thus, the variable that we actually want to retrieve with optical flow is the projection on the image plane of the 3D motion in the scene, usually called motion field. A major issue that occurs with optical flow algorithms is that often the intensity changes are not happening due to pixel displacement

but because of other disturbing phenomena like lighting changes, reflection effects or modifications of the properties of the objects affecting their light emission or reflectance.

The optical flow methods try to calculate the motion between two image frames which are taken at times t and t+Δt at every pixel position. The equation that describes the optical flow solver is Equation 1. I is the image where the optical flow is performed, x represents the movement in the x axis, y in the y axis, t the time and Δx, Δy and Δt are the partial derivatives of each one of them which are used to detect the movement. These methods are called differential since they are based on local Taylor series approximations of the image signal. That is, they use partial derivatives with respect to the spatial and temporal coordinates.

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t) \tag{1}$$

2.2 DIFFERENT OPTICAL FLOW ALGORITHMS

2.2.1 *Lukas and Kanade*

2.2.1.1 *Mono-scale*

One of the most famous algorithms that is used for optical flow estimation is the Lucas Kanade mono-scale algorithm (L&K) [42]. This algorithm is a gradient based one. The velocities are computed from the spatial and temporal derivatives of the image's brightness. In order to obtain correct optical flow values using gradient-based optical flow algorithms, certain assumptions have to be met such as the assumption that the flow is not changing a lot between consecutive frames and that the pixel brightness remains the same (const in Equation 2). With these two assumptions Equation 2 derives (t is the time for the first frame while t+1 is the time for the second frame). By taking the partial derivatives of this equation with respect to time (Equations 3, 4), Equation 5 derives.

$$I(x(t), y(t), t) = I(x(t + 1), y(t + 1), t + 1) = const \tag{2}$$

$$\frac{I(x(t), y(t), t)}{dt} = 0 \tag{3}$$

$$\frac{dI}{dx}\frac{dx}{dt} + \frac{dI}{dy}\frac{dy}{dt} + \frac{dI}{dt} = 0 \tag{4}$$

$$I_x u + I_y v + It = 0 \tag{5}$$

In Equation 5 there are two unknown variables (u,v) but with only one equation to be found. These two variable are the velocity in the x axis (u) and in the y axis (v). For this reason, a method has to be proposed to solve this problem. Lucas and Kanade considered a small neighbourhood of each pixel and they said that inside this neighbourhood, a pixel moves in the same direction as it nearest neighbours. With this assumption from Equation 5 the next step is Equation 6.

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_n) & I_y(p_n) \end{bmatrix} \times \begin{bmatrix} u \\ v \end{bmatrix} = (-1) \begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_n) \end{bmatrix} \tag{6}$$

p_1 ... p_n in 6 are the neighbouring pixels where the assumptions of (L&K) are referred to. Now, there is enough information to estimate the optical flow vectors (u,v).

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_n) & I_y(p_n) \end{bmatrix} = A, \begin{bmatrix} u \\ v \end{bmatrix} = d, \begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_n) \end{bmatrix} = b \tag{7}$$

The notations as shown in Equation 7 are used to form the following optimization problem which is shown in Equation 8.

$$Ad = B \rightarrow min\|Ad - b\|^2 \tag{8}$$

To determine the solution of d, the equation was multiplied by A^TW on both sides where W is a weight matrix which is usually called a mask of the neighborhood pixels. Thus the final equation where the velocities vectors are estimated is Equation 9. One major drawback of the algorithm is that the invertibility of the matrix A_TWA does not always guarantee a correct solution

$$(A^TWA)d = A^TWb \rightarrow \begin{bmatrix} u \\ v \end{bmatrix} = (A^TWA)^{-1}A^TWb \tag{9}$$

The serious disadvantage from which L&K suffers is that it can only detect small pixel displacements because of the assumptions made.

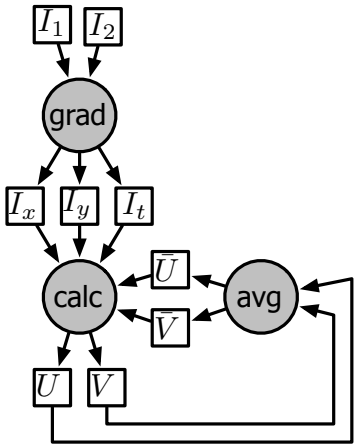

Figure 1: H&S mono-scale iterative scheme

2.2.1.2 *Multi-scale*

The multi-scale L&K method is a solution in order to tackle the bottleneck of the small pixel displacements of the L&K mono-scale algorithm. At first, an image pyramid is constructed for every image, where optical flow is performed. Then, the mono-scale algorithm is applied to every scale and the associated flow vectors are extracted. Since, the pyramid images are down-scaled, the flow vectors of every scale, correspond to larger pixel displacements for the initial image. This way larger pixel displacements are detected. In the end, all the velocity vectors are summed to provide the final velocities [43],[44].

2.2.2 *Horn and Schunck*

2.2.2.1 *Mono-scale*

The algorithm that we implement and focus in this thesis is the Horn and Schunck (H&S) [45] which is one of the most famous algorithms for optical flow and its scheme is shown in Figure 1. It is also a gradient based algorithm but in contrast to L&K, the global minimum of a function is searched. In [45] an additional assumption was proposed to solve Equation 5 which is that the optical flow should be smooth over the entire image. As a result for every pixel the computed flow in a small neighbourhood is similar. The problem formulation in the case of H&S is described by Equation 10. Parameter α is a regularization constant. Larger values of α lead to a smoother flow. Equation 10 can be minimized by solving the associated multi-dimensional Euler–Lagrange equations.

$$\int\int [(I_x u + I_y v + I_t)^2 + \alpha^2(\|\nabla_u^2\| + \|\nabla_v^2\|)]dxdy \tag{10}$$

By using Cramer's rule the following iterative scheme is derived (Equations 11 and 12) which is used to estimate the optical flow vectors and in Figure 1 is denoted as calc. The mean of the velocities is done with Equations 26 (avg in Figure 1). The derivatives I_x, I_y and I_t are computed by Equations 13, 14 and 15 (grad in Figure 1).

$$u^{(i)} = \bar{u}^{(i-1)} - I_x \frac{I_x u^{(i-1)} + I_y v^{(i-1)} + I_t}{\alpha^2 + I_x^2 + I_y^2} \tag{11}$$

$$v^{(i)} = \bar{v}^{(i-1)} - I_y \frac{I_x u^{(i-1)} + I_y v^{(i-1)} + I_t}{\alpha^2 + I_x^2 + I_y^2} \tag{12}$$

$$\begin{aligned} I_x = \frac{1}{4}(((&I_1(x+1,y) - I_1(x,y))+ \\ &(I_1(x+1,y+1) - I_1(x,y+1))+ \\ &(I_0(x+1,y) - I_0(x,y))+ \\ &(I_0(x+1,y+1) - I_0(x,y+1))) \end{aligned} \tag{13}$$

$$\begin{aligned} I_y = \frac{1}{4}(((&I_1(x,y+1) - I_1(x,y))+ \\ &(I_1(x+1,y+1) - I_1(x+1,y))+ \\ &(I_0(x,y+1) - I_0(x,y))+ \\ &(I_0(x+1,y+1) - I_0(x+1,y))) \end{aligned} \tag{14}$$

$$\begin{aligned} I_t = \frac{1}{4}(((&I_1(x,y) - I_0(x,y))+ \\ &(I_1(x,y+1) - I_0(x,y+1))+ \\ &(I_1(x+1,y) - I_0(x+1,y))+ \\ &(I_1(x+1,y+1) - I_0(x+1,y+1))) \end{aligned} \tag{15}$$

$$\bar{u} = \frac{1}{12}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 0 & 2 \\ 1 & 2 & 1 \end{pmatrix} * u \qquad \bar{v} = \frac{1}{12}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 0 & 2 \\ 1 & 2 & 1 \end{pmatrix} * v \tag{16}$$

The advantage of the H&S algorithm is that it yields a high density of flow vectors, i.e. the flow information missing in inner parts of homogeneous objects is filled in from the motion boundaries. On the negative side, it is more sensitive to noise than local methods. TVL1 [46] also belongs in the gradient based algorithm family and it originates from the H&S.

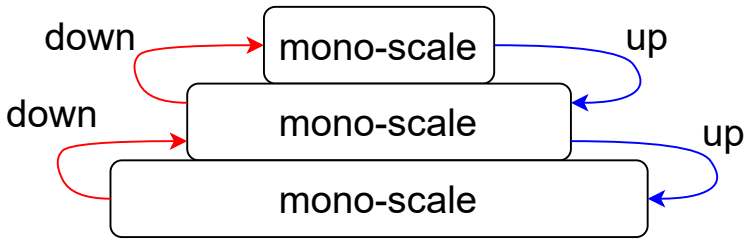

Figure 2: Pyramid of images

2.2.2.2 *Multi-scale*

Mono-scale optical flow algorithms suffer from limited pixel displacement detection [45]. Nevertheless, many applications such as meteor detection [4] require larger pixel displacement detection and the multi-scale algorithms offer a solution to this problem.

As referred in [45] the mono-scale H&S algorithm is able to detect a flow of one pixel per two consecutive frames. By taking this into account, when down-sampling the image by a specific factor which in our case is 2 and performing the H&S there, the detection might stay the same in this down-sampled scale of the image, but after up-sampling the computed velocities in the initial image size, the displacement is doubled. Considering that, a pyramid of images can be created as shown in Figure 2, whose levels and down-sampling factor can be defined with the aim of increasing the pixel displacement detection.

The description of the algorithm is shown in Figure 3. In the beginning, the two consecutive frames are down-sampled in order to create the images of the pyramid. For the down-sampling a five tap Gaussian filter is used. When this procedure has finished, the mono-scale H&S computation is performed iteratively in the smallest image (coarse level of the pyramid). The computed velocities for this level $((u, v)^{\lambda+1}_{initial})$ of the pyramid are extracted and are up-sampled (up) in order to be used in the next pyramid level. It is of great importance to take into account the motion that was extracted in the previous level, in this level too. Thus, in the beginning of the next level a warping is performed between these pre-computed velocities and the image ($I_{2_{rec}}$) that will be computed in this level. This is done to compensate the motion because of the previous extracted flow. To implement warping either bi-linear or bi-cubic interpolation is used depending on the accuracy. The same procedure followed in the previous pyramid level is followed in this level too. The only difference is that in the end of the level, the computed velocities from the previous pyramid level and this pyramid level $((du, dv)^{\lambda+1}_{final})$ have to be accumulated and then up-scaled to be used for the next pyramid warping. In the final pyramid level (fine level) where the initial images are used, the same pattern with the warping and the mono-scale kernel is followed. In the end all the velocities from all the levels are accumulated to provide the final computed flow vectors $((u, v)^{\lambda+1}_{final})$.

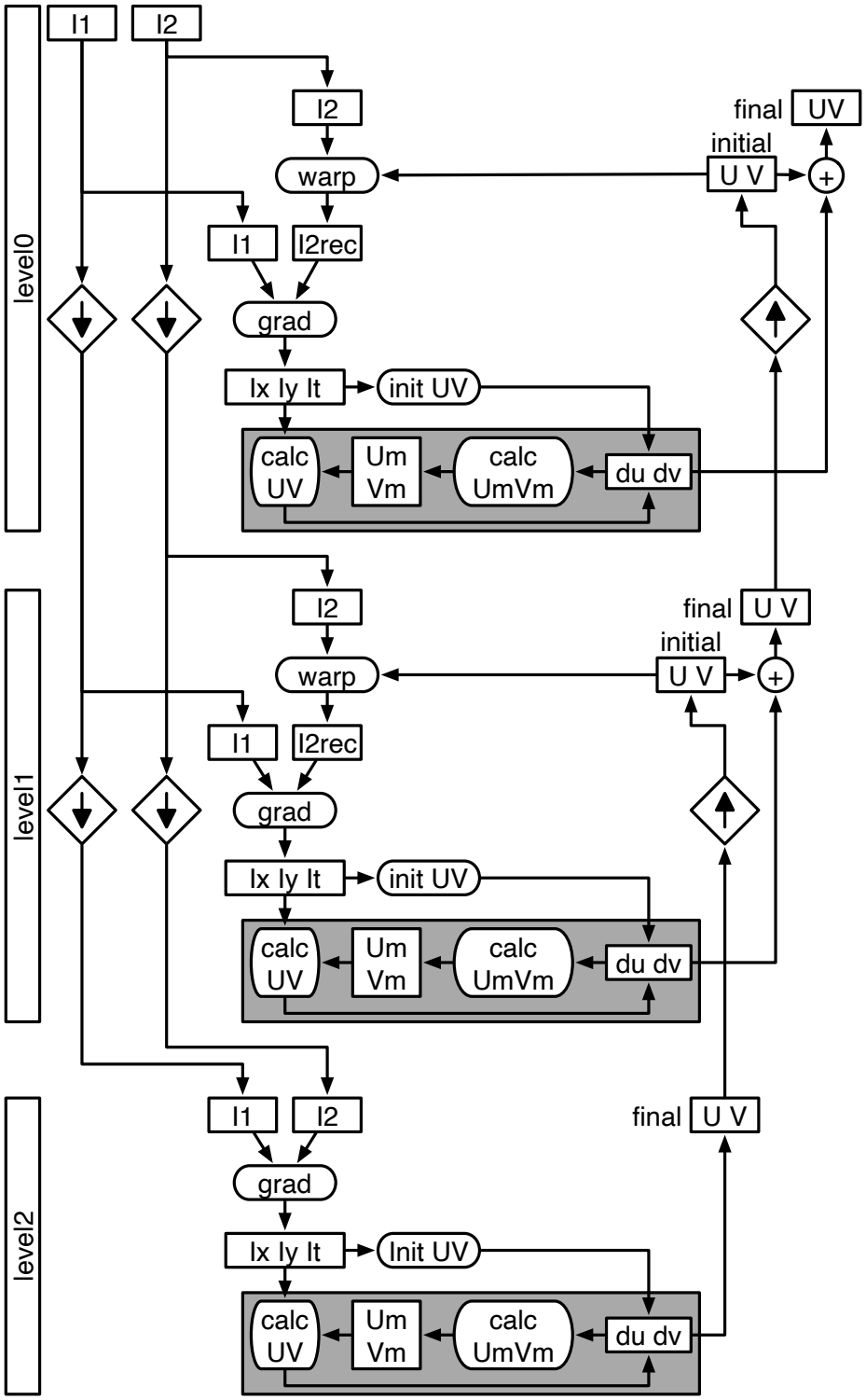

Figure 3: Description of the multi-scale H&S Algorithm

In each pyramid level the displacement detection of pixels is 1. For a three levels pyramid and for a down-sampling factor of 2, the largest total displacement that can be computed is $\sum_{\lambda=0}^{2} 2^{\lambda} = 7$ pixels. One can choose different down-sampling factors to increase the motion displacement detection but it becomes really complicated for the hardware design [47] and can lead to inaccurate results.

2.2.3 *Other Optical Flow algorithms*

Block methods algorithms are also used for optical flow [48]. There, the operation scheme is based on determining the maximum correlation of a certain area between two frames. The displacement of this area is equivalent to the determined optical flow [43]. The disadvantage of this method is that the analysis is predetermined to characteristic points. This ultimately can lead to sparse flow but with the advantage of being more prone to errors.

In [49] it is analyzed how phase and frequency are used for estimating the velocity flow. There, they make use of the fact that phase is more robust compared to the amplitude with changes in contrast, scale, orientation and speed. For the analysis Gabor filters and Fourier transform is applied in the signals. The biggest disadvantage of this method is the heavy computational effort that has to be performed.

During the last few years neural networks and especially deep neural networks (DNN) are used for the extraction of the optical flow. These approaches excel in the accuracy of the detection but they require sophisticated training and huge computational effort. More precisely in [50] the FlowNet neural network was presented. Convolutional neural networks are known to be very efficient at learning input relations given enough data for training. Thus, in the work of [50] they follow an end to end learning approach to predict optical flow which is to start with a given data set consisting of image pairs and ground truth flows which they use to train their network to predict the x and y flows directly from the images. In order to perform the optical flow they create two separate, yet identical processing streams for the two images and then they combine them at a later stage in the neural network in order to extract the flows. In [6] the new improved version of FLowNet is presented where they solve the problems of small displacements and noisy artifacts in estimated flow fields. They do that by firstly using more sophisticated data sets to train their networks. They introduce a warping operation and show how stacking multiple networks using this operation can significantly improve the results. Finally, they focus in small displacement of pixels by introducing real problem data sets to achieve that. In LiteFlowNet2 and LiteFlowNet3 [51],[52] regularisation is used to reduce the influence of noise and smooth the flow, while conserving the edges. In SpyNet [51],[53] a pyramid of images is used and small sized network for each scale which is trained independently. There, the learned convolution filters appear similar to classical spatio-temporal filters, giving insight into the method and how to improve it. However, since they compute many pyramid scales there is the possibility of loosing objects when the image is down-scaled.

2.2.4 *Advantages and disadvantages of each algorithm*

The neural networks are able to compute the optical flow with the best accuracy compared to all the other state of the art works. The drawback they have, is the enormous computation cost and the very high computation time they require for large images. The same happens with the block matching algorithms, although the accuracy they achieve is lower related to the neural networks. H&S and L&K are ideal for implementing in embedded devices because of their computation patterns. Nevertheless, the accuracy of the mono-scale version is deteriorated compared to other state of the art algorithms. Thus, the multi-scale version is considered which is able to improve drastically the accuracy and detect large pixel movements. The advantage of the H&S algorithm compared to L&K is that it is a 100% dense algorithm which means that the optical flow is detected in the whole image and not locally as in the L&K case.

In the next subsection we will present the state of the art of optical flow in embedded devices.

2.3 OPTICAL FLOW IN EMBEDDED DEVICES

Despite the impressive accuracy the DDNs achieve in computing the optical flow, their computations demands make it really challenging to implement in embedded devices. In [6] they implement in the Nvidia GTX 1080 GPU the FlowNet2 and they present a trade off between accuracy and FPS (Frames per Second). The highest FPS they achieve is 123 but for small image sizes and with limited accuracy of detection even in this powerful GPU. A recent work in implementing neural networks for optical flow in FPGAs is presented in [8]. In their work they use a multiplexing binary neural network (BNN) which is less demanding computation wise, instead of a CNN, for pyramidal feature extraction and they make the design independent of the pyramidal level number. They achieve impressive accuracy in the Middlebury data set [54], but there are no information about the resources utilization of the FGPA. The FPS they achieve for the image size they considered is deteriorated compared to other optical flow designs in embedded systems. Furthermore, a correlation is not presented between hardware utilization and throughput to see how it scales.

Many embedded works design the L&K for the optical flow detection. In [55] they use the Nvidia Jetson CPU (only the ARM CPU and not the GPU) to accelerate the L&K. They optimize the throughput with respect to the energy consumption. They also include an implementation on a Nvidia Tegra AGX in order to increase the throughput. Their design which also includes the TVL1 achieves an FPS of 25 for an image size of 960×540 pixel. In [56] they design a multi-scale L&K in a custom processor in the Artix XC7A100T-CSG324 Xilinx FPGA and in an ASIC with a 40 nm CMOS implementation. In order to increase the accuracy they implement a 4 levels pyramid but the FPS they achieve is

only 5 for a 640x480 pixel image. Their design is not scaling for larger images too. In [57] the Texas Instruments C66x, a 10 Watt embedded digital signal processor (DSP) is used for the acceleration of a pyramidal L&K. They perform an exploration concerning the number of cores needed to compute different images sizes and different iterations for a 4 levels pyramid. In [58] they use a CPU-GPU (Core 2 Quad 2.83GHz CPU and an NVidia GeForce GTX285 GPU) for the implementation of the L&K image registration algorithm for 3D computational platform stabilisation. They manage to compute 30 FPS for a 512×512×2 region of interest. In [44] the L&K is again implemented on a Titan GPU for very large image size of 3840x2160 pixels. They achieve real time computation but there are no information about accuracy. [59] design a L&K in a Virtex 6 FPGA. Their work is mainly focused on reducing the demand of external memory bandwidth of the image. They achieve that by compressing the image when they write it to the external memory. Then when they process the image they up-sample it with as much as possible minimum loss of accuracy. They perform an exploration regarding the wordlength of the data but since they use the mono-scale version their accuracy of detection is not very competitive compared to other state of the art works. In their resources utilization there is a slight increase due to the compression of the image. In [60] the authors present a mono and a multi-scale implementation of the L&K algorithm in the Virtex 4 FPGA. They explore different arithmetic formats such as fixed or floating point and they parallelize their code to increase the throughput. They perform warping in each pyramid level which highly reduces their computation speed since they want to make the design to use as less hardware as possible. The final L&K implementation which we will refer to is the one of [61]. They achieve the highest throughput of all the other state of the art works for an image with size 1024x1024 pixels. Their working clock frequency is only 90 MHz which is very impressive. However, they use two FPGAs to make the computation, one for the handling of the inputs and one for the computation of the algorithm. Some parts of the algorithm are also performed in a power PC. That is why we believe they achieve an impressive throughput. Finally, in their work there are no information about the accuracy of their algorithm.

[62] is implementing a hierarchical block matching-based optical flow algorithm in the Virtex 7 FPGA. The accuracy they achieve is higher compared to other state of the art works in FPGAs as they use a multi-scale algorithm. However, the demand of the algorithm is huge in terms of Block RAMs especially when images larger that 640x480 pixel are considered. This happens because they use on-chip memory as blocks where they store the data which they want to process to extract the flow vectors. When the number of blocks is increased with the aim of improving the accuracy of the detection, the throughput is significantly decreased due to the huge computation demand.

A multi-scale, multi-orientation phase based algorithm is employed by [63] on a Virtex 4 FPGA. For the implementation they use various fixed point formats and they warp the images in the different pyramid levels to increase accuracy. They have 4 pyramid levels

and they perform the optical flow in 640x480 pixel images. However, they do not include enough information about accuracy and they do not present how their design scales up in terms of throughput.

Along with L&K, H&S is the most famous algorithm which is implemented widely in embedded devices. [64] designed a mono-scale H&S on the Jetson TK1 board. They applied various optimizations, such as half precision IEEE arithmetic format, SIMD and various pipeline patterns. They tried also different iteration numbers and different image sizes. They achieved 25 frames per second for up to 8Mpix images for 0.35J per image. There were no information provided for the impact of the arithmetic format in the quality of the detection. [3] is using the Jetson AGX Xavier GPU for the implementation and it achieves remarkable throughput for large images by using half precision floating point and an alternative pipeline pattern. [65] is implementing the same algorithm but on an FPGA. The computation time their algorithm needs is very high compared to other state of the art works. They also store the intermediate computed velocities in the Block RAMs which leads to huge cost of on chip memory or the ability to compute only small images. [66] on the other side is able to compute full HD images with a throughput of 283 MPixel/sec. However, it is not using techniques to reduce the resources utilization or to provide a scalable architecture which would fit in a smaller FPGA than the one it is using. There are no information regarding the accuracy achieved too. [67] is implementing exactly the same algorithm and it also achieves impressive throughput compared to other state of the art works. However compared to [66], they deal with the iterations number, the floating or fixed point format and the convergence. Moreover, they claim that in order to detect large pixel displacements they perform the mono-scale algorithm very fast. The last two works are not exploring the trade-offs that have to be made between the external memory bandwidth and the on chip memory. [43] is implementing a multi-scale H&S algorithm on the ZCU 104 platform. The pyramid's levels are two and for the motion compensation the bi-linear interpolation. The throughput they achieve is very high but this comes with an important cost in the DSPs units of the FPGA. They use both vectorization and deep pipeline techniques to accelerate the algorithm. They use fixed point arithmetic and a simple convolution algorithm to down-sample the images. However, since they use two pyramid levels the accuracy is reduced compared to previous state of the art works with more scales. Furhtermore, they perform the interpolation in the end of the pipeline of each pyramid level and then they create the new image for the fine level. This also affects negatively the accuracy. Finally, as all the previous H&S designs in FPGAs they use computation reuse for the derivatives.

In this thesis we will explore and implement the multi-scale H&S algorithm in FPGAs. We believe that since the previous state of the art works focus solely either on the accuracy or the throughput part, there should be a work which would offer a clear trade off between them. The multi-scale H&S is the kind of algorithm which offers this advantage as it can be tuned in many ways.

2.4 DIFFERENT COMPONENTS OF THE MULTI-SCALE H&S

In Figure 3 the different blocks that form the described algorithm in this thesis are presented. The hierarchical H&S algorithm is a multi-rate image processing algorithm [68] which consists different processing units.

The mono-scale H&S kernel is an Iterative Stencil Loop (ISL) algorithm. ISL is a kind of algorithms where a grid is iteratively updated cell by cell according to a multidimensional local fixed input array from the starting grid. Usually, the computation is done for all the cells of the grid for iteration $i-1$ before calculating all the cells for iteration i. Instead of doing the computation with all the grid there is a possibility to compute a subset of cells for the grid in iteration $i-1$ in order to calculate some adequate cells for the iteration i. Each approach has a different impact in the implementation as we will describe in the next sections.

There are already a lot of works optimizing the above kind of algorithms especially for CPU and GPUs [69]–[74] in terms of parallelism, efficient blocking to reduce memory bandwidth and data locality.

More recently, for FPGAs, there was a great effort to optimize the ISL. These works [66], [67], [75]–[78] take advantage of the deep pipeline approach between consecutive iterations in a similar way as proposed by [79], [80] who also performs the acceleration with the help of OpenCL. Sano [81] in his work combines the deep pipeline and the parallel approaches in multi-FPGA architectures. Zohouri [82], [83] combined temporal and spatial parallelism targeting 2D and 3D iterative stencils in order to deal with different input sizes. In his work, shift registers were used as on-chip buffers to provide the PEs with the adequate data. A similar strategy with [79] was followed by Dest [84] and Chi [85], [86]. Chi proposed a design framework which also solves the problem of halo regions in iterative stencil computations. Nacci [87] followed a different approach compared to the aforementioned works. Instead of doing the computation frame by frame for all the c_a points of the grid (in a current iteration), he takes advantage of the fact that c_a need only a neighborhood of c_{a-1} to be computed. He performs the desired processing by repeatedly applying a cone to portions of the input matrix. In that way he reduces the external memory bandwidth rate with an overhead for on chip memory and computation time. A similar approach was also followed by [88], [89]. Waidyasooriya in his work [90] uses Multi-FPGAs architectures and OpenCL to accelerate Stencil Computation and he also targets very large inputs which he handles with spatial pipelining. Finally, in the work of [91] there is a description of how stencil computation should be handled with high level synthesis tools.

Warping, with the appropriate designing can be considered as a stencil algorithm. The same happens with the down-sampling and the up-sampling core. Still, with the last two algorithms there is a fundamental difference. The input rate of the two algorithms is different from the output rate they produce. This makes them multi-rate algorithms.

Table 1: Advantages of the FPGA State of the Art Works

algorithm	Accuracy	Image Size	throughput	Resources	Tunability
H&S [66]	NA	✓✓✓✓	✓✓✓✓	✓✓	✓✓
H&S [67]	✓	✓✓✓✓	✓✓✓✓	✓✓	✓✓✓
H&S [95]	✓	✓✓✓✓	✓✓	✓✓	✓
H&S [96]	✓	✓✓✓✓	✓✓✓	✓✓✓✓	✓
MH&S [43]	✓✓	✓✓✓✓	✓✓✓✓	✓✓✓	✓✓✓
L&K [59]	✓✓	✓✓	✓✓	✓✓✓✓	✓
L&K [61]	NA	✓✓✓✓	✓✓✓✓	✓✓✓✓	✓✓
L&K [97]	✓	✓✓✓	✓✓	✓✓	✓
ML&K [43]	✓✓	✓✓✓✓	✓✓✓✓	✓✓✓	✓✓
ML&K [60]	✓✓	✓✓	✓	✓✓	✓
ML&K [43]	✓✓	✓✓✓✓	✓✓✓✓	✓✓✓	✓✓✓
MPB [47]	✓✓	✓✓	✓	✓✓	✓
HBM [62]	✓✓✓✓	✓	✓	✓✓	✓
DNN [8]	✓✓✓✓	✓	✓✓	NA	NA

Multi-rate algorithms are widely used in image processing pipelines and a lot of research groups concentrate their efforts on optimizing these kind of algorithms. In [92] [93] [94] they are using static scheduling for accelerating the aforementioned kind of algorithms. Despite the benefits of static scheduling there are no FPGA image processing compilers that can statically schedule realistic size multi-rate applications and they are relying on on the inter-stage FIFO approach which highly increases the usage of Block RAMs and decreases the throughput. There is not also the option of vectorizing the computation with the aim of increasing the throughput. The only work which concentrates on accelerating multi-rate algorithms for FPGAs is the one of [68]. In their work they effectively pipelining stencils with different rates and they try to keep the resources usage low. However, they do not take into account the external memory reads and writes which highly impacts the throughput. They also do not explore the different trade-offs that have to be explored between the external memory interaction and the Block RAMs.

2.5 DISCUSSION ON THE STATE OF THE ART IN FPGAS

Since in this thesis we are designing the optical flow multi H&S in an FPGA, it is important to summarize what other state of the art works are lacking and why our work is essential.

In table 1 we are presenting the strong points of the state of the art FPGA implementations. In this table, we include information about the algorithm used, the image size, the throughput, the FPGA resources utilization and the tunability, meaning from how many options the designer can choose in order to tune the design according to its needs. For example, if there is the option to parallelize the design or to use fewer hardware in order to compute the flow in real time. We grade the strength of each implementation in each domain. We can see from this table that the DNN implementation and the hierarchical block matching implementations achieve the best accuracy detection of the optical flow. Especially the DNN design achieves the highest accuracy between all the state of the art works in the Middlebury data set of images [54]. We also see that the mono-scale H&S designs achieve remarkable throughput but the work of Ishii[61] is achieving the highest throughput among all works for a mono-scale L&K algorithm. Nevertheless he uses two FPGAs for some part of the computation of the algorithm. [43] is implementing multi-scale H&S and L&K algorithms and it performs less iterations in the finest level which highly increases throughput. The gradient based designs are able to compute large images, while this is not the case for HBM and DNNs. Most of the works do not offer tunable designs which can fit to smaller FPGA devices, something we consider very important as with limited resources, still real time processing can be achieved with acceptable accuracy. Finally, it is important to point out, that designs with high accuracy are unable to compute the optical flow with high throughput for large images.

2.6 CONCLUSION

In the beginning of the second chapter of this thesis we have made an introduction to the optical flow algorithms. The definition of the optical flow was given. Different algorithms were presented and the advantages and disadvantages of each one of them were given. Their implementation in embedded devices was discussed and it was pointed out how demanding it is to design effective optical flow algorithms for them. We described the multi-scale H&S algorithm which is the optical flow algorithm we will explore and design in this thesis. The key characteristics of the algorithm were explained and the main challenges that we have to overcome in order to have a sustainable architecture. We referred on how other state of the art works dealt with these challenges and why there is a need for a new FPGA optical flow design dealing with all the factors that impact the design.

In the third chapter we will explore the accuracy of the multi-scale algorithm. We will see why the mono-scale architecture is not enough for detecting the optical flow and how other various factors impact the accuracy. We will use smaller arithmetic formats and we will use a very known sequence of images to evaluate our work.

3

ACCURACY

An increase in the word length of the data representation usually leads to an improvement in the extracted information from a computer vision algorithm and better precision for the human eye as it is able to detect the difference in accuracy of the pixels of the image. A fundamental part when designing image processing algorithms for embedded applications is the precision of the value representing the pixels of the image compared to a ground truth [54] and it has a key impact on the image quality level which the design achieves [98]. Addressing the issue of accuracy of the observed scene in optical flow is tremendously challenging and it requires careful steps and considerations [99]. Accuracy in optical flow depends on the observed scene, however in any case the accuracy can be augmented with careful handling of certain parameters of the design. In this chapter, an accuracy exploration analysis will be performed regarding the optical flow Hierarchical Horn and Schunck algorithm. Several factors are essential for the tracking of the accuracy in a multi-scale algorithm. Thus, the exploration will include different factors of the algorithm, such as the number of the pyramid levels, the iterations number in each pyramid level and the interpolation algorithm for the warping. As reducing the number of iterations play a significant role in throughput, in this chapter the impact it has in accuracy and convergence will be explored. Embedded applications usually demand the hardware to remain low in order be sustainable in terms of resources usage design. Since we are talking about embedded applications, the impact of reducing the arithmetic format will be explored [100],[101]. Consequently, in this thesis a deep exploration is performed in the multi-scale algorithm concerning the arithmetic formats which to the best of our knowledge has never been done before for a pyramid algorithm. For the exploration results the Arria 10 Han Pilot Platform was used and the components were designed with the help of the FloPoCo library [102]. Since decreasing the data size, decreases the stretching of the external memory bandwidth which potentially leads to higher throughput, the possibility of using smaller arithmetic formats in the last scale of the image which takes the longest time is explored in terms of the accuracy. Finally, a thorough comparison will be made with the state of the art and the most known sequence of images will be used for fair comparisons[54]. The first section will be about the evaluation metrics which are widely used in the literature to compare the different designs[97],[67].

3.1 EVALUATION METRICS

The optical flow is computed between two images. For evaluating the accuracy of the optical flow, it have to be compared with a reference value[98],[103]. However, evaluating the result is subjective, it depends on the observed scene but also on the application where optical flow is used[104],[105]. Nevertheless, by comparing our results with the rest state of the art designs, we can get an idea if our accuracy is acceptable on a very common data set. In order to do a fair comparison of the result with other state of the art works, certain metrics that are used widely will be used in this thesis too.

3.1.1 *Peak Signal to Noise Ratio*

The Peak Signal to Noise Ratio (PSNR) metric defined by Equation 18, computes the peak signal-to-noise ratio, in decibels, between two images[106],[107]. The higher the PSNR is, the better the quality of the optical flow is. The Maximum Square Error (MSE) [108],[109] which is shown in Equation 17 represents the cumulative squared error between the two images, whereas PSNR represents a measure of the peak error. The lower the value of MSE is, the lower the error is.

MSE requires two images for the computation of its value. The first image (I_1) is the one against which the optical flow is computed. The second image has to be created (I_2). Thus the optical flow vectors are used for the creation of this image. The initial image, where optical flow is computed, is interpolated by using bi-cubic interpolation with the computed velocities of the optical flow. The warped final image is used for the computation of the MSE metric. W is the width of the image while H is the height. Since the images that are considered, are gray scale images with a representation of 8 bits, R is the maximum fluctuation in the input image data type which, in our case, is unsigned integer 8 bit, so 255.

$$MSE = \frac{\sum_{w,h}[I_1(w,h) - I_2(w,h)]^2}{W \cdot H} \tag{17}$$

$$PSNR = 10 \cdot \log(\frac{R^2}{MSE}) \tag{18}$$

3.1.2 *Average Angular Error*

As explained in [54] the PSNR metric is not a very accurate metric because interpolation affects the accuracy. Thus, the quality of the interpolation highly impacts the result and in this thesis we will avoid using this metric. A more accurate metric for the evaluation

of the optical flow is the Average Angular Error (AAE) [56],[110],[111]. The AAE between two flows is the angle in 3D space between them ((u, v, 1.0) and (u_r, v_r, 1.0)). (u, v) is the computed flow and (u_r, v_r) is the ground truth flow (ground truth is the real optical flow vectors between the two images [54]). The AAE is computed by normalizing the optical flow vectors, taking the dot product, and then taking the inverse cosine of their dot product. AAE is described by Equation 19.

$$E_{AAE} = \frac{1}{W \cdot H} \sum_{W,H} \arccos \frac{1 + u_r \cdot u + v_r \cdot v}{\sqrt{(1 + u_r^2 + v_r^2) \cdot (1 + u^2 + v^2)}} \tag{19}$$

3.1.3 *Average Endpoint Error*

In AAE normalizing results leads in a situation, in which errors from small flows have the same influence in the final result as errors in large flows[54]. This is why a second metric is used which is also used widely in literature. This is the Average Endpoint Error (AEE)[112]. AAE is computed by Equation 20. The Euclidean norm is used to compute the average error between the obtained optical flow and the ground truth in pixels.

$$E_{AEE} = \frac{1}{W \cdot H} \sum_{W,H} \sqrt{(u^2 - u_r^2) + (v^2 - v_r^2)} \tag{20}$$

3.1.4 *Exploration Methodology*

For the exploration of the number of the pyramid levels and the iteration numbers in each pyramid level, single precision floating point format was used and bi-cubic interpolation which is widely used by the community for the experiments of the optical flow [60][113][63][47]. By doing that, we first determine these two factors, without reducing the accuracy of the detection by using smaller arithmetic formats or less accurate interpolation algorithms. Following that, we determine the interpolation factor but at this time we take into account the reduced floating point formats as it is essential for the algorithm selection. Finally, we evaluate the impact in the accuracy of reduced size floating point numbers and of trans-floating representations.

3.2 LEVELS OF THE PYRAMID AND ITERATIONS IN EACH LEVEL

The problem with the mono-scale algorithms is that they can not detect pixel displacement larger than one pixel per frame [67]. As a result, when someone wants to detect displacements larger than one pixel per frame alternative approaches have to be followed. One approach that is widely used in the literature is to perform the mono-scale algorithm

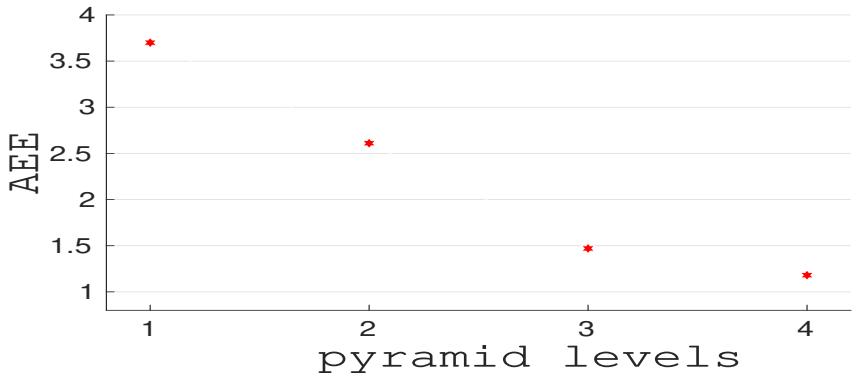

Figure 4: Number of pyramid level

extremely fast[66],[44],[114]. By doing that, many frames per second can be computed [67]. The largest detectable pixel displacement is still one pixel per frame but by computing many frames per second it is assumed, that the pixel movement between consecutive frames is not larger than one. This approach demands cameras that are able to produce this number of frames per second which are very expensive. Another significant problem these cameras face, is the noise that is created due to the demand for high frame rate which results to too few photons been detected[67].

In this thesis we address this problem by using the multi-scale H&S algorithm[3],[4],[115]. Multi-scale algorithms are used to detect larger pixel displacement by creating a pyramid of the images [116],[117],[118]. In each level of the pyramid the mono-scale algorithm is performed and finally the computed velocities in each level are summed for the final output [47]. A question that arises from this approach and which has not yet been answered by the previous multi-scale designs is how many levels should the pyramid be in order to achieve enough motion detection. We address this problem by testing different known sequences of images and we then choose the number of the levels of the pyramid. All the images of the Middlebury data set were tested [54]. The mathematical relation connecting the number of levels of the pyramid which is known as level constraint (λ) and the maximum pixel movement detection is given by Equation 21.

$$\lambda = \log_2(max\_pixel\_motion) \tag{21}$$

In Figure 4 the impact of the number of pyramid levels on AEE is presented on the Groove3 sequence of images. Thus, AAE is used to determine the accuracy. It can be seen that better convergence is achieved with four and three pyramid levels. In this thesis we will stay with the 3 levels pyramid because the value of the AAE for level 3 and 4 pyramids are not very far from each other and we consider the more pyramid levels as future work.

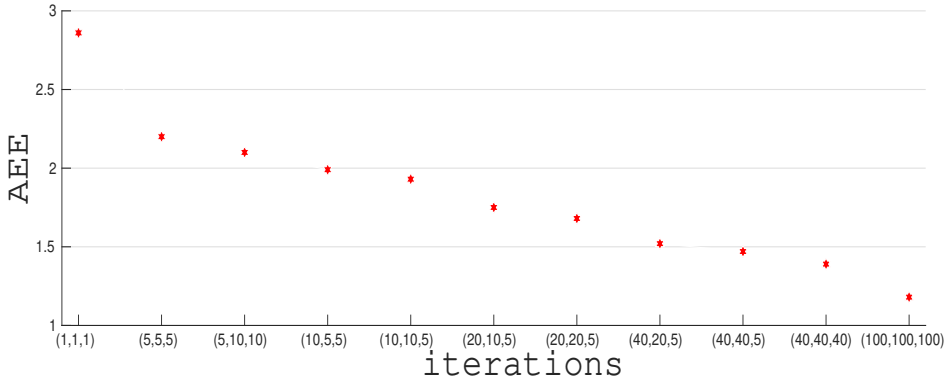

Figure 5: Number of iterations in each pyramid level

Now that the number of the pyramids levels has been decided it is of great importance to determine how many iterations in each level of the pyramid should be performed. In Figure 5 the exploration that has been performed regarding the number of iterations is shown (G3 sequence). It is shown in this graph that performing more iterations in the finest level of the pyramid does not lead to better accuracy. This is expected as in the biggest image, smaller pixel displacement is detected. On the contrary, when more iterations are performed in the smallest image, AAE is reduced and so better accuracy is achieved. The ideal in terms of accuracy would be to perform 100 iterations in each level as shown in the graph but this comes with a huge cost in the throughput and computation as we will see in the next chapter. In this thesis we choose to perform (20,10,5) iterations (20 iterations in pyramid level 2, 10 iterations in pyramid level 1, 5 iterations in pyramid level 0) when throughput is important or (40,20,5) when accuracy is the primary goal. Nevertheless, the accuracy is better in both cases than almost all the state of the art works as we will show later. In the future we plan to remove completely the computation in the finest level of the pyramid and increase the pyramids' levels as the computation in the smallest image takes less time.

3.3 INTERPOLATION

In the multi-scale H&S algorithm a crucial part is the warping[119],[120]. Bi-cubic [121],[122] and bi-linear [123] interpolations are used in this thesis. We perform interpolation in the beginning of each pyramid level. Blachut in his work [43] who is implementing exactly the same algorithm, is performing interpolation in the end of each pyramid level and then he up-samples the results with the aim of creating the new image for the next pyramid level. Thus, for a 2 levels pyramid he is only performing one interpolation in the end of the coarse level. This reduces accuracy and he has to perform pre-processing in the image to achieve good accuracy. This is not the case in our work. We perform the interpolation

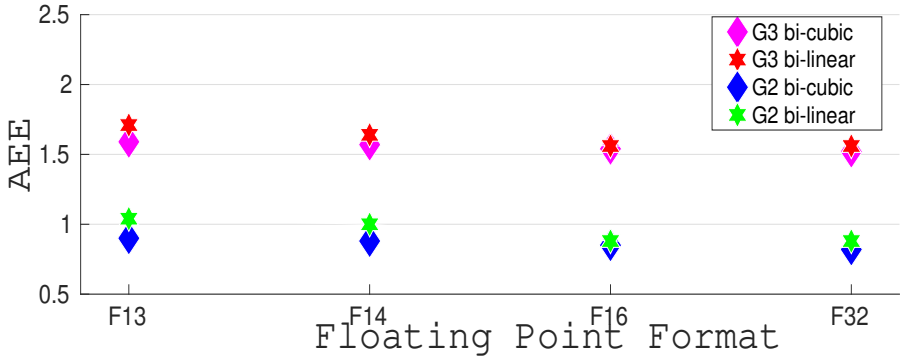

Figure 6: Interpolation's accuracy

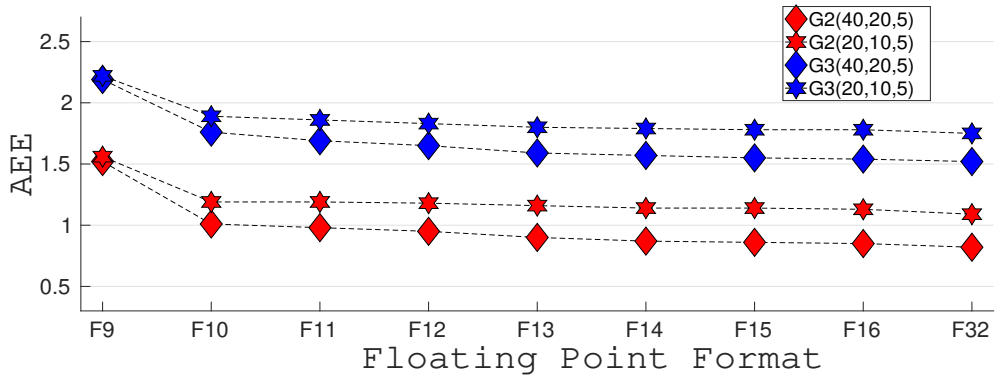

Figure 7: AEE for G3 and G2 with different Floating Point formats

in the beginning of each pyramid level and in this case there is no need for recreating the image.

In Figure 6 the results regarding the bi-linear (dotted line) and the bi-cubic interpolation type (continuous line) are shown against the AEE. The results for the Groove 3 and Groove 2 sequences of images are shown. We include the floating point format used as it is essential for the interpolation type. In the cases of F_{32} IEEE single precision floating point arithmetic format and F_{16} half precision arithmetic format [124] the difference between the two interpolations types is smaller than 0.04. In the case of F_{13} (sign=1, exponent=5, mantissa=7) the difference is up to 0.15. Thus, for small floating point arithmetic in this thesis we are using the bi-cubic interpolation and for bilinear F_{16} and F_{32}.

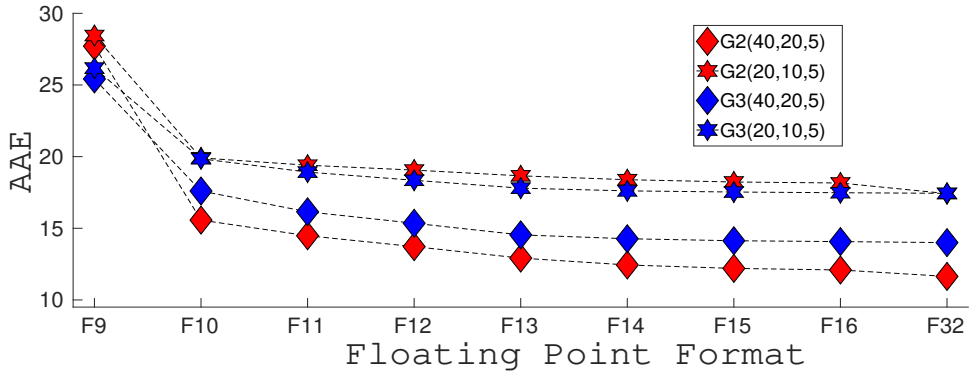

Figure 8: AAE for G3 and G2 with different Floating Point formats

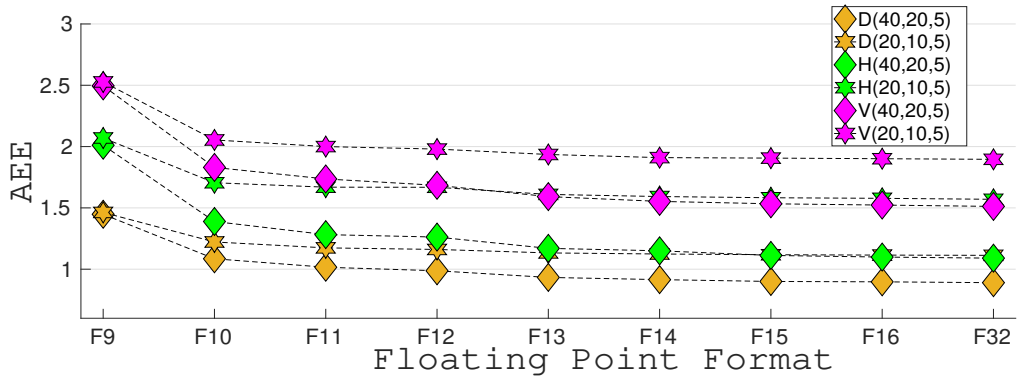

Figure 9: AEE for H, D and V with different Floating Point formats

3.4 FLOATING POINT FORMATS

One of the basic issues the designer has to consider when designing embedded implementations for image processing algorithms is the arithmetic format of the data that are processed because when reducing the word length of the data less hardware resources are used. In this section, an exploration is performed regarding the arithmetic format that will be used in the design. Floating point arithmetic was used for the design of the algorithm. Fixed point arithmetic was also considered but we need more bits and iterations compared to floating point arithmetic as also explained by other state of the art works [67] to achieve convergence and we leave this as potential future work.

Floating-point numbers are usually a subset of rational numbers, with some additional values for handling exceptions (e.g. infinities). A radix β is associated to a floating-point arithmetic, and its finite numbers can be represented as $S.M \cdot \beta^E$ with M (mantisa) and E (exponent) two integers and S the sign bit. The most common radix is β=2 which is used widely in computer engineering and which we will use [125],[126].

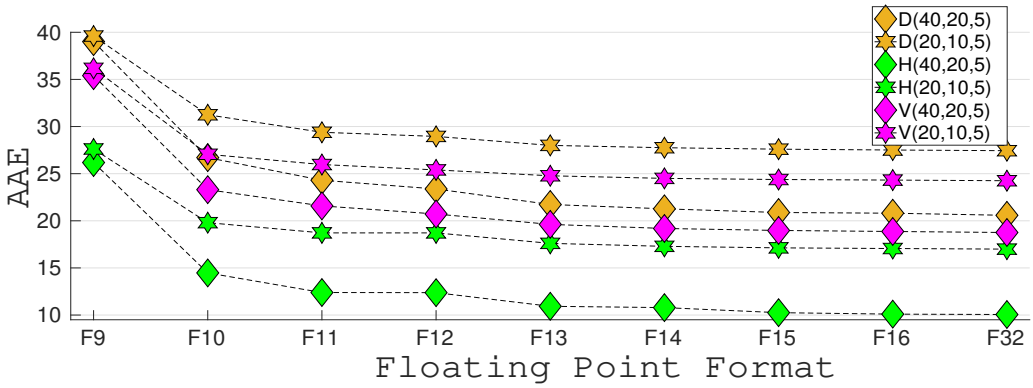

Figure 10: AAE for H, D and V with different Floating Point formats

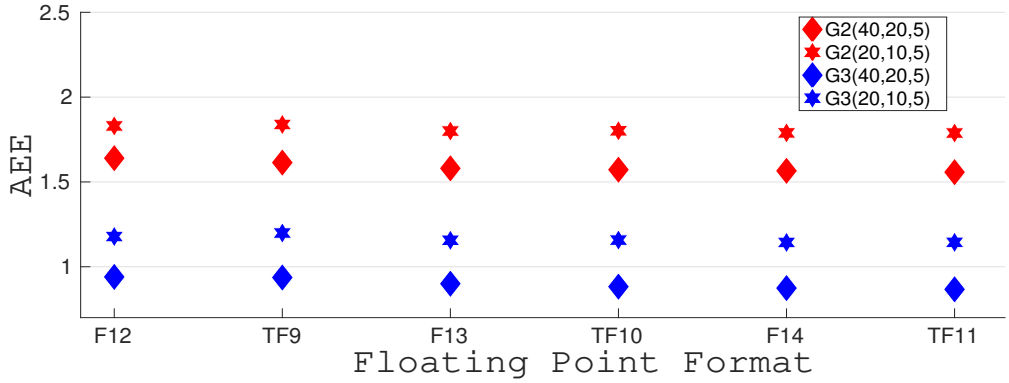

Figure 11: AEE for G3 and G2 with trans Floating Point format

A very important parameter associated with floating point arithmetic is the lowest and largest allowable exponents, e_{min} and e_{max}. In this thesis the largest allowable exponent is essential as the largest possible number that may occur has to be able to be represented. As the input images are 8 bit (unsigned) we select at least 5 bits for the exponent to cover the largest number.

In Figures 7,8,9 and 10 the exploration we performed regarding the floating point formats is shown. The five most used by other state of the art works [62] image data sets from the Middlebury data set are used for the presentation of the results (Groove 3 (G3), Groove 2 (G2), Hydrangea (H), Dimetrodon (D) and Venus (V)). F_{32} is the single precision floating point format. For all the rest floating point numbers, 1 bit is dedicated for the signs, 5 bit for the exponents and the rest bits are devoted for the mantissas. For example F_9 is S=1, E=5, M=3. Rounding to the nearest was also used [127],[128]. In these figures the AEE and AAE metrics for error are used for the evaluation of the result and we used (20,10,5) and (40,20,5) for the levels (λ_2,λ_1,λ_0).

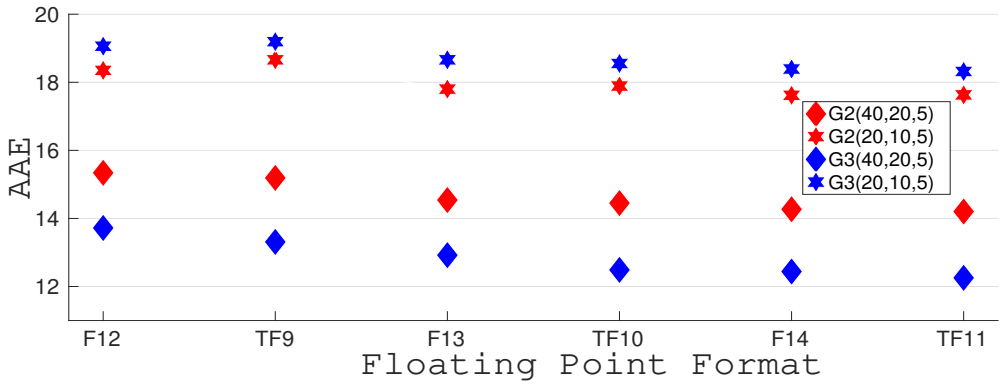

Figure 12: AAE for G3 and G2 with trans Floating Point format

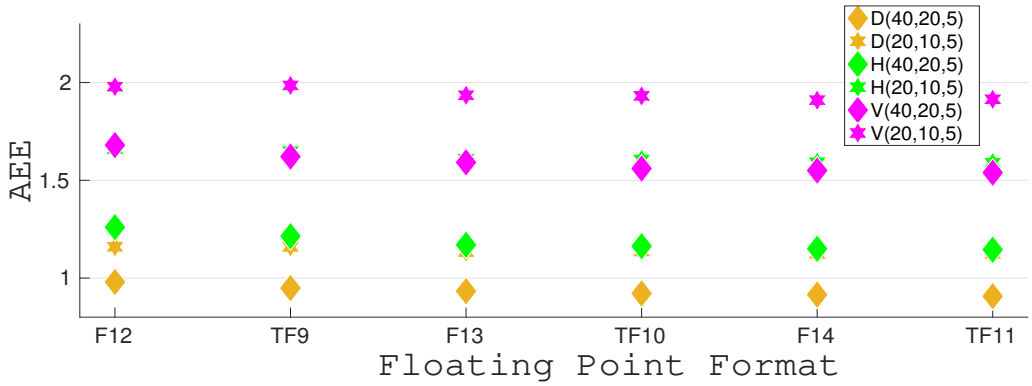

Figure 13: AEE for D, H and V with trans Floating Point format

It is shown that in all cases after F_{14} and for larger representations the error difference is negligible. For F_9 the accuracy is much worse that the other formats but still better compared to other state of the art works. Nevertheless F_{10} provides relatively not so different accuracy compared to larger formats and we will use it in our designs. F_8 was also tested but the accuracy achieved is totally unacceptable. We tried different quantification factors to fit the data in the the F_8 (1.4.3) but none of them worked. It has to be mentioned that all the results were obtained by running the designs on the Arria 10 FPGA with the help of the FloPoCo library [102] and not by simulations. Another point that has to be mentioned is that when more iterations are performed in the coarse level, the accuracy is significant improved as shown in Figures 7,8,9 and 10.

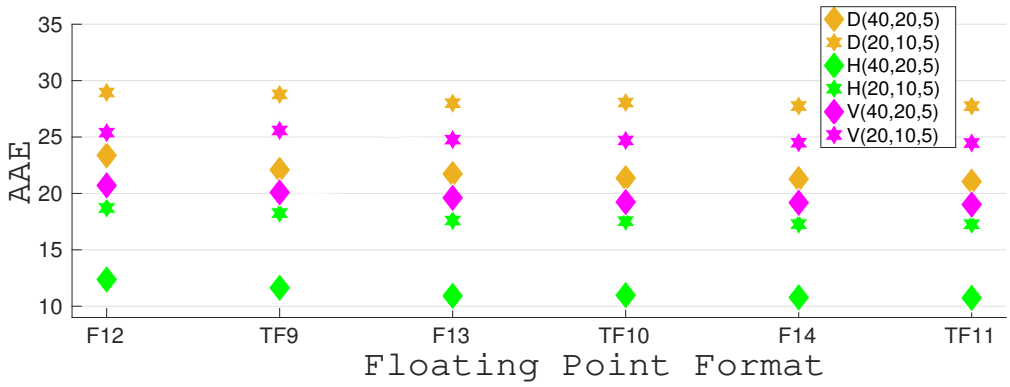

Figure 14: AAE for D, H and V with trans Floating Point format

3.5 TRANS-FLOATING FORMATS

In order to further accelerate the throughput of our algorithm and encouraged by the fact that the accuracy is not dominated by the finest level of the pyramid, we decide to use smaller arithmetic format for the finest level of the pyramid, where the computation time is the longest [64],[55]. Thus, for the levels 2 and 1 we use F_{16} and for level one we use either F_{11}, F_{10} or F_9. The final result is represented with either F_{11}, F_{10} or F_9 because otherwise the impact in the external memory bandwidth would remain the same (if F_{16} was used for the result). When we switch from F_{16} to the other formats in level 0 we truncate the extra bits in the mantissa. We have to mention here that smaller arithmetic formats were tested. However the results were not acceptable at all. One can also try larger arithmetic formats than F_{11} in level 0, however this will only increase the LUTs utilization and decrease vectorization as we will explain in the chapter 5 and as explained in [129]. Finally, one solution we tried is to fit the results of the finest level in F_8 which is very friendly to the external memory bus but the loss in accuracy was very high and we consider this approach as future work.

In Figures 13 and 14 the results concerning the AEE and AAE are shown for TF9 (F_{16}, F_{16}, F_9) , TF10 (F_{16}, F_{16}, F_{10}) and TF11 (F_{16}, F_{16}, F_{11}) and for (40,20,5) and (20,10,5) iterations. It is shown that the TF_9 accuracy is higher than that of F_{12}, TF_{10} has better accuracy than F_{13} and TF_{11} has better accuracy than F_{14}.

3.6 COMPARISON WITH STATE OF THE ART

In this section we will compare our work with all the other state of the art works in terms of accuracy (AEE,AAE). It is very difficult to do a fair comparison with other state of the art works, because not all of them are using the same set of images. Only designs implemented on FPGAs are included. Deep neural network-based methods are not included.

Table 2: Comparison with other State of the Art (AAE)

Implem.	algo.	Format	V	H	G2	G3	AVG
ours(20,10,5)	MH&S bi-lin	F_{16}	24.31	17.05	18.16	17.48	19.25
ours(40,20,5)	MH&S bi-lin	F_{16}	18.86	10.09	12.09	14.07	13.75
ours(20,10,5)	MH&S bi-cub	F_{13}	24.77	17.6	18.66	17.80	19.7
ours(40,20,5)	MH&S bi-cub	F_{13}	19.61	10.92	12.92	14.54	14.49
ours(20,10,5)	MH&S bi-cub	F_{10}	27.05	19.80	19.91	19.85	21.65
ours(40,20,5)	MH&S bi-cub	F_{10}	23.30	14.47	15.57	17.60	17.73
ours(40,20,5)	MH&S bi-cub	F_{16} F_{16} F_{10}	19.23	10.97	12.48	14.45	14.28
[62]	Block	-	6.41	14.80	5.80	10.90	9.47
[43]	L&K	$Q_{1.4.8\rightarrow1.19}$	41.92	34.51	38.22	37.36	38.02
[43]	ML&K	$Q_{1.4.8\rightarrow1.19}$	28.14	18.08	17.81	24.88	22.22
[43]	MH&S	$Q_{1.4.8\rightarrow1.19}$	29.63	18.60	17.90	26.76	23.2
[56]	L&K	$Q_{8.8\rightarrow44.20}$	24.16	19.32	11.51	16.05	17.75
[56]	ML&K	$Q_{8.8\rightarrow44.20}$	16.21	8.28	5.50	10.08	10.01
[95]	H&S	-	-	40.30	-	-	40.30
[95]	H&S	-	-	36.93	-	-	36.93
[96]	H&S	$Q_{8\rightarrow28}$	26.12	25.23	26.88	26.64	26.21
[96]	H&S	$Q_{8\rightarrow28}$	26.88	25.56	27.08	26.89	26.6

This is done because they are not able to compute in real time the optical flow for embedded devices [50],[130],[131] and we only include works that do the evaluation in real time .

For the evaluation we use the images from the Middlebury [54] data set which is the most known data set of images for evaluating optical flow. In tables 3 and 2 the results from the comparison between our work and the rest state of the art papers are shown. For the comparison we used the AAE and AEE error metrics which are the most widely used metrics and the ones that most works use for their evaluation.

In these two tables we can see that our F_{10} (40,20,5) design achieves comparable and in many cases better accuracy than the previous state of the art works which also use larger arithmetic formats. In fact, this design is only outperformed in terms of accuracy by Seyid [62] and Smets works [56]. However, Smets in his work is using a much larger arithmetic format and a 4 levels pyramid which reduces the throughput of the design significantly. Seyid's work achieves impressive accuracy results but his implementation is not scaling for images larger than 640x480 image pixels because too many Blocks RAMs are used

Table 3: Comparison with other State of the Art (AEE)

Implem.	algo.	Format	V	H	G2	G3	AVG
ours(20,10,5)	MH&S bi-lin	F_{16}	1.90	1.57	1.13	1.78	1.59
ours(40,20,5)	MH&S bi-lin	F_{16}	1.53	1.09	0.85	1.54	1.25
ours(20,10,5)	MH&S bi-cub	F_{13}	1.93	1.61	1.15	1.80	1.62
ours(40,20,5)	MH&S bi-cub	F_{13}	1.59	1.17	0.90	1.59	1.31
ours(20,10,5)	MH&S bi-cub	F_{10}	2.05	1.70	1.19	1.89	1.70
ours(40,20,5)	MH&S bi-cub	F_{10}	1.83	1.38	1.01	1.75	1.49
ours(40,20,5)	MH&S bi-cub	F_{16} F_{16} F_{10}	1.16	1.56	1.16	0.88	1.57
[62]	Block	-	0.47	1.98	0.42	0.99	0.97
[43]	L&K	$Q_{1.4.8\rightarrow1.19}$	3.33	2.47	2.22	3.11	2.78
[43]	ML&K	$Q_{1.4.8\rightarrow1.19}$	2.53	1.45	1.19	2.35	1.88
[43]	H&S	$Q_{1.4.8\rightarrow1.19}$	2.97	2.41	1.90	3.02	2.57
[43]	MH&S	$Q_{1.4.8\rightarrow1.19}$	2.31	1.40	1.21	2.38	1.82
[95]	H&S	-	-	2.71	-	-	2.71
[95]	H&S	-	-	2.21	-	-	2.21
[96]	H&S	$Q_{8\rightarrow28}$	0.63	2.34	2.23	1.56	2.53
[96]	H&S	$Q_{8\rightarrow28}$	0.65	2.43	2.34	1.66	2.62

and so the design is not implementable. This is the disadvantage of the blocking match algorithm. Our trans-floating F_{16}, F_{16}, F_{10} design also largely outperforms the previous state of the art works except of the two aforementioned ones. However, as we will show in the chapter 4 the acceleration in the finest level of the image is highly increased.

3.7 CONCLUSIONS

To recap, in this chapter the main objective was to explore the accuracy of the multi-scale H&S algorithm. The first thing that was determined was the number of the levels of the pyramid in our multi-scale algorithm. It was shown that by increasing the number of levels, the AEE error was decreasing and the mono-scale architecture was significantly outperformed in terms of accuracy. Following that, an exploration has been performed regarding the iterations number in each pyramid level. By trying different iterations numbers, we deduced that by performing more iterations in the coarse level, the convergence is better. We also show that it is not necessary to compute many iterations in the finest pyramid level as the accuracy is not significantly improved because only small pixel move-

ment is detected. We tried two interpolation algorithms and we saw the impact they have in the convergence. Bi-linear interpolation is effective for F_{32} and F_{16} while for the rest formats we turned to bi-cubic interpolation. As we are talking for embedded architectures, we tried different floating point formats and we showed that even with F_{10} the accuracy is acceptable. Trans-floating point arithmetic was employed and the results were evaluated to check if using them is beneficial for the design. Finally, a comparison with the state of the art was presented where we saw that our designs achieve comparable accuracy. As future work, we plan to implement a 4 or even 5 levels pyramid and remove completely the computation in level 0 which is the level which requires the most computation time.

In the next chapter, the throughput of our algorithm will be explored. As it is a multi-scale algorithm a lot of tradeoffs have to be considered. The accuracy results will be used to accelerate the architecture but other things will be examined too, such as the depth of the pipeline, vectorization and how to deal with multi-rate architectures.

4 THROUGHPUT

In chapter 3 an analysis regarding the accuracy of the flow detection in the multi-scale H&S algorithm was performed. Another very important factor that has to be taken into account when implementing image processing algorithms for FPGAs is to make them able to perform the computation in real time[132] [7]. This is crucial because computing image processing algorithms in real time is used in domains such as autonomous driving, object detection, security and health applications. Making the implementation of an image processing algorithm able to perform in real time can be very challenging especially when multi-rate image processing algorithms which include huge pipelines and costly computations are considered [133][132][134]. In order to address the bottleneck of speed, a lot of widely used hardware techniques such as parallelisation have to be pondered and combined with new techniques. The multi-scale Horn and Schunck algorithm which is considered in this thesis, is a multi-rate and a multi-level algorithm which makes it a challenging algorithm to accelerate in FPGAs [68]. In this chapter, a lot of techniques will be discussed in order to accelerate effectively every component of the multi-scale Horn and Schunck algorithm. Pipeline and parallelism which are widely used to accelerate algorithms in FPGAs, will be employed for each component as well as smaller floating point formats [100],[129]. Furthermore, efficient architectures are explored in order to achieve pipeline and parallelism between the different components which will lead to the further improvement of the throughput. For the design of the algorithm VHDL is used and also the Intel FPGA SDK for OpenCL [135]. Finally it will be shown that real time speed is accomplished even when images of size 1024×1024 are considered.

4.1 MONOSCALE HORN AND SCHUNCK

As detailed in subsection 2.2.2.2 and in Figure 3, one of the components of the multi-scale H&S algorithm is the mono-scale one. The mono-scale H&S algorithm is an Iterative Stencil Loop algorithm (ISL) and so iterative stencil techniques to accelerate the computation are applied [80][82][75][136][137]. A significant attention was given in ISL algorithms during the last years in high performance computing (HPC) and especially in FPGAs in HPC [85] as they offer high throughput possibilities.

4.1.1 *Pipeline*

In order to attain real time processing in the mono-scale H&S, an output of at least one velocity computation per clock cycle has to be achieved in the mono-scale component. Consequently, pipeline should be employed.

The most classic technique to accelerate an ISL, such as the mono-scale H&S algorithm in FPGAs is the deep pipeline technique [75][81]. This technique is also referred as temporal pipeline or full pipeline computation in other works. In the deep pipeline approach, stencil computation is streamed and pipelined over successive iterations. The proposed architecture for the mono-scale Horn and Schunck core is shown in Figure 15.

Horn and Schunck [45] requires a convolution with a 3×3 mask in order to get the average velocity between the neighbored velocities which were computed in the previous iteration ($u_{i,j}^{t-1}$). The computed value will be fed to the main processing unit (H&S) where the estimation of the velocity for the next iteration is computed ($u_{i,j}^{t}$). As shown in Figure 15 one velocity is read from the external memory. This velocity is propagated through the registers and the shift buffers until it is no more needed for the computation of the next iteration velocity. Since the memory layout of the Horn and Schunck supports consecutive memory reads, the efficiency of the memory controller is 100 % and there are no stalls in the pipeline [82]. This is crucial as the throughput of the cores is almost 1 velocity per clock cycle (there are some stalls which can not be avoided because of the refresh of the memory). The length of the shift buffers is $SB_l = W - 3$, where W is the width of the image.

With the deep pipeline technique multiple iterations can be computed simultaneously when a velocity is read from the memory before written back to the external memory [85]. The number of iterations which are computed, is equal to the number of the H&S cores which are cascaded. If deep pipeline is not applied then, only one iteration can be performed when data are written from the external memory before written back to the external memory. This leads to a significant reduce of throughput, as the goal of the computation of one velocity per clock cycle is diverged. Another point that has to be mentioned is that when a very large number of iterations is required then in most cases it is not possible to create a deep pipeline line to compute all the iterations at once. However, even in this case, the proposed technique accomplishes much better throughput.

With the deep pipeline technique the drawback of the low external memory bandwidth of the FPGAs compared to GPUs and CPUs is overcame. The potential bottleneck of this method might be the increase in the need of Block RAM [66].

Another technique that is considered by other works to accelerate ISL is the scalable computation. The main idea of the pipeline of this method is similar to that of the deep pipeline but the handling of the on chip memory is more complex which requires more hardware resources [138]. However the throughput achieved is similar to that of the deep pipeline [87].

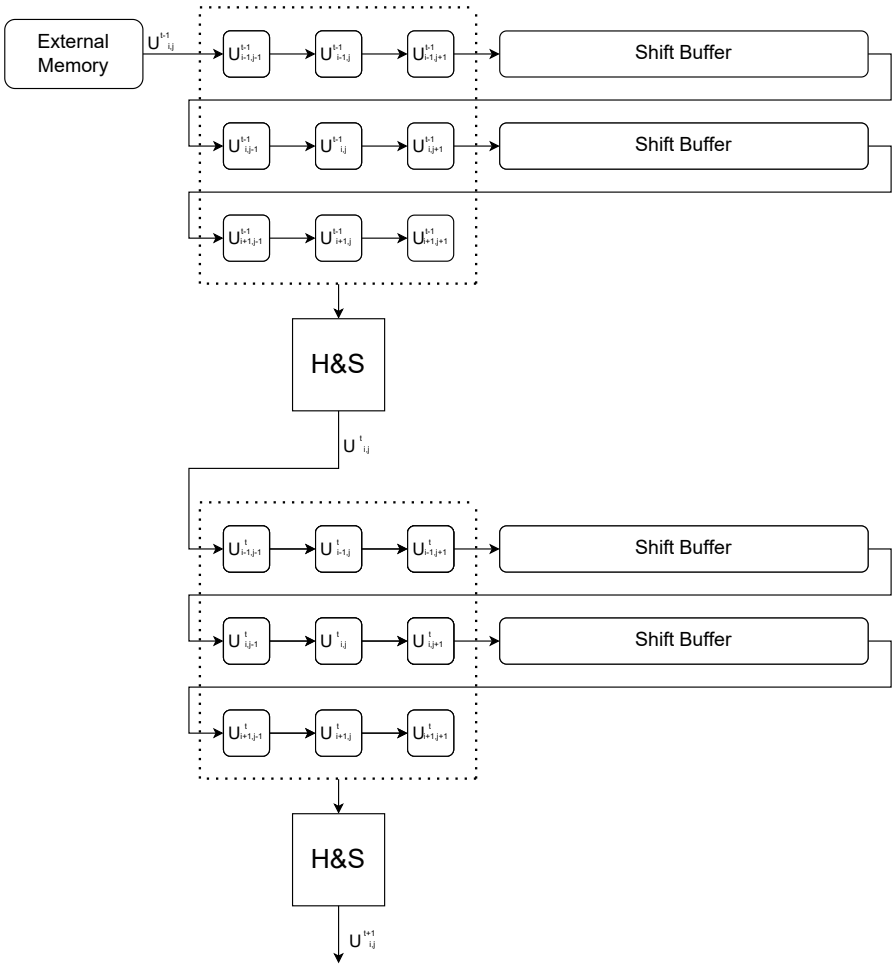

Figure 15: Deep Pipeline Architecture

By taking all these into account, depending on the number of H&S cores Π used and the number of iterations ITER, the total time T for the mono-scale algorithm calculation of an image size with height of H and width of W, can be estimated by (22).

$$T = \frac{H \cdot W \cdot ITER}{f \cdot \Pi} \tag{22}$$

The extra latency added in order to fill the shift buffers is not taken into account as it is negligible (we measured it less than 1%) compared to the total computation time. f is the working frequency of the design.

4.1.2 *Vectorization*

With the deep pipeline technique the external memory bandwidth is not fully utilized as only one pixel is read in every clock cycle but the bus of the external memory allows for simultaneous reads and writes. In order to further accelerate the computation of the mono-scale H&S algorithm, this free margin of the bus has to be taken advantage of [80]. This is done with the vectorization or pixel parallelization of the processing velocities. In such manner the throughput is increased to more than one velocities per clock cycle. The proposed vectorized technique with a throughput of 2 velocities per clock cycle is shown in Figure 16. The Horn and Schunck cores have to have access to $2 \times Q$ velocities in order to compute the mean of the neighbored velocities where Q is the level of vectorization. The length of the shift buffers in this case is $SB_l = \frac{W}{Q} - Q \times 2$. Q is always chosen to be a multiple of power of two as these numbers are better handled by the external memory bandwidth and this is what is suggested by the Intel manual [135]. The number of the parallel H&S cores needed in this case is Q. The number of data needed to be read and written back from and to the external memory is also Q. Finally, the velocities which are read are consecutive (the memory layout remains the same as the velocities are read horizontally and not both vertically and horizontally) which makes the reads and writes burst friendly for the external memory.

The total computation time by taking into account both the vectorization and the deep pipeline approach can be estimated by Equation (23).

$$T = \frac{H \cdot W \cdot ITER}{f \cdot \Pi(Q)} \tag{23}$$

It is shown that the computation time can be reduced drastically by increasing vectorization, as $\Pi(Q)$ ($\Pi(Q)$ is the total number of the H&S cores with a vectorization of Q) is increased. Another major point that has to be stated is that a trade off can be made between the depth of the pipeline and the level of vectorization. This might have a substantial effect when either the external memory bandwidth or the on chip memory is the bottleneck. The same computation time can be accomplished by both increasing the depth of the pipeline and decreasing vectorization or the opposite. If both approaches are combined then the computation time can be decreased remarkably.

4.2 WARPING

The next part of the multi-scale H&S algorithm that is studied is the warping core. Warping, as in the case of the mono-scale H&S requires a neighborhood of pixels in order for the interpolation operation to be computed. However the memory layout of the warping is not standard which means that not consecutive pixels are read [64][60]. The pixels that are read from the external memory depend on the velocities computed in the previous

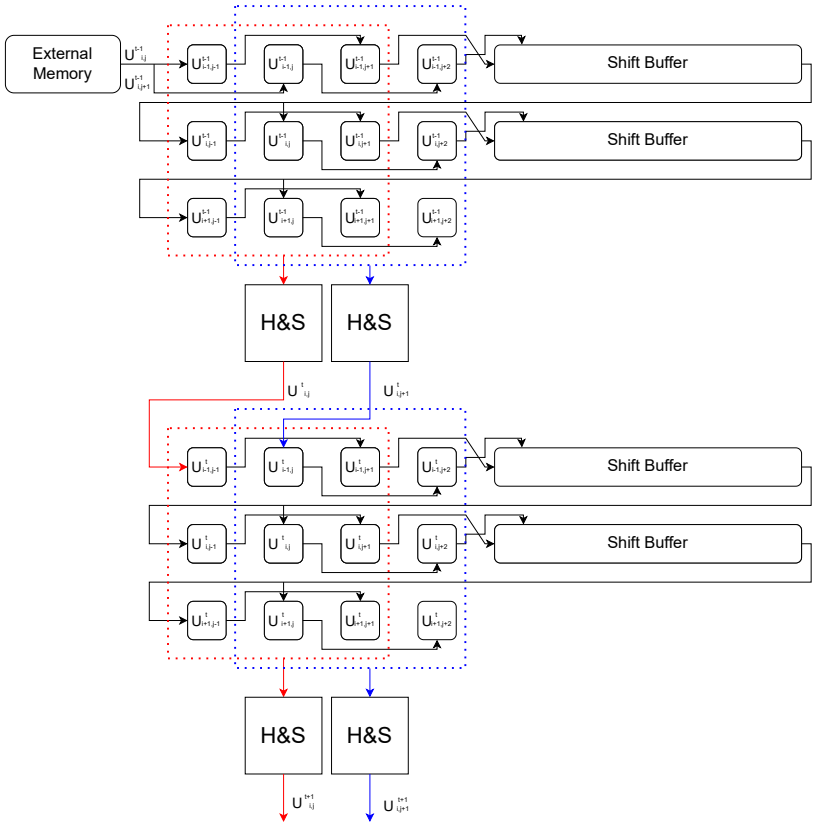

Figure 16: Vectorized Architecture

pyramid level. Consequently, a continuous streaming of computations can not be supported (because of the random reads in the external memory) and therefore a solution has to be proposed in order to fulfill the goal of calculating at least one pixel per clock cycle.

For the warping core two, cases have been explored, the bi-linear and the bi-cubic interpolation [60]. For the bi-linear (resp. bi-cubic) interpolation, a neighbourhood of 2x2 (resp. 4x4) pixels in the input image is required as show in Figure 17. As mentioned before, the goal is to interpolate one pixel per clock cycle, so it has to be ensured that in every clock cycle all the required neighboured pixels are already read from the external memory and available in the on chip memory for the interpolation and at the time that they are needed. Hopefully, the computed velocities in the previous levels have an upper and a lower bound. Consequently, in this thesis we propose a new architecture to achieve the continuous computation of pixels which is show in Figure 17. In order to accomplish that, all the adequate pixels are available the max velocity range has to be taken into account. For example, in the warping of the first level, the velocities computed in the previous levels are in the range of $((V, U) \leqslant |2|)$ where V is the velocity in the y axis and U is the

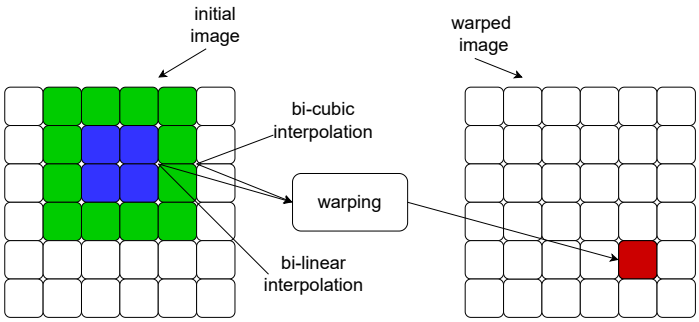

Figure 17: Interpolation Pattern

velocity in the x axis. So the largest velocity range is 2 and -2. This means, that for the bi-linear interpolation 5 lines (-2-0 to +1+2 for the 2x2 neighborhood) and for the bi-cubic 7 lines have to be stored in the Block RAM to ensure consecutive reads from the external memory and consecutive computations.

The worst case scenario occurs when the optical flow vectors summed from levels 2 and 1 have to be interpolated with the input image in level 0 . In this case, the summed optical flow vectors from level 2 and 1 are $(V, U) < |6|$. So a pixel in position (i, j) needs to have access from the pixel $(i-7, j-7)$ to pixel $(i+8, j+8)$ in the bi-cubic interpolation and from $(i-6, j-6)$ to $(i+7, j+7)$ for the bi-linear interpolation. This means that for bi-linear (resp. bi-cubic) interpolation, 14 (resp. 16) lines have to be stored in 14 (resp. 16) shift buffers. In every clock cycle a new pixel is read from the external memory and all the remaining pixels inside the Shift Buffers are moved one position to the right so that all the required pixels for the interpolation are available without latency. In order to choose the right neighbouring pixels in the two cases, the integer parts of the velocity vectors are needed and the interpolation is done with the fraction part of the velocities. For the interpolation 2 multiplexers 16:1 are required one for each dimension. Furthermore, the fraction part of the velocities is required for the interpolation.

As in the case of the mono-scale H&S core and in all the ISL algorithms, vectorization can be applied. In order to process Q pixels per clock cycle, we have to parallelize the computation. Thus, the neighbouring pixels required for the interpolation are from $(-i-7, j-7-Q)$ to $(i+8, j+8+Q)$ for the bi-cubic interpolation and from $(-i-6, j-6-Q)$ to $(i+7, j+7+Q)$ for the bi-linear interpolation. In that case, Q pixels per clock cycle are read from the memory.

In Figure 18 the warping operation is shown for velocities in the range of $((V, U) \leqslant |1|)$. The first multiplexer is used to choose the correct neighborhood in the horizontal axis and its output is a 2×4 matrix. The second multiplexer is used to select the correct neighborhood in the vertical axis. With this architecture a throughput of at least one pixel per clock cycle is guaranteed because the reads are consecutive from the external memory and the anomalies are bypassed by the multiplexers.

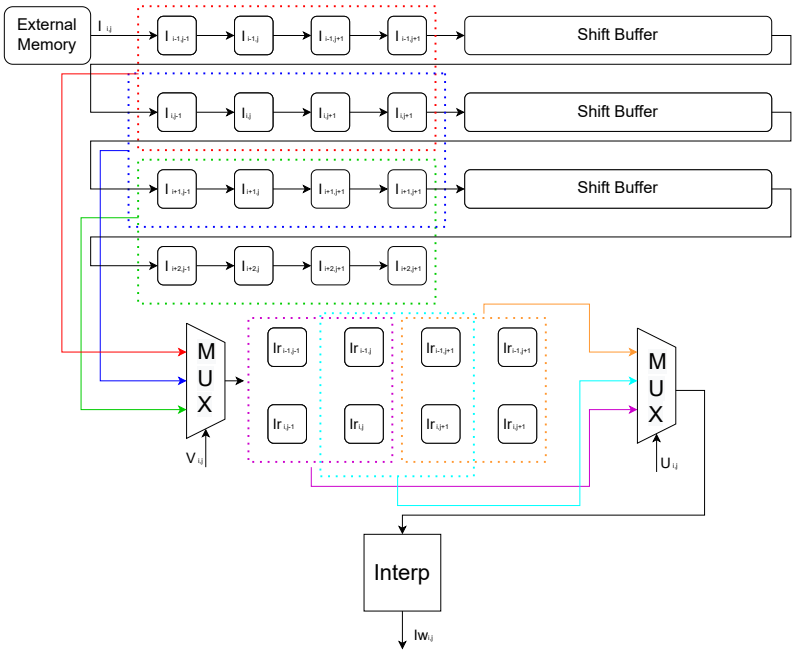

Figure 18: Warping Architecture

4.3 DOWN-SAMPLING

In this section the new architecture proposed in this thesis for the down-sampling component will be discussed. Down-sampling is computed with a convolution of the initial image with a 5×5 Gaussian filter to produce the down-sampled image. In this thesis, the down-sampled image has ×2 smaller width and ×2 smaller height compared to the initial image. However, down-sampling is not a classic stencil algorithm [68]. For every four new pixels that are read from the external memory, there is only one pixel of output. This indicates, that in order to achieve a throughput of 1 pixel per clock then at least 4 pixels have to be read from the external memory.

A very significant issue that has to be pointed out is that when the pixels of the even lines are read then no down-sampling operation can be performed [68][120]. This is due to the fact that down-sampling has a stride of two both in the vertical and in the horizontal direction as it is shown in Figure 19. Thus, in the odd lines of the initial image the processing has to be doubled with the aim of sustaining the throughput of one pixel per clock. This is the technique that is proposed by most of the previous works.

An alternative strategy that is proposed by [43] is to read the horizontal lines two by two in order to overcome the stride of 2. The output image is then stored in the same size of matrices as the inputs (the size of the output image is four times smaller) but in a sparse format in order to be computed without any modifications by the rest cores of the

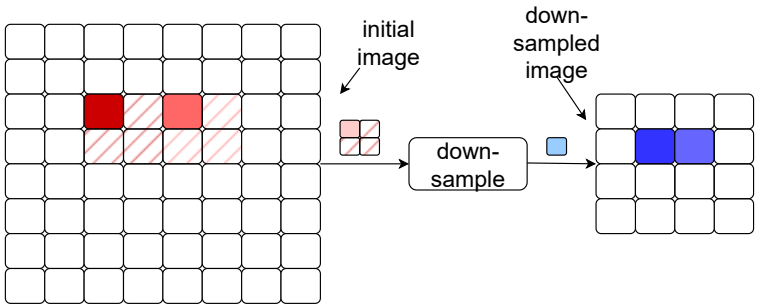

Figure 19: Down-Sampling Pattern

designs of the work. This has a major disadvantage, which is that "empty" pixels are read from the external memory which will not be used, thus overusing the external memory bandwidth.

Algorithm 1 Pseudo code of down-sampling

```
1: for (i=0; i<H; i++) do
2:     for (j=0; j<W; j++) do
3:
4:         if (i−1)%2 == 0 then I^ds_{i/2,j} = C_{0,0} I_{i,j} + C_{0,1} I_{i,j+1} + C_{1,0} I_{i+1,j} + C_{1,1} I_{i+1,j+1}
5:     end for
6: end for
```

Algorithm 1 shows the pseudo code of a simple down-sampling core with a 2×2 convolution for a 1024×1024 image. I^{ds} is the down-sampled image, I is the initial image, C are the weights of the convolution and (i,j) are the coordinates for the vertical and horizontal dimensions. As stated before, it is obvious from line 4 that the operators perform the computation only in the odd lines of the image as they wait for the adequate data (the next horizontal image line) to fill the shift buffers. Thus the operators remain unused half the time of the computation.

It is of great significance to mention that if the pipeline of the down-sampling feeds another operation, then the next operation will get new data as input only half of the total time. This significantly reduces the throughput unless the hardware for the operations is doubled in order to have an output of at least one pixel per clock as mentioned before.

By taking all the previous into account in this thesis a new architecture for the down-sampling is proposed which overcomes this problem. The new architecture is presented in Figure 20. In this figure, the red arrows denote the pixel that are written in a 1X4 format to the Block RAM and which are read from the external memory (pixel). With the decoder and the choose signal (ch) the pixels are written either in the one Block RAMs or to the other. If the pixels are written to the first Block RAM, then the pixels that are needed for the down-sampling are read from the second Block RAM. The blue arrows

Algorithm 2 Pseudo code of up-sampling

```
for (i=0; i<H; i++) do
    for (j=0; j<W; j++) do
```

$U^n_{2i,2j} = f(U_{i,j}, U_{i,j+1}, U_{i+1,j}, U_{i+1,j+1})$
$U^n_{2i,2j+1} = f(U_{i,j}, U_{i,j+1}, U_{i+1,j}, U_{i+1,j+1})$
$U^n_{2i+1,2j} = f(U_{i,j}, U_{i,j+1}, U_{i+1,j}, U_{i+1,j+1})$
$U^n_{2i+1,2j+1} = f(U_{i,j}, U_{i,j+1}, U_{i+1,j}, U_{i+1,j+1})$

```
    end for
end for
```

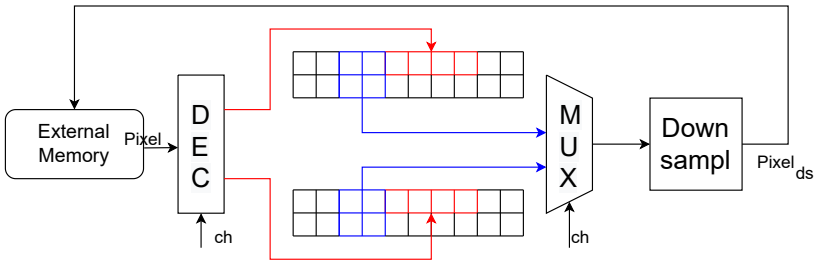

Figure 20: Down-sampling

denote the pixels in a 2x2 format that are read and that are fed to the down-sampling core. With this architecture the memory reads from the external memory are consecutive as the two Block RAMs read only neighborhood pixels from it. The final down-sampled pixel ($Pixel_{ds}$) is written back in the memory.

With the proposed architecture the aim of a steady output of at least one pixel per clock is attained. In this manner there is no reason for replicating the processing units to achieve the same throughput as the obstacle of the even lines is bypassed.

Another solution that was examined is to read from the external memory $2\times2\times Q$ pixels in a similar way as it was done in [43] . The first 2 corresponds to stride in the vertical dimension, the second 2 corresponds to the stride of 2 in the horizontal dimension and Q to the vectorization. Although this technique overcomes the need for replicating the processing units to maintain a steady output of one pixel per clock, it seriously reduces the throughput of the external memory as non consecutive pixels are read.

In the multi-scale H&S algorithm there is no component which is fed from the down-sampling core. However in many image processing algorithms this core is the forefront unit or a significant part of a large pipeline [132][133] and that is why it is believed that this architecture can lead to the saving of significant hardware.

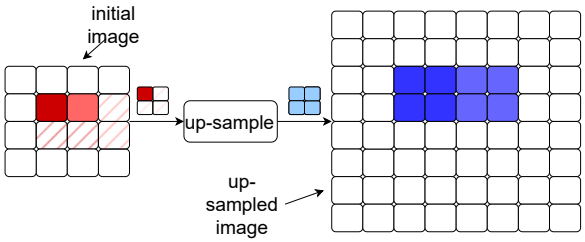

Figure 21: Up-Sampling Pattern

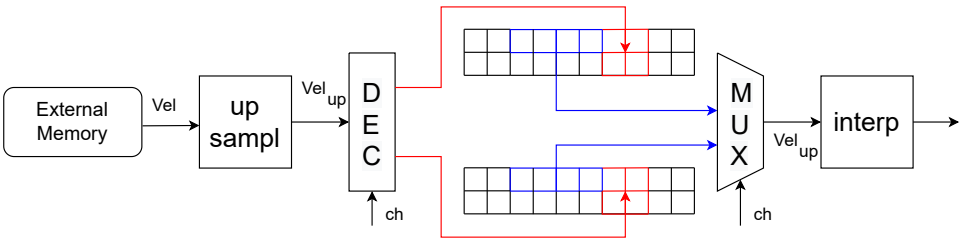

Figure 22: Up-sampling

4.4 UP-SAMPLING

Up-sampling is the first component in the pipeline in each level of the pyramid. Hence, it is of great importance to design this component without adding additional stalls to the total computation chain. In the up-sampling, every new up-sampled velocity is computed by interpolating a 2×2 neighbourhood of velocities in the initial grid of velocities as it is shown in Figure 21(the initial grid is $\times2$ smaller in both the height and width than the up-sampled grid). This means that for every new velocity read, four velocities are produced [68]. Algorithm 2 describes a simple up-sampling algorithm. $U^n_{2i,2j}$ are the up-sampled velocities and f is the function of the initial grid velocities($U_{i,j}$) to compute the up-sampling.

As far as we know, in all the previous implementations, for every new velocity input, four outputs are produced in a 2×2 matrix format. Consequently, the components that follow the up-sampling have to perform their computations with an input of 2×2 matrix formats (similar to down-sampling). This causes stalls and degrades the performance of the external memory bandwidth usage when the following components need to access the memory in 2×2 matrix formats. This happens because no consecutive reads are done to the external memory. Thus, it affects considerably the throughput and an alternative approach has to be enacted in order to fulfill the goal of computing at least one pixel per clock.

In the multi-scale H&S algorithm up-sampling is the leading the interpolation core [2]. Therefore, the interpolation cores needs to access non neighbourhood pixels in the memory. This situation makes it even more imperative to propose an alternative imple-

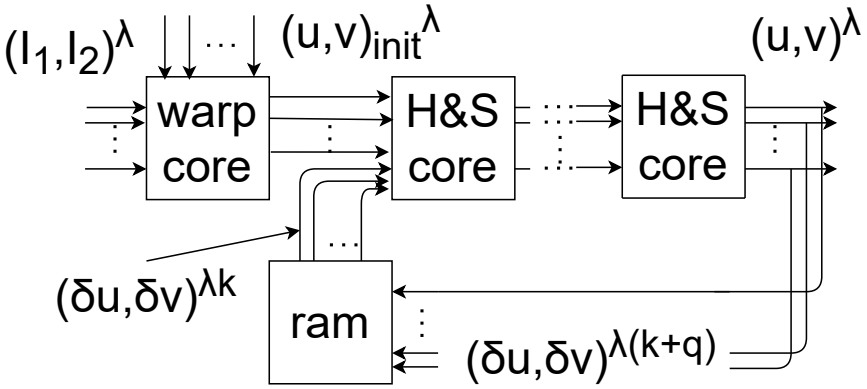

Figure 23: Partial pipeline parallel architecture

mentation for the up-sampling compared to all the previous state of the art architectures in order to overcome this issue and increase the throughput.

In Figure 22 the new architecture of the up-sampling component is presented. The red arrows denote the up-sampled velocities which are written in the same way as in the case of down-sampling in the Block RAMs. For every new velocity that the up-sampling core consumes its output is a 2x2 velocity matrix. With the decoder, this matrix is either written to the first or to the second Block RAMs. However, the components that are lead from the up-sampling core need an input of 1Xn and not a 2xn input to function without stalls. The blue arrows show how the up-sampled velocities are forwarded to the interpolation unit (next steps of the algorithm) and how the proposed architecture shown in Figure 22 serves the goal of an output of 1xn velocities.

To sum up this subsection, with the proposed implementation, the next component in the pipeline which follows the up-sample component gets as an input an $1\times Q$ matrix rather than a $2\times \frac{Q}{2}$. This is very effective for the interaction with the external memory bandwidth as fully consecutive reads are made and a throughput of one or more velocities per clock can be computed without stalls [85].

4.5 MULTI-SCALE H&S

The aim of this section is to efficiently pipeline and parallelize all the components in each pyramid level in order to achieve a throughput of one or more velocities per clock cylce increasing in this way the throughput.

4.5.1 *Pipeline and Parallelization*

The disadvantage of the FPGAs compared to CPUs and GPUs is the reduced external memory bandwidth [82]. However, the Block RAM is restricting this drawback as more

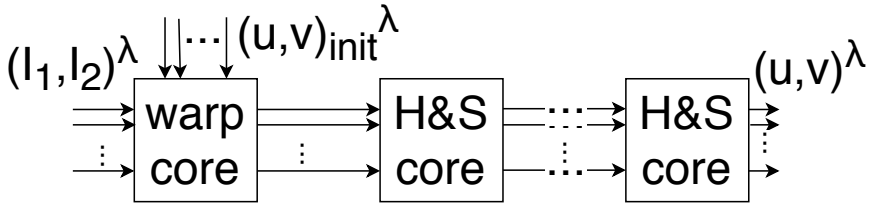

Figure 24: Fully pipeline architecture

data can be kept in the FPGA chip. Accordingly, to increase the throughput, the interaction with the external memory should be lessened and the data should be kept in the on-chip memory until they are no more needed [60][43]. This way the bottleneck of the external memory bandwidth can be bypassed and the computations can be fused between the different component of the algorithm.

In sections 4.1 and 4.2 the architectures proposed in this thesis for the warping and the mono-scale H&S architectures are able to compute one or more outputs per clock and therefore these components can be pipelined. This has a major advantage: interpolated pixels do not need to be written back to the external memory and then read again, but they can be directly processed by the H&S core in a pipeline way.

In Figure 23 and in Figure 24 the two new architectures proposed in this thesis for fusing the computation between the warping and the mono-scale H&S are presented. In the case where the number of the H&S cores is fewer than the number of iterations in this specific level of the pyramid, the Partial Pipeline Parallel (PPP) architecture is employed (Figure 23). If the number of the H&S cores is Π and the number of iterations is ITER, then in each read from the external memory Π iterations are fused and computed. When ITER=Π then the Fully Pipeline Parallel (FPP) architecture is designed (Figure 24). In this case, with one read of the data from the external memory all the iterations are performed. Both designs can be vectorized with the aim of further reducing the computation time.

4.5.2 *Multi-rate Architecture*

The leading component in each level of the pyramid is the up-sampling core. Up-sampling makes the multi-scale H&S algorithm a multi-rate algorithm. Multi-rate algorithms are the kind of algorithms where some stages of the algorithm produce matrices of outputs which are of different size from their inputs matrices [133]. This is very common in many image processing algorithms. The up-sampling and the down-sampling cores are such cases and it is very frequent to find them in an image processing pipeline. Up-sampling is getting a new velocity as an input and it up-scales by 2 in both dimensions producing a square as an output. Pipelining multi-rate architectures is challenging because these anomalies of inputs and outputs have to be addressed [68].

In Sec. 4.4 the new proposed architecture of this thesis for the up-sampling core was presented. With the aforementioned design the square output of the up-sampling is transformed to a $1\times$ $2\times$Q output. Thus, this output can be fused directly to the input of the warping core and the read from the external memory remain completely consecutive. As a result, stalls because of irregular memory reads are not created in the pipeline.

In all the previous works the multi-rate algorithms are treated either with sparse matrices or with non consecutive reads from the external memory. For example in the work of [43][68], in order to reuse the same resources in each level of the pyramid the storage in the external memory of the down-sampled images is done as all the down-sampled images sizes is the same with the input images. In this way pixels which are "empty" are processed as normal pixels. However this reduces the computation time dramatically [43].

4.5.3 *Multi-level Architecture*

In the multi-scale Horn and Schunck algorithm more iterations are performed in the coarse level (small images) and less iterations in the fine level (large images) as in this way the convergence is better as it was explained in Chapter 3. The iteration factor between the levels of the pyramid in this thesis is $\times 2$ and $\times 4$. The architectures used in all the levels, except the first level where the fully pipeline parallel is used, is the partial parallel pipeline architecture. If the fully pipeline parallel design is used in all the levels, then unnecessary iterations are performed in the finest pyramid level (increasing the number of iterations in this level does not increase accuracy). This happens because, if in level 2, 20 iterations are performed then in level 0 20 iterations will be performed. However, 20 iterations in level 0 are useless, and the Horn and Schunck cores in this level should be parallelized with the aim of computing the iterations in level 0 faster.

In this work the same vectorization factor is used in all the levels of the pyramid. With the proposed architectures of all the cores in this thesis dense inputs are created and the stride of computation is always 1 which enables the same vectorization in all the levels. Contrary to this thesis, in the work of [43], the vectorization factor changes between the levels, as empty pixels are processed because a stride larger than one is considered (2 for level 1 and 4 for level 0). Hence, no matter that their processing units compute 4 pixels per clock, the actual computation per clock in level 1 is 2 and in level 2 is one. In this thesis the only thing that changes between the levels is the depth of the cyclic buffers which is adapted to the size of the images.

It is obvious from the above that the computation time required in the finest levels is longer, as the sizes of the images are larger and the vectorization the same.

4.5.4 *Computation Time*

By taking all the previous into account, the total computation time in order to compute the whole pyramid is given by Equation (24). Height and width are the sizes of the image and t_0, t_1 and t_2 are the number of iterations in level 0, 1 and 2 respectively. lat is the time needed to fill the shifting buffers of the pipeline but this time is negligible compared to the total time and can be ignored.

$$T = \frac{H \cdot W \cdot (t_0 + \frac{t_1}{4} + \frac{t_2}{16})}{f \cdot \Pi(Q)} + lat \cdot \frac{1}{f} \tag{24}$$

4.5.5 *Trans-floating architecture*

As mentioned in the subsection 4.5.2 the finest level of the pyramid is the one which demands the most computation time. Nevertheless, this level is the one where the smallest displacement of the pixel is detected. In this manner, methods which would accelerate this level of the pyramid where considered in this thesis. The main bottleneck of the acceleration of this part is the external memory bandwidth which does not allow for the vectorization of the processing beyond 4 for the Arria 10 FPGA if single precision arithmetic is considered. Hence, the number of the H&S cores which are used in this level is the same as in level 1 and 2 where the computation time is lower.

Therefore, smallest arithmetic formats were considered in order to overcome the bottleneck of external memory bandwidth [139], increase vectorization and as a result the number of the H&S cores which is the ultimate goal. The last 5 years this technique has already been applied to deep neural networks [140][51][141]. The formats that were considered are the half precision floating point format (F_{16}) [142], F_{15}, F_{14}, F_{13}, F_{12}, F_{11} , F_{10} , F_9 and F_8 (their impact in accuracy was evaluated in chapter 3). All these formats enable at least the doubling of the vectorization and especially the F_9 enables to increase three times the vectorization as the F_{32} demands 32 bits for one read or write while the F_9 demands 27 bits for 3.

In Figure 25 the architecture proposed in this thesis to support a smaller arithmetic format in level 0 is shown. In this figure, in level 1 F_{16} computation is used while in level 0 the F_{10} arithmetic format. The vectorization of level one is 1 while in level 0 is 4. The up-sampling core is designed as proposed in section 4.4 which enables external memory reads of consecutive addresses which leads to increased throughput. Moreover, the computed velocities in level 1 are not written back to the external memory but processed immediately by the cores of level 0. Consequently, level 0 and level 1 are computed at the same time and as a result the computation time is further improved. In level 2 the same cores which are used in level 1 are utilized.

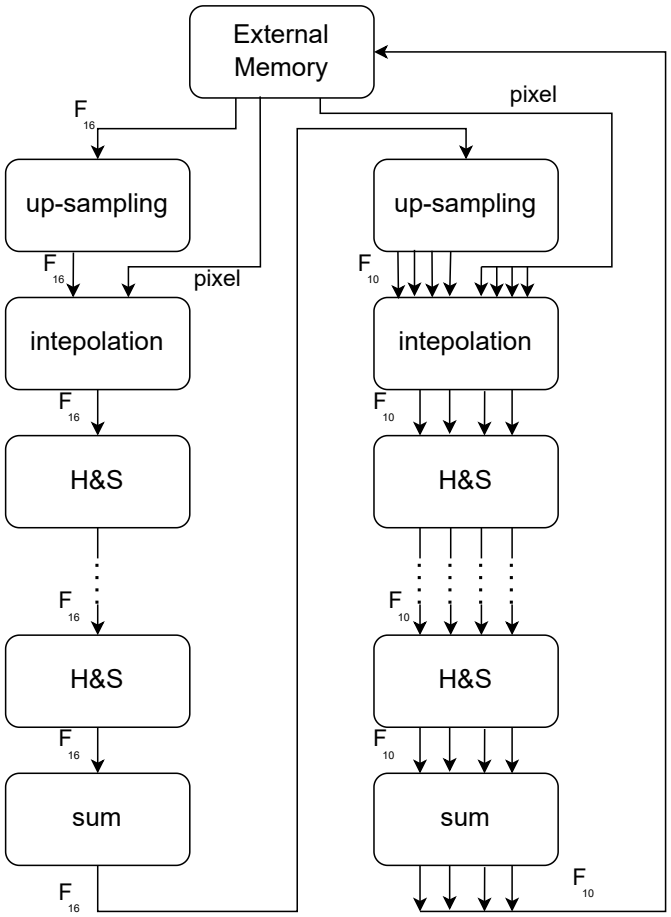

Figure 25: Trans-floating architecture

A more efficient design in terms of hardware resources would be to use MAC units which would be able to compute the velocities in more than one arithmetic formats. For example the ideal would be to have a MAC which would be able to compute 1 MAC in F_{16} in level 1 and 2 MACs in F_{10} in level 0 as it was done by [143] for F_{32} and F_{16}. Nevertheless, the MAC units used in this thesis are the standard MAC single precision floating point units provided by intel and the floating point ones which enable the computation of smaller formats provided by the open source Flopoco library [102]. This means that the H&S cores used in level 2 and level 1 can not be reused in level 0 as they do not support trans-floating computations. Making the MAC units able to compute trans-floating numbers is crucial as the same MAC units will be used in all the levels, keeping the throughput in the same level but decreasing at the same time the resources used.

4.5.6 *Computation Time for the Trans-floating architecture*

Equation (24) is transformed to Equation (25) as now the H&S cores which compute level 0 (Π_0) have to be taken into account. Moreover, level 1 is computed at the same time as level 0. It is clear from this equation that since the number of $\Pi_0 > \Pi_{1,2}$ then the computation time is decreased drastically compared to the proposed architecture where not trans-floating point formats are used. Level 0 of the pyramid is no more the dominant part by far which requires the most computation time.

$$T = \frac{H \cdot W \cdot (\frac{t_2}{16})}{f \cdot \Pi_{2,1}(Q_{2,1})} + \frac{H \cdot W \cdot t_0}{f \cdot \Pi_0(Q_0)} + lat \cdot \frac{1}{f} \tag{25}$$

4.6 THROUGHPUT RESULTS AND COMPARISON WITH STATE OF THE ART

In the previous sections all the designs that were followed of the different cores of the multi-scale H&S algorithm were presented. The next step in this section, is to present the results of the implementations and to compare with the other state of the art designs in order to see if the proposed designs are efficient in terms of throughput.

4.6.1 *Results of implementation*

For the implementation of the algorithm, the FPGA Intel Arria 10 Han Pilot platform was used and the intel Opencl 19.1. The Arria 10 is equipped with 1518 DSPs and a peak external memory bandwidth rate of 17 GB/sec. Smaller precision floating point numeric formats were built with the help of the FloPoCo library [102] and were inserted as libraries in the Opencl software as custom RTL designs. Detailed instructions in order to add custom RTL in the Opencl software is provided in the manual [135]. 2 iteration factors are considered, the $\times 2$ and $\times 4$. With the $\times 2$, 20 iterations are performed in level 2, 10 iterations in level 1 and 5 iterations in level 0. With the $\times 4$, 40 iterations are performed in level 2, 20 iterations in level 1 and 5 iterations in level 0.

In Table 4 the information about the throughput of the multi-scale H&S algorithm for an image size of 1024x1024 pixels when the same arithmetic format is used in all the pyramid levels is shown (FPS is the frames per second, $\times 2$ is for (20,10,5) iterations and $\times 4$ is for (40,20,5) iterations. When single precision numeric format (F_{32}) is considered the maximum vectorization that can be applied is (Q=2) as the external memory bandwidth of the Han Pilot Platform is saturated. With the half precision format (F_{16}) vectorization is increased to 4 and with F_{10} to 8. From Table 4 it is obvious that by increasing the vectorization, throughput is increased sub-linearly. The ideal increase would be linear. However, as resource usage is increased the working frequency of the clock is decreased. Furthermore, by increasing vectorization, this does not mean that the maximum theoretical bandwidth

Table 4: Throughput Results with the Same Numeric Format

Format	iter	#H&S Cores	Throughput (Mpixel/s)	FPS
F_{32}	×2	5	130	124
F_{32}	×2	10	247	216
F_{32}	×4	5	89	85
F_{32}	×4	10	172	164
F_{16}	×2	10	270	257
F_{16}	×2	20	449	428
F_{16}	×4	10	165	157
F_{16}	×4	20	334	319
F_{10}	×2	10	235	230
F_{10}	×2	40	731	698
F_{10}	×4	10	172	164
F_{10}	×4	40	553	528

Table 5: Throughput Results with the Smaller Numeric Format in level 0

Format	iter	#H&S Cores	Throughput (Mpixel/s)	FPS
F_{32} F_{32} F_{16}	×2	$10_{2,1}$ 20_0	363	346
F_{32} F_{32} F_{16}	×4	$10_{2,1}$ 20_0	226	215
F_{16} F_{16} F_{10}	×2	$20_{2,1}$ 40_0	580	553
F_{16} F_{16} F_{10}	×4	$20_{2,1}$ 40_0	422	402

of the external memory given by the manufacturer of the device will be reached as the efficiency of the memory controller is dropping as it is thoroughly explained in [144][145].

In Table 5 the information about the throughput of the multi-scale H&S algorithm for an image size of 1024x1024 pixels when a smaller arithmetic format is applied in the finest pyramid levels is shown. F_{32} F_{32} F_{16} means that in level 2 and 1 F_{32} is used, while in level 0 F_{16}. $10_{2,1}$ denotes that in level 2 and 1 the same 10 cores are used while 20_0 means that in level 0, 20 cores are used. It is shown that the algorithm is further accelerated compared to the case were the same format is used in all the pyramid levels. Nevertheless this comes with a cost in accuracy which was explained thoroughly in chapter 3. In the case of F_{32} F_{32} F_{16} a vectorization of 4 is achieved in level 0 while with F_{16} F_{16} F_{10} a vectorization of 8 in level 0 as explained in section 4.5.5.

4.6.2 *Comparison with State of the Art*

In Table 6 we make a comparison in terms of throughput of the works presented in this thesis with all the state of the art optical flow algorithms implemented in FPGA during the last 15 years. In this table information are given about the algorithm used, the arithmetic format, the throughput and the frames per second achieved. It can be seen from this table that the designs proposed in this thesis achieve comparable throughput and in most cases outperform in terms of throughput all the previous state of the art optical flow algorithms.

More precisely, the $\times 4$ designs (v_1,v_2,v_3 and v_4) achieve real time speed for an image size of 1024X1024. The fastest trans-floating $\times 4$ design in terms of throughput is only slower to Blachut's [43] and Ishiis [61] designs. However, Blanchut's implementation requires a very large usage of resources and the scales of the pyramid is only restricted to two. The number of iterations implemented in level 0 is 5, while in level 1 is 10. Ishii on the other side is not performing the whole computation in a single FPGA but in two and a PC. V_5 is the fastest $\times 2$ design proposed in this thesis and is only second to Ishii's implementation.

It can be seen by Table 4 and Table 5 that with trans-floating and smaller floating point formats than single precision floating point arithmetic, the throughput can be significantly boosted as more words can be read and written from and to the external memory. As also shown in Table 6 these smaller floating point formats lead to very fast designs compared to the previous state of the art implementations, even compared with mono-scale designs as less iterations are performed in the finest pyramid level which demands the most computation time or designs with a fewer number of scales. Smaller floating point formats achieve similar or better throughput compared to designs which use fixed point numeric with more bits and variable word length inside the computation.

4.7 CONCLUSION

To conclude this chapter, the thorough explanation of the design of every component of the multi-scale Horn and Schunck algorithm was done. The primary goal of each design was to achieve one or more pixel computations per clock cycle for every component with the aim of boosting throughput. Moreover, external memory bandwidth was taken into account as it poses the main obstacle in order to accelerate the algorithm. The next step was the connection of the different components in a very large pipeline in each pyramid level. The classic deep pipeline technique which is used in accelerating iterative stencil algorithms was used and the computation was vectorized to saturate the external memory bandwidth. Solutions were provided to overcome the issue of multi-rate architectures and a technique to deal with the irregular reads of the external memory was proposed. Smaller arithmetic floating point formats were also considered. As the finest pyramid level is the one which demands the most computation time, smaller arithmetic formats were used

Table 6: Comparison with other State of the Art

Implem.	algo.	Format	Thr. (Mpixel/s)	FPS
This work v_1	MH&S ($\times$4)	F_{16}	165	157
This work v_2	MH&S ($\times$4)	F_{10}	553	528
This work v_3	MH&S ($\times$4)	F_{32} F_{32} F_{16}	226	215
This work v_4	MH&S ($\times$4)	F_{16} F_{16} F_{10}	422	402
This work v_5	MH&S ($\times$2)	F_{16} F_{16} F_{10}	580	553
Bahar [146]	H&S	-	79	1029
Komor. [67]	H&S	$Q_{1.4.8\rightarrow1.19}$	174	84
Kunz [66]	H&S	Q	283	30
Johnson [96]	H&S	$Q_{8\rightarrow28}$	418	200
Blachut [43]	MH&S($\times$2)	$Q_{1.4.8\rightarrow1.19}$	498	60
Diaz [97]	L&K	-	82	170
Seong [59]	L&K	$Q_{4.6\rightarrow26}$	94	196
Ishii [61]	L&K	$Q_{10\rightarrow32}$	1048	1000
Barranco [60]	ML&K	$Q_{9.0\rightarrow29.8}$	9.8	32
Blachut [147]	ML&K	-	46	50
Blachut [43]	ML&K	$Q_{1.4.8\rightarrow1.19}$	498	60
Tommasi [63]	MPB	$Q_{8.0\rightarrow8.4}$	10	32
Seyid [62]	HBM	$Q_{10\rightarrow32}$	12	39

in this level in a trans-floating architecture. Comparison with all the other state of the art optical flow designs in FPGAs was done and it was shown that the fastest design proposed in this thesis achieves the fastest throughput compared to all the previous single FPGA implementations. Furthermore, all the designs are able to compute an image of size 1024x1024 pixels in real time. However, some applications such as comets detection and image processing in health application request speeds higher than real time processing. The designs of this thesis are able to provide this throughput.

In the next chapter, the impact of all these designs in the resource usage will be discussed. The usage of DSPs, Block RAMs and LUTs will be analyzed along with need of external memory bandwidth . The challenges that were faced during the design of the different components will be presented and how all these obstacles were overcame.

5 HARDWARE RESOURCES UTILIZATION

In the previous chapter, a throughput analysis of the multi-scale Horn and Schunck algorithm was performed. However, when designing an algorithm in FPGAs increasing or decreasing the throughput has a major effect in the hardware resources usage[67]. Hardware resources in FPGAs are not unlimited and subsequently they have to be taken seriously into account when designing an algorithm. Increasing the throughput drastically might potentially lead to designs that may not be able to fit to a specific FPGA device or take more space than intended compared to other accelerators that form the whole system. In this chapter an analysis will be made considering the hardware resources usage by the multi-scale H&S algorithm. DSPs, LUTs and Block RAMs utilization will be discussed. Interaction with the external memory bandwidth is considered in this thesis as hardware resources as it is one of the main bottlenecks when trying to accelerate the multi-scale H&S algorithm. Solutions will be provided for the effective usage of all these resources in the FPGA considering the currently studied algorithm. It will be discussed how pipeline and parallelism in each component affect the resources usage. As different arithmetic formats are used, their impact will also be pointed out in terms of DSPs, LUTs and Block RAMs. Moreover, it will be shown how larger sized images have a different impact in the Block RAMs utilization. A comparison will be made with all the recent FPGA state of the art works in terms of resources. Finally it will be shown that even with a limited usage of the whole FPGA area, real-time computation can be achieved for an image of size 1024X1024 pixels. For the design of the components with different floating point numbers, the open source FloPoCo library was used [102].

5.1 MONO-SCALE HORN AND SCHUNCK AGORITHM

The first algorithm that will be discussed in terms of resources is the mono-scale H&S algorithm [45]. This algorithm is an iterative stencil loop algorithm[69][70]. In this algorithm, first a convolution is performed between the neighboring velocities computed in the previous iterations as shown in Equations 26 to extract a mean of the previous iteration velocity. Following that, the mean velocity that has been computed is provided to the Equations 27 and 28 in order to compute the velocities of the next iterations.

In the beginning of the algorithm the derivatives I_x, I_y and I_t are computed. These derivatives are used for the computation of the $I_d = \frac{1}{\alpha^2 + I_x^2 + I_y^2}$ term. As it can be seen in Equations 27 and 28 the derivatives computation and the division (I_d) is repeated in every iteration. Computing the derivatives and the division in every iteration is costly in terms

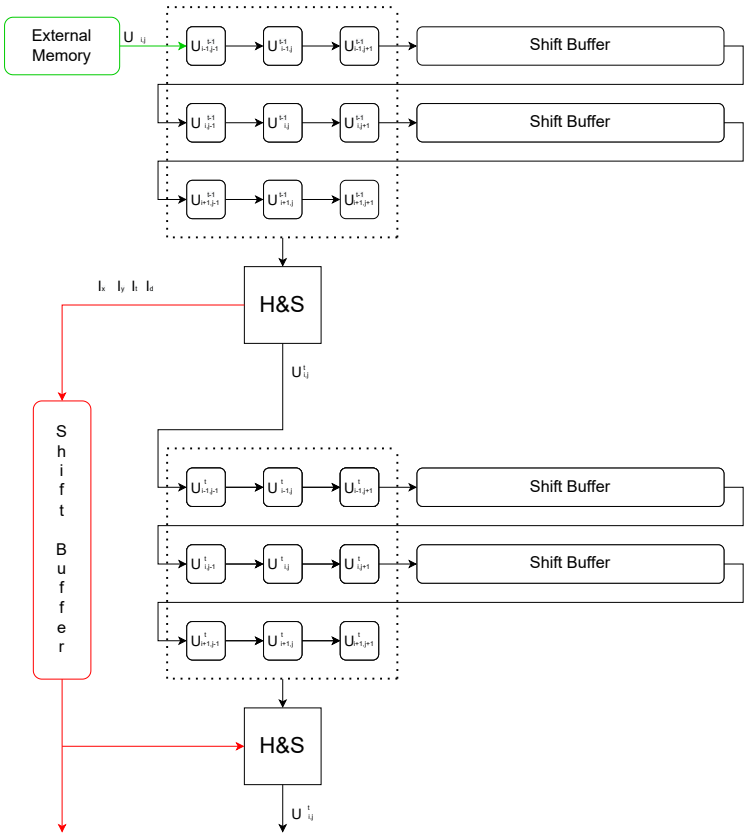

Figure 26: Derivatives Reuse

of DSPs[66]. By taking that into consideration a new solution is proposed in this thesis for inter-iteration computation reuse. This means that the derivatives and the division are computed in the first iteration, they are streamed with shift registers and they are reused by the next iteration as it is shown in Figure 26 (red path of the Figure 26). So, the idea is to store the results of the division and the other derivatives and use them in the next iteration. This leads to a decrease in DSPs and LUTs consumption but with a cost in Block RAMs. Moreover, this leads to a saving in external memory bandwidth as for the computation of the derivatives the pixels of the image have to be read from the external memory[67] (green path of the Figure 26).

Computation reuse can be also applied in Equations 26 as it is done in CPUs and GPUs for stencils with shared coefficients [143],[148]. We employ computation reuse in the convolution of the H&S core which to the best of our knowledge has never been done before. In the Equation (26) there are computations that are repeated both vertically and horizontally. In most of the previous implementations in FPGAs of Iterative Stencil Loops (ISL) algorithms with shared constant coefficients [82],[138],[149] the reuse computation is only limited to point-wise operations. Nevertheless, we extend it to include reduction

Table 7: Mono-scale Core

Mode	DSPs	LUTs	FFs	Block RAMs
With Computation Reuse	19	1677	12871	40
Without Computation Reuse	36	2010	11242	31

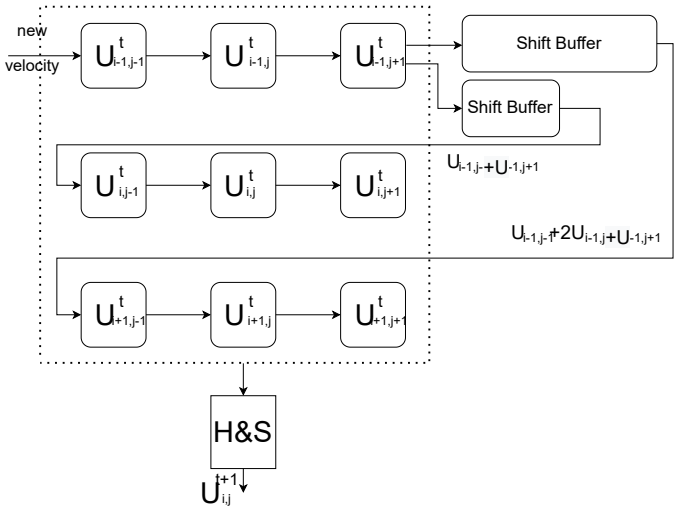

Figure 27: Deep Pipeline Architecture

operations [86] [150]. The total number of additions in Equation (26) are 14. However, the computations $U_{i-1,j-1} + U_{i-1,j+1}$ (multiplied by 2) and $U_{i-1,j-1} + 2 \cdot U_{i-1,j-1} + U_{i-1,j+1}$ are repeated vertically. Thus, we compute these 2 additions and we stream them, instead of the velocities. This technique requires one more shift buffer compared to [66] but it comes with a save of 6 additions in total for each H&S core compared to [66] and [67]. The architecture of the computation reuse in the convolution is shown in Figure 27.

In table 7 it is shown that with the computation reuse there is a 48% save in the use of DSPs and 17% in the use of LUTs but with a cost of 29% increase in the number of Block RAMs and 14% in the FFs. The image size is 2048x2048 pixels.

$$\bar{u} = \frac{1}{12}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 0 & 2 \\ 1 & 2 & 1 \end{pmatrix} * u \qquad \bar{v} = \frac{1}{12}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 0 & 2 \\ 1 & 2 & 1 \end{pmatrix} * v \tag{26}$$

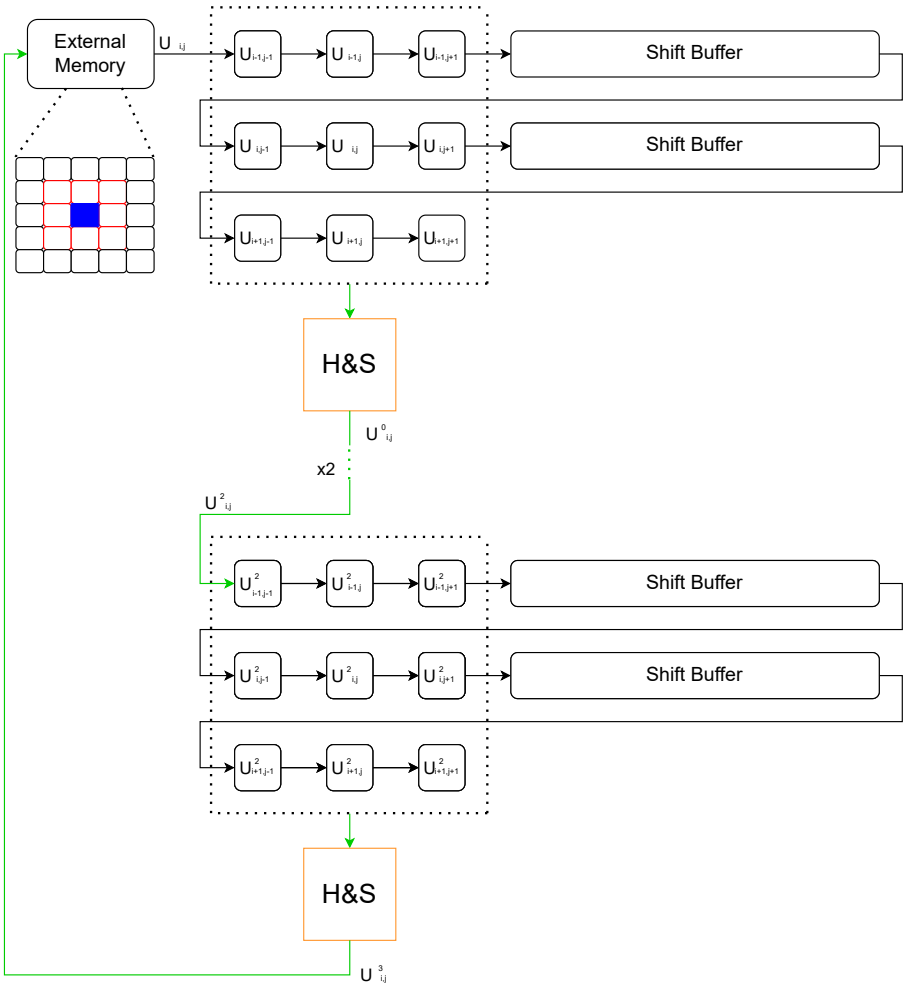

Figure 28: Deep Pipeline Architecture for 4 Iterations

$$u^{(i)} = \bar{u}^{(i-1)} - I_x \frac{I_x u^{(i-1)} + I_y v^{(i-1)} + I_t}{\alpha^2 + I_x^2 + I_y^2} \tag{27}$$

$$v^{(i)} = \bar{v}^{(i-1)} - I_y \frac{I_x u^{(i-1)} + I_y v^{(i-1)} + I_t}{\alpha^2 + I_x^2 + I_y^2} \tag{28}$$

As the mono-scale H&S algorithm is an iterative stencil loop algorithm, the deep pipeline and the vectorization techniques will be applied [80],[82],[81],[84],[76]. This two techniques affect widely the external memory bandwidth and the Block RAMs memory utilization [85].

The same number of pixels can be processed at the same time either by increasing vectorization [151] or the depth of the pipeline [85],[152]. For example, if 4 iterations have

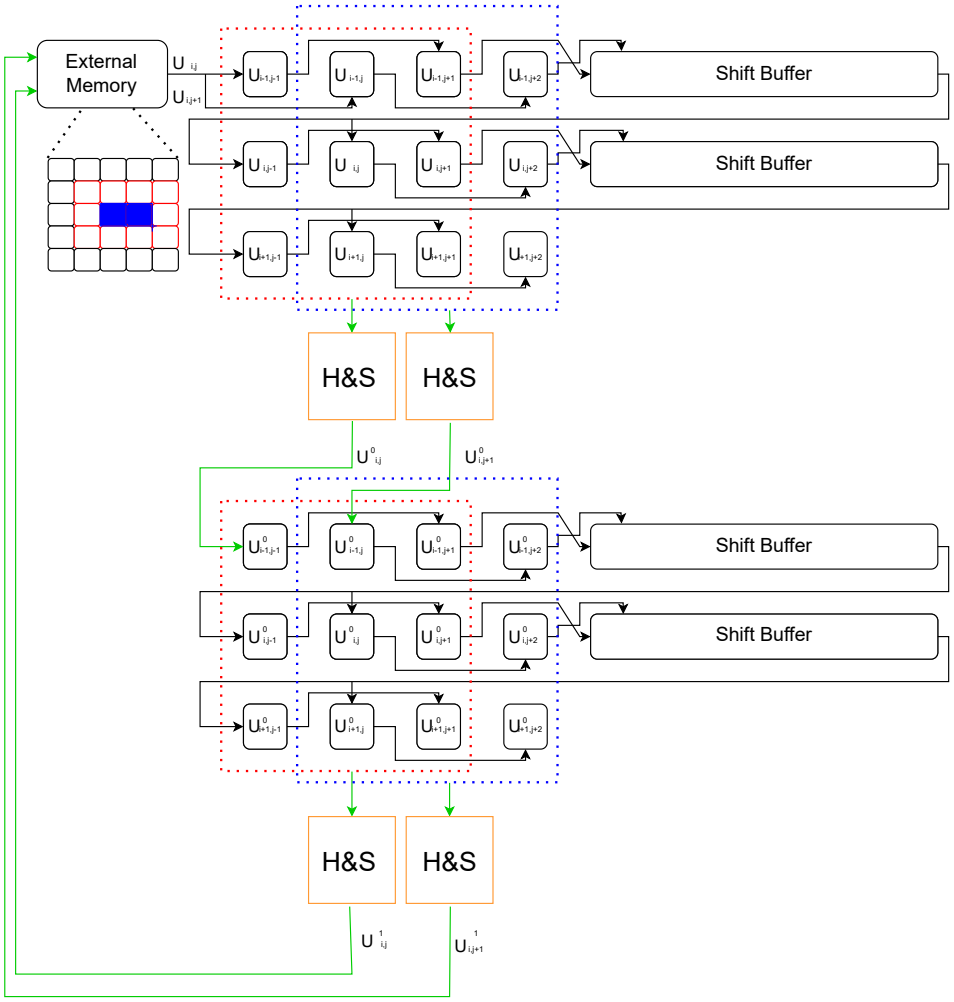

Figure 29: Vectorized-Deep Pipeline Architecture for 4 Iterations

to be computed then this can be done either with an architecture with a depth of pipeline of 4 units and a vectorization of 1 as shown in Figure 28 or with a depth of pipeline of 2 units and vectorization of 2 as shown in Figure 29 (in the two schematics the green is the depth of the pipeline and orange is the vectorization). The shift buffers required in the first case is twice as much as the shift buffers required by the second architecture. However, the external memory bandwidth of the first architecture requires half the bandwidth required by the second architecture due to less vectorization in the memory. In this thesis, the second architecture is used, as the only accelerator in the system is the multi-scale H&S accelerator and there is no need for other algorithms to access the memory. Thus, all the bandwidth is dedicate for the multi-scale H&S algorithm.

Table 8: Mono-Scale Architecture

Arch.	DSPs	LUTs	FFs	Block RAMs	Mem. Band.(GB/s)
Π=4 Q=2	144	5132	65011	94	4.3
Π=4 Q=1	144	7012	71051	127	2.1

In table 8 the two architectures' resources demand is shown. The working clock frequency in both cases is 250 Mhz. Π is the depth of the pipeline and Q is the vectorization. In the second case the demand for Block RAMs is increased by 35 % while the demand for external memory is reduced by 52 %. Following that, when saving Block RAMs is important, the first architecture should be chosen while the second one should be preferred when external memory bandwidth is the bottleneck.

As a new proposal, in this thesis for the design of the mono-scale architecture, different floating point formats were used. The different arithmetic formats impact the resources utilization[153],[154]. In table 9 the results of the designing of the mono-scale core with different arithmetic formats is shown. The image size is 2048x2028 pixels. For the F_{10} and F_9 designs the option not to use DSPs was preferred. This was done because the mantissa size in both cases is very small (4 and 3 bits respectively) and as a result the cost of designing the arithmetic units with only LUTs is less costly [129]. It can be seen from this table that when decreasing the arithmetic format, the number of the Block RAMs is decreased and also the number of LUTs and FFs. The number of DSPs is the same for all arithmetic formats except that of F_{32}, F_{10} and F_9. This happens because the Arria 10 FPGA does not have dedicated floating point units for smaller arithmetic floating point formats [129],[141],[155]. In the future computations will be coupled in the same DSPs in order to decrease the number of DSPs in a similar way as it was done in [155] ,[154],[156] and [157]. Another important factor that has to be pointed out is that when using smaller floating point formats, more pixels can be read parallelly from the external memory [140], [158], something that highly increases the throughput as vectorization is increased.

5.2 WARPING

The next algorithm that is discussed is the warping. In this thesis bi-linear and bi-cubic interpolation has been used for the warping. The bi-linear interpolation requires a 2x2 neighbourhood of pixels and the computation of each point is done according to Equations 29,30 and 31. f_{00} f_{01}, f_{10} and f_{11} are the neighboring pixels from the 2x2 neighborhood, U_f and V_f are the fraction parts of the velocities of the previous pyramid level and F is the new computed warped pixel.

Table 9: Mono-Scale Core with Different Floating Point Formats

Floating Point	DSPs	LUTs	FFs	Block RAMs
F_{32}	19	1677	12081	38
F_{16}	10	1241	70991	21
F_{15}	10	1202	68531	21
F_{14}	10	1125	67122	20
F_{13}	10	1078	65012	20
F_{12}	10	1051	63912	20
F_{11}	10	1002	61239	18
F_{10}	0	2749	58544	15
F_9	0	2431	56923	15

$$F_0 = (1 - U_f)f_{00} + U_f f_{01} \tag{29}$$

$$F_1 = (1 - U_f)f_{10} + U_f f_{11} \tag{30}$$

$$F = (1 - V_f)F_0 + V_f F_1 \tag{31}$$

The bi-cubic interpolation is performed with a 4x4 neighbourhood of pixels according to the Equations 32,33,34,35 and 36. It is clear from the Equations that the demand for DSPs in the bi-cubic interpolation is higher than the bi-linear interpolation as also stated in [159]. When vectorization is applied then the demand of DSPs for the bi-cubic interpolation, requests the highest amount of DSPs between all the components.

$$\begin{aligned} F_0 = f_{01} + 0.5U_f(f_{02} - f_{00} + U_f(2f_{00} - 5f_{01} + \\ 4f_{02} - f_{03} + U_f(3(f_{01} - f_{02}) + f_{03} - f_{00}))) \end{aligned} \tag{32}$$

$$\begin{aligned} F_1 = f_{11} + 0.5U_f(f_{12} - f_{10} + U_f(2f_{10} - 5f_{11} + \\ 4f_{12} - f_{13} + U_f(3(f_{11} - f_{12}) + f_{13} - f_{10}))) \end{aligned} \tag{33}$$

$$\begin{aligned} F_2 = f_{21} + 0.5U_f(f_{22} - f_{20} + U_f(2f_{20} - 5f_{21} + \\ 4f_{22} - f_{23} + U_f(3(f_{21} - f_{22}) + f_{23} - f_{20}))) \end{aligned} \tag{34}$$

$$\begin{aligned} F_3 = f_{21} + 0.5U_f(f_{22} - f_{20} + U_f(2f_{20} - 5f_{21} + \\ 4f_{32} - f_{33} + U_f(3(f_{31} - f_{32}) + f_{33} - f_{30}))) \end{aligned} \tag{35}$$

Table 10: Bi-cubic Interpolation Core with Different Floating Point Formats

Floating Point	DSPs	LUTs	FFs	Block RAMs
F_{32}	67	70700	49659	63
F_{16}	35	40898	27911	37
F_{15}	35	39147	27015	37
F_{14}	35	38214	26801	34
F_{13}	35	37109	26491	31
F_{12}	35	36813	25817	30
F_{11}	35	36412	24901	28
F_{10}	35	31471	31231	25
F_9	35	25411	29031	25

Table 11: Bi-linear Interpolation Core with Different Floating Point Formats

Floating Point	DSPs	LUTs	FFs	Block RAMs
F_{32}	10	16849	9520	68
F_{16}	5	9122	5931	40
F_{13}	5	8179	4701	34

$$F = F_1 + 0.5V_f(F_2 - F_0 + V_f(2F_0 - 5F_1 + 4F_2 - F_3 + V_f(3(F_1 - F_2) + F_3 - F_0))) \tag{36}$$

In Section 4.2 the importance of reading consecutive addresses from the external memory is pointed out in order to achieve the goal of the computation of one or more pixels per clock cycle [144][142][160]. To fulfill this, it is mandatory to store in the on-chip memory all the possible pixels that might potentially be the adequate neighboring pixels. This leads to an increase in the RAM usage. A different approach is followed by the work of [43] where non consecutive pixels are read from the external memory which highly reduces the throughput but it saves on-chip memory.

Another important matter that has to be pointed out is that the multiplexers which are used for the choice of the right pixels, are $2\rightarrow1$ instead of $8\rightarrow1$ with the aim of reducing the consumption of LUTs. The choice of the correct pixels is done with the integer part of the velocities computed in the previous levels (U_i).

As in the case of the mono-scale algorithm different arithmetic floating point formats were used for the design of the warping core. In table 10 the hardware resources utilization is shown for the bi-cubic interpolation [159] and in table 11 the results are shown for the bi-linear [43] interpolation. The image size in both cases is 2048x2028 pixels. It is clear from the two matrices that bi-cubic interpolation demands more resources. By using smaller floating point formats, the DSPs usage is not affected (only if F_{32} is used) but the number of LUTs and Block RAMs is highly reduced as also stated in [129]. F_{10} and F_9 in the warping case are designed with the use of DSPs because otherwise the ALUTs usage is enormous. The warping core, when bi-cubic interpolation is used, is the one which demands the largest resources utilization. Nevertheless, as we show in chapter 3 for F_{32} and F_{16} there is no need for the bi-cubic interpolation and we will not use it in this too formats to save resources.

5.3 DOWN-SAMPLING

The initial component of the multi-scale H&S algorithm is the down-sampling core. Down-sampling is a component which is widely used in image processing pipelines [68], [161]. In this thesis, it is computed by applying a 5x5 gaussian filter in the initial image in order to provide the final down-sampled image.

As we have seen, in the works of [68],[60],[162] down-sampling is done line by line in the initial image. This means that, as explained in algorithm 1 and in section 4.3 in the odd lines of images there is no computation performed. Thereafter, to achieve an output of Q pixels per clock cycle, vectorization should be 2×Q as the computation in the even lines should be doubled. This impacts the resources consumption by 2 in terms of DSPs usage.

Algorithm 3 Pseudo code of Shift Buffers with 2 clocks

```
1: int ShiftBuf[width]
2:
3: if ch == 0 then par = 2
4: if ch == 1 then par = 4
5: #pragma unroll
6: for (w=0; w<width; w+=par) do
7:     #pragma unroll
8:     for (p=0; p<par; p++) do
9:         ShiftBuf[w × par + p] = ShiftBuf[(w − 1) × par + p]
10:    end for
11: end for
```

In section 4.3 the architecture for the down-sampling followed in this thesis is explained. With the proposed described approach, half the number of DSPs is saved compared to the

Table 12: Down-sampling Core

Architecture	DSPs	LUTs	FFs	Block RAMs
This Thesis	25	23133	24574	62
Classic	50	21174	18714	50

one of [68] as computations are performed in every clock cycle and in every line of the image. Nevertheless, when designing the unit in OpenCL, careful consideration should be given when handling the on chip memory. A way to implement this unit is to shift the shift registers by a different value every time. For example, when reading values from the external memory the shift is four, while when writing the shift is 2. This is done because for the computation of the down-sampling as shown in Figure 20 a 2x2 matrix is needed and at the same time 4 pixels are read from the external memory. A pseudo code for the implementation of this shifting buffers is shown in algorithm 3. In this code with the choose signal (ch) we can choose the shifting (par) of the buffer (ShiftBuf). Although this design functions wells in terms of pipeline, the problem that arises is that two clock are created by the Intel Opencl Compiler 19.1 for the shifting of the buffers. The first clock functions with a frequency of f for the double shifting of the buffers, while the second clock with $\frac{f}{2}$. Nevertheless the working frequency of the whole design is the one with the lowest clock frequency. This has a major impact in the throughput of the system as the working frequency is significantly low.

In order to overcome the above bottleneck, we provide an alternative solution for the design of the shift buffers. The pseudo code which avoids the use of two clocks is shown in algorithm 4. The shift buffers are designed with the internal Block RAMs (local int in lines 1,2,3 and 4 of the algorithm 4) as explained in the intel SDK openCL manual [135] and not as shift registers and depending on the ch value, either the first and second, or the third or fourth registers are shifted. This way there is no need for two clock working frequencies as the compiler treats the shift buffers as internal RAM and not shift registers.

With this design, the ultimate goal of at least one output per clock cycle is achieved (in the Opencl report [163] this is marked as $II=1$ and it is called loop-carried dependencies and it refers to the possible outputs per clock cycle), contrary to all the aforementioned works [68],[60],[113],[63] where more DSPs have to be used to increase the throughput but even then $II \neq 1$ as there are loop-carried dependencies.

In table 12 the resources utilization is shown for the two implementation for the down-sampling core. The first one is the one proposed in this thesis, while the second one is the classic one as also proposed by other works [68]. F_{16} floating point format was used in both cases and it is obvious from the table that the approach proposed in this thesis requires half the number of DSPs. The downside is that more Block RAMs are needed in order to avoid the issue with the two clocks.

Algorithm 4 Pseudo code of Shifth Buffers without 2 clocks

```
1: local int ShiftBuf1[width/4]
2: local int ShiftBuf2[width/4]
3: local int ShiftBuf3[width/4]
4: local int ShiftBuf4[width/4]

5: if ch == 0 then
6: #pragma unroll
7: for (w=0; w<width/4; w++) do
8:     ShiftBuf1[w × par + p] = ShiftBuf1[(w − 1) × par + p]
9:     ShiftBuf2[w × par + p] = ShiftBuf2[(w − 1) × par + p]
10: end for

11: if ch == 1 then
12: #pragma unroll
13: for (w=0; w<width/4; w++) do
14:     ShiftBuf3[w × par + p] = ShiftBuf1[(w − 1) × par + p]
15:     ShiftBuf4[w × par + p] = ShiftBuf2[(w − 1) × par + p]
16: end for
```

Another solution proposed by the work of [43] is to read not consecutive pixels from the external memory in order to perform the down-sampling. This highly reduces the throughput as the efficiency of the memory controller is affected (it is given by the OpenCL profiler [164]) and so this implementation was not used in this thesis.

5.4 UP-SAMPLING

The same bottlenecks that were faced in the case of the down-sampling in terms of resources are faced in the case of the up-sampling too. All the previous works [68],[60],[113],[63] are reading from the external memory non consecutive neighboring pixels from the external memory which highly decreases the effectiveness of the memory controller. In this thesis a different approach is followed as explained in section 4.4. This different approach suffers from the two clocks bottleneck which is bypassed with the use of more Block RAMs in the same way as it is done in the down-sampling core.

By providing a steady output of one or more pixels (1×Q) per clock cycle the following components can consume the incoming data in every clock cycle and produce an output without changing the memory layout [165].

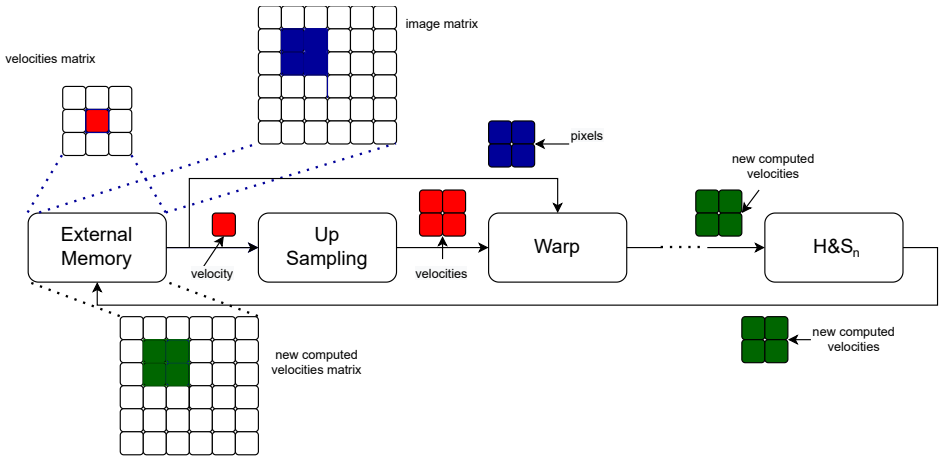

Figure 30: Pyramid level implementation of [68]

5.5 MULTI-SCALE H&S ALGORITHM

Now that the resources utilization of all the components have been explained, it will be analyzed in this section how the connection of all this components impact the resources and the design.

5.5.1 *Multi-rate Architecture*

The multi-scale H&S algorithm is a multi-rate algorithm [68], which means that the interconnection between the components needs to be synchronized according to the different rates of inputs and outputs in each component [94],[166],[167].

In all the previous works [68],[63],[168] the approach that was followed for a multi-rate algorithm design is shown in Figure 30. The algorithm because of which the system is a multi-rate algorithm is the up-sampling. For every new velocity input in the up-sampling, four new velocities are produced in a square format. The input to the warping algorithm is then, these velocities in a square format. This requires that the pixels that are read from the external memory to also be in a 2x2 format. Finally, the new computed velocities are written back to the external memory in the same format. Nevertheless, if no change in the memory layout is done for the image and the velocities to be friendly for the memory controller to this square formats, this will lead to a decrease in throughput due to the fall in the effectiveness and the stalls of the memory controller (effectiveness and stalls are given by the OpenCL Profiler). Even in the case when a different memory layout is applied, this will impact the computation time of the down-sampling as it will not be friendly for the acceleration of this part of the design as explained in section 4.3.

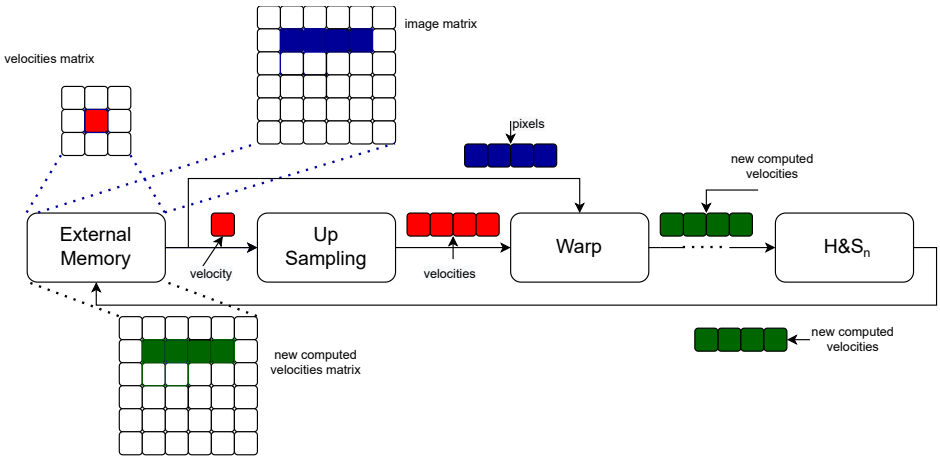

Figure 31: Pyramid level implementation

Consequently, in this thesis an alternative approach was followed which was explained thoroughly in section 4.5. This implementation is also shown in Figure 31. Here, there is no need for changing the memory layout because the pixels are read in a continuous way and not in squares as in the previous approaches. Down-sampling is also performed with the same memory layout by reading consecutive addresses. The stalls are less (4%) and the effectiveness of the controller is close to 80 %.

5.5.2 *Multi-level Architecture*

Multi-scale H&S is a multi-level architecture where the same computations are applied in every pyramid level. However, the size of the image in each pyramid level is not the same because of the scaling. Therefore, the size of the shift buffers have to be different in each level to fit the size of the image. This leads to extra Block RAMs usage which with a specific modification in the pipelines can be avoided. This modification is shown in Figure 32.

In this architecture more but smaller block RAMs are used with some extra registers and multiplexers to choose the right depth of the shift buffers depending on sizes of the image. If the computations in the largest image are made then all the Shift Registers are used (blue dotted lines denote the registers in the Figure 32 that are used in this pyramid level for the convolution of the H&S algorithm), whereas when computations in the small image are made only some of them are used (red dotted lines). The green dotted lines denote the registers used when the medium sized image is computed.

When the velocities are computed in each pyramid level, they have to be stored somewhere in order to be read in the next pyramid level for the following computations. In this thesis, two new approaches have been examined. The first one is to store the computed

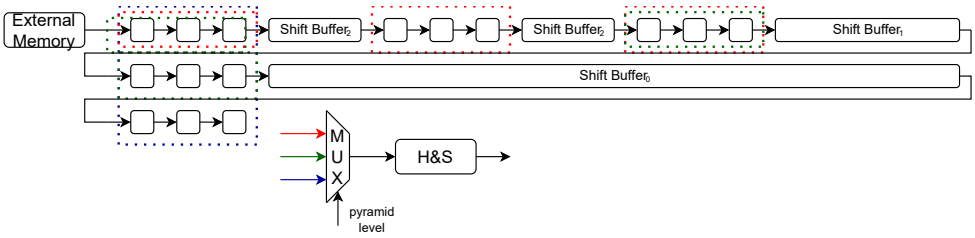

Figure 32: Shift Buffers

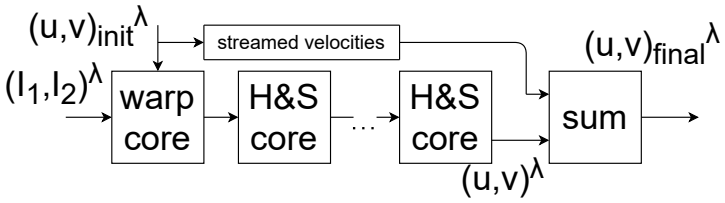

Figure 33: Streamed velocities

velocities in the on chip memory. This requires a lot of Block RAMs used but it reliefs the pressure from the external memory bandwidth. Nevertheless if large image are computed (2048x2048 pixels) on chip memory is not enough even if a large FPGA is considered like the Arria 10.

The second approach that was followed in this thesis is to store the intermediate velocities in the external memory. With this way in each pyramid level, both the two images have to be read from the external memory along with the computed velocities from the previous pyramid level in order to perform the warping. Furthermore, these velocities have to be read again after the mono-scale H&S computation in each pyramid level. This is obligatory in order to perform the summation between the velocities computed in the previous level and the ones calculated in this level. To avoid the last read, the up-sampled velocities can be streamed alongside the H&S cores in order to further reduce the interaction with the external memory as shown in Figure 33. This comes with a cost in the on-chip memory but it is negligible compared to the whole Block RAMs usage. Finally, the computed velocities are written back in the external memory.

5.6 RESOURCES UTILIZATION RESULTS AND COMPARISON WITH STATE OF THE ART

In the previous sections in this chapter the resource utilization of each component of the hierarchical H&S algorithm was analyzed. In this section, the total resources consumption of the whole implementation will be presented.

Table 13: Resources Utilization with Π=5

Format	interpolation	DSPs	ALUTs	Block RAMS	EMB
F_{32}	bi-linear	170	95k	528	9.3
F_{32}	bi-cubic	227	140k	540	9.1
F_{16}	bi-linear	95	78k	427	4.71
F_{16}	bi-cubic	125	107k	434	4.52
F_{13}	bi-linear	95	88k	422	4.41
F_{13}	bi-cubic	125	102K	423	4.38
F_{10}	bi-cubic	93	91k	386	3.2

Table 14: Resources Utilization with Π=10

Format	interpolation	DSPs	ALUTs	Block RAMS	EMB
F_{32}	bi-linear	288	167k	580	14.9
F_{32}	bi-cubic	402	238k	588	15.2
F_{16}	bi-linear	188	149K	455	8.1
F_{16}	bi-cubic	248	194k	463	8
F_{13}	bi-linear	188	131k	448	6.7
F_{13}	bi-cubic	248	184K	450	6.6
F_{10}	bi-cubic	148	158k	406	5

5.6.1 *Results of Implementation*

For the implementation of the components with floating point arithmetic, the FloPoCo library was used [102] and the Han Pilot platform. The components were inserted in the OpenCls' libraries as custom components directly from the FloPoCo libraries. All the results were taken by the report file provided by the AOC compiler and the results of the DDR4 usage were taken from the profiler report. The externals memory bandwidth of the Han Pilot Platform is 17 GByte/s (given by the manufacturer), the DSPs number is 1687, the number of ALUTs is 503360 and the number of Block RAMs is 2131 .

In tables 13,14,15 and 16 the resources consumption are shown for four arithmetic formats, for 2 interpolation types (we include the F_{32} and F_{16} bi-cubic designs to see the impact in the resources usage) and for a vectorization of Q=1, 2, 4 and 8 as the designs are scaled up to see which is the largest design that fit the Arria 10. The clock frequency is from 243 MHz to 270 MHz in all the designs.

Table 15: Resources Utilization with Π=20

Format	interpolation	DSPs	ALUTs	Block RAMS	EMB
F_{16}	bi-linear	324	235K	561	15.7
F_{16}	bi-cubic	444	267k	567	15.6
F_{13}	bi-linear	324	197k	546	13.1
F_{13}	bi-cubic	444	248K	550	12.5
F_{10}	bi-cubic	244	250k	481	7.5

Table 16: Resources Utilization with Π=40

Format	interpolation	DSPs	ALUTs	Block RAMs	EMB
F_{10}	bi-cubic	436	439k	691	16.1

In these tables, it is obvious that the DSPs number and the externals memory bandwidth requirement scales almost linearly. With the computation reuse the number of DSPs is reduced with some extra cost in Block RAMs. F_{32} is not implementable with a vectorization of 4 and 8 because of the the external memory bottleneck. For a vectorization of 8 only the F_{10} architecture is implementable and almost the whole bandwidth is utilized. When going from F_{16} to F_{13} the number of DSPs remains the same as there is no support in the Arria 10 FPGA for small arithmetic formats. However, the external memory bandwidth is reduced for the F_{13}. Finally, the mono-scale part of the F_{10} design is implemented without DSPs and that is why the DSPs usage is reduced compared to F_{13}. It is obvious and expected that switching to a smaller arithmetic formats enables the designer to fit more cores in the FPGA, save resources and scale up further the design. As a result, switching from single precision floating arithmetic to smaller floating point numbers significantly benefits the implementation. Finally, bi-linear designs save a large number of DSPs and LUTs compared to the bi-cubic ones. As we presented in chapter 3 for F_{32} and F_{16} there is no accuracy degradation if bi-linear interpolation is used and so for these two arithmetic formats only bi-linear interpolation is used.

Table 17: Resources With The Smaller Numeric Format in level 0

Format	interpolation	DSPs	ALUTs	Block RAMs	EMB
F_{16} F_{16} F_{10}	bi-cubic	640	490k	883	15.7

Table 18: Block RAMs usage with different image sizes

Format	image size	Block RAMs
F_{16}	512x512	301
F_{16}	1024x1024	455
F_{16}	2048x2048	589
F_{10}	512x512	271
F_{10}	1024x1024	406
F_{10}	2048x2048	521

In table 18 the Block RAMs usage is presented for different sizes of the image. The number of the cores in the designs is Π=10. There is a significant decrease in the on chip memory usage when smaller arithmetic formats are used. Even if a 2048x2048 pixels image is the case in a F_{10} design, the memory usage is smaller than when a 1024x1024 is considered in a F_{16} design.

In table 17 the resources usage is shown for a custom floating point format. Here the number of ALUTs is very high but also the external memory bandwidth requirement. However as shown in section 4.6 this design achieves very high throughput. Without a smaller arithmetic format in the level 0 it would not be possible to fit this number of cores as the external memory can not support this vectorization.

An important technical issue that has to be mentioned about the external memory is that because 4 reads (two images and two velocities) and 2 writes (new computed velocities) are made the efficiency of the external memory controller is dropping [144],[145]. The fact that the Han Pilot Platform is equipped with only one external memory interface which does not allow the allocation of the reads and writes in separate memory interfaces.

As future work, it is planned to couple the smaller arithmetic formats computations in the same DSPs. Furthermore, this units will be able to compute both in larger arithmetic floating point (F_{32},F_{16}) and in smaller floating point formats (F_{10},F_9). This will further reduce the resources usage as the same cores will be reused in every pyramid level in a trans-floating architecture but will keep the same throughput.

5.6.2 *Comparison with the state of the art*

The next step is to compare the design with other state of the art designs in terms of resources. The results are shown in table 19. The designs q_1 and q_2 are with Π=10, the designs q_4 and q_5 are with Π=20, q_5 is with Π=40 and q_6 is with $\Pi_{1,2}$=20 and Π_0=40. Design q_1 and q_3 are with bi-linear interpolation while the rest are bi-cubic interpolations.

Table 19: Comparison with other State of the Art

Implem.	algo.	Format	DSPs	ALUTs	B. RAMs	FPGA
q_1	MH&S	F_{16}	188	149k	455	A10
q_2	MH&S	F_{10}	148	158k	406	A10
q_3	MH&S	F_{16}	324	205k	561	A10
q_4	MH&S	F_{10}	244	309k	550	A10
q_5	MH&S	F_{10}	436	439k	691	A10
q_6	MH&S	F_{16} F_{16} F_{10}	640	490k	883	A10
[146]	H&S	-	12	3k	595k	CII
[67]	H&S	$Q_{1.4.8\rightarrow1.19}$	-	490k	1346	V7
[66]	H&S	Q	316	11k	3.5M	StrIV
[96]	H&S	$Q_{8\rightarrow28}$	237	-	467	V7
[43]	MH&S	$Q_{1.4.8\rightarrow1.19}$	523	104k	312	ZCU
[97]	L&K	-	-	-	760k	V2
[59]	L&K	$Q_{4.6\rightarrow26}$	54	16k	120	V6
[61]	L&K	$Q_{10\rightarrow32}$	80	14k	28	V2
[60]	ML&K	$Q_{9.0\rightarrow29.8}$	88	51k	244	V4
[43]	ML&K	$Q_{1.4.8\rightarrow1.19}$	861	122K	311	ZCU
[63]	MPB	$Q_{8.0\rightarrow8.4}$	52	32k	112	V4
[62]	HBM	$Q_{10\rightarrow32}$	48	36k	132	V7

The image size is 1024x1024. The clock frequency for all the designs is from 245 to 280 MHz.

It can be seen that the smallest design q_3 uses similar resources with [60], [61] and [59]. The increased usage of Block RAMs is due to the static partitioning of the Block RAMs of the intel OpenCL Compiler for the HAN Pilot Platform. However, in terms of throughput the aforementioned designs are massively outperformed as explained analytically in section 4.6 (the design of [61] is not outperformed, however the computation is not done in a single FPGA but in a PC and two FPGAs and that is why the resources remain low and the throughput very high). Moreover, these works consider smaller images than ours. Works [113], [62], [146] are using less resources, but the throughput they achieve (section 4.6) is at least 10 times less than q_3 and also the image size is smaller. Implementations [67] and [66] are implementing a mono-scale H&S algorithm and the number of their cores is Π=128 and Π=30 but without any interpolation. Nevertheless their resources utilization is

comparable with our multi-scale designs. Blachut work, which is very recent [43] and is implementing a multi-scale H&S algorithm is using similar resources as the q_4. However, our design is a 3 scales algorithm while theirs is a 2 scale algorithm and that affects the accuracy. Furthermore in Blachut's work only bi-linear interpolation is done and not in the beginning of each pyramid level but only in the end of the previous computation chain[43]. It is worth mentioning that we are as far as we know among the only ones who use floating point formats for the design of the algorithm ([67], [63] and [47] have some floating point design in their works). All the rest use fixed point formats (Q) with more bits to represent the data than ours which reduces the possible parallel interactions with the external memory and the Block RAMs usage.

5.7 CONCLUSIONS

To sum up this chapter, an extended analysis has been performed regarding the hardware resources utilization of each component of the Multi-scale H&S algorithm. Challenges in the design that were faced were pointed out, while solutions to overcome them were proposed. It was explained how computation reuse can decrease the resources usage, how smaller arithmetic formats can save external memory bandwidth and two interpolation types and their resources consumption were presented. We dealt with challenges that one faces in the multi-rate architectures and we proposed how to deal with it effectively in terms of external memory. Finally we compared our designs with other state of the art designs and we showed that for better throughput or accuracy that we achieve, less or slightly more resources are used.

In the next chapter a design space exploration will be performed for the multi-scale H&S algorithm. This exploration will take into account accuracy, throughput and hardware resources consumption analysis that was explained in this chapter and the previous ones and it will give the designer the opportunity to tune the design according to his requirements.

6 DESIGN SPACE EXPLORATION

In the previous chapters, we performed an analysis of the multi-scale H&S algorithm regarding its accuracy, throughput and resources utilization. We proposed ways to increase the accuracy of the flow detection, by performing more iterations, using a higher number of pyramid levels and trying different numeric formats. Following that, we showed the impact of these propositions in the throughput. Concerning the throughput, we provided solutions in order to cope with the multi-rate nature of the algorithm and to accelerate the computation of the ISL part of the algorithm. However, by increasing throughput, resources utilization of the FPGA are increased. Thus, techniques in order to reduce this increase were proposed. In this chapter, we take into account all this information and we perform a design space exploration concerning the multi-scale H&S algorithm. In the design space exploration we will deal with the total computation time, the resources utilization like the DSPs number and the Block RAMs, the external memory bandwidth and the accuracy of the detection.

6.1 METHODOLOGY

In order to establish the theoretical values of all the parameters of our architecture, the information provided in in the previous chapters are used. Depending on the number of the components used (for example the number of H&S cores) and the aforementioned information, a theoretical model can be established. The floating point numeric format is also taken into account as we introduced every floating point format in the model. The code of the implementation is tunable in terms of the depth of the pipeline, which means how many H&S cores are cascaded with the autorun kernel attribute (Π) provided from the OpenCL [135] and the vectorization (Q) as it was explained in subsection 4.1.2 and section 4.2. In order to verify our theoretical model, we extract the information from the model we created, and we compare it to the information we gather when we compile the tunable OpenCL code of our architecture and to the information we get when we run the design in our device.

6.1.1 *Notation*

In table 20 the notations which we use in our theoretical model are presented for the DSPs and Block RAMs usage. $H\&S_0$, $H\&S_{2,1}$, $Warp_0$ and $Warp_{2,1}$ are the H&S and warping cores DSPs and Block RAMs utilization, for levels 0, levels 2 and 1 respectively, for the

Table 20: Notations for the Theoretical Model I

	$H\&S_0$	$H\&S_{2,1}$	$Warp_0$	$Warp_{2,1}$	Sum	Up	Down
DSP	N_{HS_0}	$N_{HS_{2,1}}$	N_{warp_0}	$N_{warp_{2,1}}$	N_{sum}	N_{up}	N_{down}
Block RAMs	M_{HS_0}	$M_{HS_{2,1}}$	M_{warp_0}	$M_{warp_{2,1}}$	M_{sum}	M_{up}	M_{down}

Table 21: Notations for the Theoretical Model II

WL	Word length
Q	Vectorization
Π	Deep pipeline
Ps	Pixel size
H	Height
W	Width
f	clock frequency

trans-floating architecture. For the non trans-floating architecture, it is the same notation but without further information about the scale of the pyramid. In table 21, the rest of the notations that are used in theoretical model are presented.

6.1.2 *DSPs Utilization*

We start with the DSPs. The DSPs utilization for the non trans-floating designs can be modeled by Equations (37) and (38).

$$N = \Pi(Q) \cdot N_{HS} + Q \cdot N_{warp} + N_1 \tag{37}$$
$$N_1 = Q \cdot N_{sum} + Q \cdot N_{up} + N_{down} \tag{38}$$

We see in these equations, that the number of DSPs is proportional to the number of the H&S cores and the interpolation cores.

For the trans-floating designs the usage of DSPs is modeled by Equations 39, 40 and 41.

$$N = \Pi_0(Q_0) \cdot N_{HS_0} + Q_0 \cdot N_{warp_0} + Q_0 \cdot N_{sum} + N_1 + N_2 \tag{39}$$
$$N_1 = \Pi_{2,1}(Q_{2,1}) \cdot N_{HS_{2,1}} + Q_{2,1} \cdot N_{warp_{2,1}} \tag{40}$$
$$N_2 = Q_{2,1} \cdot N_{sum} + Q_{2,1} \cdot N_{up} + N_{down} \tag{41}$$

It can be seen, that the trans-floating format demands more DSPs compared to the non trans-floating format. This happens because, our current components from the FloPoCo

[102] library can not do the computation with different arithmetic formats. This is why in the future we plan to create cores that can compute in both the arithmetic formats used in $level_{2,1}$ and $level_0$ as it was done by [155] but for smaller floating point formats than half precision arithmetic.

The validity of the theoretical model for the DSPs number is 87.65%. It is not 100%, because the Arria 10 Han Pilot Platform instantiates some DSPs for the Static Partition [169]. Otherwise, it is able to estimate with a validity of 100% the usage of DSPs for the warping, H&S cores, down-sampling and up-sampling.

6.1.3 *Block RAMs*

As explained in subsection 5.5.2 in this thesis we use the external memory for storing the intermediate velocities between the pyramid levels for the non trans-floating implementation. The total memory M in Block RAMS number for the non-trans floating implementation is modeled by Equation (42) and Equation (43) .

$$M = \Pi(Q) \cdot M_{HS} + Q \cdot M_{warp} + M_1 + M_2 \tag{42}$$

$$M_1 = M_{sum} + \cdot M_{up} + M_{down} \tag{43}$$

The streamed velocities in each pyramid level as described in subsection 5.5.2 and shown in Figure 33 are included in the M_{HS} core. Similar to the DSPs case, the number of Block RAMs is proportional to the number of the H&S cores.

For the trans-floating design the model that gives the Block RAMs number utilization are Equation 44, 45 and 46.

$$M = \Pi_0(Q_0) \cdot M_{HS_0} + Q_0 \cdot M_{warp_0} + Q_0 \cdot M_{sum} + M_1 + N_2 \tag{44}$$

$$M_1 = \Pi_{2,1}(Q_{2,1}) \cdot M_{HS_{2,1}} + Q_{2,1} \cdot M_{warp_{2,1}} \tag{45}$$

$$M_2 = Q_{2,1} \cdot M_{sum} + Q_{2,1} \cdot M_{up} + M_{down} \tag{46}$$

As in the case of the DSPs, separate Block RAMs are used for level 0 which increases the usage of them. In the future, we plan to reuse the same Block RAMs in every pyramid level. This is something we plan to solve in the future as we plan to reuse the same Block RAMs in every pyramid level, as the same computation components will be used in every pyramid level.

The validity of the model for the number of Block RAMs utilization is 54%. This happens due to the fact of the static partitioning which instantiates 162 Block RAMs for the Han Pilot platform Arria 10 FPGA. By taking this into account the validity of the model is increased to 80%. As it is stated in [82] which are accelerating ISL with FPGAs and OpenCL, it is very challenging to create an effective model to predict the utilization of the

memory because the Block RAMs are quantified and it is difficult to predict the optimization of the intel OpenCl compiler. Vectorization also affects the number as more Block RAMs have to be used in order to satisfy the increased need of ports for the simultaneous reads and writes in the on-chip memory.

6.1.4 *External memory bandwidth*

The external memory bandwidth for the non trans-floating implementation is modeled by Equation 47.

$$EMB = 2 \cdot f \cdot ((Ps + \mathbf{2} \cdot WL + \frac{1}{4} \cdot WL) \cdot Q) \tag{47}$$

In Equation 47, 2 is because two images are read and the two computed velocities are written back to the external memory (u,v). **2** refers to the re-reads of the velocities when Π is less than the iteration numbers. This happens because, all the iterations can not be fused with only one full image read from the external memory, so both the image and the computed velocities has to be read to compute the derivatives as show in Figure 23. In any other case, this term is equal to 0 as there is no need to re-read the velocities. The $\frac{1}{4}$ term refers to the reads of the computed velocities from the previous pyramid level which in our case are not stored in the Block RAMs, but in the external memory.

In the trans-floating designs the bandwidth is modeled by Equation (48)

$$EMB = 2 \cdot f \cdot ((Ps + WL_{2,1}) + 4 \cdot (Ps + WL_0)) \cdot Q) \tag{48}$$

The extra term ($4 \cdot (8 + WL_0)$) is because both level 1 and level 0 are computed at the same time as explained in subsection 4.5.5. 4 is due to the fact, that in level 0 the vectorization is always a multiple of 4 compared to the one of level 1 as also explained in subsection 4.5.5. It can be seen that the trans-floating implementation demands more external memory bandwidth, as two pyramid levels are computed at the same time which highly impacts the bandwidth.

For the external memory bandwidth results, the OpenCL Profiler [164] provided by Intel for the FPGA was used. The validity of our theoretical model for the EMB is 72%. As stated in [144] which performed an exploration regarding the intel FPGAs external memory estimation, there is a degradation in the effectiveness of the external memory controller when multiple data are read and written back to the memory. This degradation can not be predicted in our model. This bottleneck is even higher in our case because our FPGA has only one external memory, whereas the one of [144] has two which doubles the external memory bandwidth capabilities. Another issue that we faced, is that we have to read from the external memory words that are not always power of two. To tackle this bottleneck we align data words together, as it is proposed by the Intel Manual [135].

However, this can not be added in the model. For the F_{32} and F_{16}, the prediction is better because of the aforementioned reason. The occupancy of the external memory is around 90% as given by the OpenCl Profiler. Finally, we have to mention that there are on average 5% stalls in the reads and writes which also impacts the utilization prediction.

6.1.5 *Execution Time and Throughput*

The total execution time of the whole algorithm computation is modeled by Equation (49).

$$T = \frac{H \cdot W \cdot (i_0 + \frac{i_1}{4} + \frac{i_2}{16})}{f \cdot \Pi(Q)} + \frac{(\frac{H \cdot W}{16} + \frac{H \cdot W}{4})}{f \cdot Q_{down}} \tag{49}$$

The term $\frac{(\frac{H \cdot W}{16} + \frac{H \cdot W}{4})}{f \cdot \Pi_{down}}$ refers to the computation time of the down-sampling operation, while the rest of the terms refer to computation of the three pyramid levels. Q_{down} refers to the vectorization of the down-sampling core. We can see that the computation time depends on the number of the H&S cores ($\Pi(Q)$) and the computation frequency (f).

In the case of trans-floating designs the computation time of the pyramid (the computation time for the down-sampling is the same as in the non trans-floating design) is given by Equation (50).

$$T = \frac{H \cdot W \cdot (\frac{t_2}{16})}{f \cdot \Pi_{2,1}(Q_{2,1})} + \frac{H \cdot W \cdot t_0}{f \cdot \Pi_0(Q_0)} \tag{50}$$

It can be seen by these equations, that level 0 and level 1 are computed at the same time as also described in subsection 4.5.5 and contrary to the non-transfloating implementation where each level is computed separately. This is essential as the computation of the algorithm is furthered accelerated.

The throughput of the algorithm can be modeled by Equation (51) in pixel per second

$$\mathrm{throughput} = \frac{1}{T} \cdot H \cdot W \tag{51}$$

The term $\frac{1}{T}$ corresponds to how many times the algorithm can run in one second an it also gives the value of the FPS. The throughput's validity is 79%. This happens due to the fact that the external memory highly impacts the result as it was stated in chapter 4.

6.1.6 *Design Space Exploration and OpenCL*

We implement the previous equations in a script, with the help of which we perform our design space exploration. Our theoretical model takes as inputs: the desired throughput,

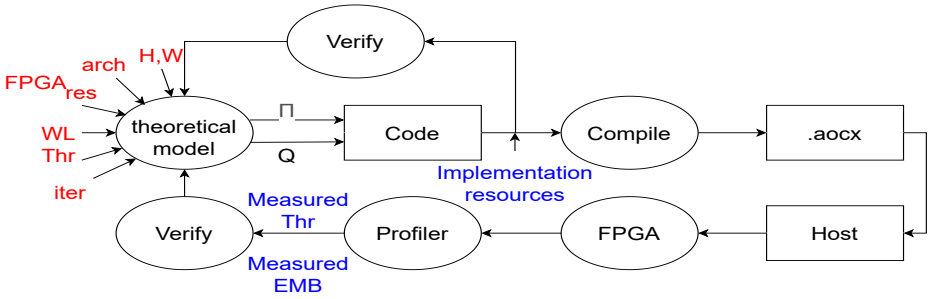

Figure 34: Design Space Exploration Steps

Algorithm 5 Algorithm of the theoretical model

Input: $H, W, thr, FPGA_{res}, arch, iter, WL$
Output: $\Pi(Q)$
Choose the correct file regarding the WL and Arch

for $\Pi(Q) = 500 \rightarrow \Pi(Q) = 1$ **do**
if $thr_{est} \approx thr(iter, H, W)$ **then**

  if $FPGA_{est} \leqslant FPGA_{res}$ **then**
    $mem[i] = (FPGA_{est}, \Pi(Q))$; return;
  end if
end if
keep mem[i] with $FGPA_{est_{min}}$

the image dimension,the iterations numbers, the FPGA devices resources, the arithmetic format and the desired architecture (trans-floating or no) as shown in Figure 34. Given these inputs, our script is able to determine: the level of the parallelism (Q) and the depth of the pipeline (Π). The algorithm for our theoretical model is described by Algorithm 5. These two factors are given to our OpenCL code and after the compilation ends, the DSPs number, the Block RAMs, the LUTs and the Registers numbers are provided by the Intel tool. The depth of the pipeline is implemented with the auto-run kernel option [83] and the vectorization by manual unroll [83] of the code. Since the compilation of the OpenCL code takes a long time, if the goal is to verify the estimation for the resources utilization, the compilation can be stopped when the resources file is ready as explained by the OpenCL manual [135] and as shown in Figure 34. Following the compilation, the host code is tuned and the executable files are ready for the testing in the FPGA device. In order to measure the external memory bandwidth and the throughput, the Intel profiler [164] is used. These two values can be used to verify our theoretical model as shown in Figure 34.

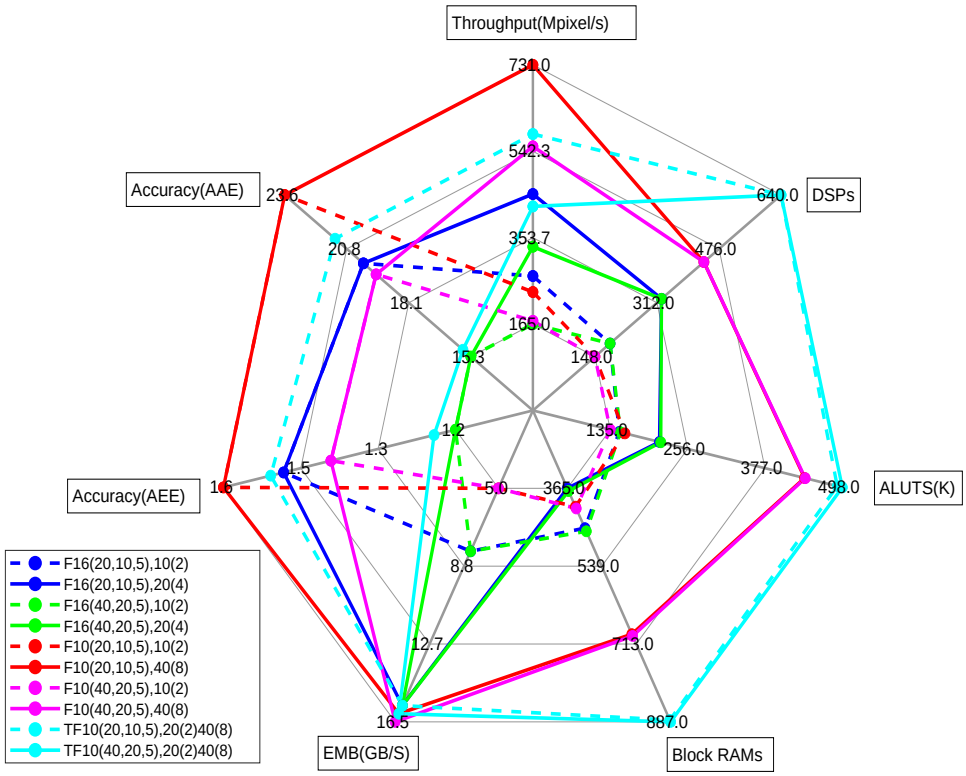

Figure 35: Results

The OpenCL code can be tuned, in order to provide a huge number of different designs, and since sometimes the goal is to verify the estimation of the theoretical model for the resources utilization, the time required for the exploration is negligible. This code helps the optical flow designer choose among a variety of options such as, the throughput, the floating point format and information about the accuracy. For example, if the goal of the designer is the high throughput, then our theoretical model suggest a large number of cores, the number of which is only constrained by the capabilities of the external memory bandwidth of the design. On the contrary, if the aim is a small design, then the smaller possible floating point format is chosen and the number of cores is set in order to provide real time computation with the least possible resources utilization. This number is strictly related to the dimensions of the computed image, as larger images demand more computation power and so more cores.

Our results, which are presented in section 6.2, are obtained by using this code. A lot of different configuration were tested and a different aim every time (for example, higher throughput).

6.2 RESULTS

In Figure 35 the FPGA running results regarding our implementations are shown in a spider plot for the Terasic Han Pilot Platform FPGA Arria 10 device. We include results regarding the throughput, DSPs, Logic Blocks, Block RAMs utilization, external memory bandwidth and accuracy. In Figure 35, FN denotes the floating point arithmetic format used, (L_2, L_1, L_0) the number of iterations in each pyramid level and Π(Q), the number of H&S cores and vectorization respectively. During this thesis, at least one thousand configurations where implemented in the FPGA with the help of the aforementioned OpenCL code, but in the spider plot we provide the information of those, which according to us are the most important.

As explained in chapter 3 and 4, implementations which achieve higher throughput, are less effective in terms of accuracy because more iterations are performed which increases the computation time. Implementations with smaller floating point formats are using less hardware resources than F_{16} in terms of DSPs and especially Logic Blocks and Block RAMs as also explained in [129] and in chapter 5. It is also obvious, that with the trans-floating (TF_{10}) architecture we achieve higher throughput compared to F_{16} as we are able to fit in the device a larger number of H&S cores. The accuracy of the detected flow is also comparable with F_{16} designs as F_{16} is used in the pyramid levels where the larger motion displacement is detected as explained in chapter 3. However, the number of Logic Blocks and Block RAMs is increased because the same resources are not used in every level of the pyramid. Our units are not able to compute both in F_{16} and F_9 and this we consider as future work in order to reduce the resources utilization.

If a high throughput design is desired, then the F_{10} design should be used with Π=40 which is denoted by the red line in the spider graph. As we have shown in chapter 4, this design outperforms most of the previous state of the art FPGA designs as we are able to fit a large number of H&S cores in the device. If accuracy is the main objective, then the F_{16} designs should be preferred as the average AAE and AEE are the smallest compared to all the designs (light green lines). When both accuracy and throughput are important then the TF_{10} offers comparable accuracy compared to F_{16} and is able to achieve a throughput up to 422 Mpixel/s (light blue lines). However, this architecture demands a large FPGA device as the needs for resources is increased and can not be fulfilled by small devices. If a small FPGA device is considered, then F_{10} with Π=10 should be the choice as the resources utilization is relatively small compared to the other designs (pink and red dotted lines) and it offers real time computation as the same time. Finally, it can be seen in the plot, that we try to optimize the designs in terms of Block RAMs utilization and that is why EMB is very high.

6.3 CONCLUSION

In this chapter, we have described how we perform the design space exploration for the multi-scale H&S algorithm. We described our theoretical model that estimate the DSPs, Block RAMs and external memory bandwidth utilization and which are able to predict the throughput of our algorithm. We detailed how our exploration is performed, how accurate our model is and how our OpenCL code can be used in order to accelerate the exploration. With this code, we have produced at least one thousand designs. The code is written in a parametric way to accelerate the exploration and to test different architectures with a different aim every time. We proposed a way, the spider plot, which contains the aggregated information of our chosen designs. The designer can use the spider plot, in order to choose among different implementations which fit the constraints that are set by his problem.

In the next chapter, we will conclude this thesis and we will describe our future work and perspectives as this is an ongoing research.

7 CONCLUSION AND FUTURE WORK

In this thesis, we have analyzed and we have proposed solutions regarding the accuracy, the throughput and the resources utilization of the design of the multi-scale H&S algorithm in an FPGA. We have used these propositions and analysis in order to perform a design space exploration of the algorithm in the HAN Pilot Platform Arria 10 FPGA. In this chapter, we will conclude this thesis contributions for the acceleration of the algorithm. Following that, we will discuss on how these contributions can be applied to other algorithms in order to accelerate their computation. Finally, we will discuss about the future work that can be conducted in the acceleration of the multi-scale H&S algorithm in FPGAs and some future ideas that can be applied to other algorithms too.

7.1 CONCLUSION

Optical flow computation remains a very important algorithm for real life applications like autonomous driving and biomedical purposes and it requires accurate and real time computations. The computations in these domains are usually performed by embedded devices as there is not always the possibility to install large computers to perform them. However, computing optical flow in real time and with embedded devices demands trade-offs to be made in the design and new solution to be proposed in order to overcome certain bottlenecks of the implementation. Concurrently, the objective of this thesis was to explore and propose solutions for the multi-scale H&S optical flow algorithm in order to be accelerated in a FPGA device. This algorithm offers high tunability to the designer which makes it an ideal algorithm to be accelerated in FPGAs.

The first thing that we explored was the accuracy. We first started with the levels of the pyramid and the iterations number in each pyramid level. We came up with the conclusion that a 3 pyramid level, more iterations in the smaller images and less in the larger ones gives us acceptable results compared to when we perform 4 levels pyramids and many iterations in each pyramid level. Bi-linear and bi-cubic interpolations were considered for warping and we found that using bi-linear interpolation for F_{32} and F_{16} is enough for the accuracy and for smaller numeric formats bi-cubic is more effective. We explored different floating point arithmetic formats and we showed that even with F_9, we detect the optical flow with higher accuracy compared to other state of the art works. Trans-floating arithmetic was used and we have shown that using smaller arithmetic formats in the finest level of the pyramid does not affect heavily the detection of the flow.

In order to accelerate the algorithm in terms of throughput, the first thing that we employed was the deep pipeline and vectorization technique for the ISL part of the algorithm. With this architecture, the iterative nature of the algorithm is tackled as the iterations are fused. For the warping, we store in the on-chip memory enough data with the aim of having a continuous pipeline without stalls. For the multi-rate nature of the algorithm, we proposed a solution for the handling of the Block RAMs in order to perform continuous reads in the external memory which highly benefits the throughput. With the trans-floating architecture, we are able to fit more cores for the computation of the finest pyramid level, which demands the most computation time. Regarding the throughput, we performed a comparison of our designs with other state of the art works and we showed that our fastest one outperforms all the single FPGA designs as far as we know in terms of throughput even for large images.

Since the increase in throughput demands more hardware resources, we performed an exploration regarding the FPGA resources utilization. We extensively used computation reuse in the H&S core to decrease the number of DSPs usage. We proposed a trade off between the depth of the pipeline and the vectorization to reduce the Block RAMs utilization. For the warping, bi-linear interpolation demands far less resources compare to the bi-cubic one. For both the H&S and the warping core, we have shown the impact of the arithmetic format in the DSPs, Logic Block and Block RAMs utilization. With a smaller arithmetic format, more words can be read and written back to the external memory, which benefits vectorization and consequently throughput. The careful handling of the on-chip memory was explained for the effective design of multi-rate architectures. Moreover, the resources utilization of the trans-floating architecture was discussed and we did a thorough comparison in terms of resources of our work with other state of the art works.

Since the goal of this thesis is to perform a design space exploration on the multi-scale H&S optical flow algorithm, we used all the analysis that we have done in order to built a tool which would estimate the hardware resources regarding the throughput and the numeric format. The exploration was performed with a tunable code in OpenCL. By using this code, we produced at least one thousand architectures in order to test factors of the design. Finally, we presented a way (Figure 35) where the designer can choose between different architectures depending on its constraints.

The trans-floating architecture can be applied to many algorithms where the accuracy is not of same importance in all the levels of the algorithm. For example, it can be applied to other multi-scale image processing algorithms. With this way, higher throughput can be achieved with minimal losses in accuracy. The computation reuse in ISL algorithms is something that gained significant attention during the last few years in FPGAs as it helps in saving of DSPs and external memory bandwidth as intermediate results can be fused and not recomputed. Our proposition of handling the multi-rate architectures is able to better treat the external memory bandwidth compared to previous state of the art works which impacts positively the throughput. Multi-rate architectures are very common in

many image processing and not only pipelines and that is why we believe it is a strong contribution.

7.2 FUTURE WORK

At first, more exploration can be done regarding the multi-rate algorithm in terms of accuracy. A good improvement would be to increase the levels of the pyramid to 4 or 5 and to not do any computation in the finest level of the pyramid which demands the most computation time. This will increase the throughput of the algorithm and maybe the loss in accuracy will not be very high. Following that, fixed point arithmetic can be considered as a potential solution to decrease the usage of FPGA resources. However, more iterations might be needed to achieve convergence for the optical flow.

In the architectural part, we plan to fit the final results in the trans-floating architecture of level 0 in F_8 floating point arithmetic format. By doing that, the reads and writes from and to the external memory are more friendly to the memory bus, which might potentially increase the throughput. On the other site, the impact of this in the accuracy of the optical flow detection should be explored. An essential future work we consider, is to design IPs which will be able to make operations in more that one floating point formats. This will highly reduce the resources utilization. Since these IPs will be able to do computations in parallel for smaller floating point formats, this will increase the throughput of the computation. Moreover, we plan to use the Block RAMs in the same way in order to reduce their number too. This is a new trend in FPGAs especially for image processing application and we consider this future work very important not only for our case but for other algorithms too.

In order to compute the optical flow, other algorithms can be considered. Convolution neural networks might be a case but they demand a lot of trade offs in order to compute the flow for reasonable large images. One solution would be the binary neural networks which demand less computations. Finally, we plan to use our accelerator in a large image processing pipeline which is the meteor detection case.

BIBLIOGRAPHY

[1] N. Rambaux, J. Vaubaillon, L. Lacassagne, *et al.*, "Meteorix: a cubesat mission dedicated to the detection of meteors and space debris," in *1st ESA NEO and Debris Detection Conference*, Darmstadt, Germany: ESA Space Safety Programme Office, Jan. 2019, 9 p. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02198139.

[2] M. Millet, N. Rambaux, A. Cassagne, M. Bouyer, A. Petreto, and L. Lacassagne, "Meteorix: High performance computer vision application for Meteor detection from a CubeSat," in *44th Scientific Assembly of the Committee on Space Research (COSPAR)*, Athens, Greece, Jul. 2022. [Online]. Available: https://hal.archives-ouvertes.fr/hal-03737605.

[3] T. Romera, A. Petreto, F. Lemaitre, M. Bouyer, Q. Meunier, and L. Lacassagne, "Implementations impact on iterative image processing for embedded gpu," in *2021 29th European Signal Processing Conference (EUSIPCO)*, 2021, pp. 736–740. DOI: 10.23919/EUSIPCO54536.2021.9615947.

[4] M. Millet, N. Rambaux, A. Petreto, F. Lemaitre, and L. Lacassagne, "Meteorix - a new processing chain for real-time detection and tracking of meteors from space," *WGN, Journal of the International Meteor Organization*, vol. 49, no. 6, Dec. 2021.

[5] N. Rambaux, P. Keckhut, A. Hauchecorne, D. Galayko, G. Guignan, J. Vaubaillon, L. Lacassagne, M. Birlan, P. Boisse, M. Capderou, *et al.*, "Meteorix: A cubesat mission dedicated to the detection of meteors," *42nd COSPAR Scientific Assembly*, vol. 42, Bo–2, 2018.

[6] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "Flownet 2.0: Evolution of optical flow estimation with deep networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2462–2470.

[7] T.-W. Hui, X. Tang, and C. C. Loy, "Liteflownet: A lightweight convolutional neural network for optical flow estimation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8981–8989.

[8] Y. Ling, Y. Yan, K. Huang, and G. Chen, "Flowacc: Real-time high-accuracy dnn-based optical flow accelerator in fpga," in *2022 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2022, pp. 112–115. DOI: 10.23919/DATE54114.2022.9774506.

[9] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3354–3361. DOI: 10.1109/CVPR.2012.6248074.

[10] A. Giachetti, M. Campani, and V. Torre, "The use of optical flow for road navigation," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 1, pp. 34–48, 1998. DOI: 10.1109/70.660838.

[11] D. Kapsalis, O. Sename, V. Milanés, and J. J. Martinez Molina, "Gain-Scheduled Steering Control for Autonomous Vehicles," *IET Control Theory and Applications*, vol. 14, no. 20, pp. 3451 –3460, Feb. 2021. DOI: 10.1049/iet-cta.2020.0698. [Online]. Available: https://hal.univ-grenoble-alpes.fr/hal-02904666.

[12] Z. Sun, G. Bebis, and R. Miller, "On-road vehicle detection: A review," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 5, pp. 694–711, 2006. DOI: 10.1109/TPAMI.2006.104.

[13] H. Chao, Y. Gu, and M. Napolitano, "A survey of optical flow techniques for robotics navigation applications," *Journal of Intelligent & Robotic Systems*, vol. 73, no. 1, pp. 361–372, 2014.

[14] W. Enkelmann, "Obstacle detection by evaluation of optical flow fields from image sequences," *Image and Vision Computing*, vol. 9, no. 3, pp. 160–168, 1991.

[15] M. Xavier, A. Lalande, P. M. Walker, F. Brunotte, and L. Legrand, "An adapted optical flow algorithm for robust quantification of cardiac wall motion from standard cine-mr examinations," *IEEE Transactions on Information Technology in Biomedicine*, vol. 16, no. 5, pp. 859–868, 2012. DOI: 10.1109/TITB.2012.2204893.

[16] T.-C. Huang, C.-K. Chang, C.-H. Liao, and Y.-J. Ho, "Quantification of blood flow in internal cerebral artery by optical flow method on digital subtraction angiography in comparison with time-of-flight magnetic resonance angiography," *PloS one*, vol. 8, no. 1, e54678, 2013.

[17] D. Rueckert, L. Sonoda, C. Hayes, D. Hill, M. Leach, and D. Hawkes, "Nonrigid registration using free-form deformations: Application to breast mr images," *IEEE Transactions on Medical Imaging*, vol. 18, no. 8, pp. 712–721, 1999. DOI: 10.1109/42.796284.

[18] A. Sotiras, C. Davatzikos, and N. Paragios, "Deformable medical image registration: A survey," *IEEE Transactions on Medical Imaging*, vol. 32, no. 7, pp. 1153–1190, 2013. DOI: 10.1109/TMI.2013.2265603.

[19] F. Amat, E. W. Myers, and P. J. Keller, "Fast and robust optical flow for time-lapse microscopy using super-voxels," *Bioinformatics*, vol. 29, no. 3, pp. 373–380, 2013.

[20] D. Fortun, C. Chen, P. Paul-Gilloteaux, F. Waharte, J. Salamero, and C. Kervrann, "Correlation and variational approaches for motion and diffusion estimation in fluorescence imaging," in *21st European Signal Processing Conference (EUSIPCO 2013)*, 2013, pp. 1–5.

[21] J. Delpiano, J. Jara, J. Scheer, O. A. Ramírez, J. Ruiz-del Solar, and S. Härtel, "Performance of optical flow techniques for motion analysis of fluorescent point signals in confocal microscopy," *Machine Vision and Applications*, vol. 23, no. 4, pp. 675–689, 2012.

[22] D. Fortun, P. Bouthemy, P. Paul-Gilloteaux, and C. Kervrann, "Aggregation of patch-based estimations for illumination-invariant optical flow in live cell imaging," in *2013 IEEE 10th International Symposium on Biomedical Imaging*, 2013, pp. 660–663. DOI: 10.1109/ISBI.2013.6556561.

[23] K. Liu, S. S. Lienkamp, A. Shindo, J. B. Wallingford, G. Walz, and O. Ronneberger, "Optical flow guided cell segmentation and tracking in developing tissue," in *2014 IEEE 11th International Symposium on Biomedical Imaging (ISBI)*, 2014, pp. 298–301. DOI: 10.1109/ISBI.2014.6867868.

[24] N.-M. T. Kokolakis, A. Kanellopoulos, and K. G. Vamvoudakis, "Bounded rational unmanned aerial vehicle coordination for adversarial target tracking," in *2020 American Control Conference (ACC)*, 2020, pp. 2508–2513. DOI: 10.23919/ACC45564.2020.9147737.

[25] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, 2005, 886–893 vol. 1. DOI: 10.1109/CVPR.2005.177.

[26] A. Basset, P. Bouthemy, and C. Kervrann, "Recovery of motion patterns and dominant paths in videos of crowded scenes," in *2014 IEEE International Conference on Image Processing (ICIP)*, 2014, pp. 184–188. DOI: 10.1109/ICIP.2014.7025036.

[27] N. Kiryati, T. R. Raviv, Y. Ivanchenko, and S. Rochel, "Real-time abnormal motion detection in surveillance video," in *2008 19th International Conference on Pattern Recognition*, 2008, pp. 1–4. DOI: 10.1109/ICPR.2008.4761138.

[28] M. Tassano, J. Delon, and T. Veit, "Fastdvdnet: Towards real-time video denoising without explicit motion estimation," 2019.

[29] C. Liu and W. T. Freeman, "A high-quality video denoising algorithm based on reliable motion estimation," in *European conference on computer vision*, Springer, 2010, pp. 706–719.

[30] C. Zuo, Y. Liu, X. Tan, W. Wang, and M. Zhang, "Video denoising based on a spatiotemporal kalman-bilateral mixture model," *The Scientific World Journal*, vol. 2013, 2013.

[31] H. Wang, A. Kläser, C. Schmid, and C.-L. Liu, "Action recognition by dense trajectories," in *CVPR 2011*, 2011, pp. 3169–3176. DOI: 10.1109/CVPR.2011.5995407.

[32] M. Jain, H. Jégou, and P. Bouthemy, "Better exploiting motion for better action recognition," in *2013 IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 2555–2562. DOI: 10.1109/CVPR.2013.330.

[33] W. Hu, N. Xie, L. Li, X. Zeng, and S. Maybank, "A survey on visual content-based video indexing and retrieval," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 41, no. 6, pp. 797–819, 2011. DOI: 10.1109/TSMCC.2011.2109710.

[34] D. Shulman and J.-Y. Herve, "Regularization of discontinuous flow fields," in *[1989] Proceedings. Workshop on Visual Motion*, 1989, pp. 81–86. DOI: 10.1109/WVM.1989.47097.

[35] R. Gal, N. Kiryati, and N. Sochen, "Progress in the restoration of image sequences degraded by atmospheric turbulence," *Pattern Recognition Letters*, vol. 48, pp. 8–14, 2014, Celebrating the life and work of Maria Petrou, ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2014.04.007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167865514001251.

[36] T. Corpetti, E. Memin, and P. Perez, "Dense estimation of fluid flows," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 3, pp. 365–380, 2002. DOI: 10.1109/34.990137.

[37] D. Heitz, E. Mémin, and C. Schnörr, "Variational fluid flow measurements from image sequences: Synopsis and perspectives," *Experiments in fluids*, vol. 48, no. 3, pp. 369–393, 2010.

[38] N. Maurice, F. Lemaitre, J. Sopena, and L. Lacassagne, "Lsl3d: A run-based connected component labeling algorithm for 3d volumes," in *Image Analysis and Processing. ICIAP 2022 Workshops*, P. L. Mazzeo, E. Frontoni, S. Sclaroff, and C. Distante, Eds., Cham: Springer International Publishing, 2022, pp. 132–142, ISBN: 978-3-031-13324-4.

[39] F. Lemaitre, A. Hennequin, and L. Lacassagne, "Taming voting algorithms on gpus for an efficient connected component analysis algorithm," in *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021, pp. 7903–7907. DOI: 10.1109/ICASSP39728.2021.9413653.

[40] P. Turaga, R. Chellappa, and A. Veeraraghavan, "Advances in video-based human activity analysis: Challenges and approaches," in *Advances in Computers*, ser. Advances in Computers, M. V. Zelkowitz, Ed., vol. 80, Elsevier, 2010, pp. 237–290. DOI: https://doi.org/10.1016/S0065-2458(10)80007-5. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0065245810800075.

[41] 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, Image Understanding for Real-world Distributed Video Networks, ISSN: 1077-3142. DOI: https://doi.org/10.1016/j.cviu.2015.02.008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1077314215000429.

[42] B. D. Lucas, T. Kanade, *et al.*, *An iterative image registration technique with an application to stereo vision*. Vancouver, 1981, vol. 81.

[43] K. Blachut and T. Kryjak, "Real-time efficient fpga implementation of the multi-scale lucas-kanade and horn-schunck optical flow algorithms for a 4k video stream," *Sensors*, vol. 22, no. 13, 2022, ISSN: 1424-8220. DOI: 10.3390/s22135017. [Online]. Available: https://www.mdpi.com/1424-8220/22/13/5017.

[44] 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, pp. 713–730, 2016.

[45] B. K. Horn and B. G. Schunck, "Determining optical flow," *Artificial Intelligence*, vol. 17, no. 1, pp. 185 –203, 1981.

[46] C. Zach, T. Pock, and H. Bischof, "A duality based approach for realtime tv-l 1 optical flow," in *Joint pattern recognition symposium*, Springer, 2007, pp. 214–223.

[47] M. Tomasi, M. Vanegas, F. Barranco, J. Diaz, and E. Ros, "Real-Time Architecture for a Robust Multi-Scale Stereo Engine on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 12, pp. 2208–2219, 2012.

[48] P. Anandan, *Measuring visual motion from image sequences*. University of Massachusetts Amherst, 1987.

[49] 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, 1990.

[50] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazirbas, V. Golkov, P. v. d. Smagt, D. Cremers, and T. Brox, "Flownet: Learning optical flow with convolutional networks," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 2758–2766. DOI: 10.1109/ICCV.2015.316.

[51] Z. Sun and H. Wang, "Deeper spatial pyramid network with refined up-sampling for optical flow estimation," in *Pacific Rim Conference on Multimedia*, Springer, 2018, pp. 492–501.

[52] T.-W. Hui, X. Tang, and C. C. Loy, "A lightweight optical flow cnn—revisiting data fidelity and regularization," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 8, pp. 2555–2569, 2020.

[53] A. Ranjan and M. J. Black, "Optical flow estimation using a spatial pyramid network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4161–4170.

[54] D.Scharstein and R. Szeliski, "A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms," *International Journal of Computer Vision*, vol. 47, pp. 7–42, 2002.

[55] A. Petreto, T. Romera, F. Lemaitre, I. Masliah, B. Gaillard, M. Bouyer, Q. L. Meunier, and L. Lacassagne, "A new real-time embedded video denoising algorithm," in *2019 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, 2019, pp. 47–52. DOI: 10.1109/DASIP48288.2019.9049189.

[56] S. Smets, T. Goedemé, and M. Verhelst, "Custom processor design for efficient, yet flexible lucas-kanade optical flow," in *2016 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, 2016, pp. 138–145. DOI: 10.1109/DASIP.2016.7853810.

[57] F. Zhang, Y. Gao, and J. D. Bakos, "Lucas-kanade optical flow estimation on the ti c66x digital signal processor," in *2014 IEEE High Performance Extreme Computing Conference (HPEC)*, 2014, pp. 1–6. DOI: 10.1109/HPEC.2014.7040984.

[58] B. Duvenhage, J. P. Delport, and J. de Villiers, "Implementation of the lucas-kanade image registration algorithm on a gpu for 3d computational platform stabilisation," in *Proceedings of the 7th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa*, ser. AFRIGRAPH '10, Franschhoek, South Africa: Association for Computing Machinery, 2010, 83–90, ISBN: 9781450301183. DOI: 10.1145/1811158.1811172. [Online]. Available: https://doi.org/10.1145/1811158.1811172.

[59] H. Seong, C. E. Rhee, and H. Lee, "A Novel Hardware Architecture of the Lucas–Kanade Optical Flow for Reduced Frame Memory Access," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 26, no. 6, pp. 1187–1199, 2016.

[60] 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.

[61] I. Ishii, T. Taniguchi, K. Yamamoto, and T. Takaki, "High-Frame-Rate Optical Flow System," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 1, pp. 105–112, 2012.

[62] 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, 2018.

[63] 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.

[64] A. Petreto, A. Hennequin, T. Koehler, T. Romera, Y. Fargeix, B. Gaillard, M. Bouyer, Q. L. Meunier, and L. Lacassagne, "Energy and Execution Time Comparison of Optical Flow Algorithms on SIMD and GPU architectures," in *2018 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, 2018, pp. 25–30.

[65] R. Rustam, N. H. Hamid, and F. A. Hussin, "FPGA-based hardware implementation of optical flow constraint equation of Horn and Schunck," in *2012 4th International Conference on Intelligent and Advanced Systems (ICIAS 2012)*, vol. 2, 2012, pp. 790–794.

[66] M. Kunz, A. Ostrowski, and P. Zipf, "An FPGA-optimized architecture of horn and schunck optical flow algorithm for real-time applications," in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, 2014, pp. 1–4.

[67] 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.

[68] D. Huff, S. Dai, and P. Hanrahan, "Clockwork: Resource-efficient static scheduling for multi-rate image processing applications on fpgas," in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2021, pp. 186–194. DOI: 10.1109/FCCM51124.2021.00030.

[69] A. Nguyen, N. Satish, J. Chhugani, C. Kim, and P. Dubey, "3.5-d blocking optimization for stencil computations on modern cpus and gpus," in *SC '10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, 2010, pp. 1–13. DOI: 10.1109/SC.2010.2.

[70] G. Jin, T. Endo, and S. Matsuoka, "A parallel optimization method for stencil computation on the domain that is bigger than memory capacity of gpus," in *2013 IEEE International Conference on Cluster Computing (CLUSTER)*, 2013, pp. 1–8. DOI: 10.1109/CLUSTER.2013.6702633.

[71] J. Pekkilä, M. S. Väisälä, M. J. Käpylä, M. Rheinhardt, and O. Lappi, "Scalable communication for high-order stencil computations using cuda-aware mpi," *Parallel Computing*, vol. 111, p. 102 904, 2022.

[72] L. Peng, R. Seymour, K. ichi Nomura, R. K. Kalia, A. Nakano, P. Vashishta, A. Loddoch, M. Netzband, W. R. Volz, and C. C. Wong, "High-order stencil computations on multicore clusters," in *2009 IEEE International Symposium on Parallel Distributed Processing*, 2009, pp. 1–11. DOI: 10.1109/IPDPS.2009.5161011.

[73] X. Liu, Y. Liu, H. Yang, J. Liao, M. Li, Z. Luan, and D. Qian, "Toward accelerated stencil computation by adapting tensor core unit on gpu," in *Proceedings of the 36th ACM International Conference on Supercomputing*, 2022, pp. 1–12.

[74] A. Gorobets and P. Bakhvalov, "Heterogeneous cpu+ gpu parallelization for high-accuracy scale-resolving simulations of compressible turbulent flows on hybrid supercomputers," *Computer Physics Communications*, vol. 271, p. 108 231, 2022.

[75] R. Cattaneo, G. Natale, C. Sicignano, D. Sciuto, and M. D. Santambrogio, "On how to accelerate iterative stencil loops: A scalable streaming-based approach," *ACM Trans. Archit. Code Optim.*, vol. 12, no. 4, Dec. 2015, ISSN: 1544-3566. DOI: 10.1145/2842615. [Online]. Available: https://doi.org/10.1145/2842615.

[76] G. Natale, G. Stramondo, P. Bressana, R. Cattaneo, D. Sciuto, and M. D. Santambrogio, "A polyhedral model-based framework for dataflow implementation on fpga devices of iterative stencil loops," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2016, pp. 1–8. DOI: 10.1145/2966986.2966995.

[77] X. Tian, Z. Ye, A. Lu, L. Guo, Y. Chi, and Z. Fang, "Sasa: A scalable and automatic stencil acceleration framework for optimized hybrid spatial and temporal parallelism on hbm-based fpgas," *arXiv preprint arXiv:2208.10770*, 2022.

[78] K. Kamalakkannan, G. R. Mudalige, I. Z. Reguly, and S. A. Fahmy, "Fpga acceleration of structured-mesh-based explicit and implicit numerical solvers using sycl," in *International Workshop on OpenCL*, ser. IWOCL'22, Bristol, United Kingdom, United Kingdom: Association for Computing Machinery, 2022, ISBN: 9781450396585. DOI: 10.1145/3529538.3530007. [Online]. Available: https://doi.org/10.1145/3529538.3530007.

[79] H. M. Waidyasooriya, Y. Takei, S. Tatsumi, and M. Hariyama, "Opencl-based fpga-platform for stencil computation and its optimization methodology," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 5, pp. 1390–1402, 2017. DOI: 10.1109/TPDS.2016.2614981.

[80] H. R. Zohouri, N. Maruyama, A. Smith, M. Matsuda, and S. Matsuoka, "Evaluating and optimizing opencl kernels for high performance computing with fpgas," in *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2016, pp. 409–420. DOI: 10.1109/SC.2016.34.

[81] K. Sano, Y. Hatsuda, and S. Yamamoto, "Multi-fpga accelerator for scalable stencil computation with constant memory bandwidth," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 695–705, 2014. DOI: 10.1109/TPDS.2013.51.

[82] H. R. Zohouri, A. Podobas, and S. Matsuoka, "High-performance high-order stencil computation on fpgas using opencl," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2018, pp. 123–130. DOI: 10.1109/IPDPSW.2018.00027.

[83] H. R. Zohouri, A. Podobas, and S. Matsuoka, "Combined spatial and temporal blocking for high-performance stencil computation on fpgas using opencl," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2018, 153–162.

[84] G. Deest, T. Yuki, S. Rajopadhye, and S. Derrien, "One size does not fit all: Implementation trade-offs for iterative stencil computations on fpgas," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017, pp. 1–8. DOI: 10.23919/FPL.2017.8056781.

[85] Y. Chi, J. Cong, P. Wei, and P. Zhou, "Soda: Stencil with optimized dataflow architecture," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018, pp. 1–8. DOI: 10.1145/3240765.3240850.

[86] Y. Chi and J. Cong, "Exploiting computation reuse for stencil accelerators," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6. DOI: 10.1109/DAC18072.2020.9218680.

[87] A. A. Nacci, V. Rana, F. Bruschi, D. Sciuto, P. di Milano, I. Beretta, and D. Atienza, "A high-level synthesis flow for the implementation of iterative stencil loop algorithms on fpga devices," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2013, pp. 1–6. DOI: 10.1145/2463209.2488797.

[88] A. Akin, I. Beretta, A. A. Nacci, V. Rana, M. D. Santambrogio, and D. Atienza, "A high-performance parallel implementation of the chambolle algorithm," in *2011 Design, Automation Test in Europe*, 2011, pp. 1–6. DOI: 10.1109/DATE.2011.5763232.

[89] V. Rana, A. A. Nacci, I. Beretta, M. D. Santambrogio, D. Atienza, and D. Sciuto, "Design methods for parallel hardware implementation of multimedia iterative algorithms," *IEEE Design Test of Computers*, vol. 30, no. 04, pp. 71–80, 2013, ISSN: 1558-1918. DOI: 10.1109/MDT.2012.2223191.

[90] H. M. Waidyasooriya and M. Hariyama, "Multi-fpga accelerator architecture for stencil computation exploiting spacial and temporal scalability," *IEEE Access*, vol. 7, pp. 53 188–53 201, 2019. DOI: 10.1109/ACCESS.2019.2910824.

[91] J. Cong, J. Lau, G. Liu, S. Neuendorffer, P. Pan, K. Vissers, and Z. Zhang, "Fpga hls today: Successes, challenges, and opportunities," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 15, no. 4, 2022, ISSN: 1936-7406. DOI: 10.1145/3530775. [Online]. Available: https://doi.org/10.1145/3530775.

[92] J. Pu, S. Bell, X. Yang, J. Setter, S. Richardson, J. Ragan-Kelley, and M. Horowitz, "Programming heterogeneous systems from an image processing dsl," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 3, pp. 1–25, 2017.

[93] N. Chugh, V. Vasista, S. Purini, and U. Bondhugula, "A dsl compiler for accelerating image processing pipelines on fpgas," in *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation*, 2016, pp. 327–338.

[94] J. Hegarty, R. Daly, Z. DeVito, J. Ragan-Kelley, M. Horowitz, and P. Hanrahan, "Rigel: Flexible multi-rate image processing hardware," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, pp. 1–11, 2016.

[95] B. Johnson and S. R. J, "A high throughput fully parallel-pipelined fpga accelerator for dense cloud motion analysis," in *2016 IEEE Region 10 Conference (TENCON)*, 2016, pp. 2589–2592. DOI: 10.1109/TENCON.2016.7848505.

[96] B. Johnson, S. Thomas, and R. Sheeba, "A high-performance dense optical flow architecture based on red-black sor solver," *Journal of Signal Processing Systems*, pp. 357–373, 2020.

[97] 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, ISSN: 1077-3142. DOI: https://doi.org/10.1016/j.cviu.2008.05.006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1077314208000684.

[98] H.-H. Nagel and W. Enkelmann, "An investigation of smoothness constraints for the estimation of displacement vector fields from image sequences," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 5, pp. 565–593, 1986. DOI: 10.1109/TPAMI.1986.4767833.

[99] J. R. Bergen, P. Anandan, K. J. Hanna, and R. Hingorani.

[100] S. Piskorski, L. Lacassagne, S. Bouaziz, and D. Etiemble, "Customizing CPU Instructions for Embedded Vision Systems," in *2006 International Workshop on Computer Architecture for Machine Perception and Sensing*, 2006, pp. 59–64.

[101] B. Barrois, O. Sentieys, and D. Menard, "The hidden cost of functional approximation against careful data sizing — a case study," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, 2017, pp. 181–186. DOI: 10.23919/DATE.2017.7926979.

[102] F. de Dinechin and B. Pasca, "Designing Custom Arithmetic Data Paths with FloPoCo," *IEEE Design Test of Computers*, vol. 28, no. 4, pp. 18–27, 2011.

[103] C. Rhemann, C. Rother, J. Wang, M. Gelautz, P. Kohli, and P. Rott, "A perceptually motivated online benchmark for image matting," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 1826–1833. DOI: 10.1109/CVPR.2009.5206503.

[104] S. Baker and T. Kanade, "Limits on super-resolution and how to break them," in *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No.PR00662)*, vol. 2, 2000, 372–379 vol.2. DOI: 10.1109/CVPR.2000.854852.

[105] A. Wedel, D. Cremers, T. Pock, and H. Bischof, "Structure- and motion-adaptive regularization for high accuracy optic flow," in *2009 IEEE 12th International Conference on Computer Vision*, 2009, pp. 1663–1668. DOI: 10.1109/ICCV.2009.5459375.

[106] Z. Wang and A. Bovik, "A universal image quality index," *IEEE Signal Processing Letters*, vol. 9, no. 3, pp. 81–84, 2002. DOI: 10.1109/97.995823.

[107] H. Sheikh, M. Sabir, and A. Bovik, "A statistical evaluation of recent full reference image quality assessment algorithms," *IEEE Transactions on Image Processing*, vol. 15, no. 11, pp. 3440–3451, 2006. DOI: 10.1109/TIP.2006.881959.

[108] Z. Wang and A. C. Bovik, "Mean squared error: Love it or leave it? a new look at signal fidelity measures," *IEEE Signal Processing Magazine*, vol. 26, no. 1, pp. 98–117, 2009. DOI: 10.1109/MSP.2008.930649.

[109] S. Chang, B. Yu, and M. Vetterli, "Adaptive wavelet thresholding for image denoising and compression," *IEEE Transactions on Image Processing*, vol. 9, no. 9, pp. 1532–1546, 2000. DOI: 10.1109/83.862633.

[110] M. Liu and T. Delbruck, "Block-matching optical flow for dynamic vision sensors: Algorithm and fpga implementation," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2017, pp. 1–4. DOI: 10.1109/ISCAS.2017.8050295.

[111] J. Barron, D. Fleet, S. Beauchemin, and T. Burkitt, "Performance of optical flow techniques," in *Proceedings 1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1992, pp. 236–242. DOI: 10.1109/CVPR.1992.223269.

[112] D. Sun, S. Roth, and M. J. Black, "A quantitative analysis of current practices in optical flow estimation and the principles behind them," *Int. J. Comput. Vision*, vol. 106, no. 2, 115–137, 2014, ISSN: 0920-5691. DOI: 10.1007/s11263-013-0644-x. [Online]. Available: https://doi.org/10.1007/s11263-013-0644-x.

[113] M. Tomasi, M. Vanegas, F. Barranco, J. Daz, and E. Ros, "Massive Parallel-Hardware Architecture for Multiscale Stereo, Optical Flow and Image-Structure Computation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 2, pp. 282–294, 2012.

[114] A. Garcia-Dopico, J. Pedraza, M. Nieto, A. Perez, S. Rodrigeuz, and J. Navas, "Parallelization of the optical flow computation in sequences from moving cameras," *EURASIP Journal on Image and Video Processing*, p. 18, 2014.

[115] S. Meyer, O. Wang, H. Zimmer, M. Grosse, and A. Sorkine-Hornung, "Phase-based frame interpolation for video," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1410–1418. DOI: 10.1109/CVPR.2015.7298747.

[116] R. Sun, P. Liu, J. Wang, C. Accetti, and A. A. Naqvi, "A 42fps full-hd orb feature extraction accelerator with reduced memory overhead," in *2017 International Conference on Field Programmable Technology (ICFPT)*, 2017, pp. 183–190. DOI: 10.1109/FPT.2017.8280137.

[117] Z. Pan, Y. Jin, X. Jiang, and J. Wu, "An fpga-optimized architecture of real-time farneback optical flow," in *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2020, pp. 223–223. DOI: 10.1109/FCCM48280.2020.00054.

[118] V. Suse and D. Ionescu, "A real-time reconfigurable architecture for face detection," in *2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, 2015, pp. 1–6. DOI: 10.1109/ReConFig.2015.7393281.

[119] K. Daniilidis, A. Makadia, and T. Bulow, "Image processing in catadioptric planes: Spatiotemporal derivatives and optical flow computation," in *Proceedings of the IEEE Workshop on Omnidirectional Vision 2002. Held in conjunction with ECCV'02*, 2002, pp. 3–10. DOI: 10.1109/OMNVIS.2002.1044483.

[120] Y. Hu, R. Song, and Y. Li, "Efficient coarse-to-fine patch match for large displacement optical flow," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 5704–5712. DOI: 10.1109/CVPR.2016.615.

[121] H.-T. Yau, M.-T. Lin, and M.-S. Tsai, "Real-time nurbs interpolation using fpga for high speed motion control," *Computer-Aided Design*, vol. 38, no. 10, pp. 1123–1133, 2006.

[122] M. A. Nuño-Maganda and M. O. Arias-Estrada, "Real-time fpga-based architecture for bicubic interpolation: An application for digital image scaling," in *2005 International Conference on Reconfigurable Computing and FPGAs (ReConFig'05)*, IEEE, 2005, 8–pp.

[123] K. S. Rani and W. J. Hans, "Fpga implementation of bilinear interpolation algorithm for cfa demosaicing," in *2013 International Conference on Communication and Signal Processing*, IEEE, 2013, pp. 857–863.

[124] D. Etiemble, S. Bouaziz, and L. Lacassagne, "Customizing 16-bit floating point instructions on a nios ii processor for fpga image and media processing," in *3rd Workshop on Embedded Systems for Real-Time Multimedia, 2005.*, 2005, pp. 61–66. DOI: 10.1109/ESTMED.2005.1518073.

[125] D. Zuras, M. Cowlishaw, A. Aiken, M. Applegate, D. Bailey, S. Bass, D. Bhandarkar, M. Bhat, D. Bindel, S. Boldo, *et al.*, "Ieee standard for floating-point arithmetic," *IEEE Std*, vol. 754, no. 2008, pp. 1–70, 2008.

[126] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM computing surveys (CSUR)*, vol. 23, no. 1, pp. 5–48, 1991.

[127] F. de Dinechin, B. Pasca, O. Cret, and R. Tudoran, "An fpga-specific approach to floating-point accumulation and sum-of-products," in *2008 International Conference on Field-Programmable Technology*, 2008, pp. 33–40. DOI: 10.1109/FPT.2008.4762363.

[128] F. de Dinechin, C. Klein, and B. Pasca, "Generating high-performance custom floating-point pipelines," in *2009 International Conference on Field Programmable Logic and Applications*, 2009, pp. 59–64. DOI: 10.1109/FPL.2009.5272553.

[129] R. DiCecco, L. Sun, and P. Chow, "Fpga-based training of convolutional neural networks with a reduced precision floating-point library," in *2017 International Conference on Field Programmable Technology (ICFPT)*, 2017, pp. 239–242. DOI: 10.1109/FPT.2017.8280150.

[130] D. Sun, X. Yang, M.-Y. Liu, and J. Kautz, "Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8934–8943. DOI: 10.1109/CVPR.2018.00931.

[131] J.-M. Lin, K.-T. Lai, B.-R. Wu, and M.-S. Chen, "Efficient two-stream action recognition on fpga," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2021, pp. 3070–3074. DOI: 10.1109/CVPRW53098.2021.00343.

[132] S. W. Hasinoff, D. Sharlet, R. Geiss, A. Adams, J. T. Barron, F. Kainz, J. Chen, and M. Levoy, "Burst photography for high dynamic range and low-light imaging on mobile cameras," *ACM Transactions on Graphics (ToG)*, vol. 35, no. 6, pp. 1–12, 2016.

[133] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden, "Pyramid methods in image processing," *RCA engineer*, vol. 29, no. 6, pp. 33–41, 1984.

[134] T. Mertens, J. Kautz, and F. Van Reeth, "Exposure fusion: A simple and practical alternative to high dynamic range photography," in *Computer graphics forum*, Wiley Online Library, vol. 28, 2009, pp. 161–171.

[135] *Intel® fpga sdk for opencl™ software technology*, Intel Corporation, 2019.

[136] G. Singh, D. Diamantopoulos, C. Hagleitner, J. Gómez-Luna, S. Stuijk, O. Mutlu, and H. Corporaal, "Nero: A near high-bandwidth memory stencil accelerator for weather prediction modeling," in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, IEEE, 2020, pp. 9–17.

[137] G. Singh, D. Diamantopoulos, J. Gómez-Luna, C. Hagleitner, S. Stuijk, H. Corporaal, and O. Mutlu, "Accelerating weather prediction using near-memory reconfigurable fabric," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 15, no. 4, 2022, ISSN: 1936-7406. DOI: 10.1145/3501804. [Online]. Available: https://doi.org/10.1145/3501804.

[138] M. Koraei, O. Fatemi, and M. Jahre, "Dcmi: A scalable strategy for accelerating iterative stencil loops on fpgas," *ACM Trans. Archit. Code Optim.*, vol. 16, no. 4, Oct. 2019, ISSN: 1544-3566. DOI: 10.1145/3352813. [Online]. Available: https://doi.org/10.1145/3352813.

[139] L. Song, L. Guo, S. Basalama, Y. Chi, R. F. Lucas, and J. Cong, "Callipepla: Stream centric instruction set and mixed precision for accelerating conjugate gradient solver," *arXiv preprint arXiv:2209.14350*, 2022.

[140] P. Colangelo, N. Nasiri, E. Nurvitadhi, A. Mishra, M. Margala, and K. Nealis, "Exploration of low numeric precision deep learning inference using intel® fpgas," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2018, pp. 73–80. DOI: 10.1109/FCCM.2018.00020.

[141] A. Boutros, S. Yazdanshenas, and V. Betz, "Embracing diversity: Enhanced dsp blocks for low-precision deep learning on fpgas," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, 2018, pp. 35–357. DOI: 10.1109/FPL.2018.00014.

[142] D. Etiemble and L. Lacassagne, "Introducing image processing and simd computationswith fpga soft-cores and customized instructions," in *Ist International Workshop on Reconfigurable Computing Education, Karlsruhe, Germany, March2006*, Citeseer.

[143] T. Zhao, P. Basu, S. Williams, M. Hall, and H. Johansen, "Exploiting reuse and vectorization in blocked stencil computations on cpus and gpus," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19, Denver, Colorado: Association for Computing Machinery, 2019, ISBN: 9781450362290. DOI: 10.1145/3295500.3356210. [Online]. Available: https://doi.org/10.1145/3295500.3356210.

[144] H. R. Zohouri and S. Matsuoka, "The memory controller wall: Benchmarking the intel fpga sdk for opencl memory interface," in *2019 IEEE/ACM International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC)*, 2019, pp. 11–18. DOI: 10.1109/H2RC49586.2019.00007.

[145] Y.-k. Choi, Y. Chi, W. Qiao, N. Samardzic, and J. Cong, "Hbm connect: High-performance hls interconnect for fpga hbm," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '21, Virtual Event, USA: Association for Computing Machinery, 2021, 116–126, ISBN: 9781450382182. DOI: 10.1145/3431920.3439301. [Online]. Available: https://doi.org/10.1145/3431920.3439301.

[146] M. R. Balazadeh Bahar and G. Karimian, "High performance implementation of the horn and schunck optical flow algorithm on fpga," in *20th Iranian Conference on Electrical Engineering (ICEE2012)*, 2012, pp. 736–741. DOI: 10.1109/IranianCEE.2012.6292451.

[147] K. Blachut, T. Kryjak, and M. Gorgon, "Hardware implementation of multi-scale lucas-kanade optical flow computation algorithm—a demo," in *2018 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, 2018, pp. 60–61.

[148] P. Basu, M. Hall, S. Williams, B. Van Straalen, L. Oliker, and P. Colella, "Compiler-directed transformation for higher-order stencils," in *2015 IEEE International Parallel and Distributed Processing Symposium*, 2015, pp. 313–323. DOI: 10.1109/IPDPS.2015.103.

[149] H. M. Waidyasooriya and M. Hariyama, "Multi-fpga accelerator architecture for stencil computation exploiting spacial and temporal scalability," *IEEE Access*, vol. 7, pp. 53 188–53 201, 2019. DOI: 10.1109/ACCESS.2019.2910824.

[150] P. Basu, S. Williams, B. Van Straalen, L. Oliker, P. Colella, and M. Hall, "Compiler-based code generation and autotuning for geometric multigrid on gpu-accelerated supercomputers," *Parallel Computing*, vol. 64, pp. 50–64, 2017.

[151] M. Kowalczyk and T. Kryjak, "A comparison of real-time 4k/ultrahd connected component labelling architectures," in *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, 2021, pp. 401–401. DOI: 10.1109/FPL53798.2021.00086.

[152] J. de Fine Licht, A. Kuster, T. De Matteis, T. Ben-Nun, D. Hofer, and T. Hoefler, "Stencilflow: Mapping large stencil programs to distributed spatial computing systems," in *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2021, pp. 315–326. DOI: 10.1109/CGO51591.2021.9370315.

[153] K. Manolopoulos, D. Reisis, and V. Chouliaras, "An efficient multiple precision floating-point multiply-add fused unit," *Microelectronics Journal*, vol. 49, pp. 10–18, 2016, ISSN: 0026-2692. DOI: https://doi.org/10.1016/j.mejo.2015.10.012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0026269215002591.

[154] L. Huang, L. Shen, K. Dai, and Z. Wang, "A new architecture for multiple-precision floating-point multiply-add fused unit design," in *18th IEEE Symposium on Computer Arithmetic (ARITH '07)*, 2007, pp. 69–76. DOI: 10.1109/ARITH.2007.5.

[155] H. Zhang, D. Chen, and S.-B. Ko, "Efficient multiple-precision floating-point fused multiply-add with mixed-precision support," *IEEE Transactions on Computers*, vol. 68, no. 7, pp. 1035–1048, 2019. DOI: 10.1109/TC.2019.2895031.

[156] C. Wu, M. Wang, X. Chu, K. Wang, and L. He, "Low-precision floating-point arithmetic for high-performance fpga-based cnn acceleration," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 15, no. 1, 2021, ISSN: 1936-7406. DOI: 10.1145/3474597. [Online]. Available: https://doi.org/10.1145/3474597.

[157] M. Sun, Z. Li, A. Lu, Y. Li, S.-E. Chang, X. Ma, X. Lin, and Z. Fang, "Film-qnn: Efficient fpga acceleration of deep neural networks with intra-layer, mixed-precision quantization," in *Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '22, Virtual Event, USA: Association for Computing Machinery, 2022, 134–145, ISBN: 9781450391498. DOI: 10.1145/3490422.3502364. [Online]. Available: https://doi.org/10.1145/3490422.3502364.

[158] J. Fowers, K. Ovtcharov, M. Papamichael, *et al.*, "A configurable cloud-scale dnn processor for real-time ai," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018, pp. 1–14. DOI: 10.1109/ISCA.2018.00012.

[159] S. Boukhtache, B. Blaysat, M. Grédiac, and F. Berry, "Alternatives to bicubic interpolation considering fpga hardware resource consumption," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 2, pp. 247–258, 2021. DOI: 10.1109/TVLSI.2020.3032888.

[160] J. Stanisz, K. Lis, T. Kryjak, and M. Gorgon, "Optimisation of the pointpillars network for 3d object detection in point clouds," in *2020 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA)*, 2020, pp. 122–127. DOI: 10.23919/SPA50552.2020.9241265.

[161] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe, "Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines," *SIGPLAN Not.*, vol. 48, no. 6, 519–530, 2013, ISSN: 0362-1340. DOI: 10.1145/2499370.2462176. [Online]. Available: https://doi.org/10.1145/2499370.2462176.

[162] J. Pu, S. Bell, X. Yang, J. Setter, S. Richardson, J. Ragan-Kelley, and M. Horowitz, "Programming heterogeneous systems from an image processing dsl," *ACM Trans. Archit. Code Optim.*, vol. 14, no. 3, Aug. 2017, ISSN: 1544-3566. DOI: 10.1145/3107953. [Online]. Available: https://doi.org/10.1145/3107953.

[163] *Introduction to intel® fpga sdk for opencl™ pro edition best practices guide*, Intel Corporation, 2019.

[164] *Intel® fpga sdk for opencl™ standard edition: Programming guide*, Intel Corporation, 2019.

[165] C. Ferry, T. Yuki, S. Derrien, and S. Rajopadhye, "Increasing fpga accelerators memory bandwidth with a burst-friendly memory layout," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2022. DOI: 10.1109/TCAD.2022.3201494.

[166] J. Hegarty, R. Daly, Z. DeVito, J. Ragan-Kelley, M. Horowitz, and P. Hanrahan, "Rigel: Flexible multi-rate image processing hardware," *ACM Trans. Graph.*, vol. 35, no. 4, 2016, ISSN: 0730-0301. DOI: 10.1145/2897824.2925892. [Online]. Available: https://doi.org/10.1145/2897824.2925892.

[167] J. Thomas, P. Hanrahan, and M. Zaharia, "Fleet: A framework for massively parallel streaming on fpgas," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 639–651.

[168] M. B. S. Ahmad, J. Ragan-Kelley, A. Cheung, and S. Kamil, "Automatically translating image processing libraries to halide," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 6, pp. 1–13, 2019.

[169] *Terasic han pilot platform demonstration manual*, 2019.

DECLARATION

Paris, May 2023

Ilias Bournias