Computer Engineering
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ce.et.tudelft.nl/

2015

MSc THESIS

Exploring Depth Estimation on Intel IPU Platform

Dan Iorga

Abstract

Numerous applications from autonomous vehicles to surveillance systems can benefit from "seeing in 3D". A crucial element of sight is depth detection since this enables evaluation of position and shape. The depth at which objects are located can be estimated by using two or more cameras and comparing the resulting images. Despite the increasing popularity of these algorithms, available solutions require expensive high-end platforms, which are not suitable for commercial applications. These motivate the search for a cheaper alternative and in this work, we provide a low-cost, low-power embedded solution to determine the depth at high speed.
A highly parallel but not inherently complex processor architecture is required for this tasks. VLIW processors are the best choices, and the Intel Image Processing Unit is an excellent option. By adding different functional units to it and adjusting an algorithm to take full advantage of them, a 640x480 image pair with 64 disparities can be processed at 36.75fps, which is an improvement of 23% compared to the best state-of-the-art image processor.

CE-MS-2015-07

Faculty of Electrical Engineering, Mathematics and Computer Science

Exploring Depth Estimation on Intel IPU Platform

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

EMBEDDED SYSTEMS

by

Dan Iorga
born in Tulcea, Romania

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Exploring Depth Estimation on Intel IPU Platform

by Dan Iorga

Abstract

Numerous applications from autonomous vehicles to surveillance systems can benefit from ”seeing in 3D”. A crucial element of sight is depth detection since this enables evaluation of position and shape. The depth at which objects are located can be estimated by using two or more cameras and comparing the resulting images. Despite the increasing popularity of these algorithms, available solutions require expensive high-end platforms, which are not suitable for commercial applications. These motivate the search for a cheaper alternative and in this work, we provide a low-cost, low-power embedded solution to determine the depth at high speed.

A highly parallel but not inherently complex processor architecture is required for this tasks. VLIW processors are the best choices, and the Intel Image Processing Unit is an excellent option. By adding different functional units to it and adjusting an algorithm to take full advantage of them, a 640x480 image pair with 64 disparities can be processed at 36.75fps, which is an improvement of 23% compared to the best state-of-the-art image processor.

Laboratory	:	Computer Engineering
Codenumber	:	CE-MS-2015-07

Committee Members :

Contents

List of Figures

List of Tables

List of Acronyms

ASIC	Application-Specific Integrated Circuit
ASIP	Application-Specific Instruction Set Processor
BAMEM	Block Access Memory
BFA	Bilateral Filter Accelerator
CHDL	Configurable Hardware Description Language
DMA	Direct Memory Access
DS3	DeepSee3
FU	Functional Units
HSD	Hive System Description
ILP	instruction-Level Parallelism
IPU	Image Processing Unit
IS	Issue Slot
Mde/s	Millions of Disparity Evaluations per Second
LUT	Lookup Table
RA	Region Aggregation
RAM	Random Access Memory
RTL	Register-Transfer Level
RMS	Root MeanSquared
SAD	Sum of Absolute Differences
SGM	Semi-Global Matching
SIMD	Single Instruction, Multiple Data
SSE	Streaming SIMD Extensions
TIM	The Incredible Machine
VLIW	Very Long Instruction Word
VMEM	Vector Memory
WTA	Winner Takes it All

Acknowledgements

I would like to thank my parents for supporting me in all my years of education and for always trusting me to make the right decision.

I am very grateful for all the support I have received from the people at Intel. Lu Yi for explaining the details of the processor, showing me how to use the tools and helping me find every document I required, Edwin Van Dalen for always finding the time to provide invaluable guidance and Rik-Jan Zwartenkot for helping me understand how a company works. I am also grateful to Rahul and Jeffrey for making my experience here a fun one.

I would also like to thank Koen Bertels and Razvan Nane for assisting me in creating a technical document. They gave me the freedom to work in what direction I wanted but at the same time made sure that I follow a scientific approach.

Dan Iorga
Delft, The Netherlands
September 7, 2015

Introduction 1

Nowadays, with the fast moving advance of technology, vehicles such as drones or self-driving cars that need to "see in 3D" are increasing in popularity. One way for devices to gather such information is through stereo vision, which is a technique aimed at inferring depth from two or more passive cameras by finding corresponding points in the images. The information gathered this way is required in numerous machine vision applications to perform more advanced tasks. Being a widely researched topic, such applications are increasing in popularity and, as a result, more solutions are becoming available.

Figure 1.1: Depth information being used for gesture recognition

One category of devices that can use this type of information is composed of laptops and tablets. For example, laptops equipped with such cameras can recognise and respond to facial expressions and hand gestures, so users can interact with them more naturally like in figure 1.1. Since software development kits for camera-equipped platforms are starting to become available, it is very probable that we can expect the range of applications to increase.

A system of cameras can also be mounted on other devices for different utilisation. This has proven useful in applications such as collision avoidance [3], traffic conflict analysis [4] or 3D reconstruction [5]. Since this are mostly real-time systems[1], it is desired that the camera information be processed as fast as possible.

These more advanced applications that require depth as input are usually mapped to other hardware elements such as GPUs or other devices that are specialised in graphical processing. The main reason Intel is interested in this algorithm is because it is an essential processing element in their RealSense technology that is used for applications such as gesture control [6].

1.1 Problem statement

The algorithms used to extract the depth information from video stereo camera systems require a high computational load, explaining why existing solutions require expensive

[1]In this case, real time should be more than 25 frames per second, in order to create a smooth sensation for the human eye

high-end platforms that are not suitable for commodity hardware. There is a large number of articles that provide solutions based on FPGA and GPU that can achieve high frame rates and excellent image quality. However, their high-cost is not suitable for entry-level solutions.

Application specific integrated circuits can also be used to detect depth and have the advantage of offering superior performance. Currently, Intel utilises such a solution to obtain a high number of frames per second. However, the ASIC manufacturing costs are high, preventing the adoption of stereo vision applications for low-end solutions. A less expensive solution would facilitate the integration of stereo vision solutions into commodity hardware. Therefore, in this thesis we investigate the tradeoffs in terms of performance, accuracy and power of porting the available algorithm used in the ASIC implementation on a custom-designed image processor.

By eliminating the ASIC, it is expected to achieve a lower frame rate since the same level of parallelism can not be reached. This decrease in performance is tolerable as long as the frame rate remains above the speed of the human vision system and does create a bottleneck for the rest of the computer vision system.

1.2 Approach

The use of a programmable embedded processor in computer vision systems allows to introduce new algorithm modifications quickly by just adjusting the application code without requiring to redesign the complete hardware architecture.

Very few depth detection solutions with of the shelf processors exist, and unfortunately they are not able to provide sufficient speed for most applications. Partitioning the algorithm on multiple cores, using Single Instruction, Multiple Data (SIMD) extensions or extending the processor with new functional units are approaches that have proven useful in the past for reaching greater performance.

Very Long Instruction Word (VLIW) processors can be very efficient when it comes to multimedia applications [7] motivating the search using such a solution. In order to reach a high frame rate, the algorithm needs to be adapted to take advantage of the capabilities of the platform while maintaining the same image quality. A large number of parameters present in the algorithm provide multiple opportunities for this.

The most computational intensive parts of the algorithm are dealt with by extending the hardware with different processing elements. Despite increasing the area of the processor, this can provide a significant speedup. Experiments with various hardware configurations are possible by using the tools provided in the processor development kit.

1.3 Goal

Obtaining fast depth information on the Intel Image Processing Unit (IPU) while maintaining the size of the core withing a tolerable range is the primary goal of this work. As a result, we expect to provide an alternative to the previously mentioned costly platforms that can achieve good performance. Eliminating the currently used ASIC reduces the cost of the platform, especially for low-end solutions. Since the manufacturing cost of the ASIC is significant, a large core would be cheaper for the company to fabricate.

1.4 Structure

The thesis is structured the following way. Chapter 2 provides a background for depth detection algorithms and describes standardised metrics for quality and speed evaluation. The focus is more on fast and less on accurate algorithms, since obtaining a high frame rate solution is the primary objective of this work.

Chapter 3 gives an overview of related works and briefly describes the approach that each one of them used. Both FPGA and CPU implementations are considered and their performance in terms of Millions of Disparity Evaluations per Second (Mde/s) is recorded. For a more detailed survey on existing stereo vision solutions, the reader is referred to the work by Tippets and Archibald [8].

In chapter 4 the general architecture of the system on which the algorithm will be tested is described. This system contains the VLIW processor and the elements required for controlling it and for data transfer.

Chapter 5 presents the algorithm that is used on the ASIC and evaluates the initial porting of it on the VLIW processor. The bottlenecks are identified and described for a better understanding of where improvements can be provided.

In chapter 6 the identified bottlenecks are addressed, and possible solutions are explored. Both hardware and software solutions are considered, and the overall effect on the performance is examined. Software solutions are preferred because they require no extra hardware.

Chapter 7 presents the experimental setup and a summary of the results obtained and compares them to other available solutions.

Finally, chapter 8 draws the conclusions and highlights other possible improvements that have been left unexplored and that can provide better accuracy at higher speed rates.

Depth Detection Background 2

Our world is three-dimensional while images are two-dimensional; they represent the projection of the world on the 2D plane. To detect how far objects are from us, we use both eyes and calculate the difference between what the left eye and the right eye perceive. The more different the left and the right eye see the object, the closer it is to the human observer.

The book by Szeliski [9] provides an intuitive example of this concept. By holding one finger in front of our eyes and closing them alternatively, we can see how the finger jumps from one side to the other.

Similarly, at least two cameras are required to capture the 3D nature of a scene and emulate the human vision system. Knowing the exact position of the cameras and the difference between the images, the depth of a scene can be captured. There are an abundance of algorithms that can be used and [10] provides an extensive survey along with a taxonomy that has been widely adopted.

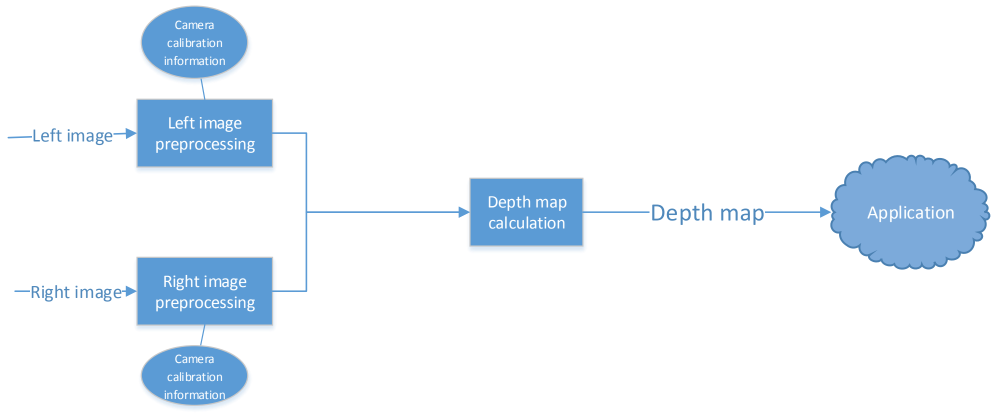


Figure 2.1: The General Data Flow of any Depth Detection Algorithm

Typical applications of such algorithms will have a structure very similar to the one in figure 2.1. Here, we can notice that the depth calculation is only a small part of a much larger application. Because of this, the depth map calculation should be done in real-time. Real-time can be defined as completing a task within an a priori specified time frame [11] which in this can should be 42ms in order to be able to achieve a frame rate of 24 fps which should be sufficient for the human vision system.

In this chapter, section 2.1 provides a brief introduction of the terminology used by this application and how a typical use case would be. In section 2.2 a short description and classification of this algorithms can be found and finally 2.3 provides the metrics for image quality and speed.

2.1 Depth Detection Terminology

The first step of such an application involves a general correction of the images so that errors caused by elements such as lenses, sensors or transmission elements are eliminated. At the same time, the camera images are aligned by applying a separate transform to each to limit the disparity search to a horizontal axis. This step is critical because it reduces the search domain from a 2D one to a 1D one [12] and is known in the literature as the epipolar constraint.

Various other steps such as white balance correction, gamma correction, dead pixel removal or downscaling are performed in this phase. These are heavily dependent on the camera, especially on the photographic system, and is beyond the purpose of this work.

The two rectified images are sent to the disparity map algorithm that detects the depth of the elements based on the differences between the two images. Information regarding the focal length and distance between the two cameras is also provided to transform the pixel difference into actual depth information as can be shown in figure 2.2:

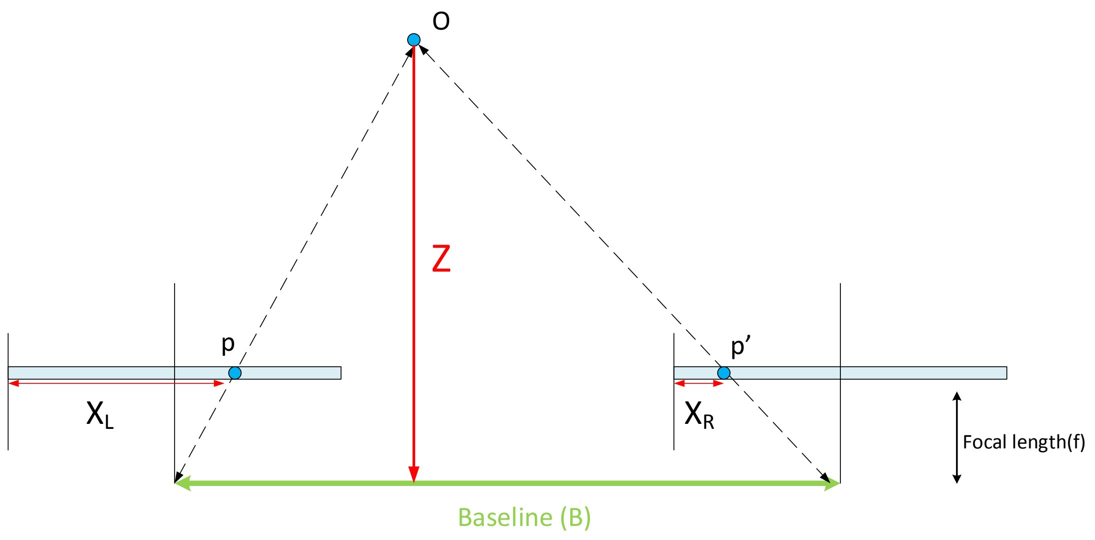


Figure 2.2: Geometrical View of the Stereo Vision Computational Elements

where:

- **O** is the object of interest
- **p** and p' are the projection of the object on left and right camera lenses
- X_L and X_R represent the position of the projection on the camera lenses
- Z is the actual distance from the object to the observer

Based on the above definitions, the distance of the pixel can be extracted easily by applying some geometrical properties that lead to equation 2.1. The equation specifies that the distance to the object is indirectly proportional to the disparity.

$$Z = \frac{b * f}{X_L - X_R} = \frac{b * f}{d} \tag{2.1}$$

where:

- **Z** is the actual depth of the object
- **d** is the measured disparity or, in other words, the difference between the relative position of the object in the right image versus the one of the object in the left image
- **b** is the baseline or the distance between the optic center of the two cameras
- **f** the focal length of the camera

The final step involves sending the depth map to other computer vision algorithms as an input.

2.2 Overview of Disparity Calculation

The term disparity was first introduced in the human vision literature to describe the difference between what the left and right eye see. This can be regarded as the inverse of the depth, and the terms are often used interchangeably.

There is an abundance of approaches that can be used to calculate the disparity map; however, the task that each one of these algorithms has to perform is actually the same operation. The correspondence between pixel (x,y) in reference image r and pixel (x',y') in matching image m has to be calculated.

This correspondence is given by

$$X_L = X_R + s * d, \qquad Y_L = Y_R$$

where $s = \pm 1$ is a sign chosen so the depth map pixel is always positive and d has been defined in equation 2.1 as the position of the best matching candidate in the reference image.

2.2.1 Processing stages

Scharstein and Szeliski [10] identify a few common steps that are present in every such algorithm. This methodology of structuring algorithms has been adopted by the majority of computer vision publications:

1. Matching cost computation;
2. Cost (support) aggregation;
3. Disparity computation / optimisation
4. Disparity refinement

Matching cost computation

In this step, a cost of matching individual pixels is defined such as similar pixels have a lower penalty in the matching process. Popular methods include the sum of the square difference, absolute difference or hamming distance [13] of the census transform.

For example, the absolute difference between pixels corresponding to the same object will likely result in a value smaller than the value between pixels corresponding to different objects.

Cost (support) aggregation

Pixels in proximity will likely have the same disparity because they belong to the same object. This provides the opportunity to reduce noise by gathering information from neighbours.

The simplest methods just sum all surrounding pixels inside a small window whereas more advanced methods sum them only if they belong to the same object. One method to determine if pixels belong to the same object is to compare colour information or intensity information.

Disparity computation / optimisation

Based on previously calculated differences between pixels the best candidate is selected. The easiest and most popular method (also known as winner take it all) just chooses the pixel with the lowest aggregated matching cost. Although other methods exist, this proves to be the simplest and most reliable one.

Disparity refinement

This is an optional step that provides further refinement. An important step that can be performed here is filling the pixels that have been detected incorrectly by gathering information from the surrounding pixels that are reliable. Favourite filters that are used to fill this incorrectly matched pixels are median filters or bilateral filters.

2.2.2 Algorithm types

There are two categories of algorithms (not mutually exclusive) that can be used to achieve the desired depth map:

- **Local algorithms** that aggregate the matching cost over a small window
- **Global algorithms** that minimise a cost function in all images being compared

Local approaches are less accurate but are a lot faster and usually can be implemented in a pipelined manner. They also require a lot less memory than global ones, which makes them more suitable for embedded systems.

Global methods are a lot more precise but rely on random access in the 2D image data and can not be performed fast on vector machines. They rely on minimising a global energy function, which in many cases has the following general form:

$$E(d) = E_{data}(d) + E_{smooth}(d)$$

Most of the top ranking algorithms are based on global methods, but there are a few local algorithms that manage to come close to the performance obtained by global methods.

A third category of algorithms called semi-global methods can be defined. These try to take the best of both worlds by performing global optimisation only over a subset of the image and usually really on techniques very similar to the one defined by Hirschmuller in [14]

2.3 Accuracy and runtime evaluation

To test the quality of the resulting depth map, the framework proposed in [10] and available at [15] is used. This framework measures the average percentage of bad pixels by comparing them to the ground truth; this has become a standard for determining the accuracy of stereo vision algorithms. The three error measurements are for all pixels (all), non-occluded pixels(noocc)[1] and discontinuities(disc)[2]. Based on this hierarchy is created, and the quality of the resulting disparity image is evaluated.

More explicitly, the following are used as quality metrics of the algorithm:

1. Root MeanSquared (RMS) error between the computed image and the ground truth

$$R = \sqrt{\frac{1}{N}\sum_{(x,y)} \|d_c(x,y) - d_t(x,y))\|^2}$$

   where N is the total number of pixels

2. Percentage of bad pixels

$$B = \frac{1}{N}\sum_{(x,y)} (\|d_c(x,y) - d_t(x,y))\| > \delta_d)$$

   where δ_d is a disparity tolerance error

Figure 2.7 shows the test images from the Middlebury website. New algorithms are ranked based on how well they compute the disparity map of these images by calculating the RMS between the ground truth images and the output of the new algorithm.

An alternative for this benchmark can be found at [16] where images captured from a car driving in the city of Karlsruhe are used. The authors claim that some of the top ranking methods from [15] are not reliable and that their method of ranking is more effective. Despite this, since the Middlebury website is more widely used, it was used as the default testing environment.

Apart from accuracy, a method to measure runtime based on image size and number of disparities is needed. For this purpose, the millions of disparity evaluations per second, proposed in [8], is used.

[1] pixels that are not available in both images
[2] pixels near edges

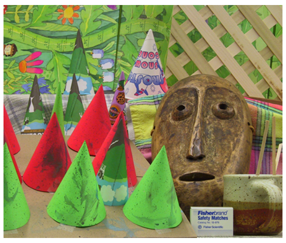


Figure 2.3: Cones test image

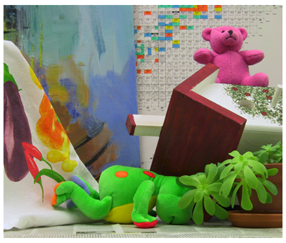


Figure 2.4: Teddy test image

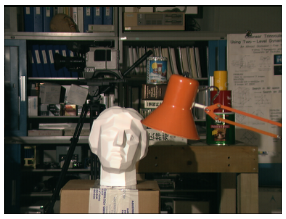


Figure 2.5: Tsukuba image

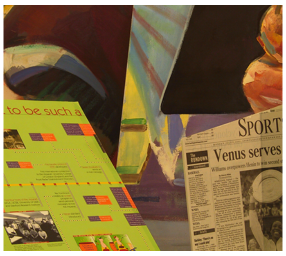


Figure 2.6: Venus image

Figure 2.7: The default four images used by Middlebury to test the resulting disparity map

$$Mde/s = \frac{W \times H \times D}{t} \times \frac{1}{1,000,000}$$

where W denotes image width, H denotes image height, D denotes the number of disparities and t denotes the total execution time in seconds.

As mentioned in the introduction, for the requirements of the Intel platform, it would be ideal to have a system that runs in real-time. The disparity map for a pair of stereo images of VGA size (640 * 480) should run at least at 25fps. In other words, the proposed system should go beyond 491Mde/s.

2.4 Conclusion

Depth can be detected efficiently estimated using a pair of stereo cameras and an efficient processing algorithm.By knowing information about the position of the cameras and analysing the resulting disparity map, an estimation of the depth of objects can be provided. Given that for our implementation speed is the primary target and not image quality, local algorithms are the best choice.

The Mde/s metric is the best option for evaluating the speed of algorithms since it has been widely adopted by most researchers and provides relevant information about the speed of implementations. Accuracy can be efficiently measured using the percentage of bad pixels metric that has been standardised due to the Middlebury test framework.

Related Work 3

Similar to other algorithms used in computer vision applications, depth detection algorithms offer a significant amount of parallelism that can be exploited for fast processing speed. Therefore, it is no surprise that the more parallelism the platform offers, the better the results.

Many FPGA and GPU implementations are able to produce excellent performance both in terms of speed and quality. These are usually orders of magnitude faster than any other implementations; however the high cost of such a platform makes it very unattractive for general-purpose commercial applications.

Currently, there are very few general purpose computers or embedded platforms that have managed to obtain reasonable results. Even taking advantage of the newest SIMD extensions and partitioning the algorithm on multiple cores is not sufficient to be able to reach real-time performance for large images and obtain good quality results.

A common trade-off to get real-time performance is to sacrifice the accuracy of the resulting image for speed. Despite the many different types of solutions, the fastest ones are based on local or semi-global approaches that can be explained by the fact that global methods are severely impacted by the memory requirements of any system and access data in a random fashion. An excellent starting point for evaluating algorithms suitable for resource constrained systems can be found in [17] and [8].

In this thesis, by utilising the Intel Image Processing Unit (IPU) an alternative to the previously mentioned costly platform is proposed, which is able to achieve significantly better performance.

This chapter is split into two sections to provide an overview of other solutions available at the moment. The first section overviews some FPGA solutions and the following one evaluates CPU-based solutions.

3.1 FPGA-based Solutions

FPGA are one of the fastest available platforms for creating depth maps. This category composes the largest amount of implementations and provides excellent performance compared to all other solutions. Despite this, the high cost makes them unattractive for a low-cost commercial solution.

Sum of Absolute Differences (SAD) and census transform are some of the most popular matching costs found on this type of platforms because of their robustness and simplicity. Some implementations even combine the two matching costs and obtain better reliability. The high speed that they are able to obtain can be explained by the fact that all of them use fixed region aggregation, which is computational inexpensive compared to other options.

Platform	Resolution	fps	Mde/s	Matching method
Altera Cyclone V [18]	2048x2048 (16)	30	2,013	SAD
Xilinx Virtex IV [18]	640 x 480 (60)	230	4,239	Census
Quartus II [19]	450 x375 (60)	599	6,062	SAD
Xilinx Virtex V [20]	1920x1200 (64)	87	12,828	ADCensus

Table 3.1: Available Field-programable gate arrays implementations

Table 3.1 presents a list of some of the fastest FPGA solutions available at the moment. The resolution and frames per second that each one is able to obtain is converted into Millions of Disparity Evaluations per Second (Mde/s) and used for ranking. All articles mention the used matching method but unfortunately, only one of the solutions described here provides measurements on all 4 images on the Middlebury framework.

An implementation on the Altera Cyclone V FPGA can be found in [18]. By using a 7x7 window for both SAD calculation and region aggregation, they are able to reach 2,013 Mde/s. This design makes efficient use of a four stage pipeline in which at every stage the number of candidates for the best disparity estimation is reduced by half. In this case, the total number of evaluated candidates determines the size of the pipeline and as a consequence the number of buffers that are used.

Jin et al. [21] implement a complete solution that contains a rectification module, a stereo processing module and a post-processing module. The census transform is used together with a modified winner takes it all module that provides sub-pixel accuracy. This is one of the few FPGA solutions that report their accuracy on all 4 images of the Middlebury evaluation framework. They are able to achieve an accuracy of 86%.

The work presented by Ambrosch et al. [19] is one of the earliest that manages to obtain excellent results by implementing sum of absolute difference on an Altera Quartus II FPGA. The key element in their implementation is that they manage to divide the image into blocks and process them in parallel. A Tree-Based minimum selector is used to select fast the best matching candidate. The authors experiment with different aggregation window sizes and finally conclude that by using a window of size 9 x 9 the optimal resource usage. Unfortunately, they only report their results on one of the images on the Middlebury test suit, which does not provide conclusive results regarding accuracy.

At the time this work was written, the fastest FPGA implementation is on Xilinx Virtex V [20]. The authors use a series of fully pipelined adders and comparator trees in order to obtain a very high throughput. They utilise the census transform combined with the absolute intensity difference from the image and fixed region aggregation in a 5x5 window. In the final step, incorrectly matched pixels are replaced by correctly matched ones using a technique called Scan-line belief Propagation. Unfortunately, the authors do not report any information regarding the quality of the resulting image.

3.2 CPU-based Solution

CPU implementations are quite slow at the moment and do not come close to their FPGA or GPU counterparts. In order to achieve reasonable performance, a large variety of techniques have been utilised. The processors that offered the highest level of parallelism are the ones with the best performance.

Table 3.2 lists the available CPU-based implementations found in literature alongside their performance results. Here we can notice that authors either use the rank transform together with Semi-Global Matching (SGM) or the Census transform with fixed Region Aggregation (RA). As described in [13] there should be no difference between the rank transform and the census transform but it is more difficult to compare fixed RA with SGM.

Platform	Resolution	fps	Mde/s	Algorithm
Freescale P4080 [22]	640x480 (91)	1.5	42	Rank + SGM
Intel Pentium IV [23]	320x240 (32)	21	51	Census + RA
TI C6416 [24]	640x480 (50)	3.8	58	Census + RA
AMD Opteron [22]	225x187 (32)	26	79	Rank + SGM
Core 2 Duo [25]	320x240 (30)	42	96	Census + RA
Tensilica LX2 [26]	640x480 (64)	20	393	Rank + SGM
Generic VLIW [27]	640x480 (64)	30	589	Rank + SGM

Table 3.2: Available general purpose processor implementations

A common technique to obtain better performance is to accelerate the computational intensive tasks using SIMD instructions. This approach has been utilised in [23] where SSE2 instructions available in Intel Pentium 4 are utilised. A similar approach but on a more powerful platform namely Intel Core 2 Duo can be seen in [25] where both cores are utilised together with the newer SSE3 instructions.

Another approach is found in [22] where they utilise a very powerful 8-core embedded platform from Freescale that runs at 1.3GHz. Despite the high computational power of such a platform, the lack of SIMD instructions prevents achieving good throughput. The authors evaluate performance both by using OpenMP and by using TBB. The same authors implement for reference purposes the algorithm on an AMD Opteron and compare the results. The quad-core AMD processor together with its SIMD instructions is able to provide better results due to its higher frequency.

A VLIW implementation with good results can be found in the work presented by Humenberger et. al. [24] [1], where they use a Texas Instruments C6416 DSP to implement a census transform based algorithm. Although this implementation is not able to reach real-time speed for VGA images, this low-cost embedded platform still manages to outperform more expensive processors such as the AMD Opteron. This implementation is also able to provide very good results, reaching a report accuracy of 91% on the Middlebury test suits

Extending the instruction set of VLIW processors with SIMD instructions has proven to be the best approach for depth estimation. The combination of this instructions together with the already available parallelism of VLIW cores is the best combination.

In one such implementation [27] they define a technique called X4 operation mode that allows the use of FU with wider inputs/outputs without increasing the number of operations. By using a bit-true, cycle-true simulator together with an enhanced instruction scheduler, they were able to achieve 30 fps for a VGA image. Despite the very good simulation results, there is no mention of synthesizing the core.

The same authors further improve their design in [26] where they modify the Tensilica LX2 by adding new instructions and modifying the memory system. The SIMD instructions they implement are able to process 64 pixels at a time, which gives them a significant advantage. Although this approach is very effective it leads to an area increase of 2.9 from the original base processor. At the current this work is being written, this is the fastest synthesizable core.

3.3 Conclusion

In order to reach high speed, most implementations started from an accurate algorithm and eliminated the components that proved difficult to implement on their platform. Unfortunately, only very few authors have reported the accuracy of their results and thus an analysis of the speed versus image quality becomes difficult. Only the maximum, accuracy of the original algorithm can be evaluated.

A large number of FPGA solutions are able to provide a high throughput due to the high parallelism that they offer. However, the high price and energy consumption of this platforms does not make them viable for low-end solutions.

Compared to this, CPU-based solutions are much slower. Their performance is usually orders of magnitude smaller than that of FPGA solutions. Despite this, some have managed to reach real-time performance for poor resolutions. SIMD-enabled VLIW cores are the only ones that are able to achieve real-time performance due to the intrinsic parallelism that they offer.

A reprogrammable embedded core would be ideal for a low-cost commercial solution. In the following chapters, such a solution is proposed.

Intel IPU Hardware Architecture 4

In this chapter, an overview of the system that will be used is described. The focus is mostly on the VLIW vector core since it is responsible for the computational intensive tasks.

Section 4.1 provides a high-level description of the system and the tools for modifying it. Section 4.2 briefly presents the different functional units and section 4.3 provides an overview of the memory system. Finally, in 4.4 the core that will be used as a starting point for exploring enhancements is presented.

4.1 High-level Architecture Description

The Intel hardware platform used in this work is composed of two main elements, a host processor and a VLIW vector coprocessor as shown in figure 4.1. The host processor is actually an untimed C model, compiled in GCC . The main tasks of the host processor is to initialise the system, upload the program kernel to the VLIW and control its execution by starting/stopping it. It is also responsible with loading the data that will be processed onto the VLIW.

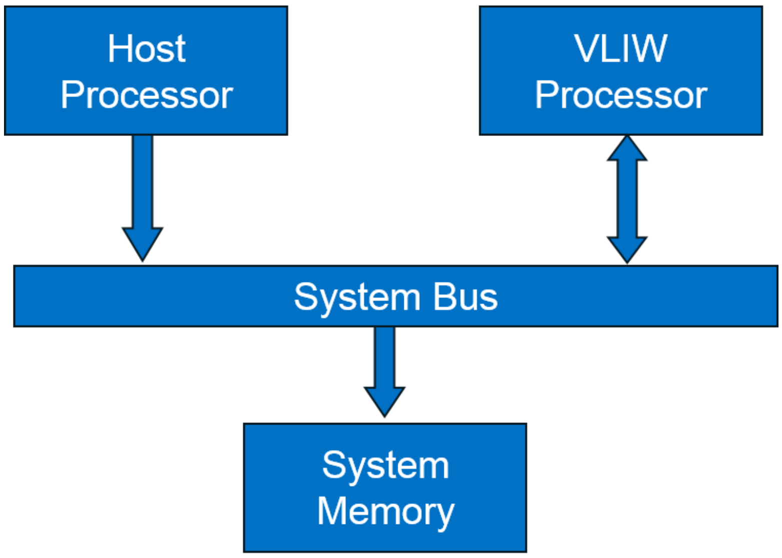


Figure 4.1: High-level View of Intel IPU Hardware Architecture

Because the VLIW core is responsible for doing computational intensive image processing, it contains multiple issue slots designed for either vector processing or scalar processing as seen in figure 4.2. There is separate data memory for the scalar issue slots and separate data memory for the vector issue slots. The program memory contains long instruction of 512 bits, sufficient for every processing element.

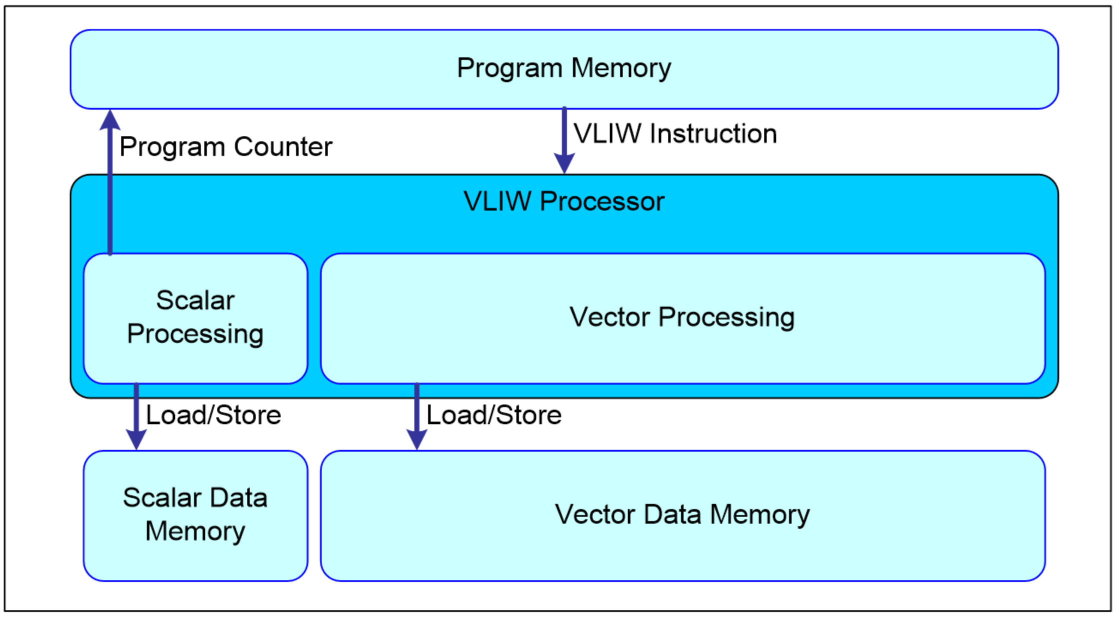


Figure 4.2: General architecture of a the VLIW processor

The process of enhancing the core can be observed in figure 4.3. The modular design is based on the fact that individual blocks that have been previously created using a hardware description language can be easily added. The Intel proprietary language called The Incredible Machine (TIM) can be used to describe the structure of the core. Another proprietary language called Hive System Description (HSD) can be used to design the connections between the core and other elements of the processor.

By knowing the exact domain in which the processor will be used, it can be easily tuned by adding new functional units to execute specific instructions. For example, for signal processing applications which are known to require a large number of multiply-accumulate operations, more such dedicated units can be added by modifying the TIM file.

In order to easily implement applications on the newly generated hardware, a retargetable compiler is required. It can be seen in figure 4.3 how the compiler uses information from the same hardware description previously mentioned.

Applications are written in ANSI-C. In order to obtain high instruction-Level Parallelism (ILP) and take full advantage of what the processor has to offer, the compiler requires a lot of help from the programmer. Intrinsics can be used to tell the compiler what functional unit to use and make its job easier. Loop unrolling and software pipelining can dramatically decrease the run-time.

The purpose of all the previous tricks is to obtain a high scheduling density and keep the issue slots as busy as possible. A good schedule is one that manages to occupy more than 70% of the issue slots at any given time during the execution.

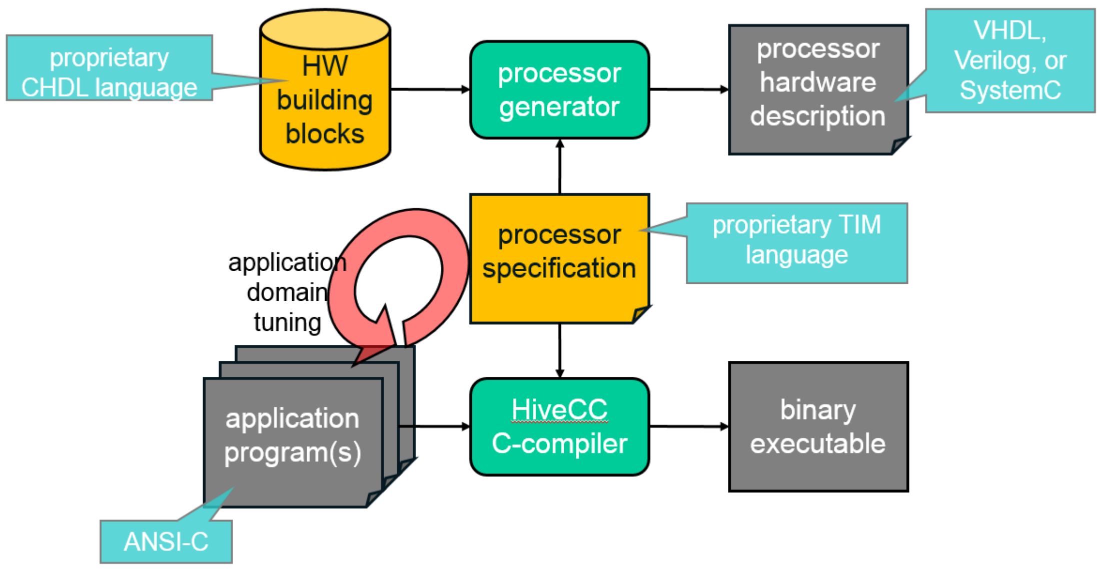


Figure 4.3: Processor generation flow

4.2 Functional Units

Each issue slot contains dedicated Functional Units (FU) supporting a variety of instructions. While some are present in most issue slots since the functionality they perform is very common (for example addition), others perform only specialised instruction and are fewer in number.

A loose categorisation of these units would be the following:

- **Load/store** units which are used to transfer information from vector memory to the registers.
- **Register transfer** units that are used to transfer information from one register to other registers. Required in order to pass information between issue slots and avoid storing intermediary data.
- **Arithmetic and logic** units which are the most numerous ones and are responsible for doing the actual computations. These units are small and have a very small latency.
- **Accelerators** are much larger units and are responsible for doing more complex operations. When operations with a high computational load become common, an accelerator is created for them.

Calculations are performed in a pipelined manner by each unit and so even though the latency for some units is quite large, efficient programming can provide a high throughput.

Figure 4.4 shows how functional units are located inside issue slots. Communication between different functional units is also displayed. In order for a functional unit to

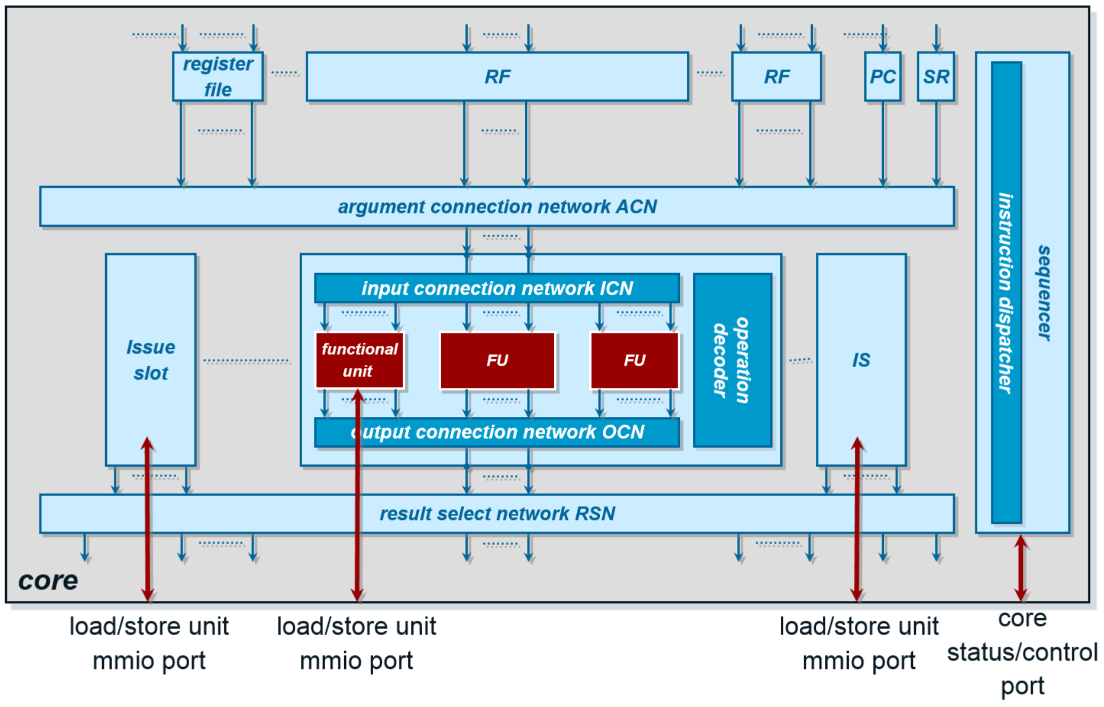

register file
RF
RF
PC
SR
argument connection network ACN
input connection network ICN
Issue slot
functional unit
FU
FU
operation decoder
IS
instruction dispatcher
sequencer
output connection network OCN
result select network RSN
core
load/store unit mmio port
load/store unit mmio port
load/store unit mmio port
core status/control port

Figure 4.4: Core architecture template

access the data it first must be stored in the register corresponding to it. An extensive interconnect network is used for this purpose.

A normal flow of operations would go in the following manner. A load/store unit would load data from the vector memory to the a register corresponding to the required processing unit, then the resulting data would be passed to another processing unit by a register transfer unit for further processing. When the output data is ready, it is stored to the vector memory by a load/store unit.

4.3 Memory Hierarchy

Image processing and computer vision systems require a large amount of fast memory in order to handle the amount of data involved. Intel IPUs contains a few dedicated types of memory that can be used for such purpose.

There are two basic vector data elements that can be used to store such information:

- A **vector** is composed of 512 bits that can be used to store 32 elements of 16 bits each. If more then 16 bits per elements are required, then two such vectors are needed and used side by side. At the moment there is no possibility to store more elements of smaller size.

- **Slice** is composed of 64 or 128 bits, depending on the processor, that can be used to store 4 or 8 elements of a vector. By storing the last or first elements of a vector in such a structure, data dependencies at the edge of vectors can be handled without using the full neighbouring vectors.

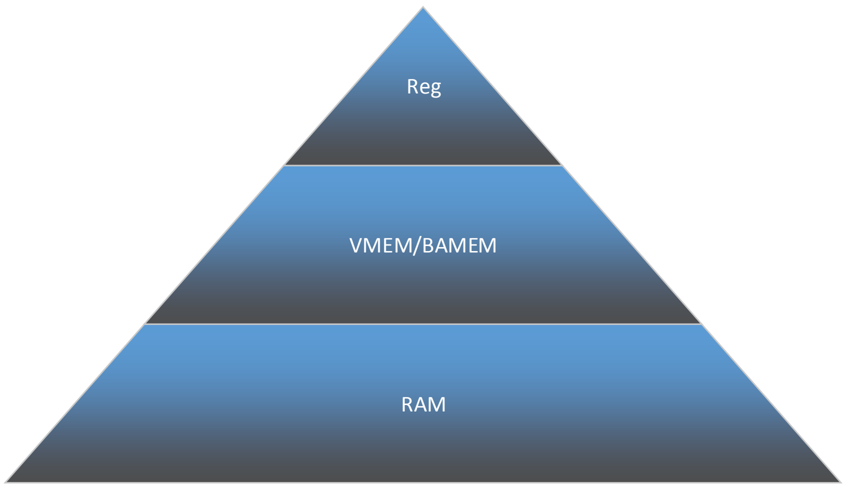


Figure 4.5: The vector memory hierarchy of a typical processor

In figure 4.5 the memory hierarchy of such a system can be observed. This structure can be adapted for the desired system according to preferences. Smaller processors for example, do not contain block access memory.

4.3.1 Registers

Each issue slot is connected to three types of registers files and can access data only through this registers as can be seen in figure 4.6. Based on their size and design purpose, 3 types of registers can be identified:

- Scalar register used for control, 32 bits width.
- Slice register that contains only a part of a vector that can be used together with a vector
- Vector registers of 512 bits that process the actual elements

Data can be transferred between registers corresponding to different issue slots using pass operations which are performed by FUs incorporated in every issue slot. Loads and stores into registers from BAMEM and VMEM is done using specialised FUs that can be found in only one issue slot.

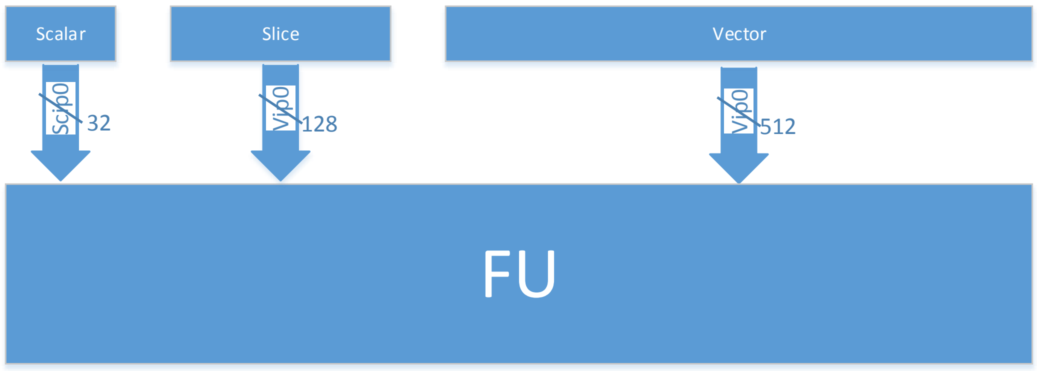


Figure 4.6: Registers and issue slot connections

A single load/store unit can be connected to an individual memory so only one element can be accessed at a time. Despite this, by pipelining a throughput of one vector per clock cycle can be read.

4.3.2 VMEM

This can be considered a type of vector cache memory that only supports 512 bits aligned accesses. In this type of memory either vectors or slices can be stored.

This type of memory can be found on the core(VMEM) or outside the core(XVMEM) and is accessed and loaded into the register using a specialised FU. Transferring data between the VMEM and the BAMEM can be done only via the registers. When the bus is free, this memory can be accessed directly by the host.

4.3.3 BAMEM

Block Access Memory (BAMEM) is a more advanced type of vector memory that supports fine access granularity of one element of a vector for both loads and stores. This means that the access address is element aligned and not vector aligned as with VMEM. It can also be configured as a Lookup Table (LUT) table. In addition to this, it also supports all the features of the VMEM [28].

The BAMEM is more expensive than the VMEM and is not implemented in all cores.

4.4 Original Core Specifications

A small core will be used as a starting point for exploring the performance of depth detection algorithms on Intel Image Processing Units. This core was originally designed to be used in wearable products but the project for it has been discontinued. Despite this, it is stable and manages to run applications without any problems.

Relying on SIMD instructions, the core contains vector memory with a maximum capacity of 2048 elements. Since each vector element contains 512 bits, which is divided into groups of 16 bits each, we can conclude that a maximum of 65,536 individual elements can be kept in memory at even given time. Image data is transferred from the host processor directly into this vector memory.

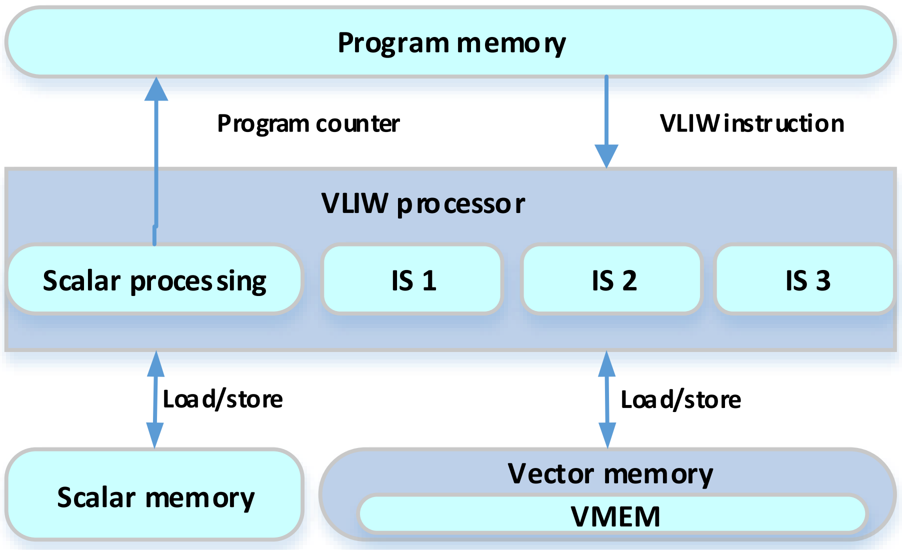


Figure 4.7: The structure of the initial core

As can be seen from figure 4.7 the core contains only 3 vector issue slots. Only the most basic functional units are present. Each issue slot is connected to a vector register file with 24 elements. There are no accelerators available and no block access memory, since this elements are very expensive.

This is a very light core compared to other image processing cores that Intel develops. This makes it very power and area efficient. It can be synthesized at 500Mhz.

4.5 Conclusion

The Intel IPU is a very efficient platform for image processing tasks and can also be an excellent candidate for depth estimation. The multiple issue slots combined with the vector processing capabilities enables a high level of parallelism that can be efficiently exploited.

We start with a light core that contains only the essential elements and add different hardware blocks in order to observe their influence on the performance. The core can be easily enhanced using the in-house tools in order to experiment with different hardware configurations. Building blocks can be easily added or eliminated and their effect on the overall performance of the application program can be easily observed using the retargetable compiler.

Original DS3 algorithm 5

This chapter investigates the original algorithm found on the DeepSee3 (DS3) chip and its performance on the Intel Image Processing Unit. Section 5.1 presents the original algorithm and evaluates the quality of the resulting image and section 5.2 presents the profiling results on the Intel core that are later used for analysing possible improvements.

The currently used solution to provide disparity maps is based on an ASIC developed by Tyzx [29] that was acquired by Intel. It offers a high frame rate of 200 frames per second at a resolution of 512x480 (52) with about 85% of the pixels detected correctly as can be seen in section 5.1.2. By using the metrics defined in chapter 2, we can evaluate the speed performance to 2555.9 Mde/s.

5.1 Algorithm Design

In [13] Zabih and Woodfill first described the census transform and the rank transform as tools that can be used to reduce the influence of illumination variances and provided a robust approach. By applying these transformations, the effect caused by different levels of illumination in the images can be reduced; therefore the match can be performed more accurately.

Studies [30] [31] have proven that this transform outperforms other alternatives when radiometric changes [1] are present. This transform is still being used today in state of the art algorithms but in combination with other elements that improve accuracy. The best example for this is the work by Xing et. all [32] where they combine colour difference information with the census transform and obtain very good robustness.

5.1.1 Algorithm steps

The main elements of the DS3 algorithm are presented in figure 5.1. In this section, each one of its elements is introduced.

The census transform. The census transform is applied to both input images to reduce the effect of radiometric changes. The transform uses a window that compares the intensity of the central pixel, with the intensity of the rest of the pixels in the window. For pixels with intensity lower than that of the central pixel, a '1' is outputted and for pixels with intensity higher a '0' is outputted. The final result is a string containing all the values.

A small example of the way the census transform would process a window of size 3 is presented:

[1]Change of light intensity in one or both of the images

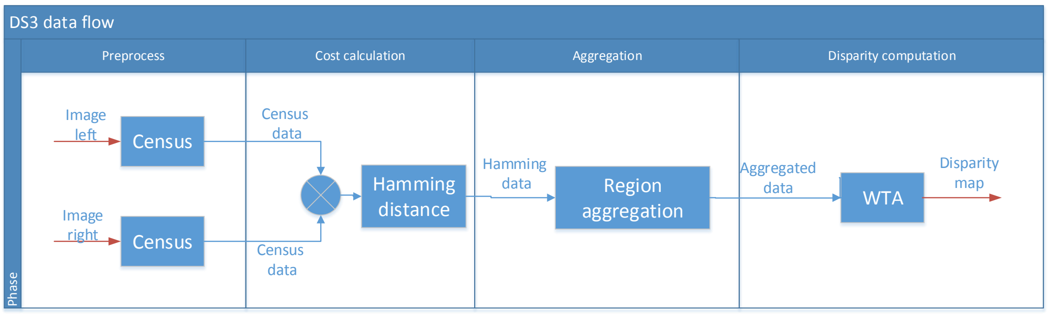


Figure 5.1: The flow of data in the DS3 algorithm

$$\begin{bmatrix} 200 & 220 & 233 \\ 220 & 230 & 210 \\ 210 & 255 & 200 \end{bmatrix} \tag{5.1}$$

The window becomes the following after the comparisons are finished.

$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & x & 1 \\ 1 & 0 & 1 \end{bmatrix} \tag{5.2}$$

And the resulting string would be:

$$1, 1, 0, 1, 1, 1, 0, 1 \tag{5.3}$$

A more formal definition of the census transform can be formulated in this manner; the census transform maps the neighbourhood of pixel P to a string representing the thresholded values by the intensity of a given pixel.

$$C(P) = \otimes_{[i,j] \in D} \xi(P, P + [i, j])$$

where $\otimes$ denotes concatenation, D denotes the window around P, and ξ denotes transform defined by

$$\xi(P, P + [i, j]) = \begin{cases} 1 & \text{, if } P > P + [i, j] \\ 0 & \text{, otherwise} \end{cases}$$

Hamming distance. Hamming distance is used to measure dissimilarity between pixels. Every pixel in the reference image is compared with multiple pixels in the target image. The search is restricted to a horizontal axis due to the epipolar constraint mentioned in previous chapters. The comparison is done by simply counting the number of different characters between the strings. The maximum distance is usually saturated to a limited number of bits to save storage space and ignore unlikely matches.

Region aggregation. Because more individual pixels can be identical, matching single ones will often lead to incorrect results. One solution to this problem is to gather more information from surrounding neighbours, seeing as they most likely belong to the same object and have similar depth.

The simplest way to do so is to sum up all pixels within a predefined small window. To further explain, the next example is provided. We try to determine the best match for window A from the left image by comparing it to two possible candidates from the right image.

$$A = \begin{bmatrix} '00' & '11' \\ '10' & '01' \end{bmatrix} \qquad B = \begin{bmatrix} '01' & '11' \\ '10' & '01' \end{bmatrix} \qquad C = \begin{bmatrix} '01' & '00' \\ '10' & '01' \end{bmatrix}$$

Window left image Window right image 1 Window right image 2

If we calculate the summed hamming distance between A and B, we get a hamming distance of 1 because there is only a distance of 1 between '00' and '01'. If we calculate the hamming distance between A and C, we get a hamming distance of 3 because there is a distance of 1 between '00' and '01' and of 2 between '11' and '00'.

Winner Takes it All (WTA). The relative position on the horizontal axis between the windows with the smallest hamming distance represent the disparity at that specific point. The inverse of this disparity represents the depth at which objects are situated.

5.1.2 Image quality

To evaluate the quality of the resulting image, the algorithm was implemented in Matlab. The resulting image was sent to the online Middlebury website for evaluation and the results can be seen in table 5.1.

The number of correctly detected pixels is about 85% for the algorithm. The quality was evaluated by a procedure described in section 2.3 where the total number of pixels that differed in intensity(distance) by more that 1 were calculated. The main problem of this algorithm is that the fixed region aggregation makes it impossible for small details to be taken into account and as such this are blurred.

The parameters that can influence the final quality of the image are the number of pixels that are compared in the census transform and the size of the aggregation window. In the original DS3 implementation, the census transform uses a window of size 7x7 in which only 24 pixels are taken into account and an aggregation window of size 7x7.

A more comprehensive study of the effects of these parameters can be found in [1] but the conclusions can be summed up as follows; if the size of the census window increases beyond a size of 7x7 or 9x9 no significant quality improvement is obtained. Similarly, the aggregation window should not be larger that 7x7 or 9x9 because small objects became blurred and they are difficult to distinguish.

Some implementations [33] use multiple predefined windows or vary the size of the window [34] in order to adapt the size of the window according to the level of detail

	Tsukuba	Venus	Teddy	Cones	Average
Errors (%)	12.5	5.8	20.5	16.2	14.9

Table 5.1: Quality metrics of the original DS3 implementations

Variable	Description
w	The width of the image
h	The height of the image
bit_depth	The number of pixels used to represent the unprocessed image
cen_cmp	The number of comparisons for the census transform
d	The number of pixels that one pixel in the reference image is compared against in the target image
clip	The amount of bits required to store a cost between pixels
vec_elements	The number of elements in a vector
aggr_win	The size of the window for the aggregation

Table 5.2: Variables used for performance evaluation

present in that particular region of the image. Such approaches are slow and the increased accuracy is not significant.

5.2 Algorithm Profiling

An implementation was created to evaluate the performance of the algorithm on the IPU and identify the bottleneck. To simplify the analysis, some useful variables are defined in table 5.2. In the rest of the chapter, all the numbers presented are for an image of size 640x480 and a total of 64 disparity candidates.

5.2.1 Data transfer

Modelling the data transfer between kernels can help better understand where the bottlenecks are present. As can be seen in table 5.3, a large amount of data is created by the hamming distance kernel, which is then moved to the aggregation kernel and finally to the winner takes it all kernel.

DMA transfers The DMA can be used to transfer data between the larger RAM memory of the system and the smaller vector memory within the core. To set up the DMA for reads, an initial latency of 400 clock cycles is required and to set it up for writes, 250 clock cycles are required. After this initial latency, memory transfers can be performed at a speed of 1 clock cycle per vector.

This can be considered the ideal case when no other traffic is present and there are no MMU misses (which can add an extra 300-400 clock cycles). The following estimates can be considered, a best case estimation and in actual use cases they can be larger.

Taking into account that a large amount of data is generated after each kernel, and the limited amount of memory available on the vector processor, line-based processing

Data element	Data output(bits)	Vectors DS3 algorithm
Image left/right	$w * h * bit\_depth$	9,600
Census data	$w * h * (cen\_cmp^2 - 1)$	19,200
Hamming data	$w * h * d * clip$	614,400
Aggregated data	$w * h * d * clip$	614,400
Disparity map	$w * h * \log d$	9,600

Table 5.3: Data transfers for a 640x480 image with a vector of 32 elements with 16 bits per element

	Census	Hamming	Aggregation	Winner
Percentages	1.24%	23.77%	61.91%	12.01%
Cycles	3,058,709	58,393,418	152,036,167	29,511,705

Table 5.4: Processing execution cycles for each kernel

is required. This approach is also found in other Intel kernels and proves to be effective. For an image of size 640*480, using line based processing, 20 vectors are transferred at a time.

By summing up all the data present in table 5.3, and taking into account that the vectors need to be both read and written from the system memory, we can evaluate that a total of 2572800 clock cycles are required for DMA transfers.

If the application runs at the same time with other applications, it is expected that the total number of clock cycles will increase. Therefore, this can be considered an optimistic estimation and in real-life use cases can increase the number of required clock cycles significantly.

5.2.2 Execution cycles

To evaluate the execution speed, each kernel is taken into account separately. The results are presented in table 5.4 and if we sum up all the numbers presented in this table we can see that it takes about 243 million clock cycles to perform just the computation part of the algorithm.

The census kernel is the fastest kernel of the entire image processing pipeline. It only processes a small amount of data as can be seen in table 5.3. Since the comparisons that need to be executed for each line are largely independent and the functional units required for this are not expensive, the census transform also achieves a high scheduling density, which also explains the high speed achieved by it.

The hamming distance kernel is slow, taking about 20% of the total execution time. This can be explained by the fact that this kernel requires unaligned loads and the processor currently does not support this so the unaligned loads have to be emulated by shuffling the elements of two successive vectors.

The most computational intensive kernel is by far the aggregation kernel. The size

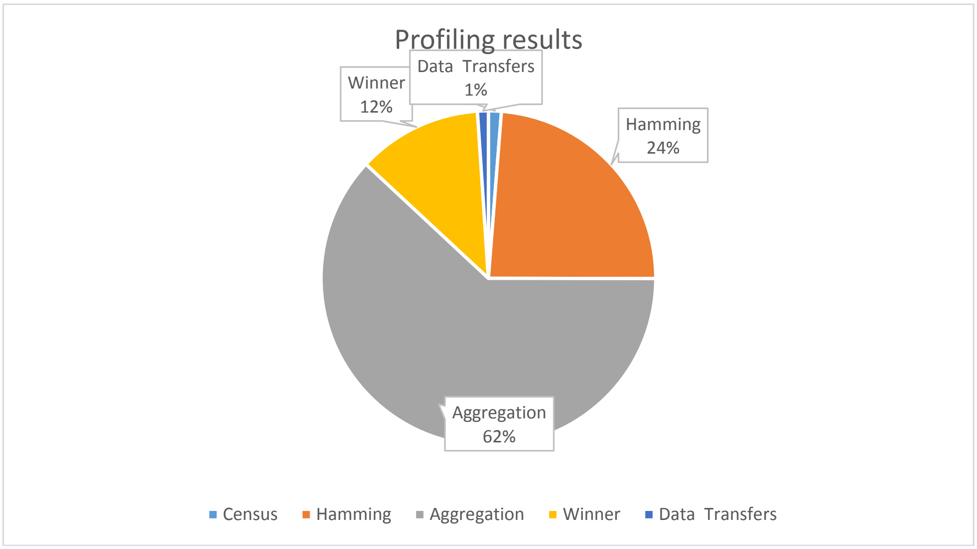


Figure 5.2: The final profile of the algorithm

of the aggregation mask is the key factor that influences the speed of this kernel as this mask has to be applied to each element created by the hamming distance kernel. For every pixel, a total of $aggr\_win^2$ has to be performed

The winner takes it all is fast compared to the other kernels. The large amount of memory accesses represents the main bottleneck of this kernel.

The results presented here can also be confirmed by works such as [35]. In this paper we have different types of aggregation methods, namely constant window aggregation, adaptive weight aggregation and cross-based aggregation for which they calculated that the kernels take up 83.88%, 93.96% and 50.28% percentage of the total execution time.

5.2.3 Profiling conclusions

In figure 5.3 a summary of the profile results on the IPU platform with colour highlighted bottlenecks can be seen. Although there are other possible variations of this algorithm, with different cost measures or different aggregation techniques, the bottlenecks will largely remain the same due to similar amounts of data elements being processed at each stage.

In this particular case, by summing up the total cycles required for processing with the ones required for data transfers, we can see that for an image of 640x480 and 64 disparities a total of 245 million clock cycles are required. By sorting in descending order the total execution cycles of each element, we can conclude that the biggest problems are as follows.

1. **Region aggregation** is the most computational intensive kernel regardless of implementation platform due to the high amount of computations that it has to perform.

2. **Hamming distance** generates an amount of data equal to $w * h * d * clip$ because

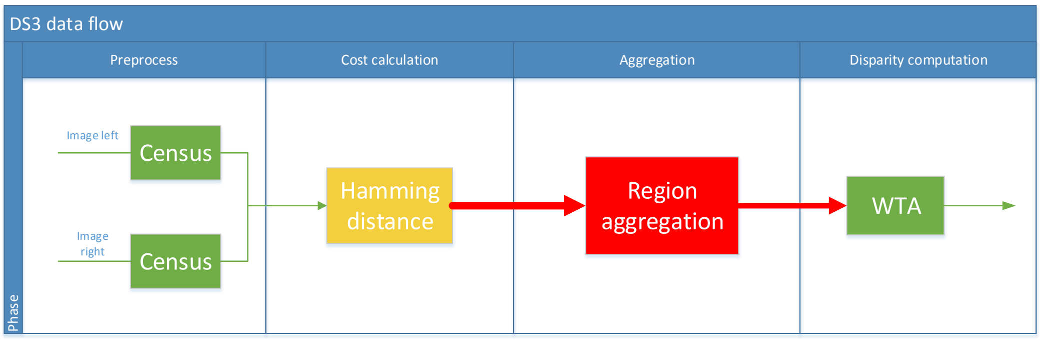


Figure 5.3: Summary of the profile. Red represents computational or data intensive elements, yellow represents medium computational intensive elements and green represents lightweight elements

a lot of comparisons have to be processed. The platform on which it is being implemented does not have specialised elements for this type of operations.

3. **Data transfers** Although the system can handle large amounts of data transfers due to an efficient DMA system, data transfers stall the core and prevent parallel processing. Eliminating them can improve parallelism.

Kernel	Census	Hamming	Aggregation	Winner	Data transfer
Percentage	1.24%	23.77%	61.91%	12.01%	1.04%

Table 5.5: The percentage of clock cycles required for each element of the algorithm

Based on this information, the next chapter explores possible solutions for every problem encountered and analyses their effect on the overall speed and quality of the result. The VLIW platform can offer significant speedup if every element is explored efficiently.

5.3 Conclusion

The DS3 ASIC is able to provide real-time performance using a very efficient algorithm. The original algorithm utilised by it needs to be adapted to the Intel IPU in order to be able to reach real-time performance. Despite the parallel nature of the algorithm, some modifications have to be made to map it efficiently on the targeted platform.

The region aggregation kernel is the most computational intensive part of the algorithm and will most likely require hardware acceleration. Since the hamming distance kernel generates a large amount of data, a solution needs to be found to prevent this becoming a bottleneck and create an efficient pipeline.

Modified DS3 algorithm 6

This chapter deals with accelerating the original DeepSea3 algorithm so that real-time performance is obtained. Software solutions are preferred due to the fact that they do not require any extra area on the core but hardware ones are used when no other option is available.

We start in section 6.1 by exploring possible improvements for the computational intensive kernels identified in the previous chapter. Section 6.2 deals with how the VLIW architecture can be further exploited by reorganising the code to take full advantage of the modified kernels.

Combining all these elements, significant speedup is obtained at the cost of increased area. Section 6.3 presents the final speedup obtained and the elements that were necessary to reach it.

6.1 Kernel Acceleration

Amdahal's law [36] states that the maximum speedup in an improved sequential program is limited by the percentage of time that is spent in that particular part of the program. Therefore we start by finding solutions for improving the execution time of the region aggregation and the hamming distance kernel.

6.1.1 Region aggregation

As can be seen from table 5.3, the aggregation kernel receives a large amount of data from the hamming distance kernel. At least $w * h * d$ data elements are received and for each one of them $aggr\_win^2 - 1$ summations has to be performed. Even by using vector instructions and processing multiple pixels at a time, this is still slow and resource demanding.

There are multiple ways in which region aggregation can be accelerated. In the literature, two main approaches can be identified:

Software approaches Most software optimizations are based on box-filtering techniques [37] or on integral images techniques [38] and provide a significant speedup by taking advantage of already calculated partial sums. Despite their effectiveness, this techniques proves difficult to implement on an SIMD processor due to the irregular way in which data is processed.

Hardware approaches A dedicated hardware block can be added to the processor in order to perform the required operation. This provides the greatest speedup at a cost of extra area.

The **hardware approach** is chosen, and a solution based on the Bilateral Filter Accelerator (BFA) is provided. This accelerator can only be found in some specialised processors but can be integrated into the core by using the tools described in chapter 4.

The bilateral filter [39] is a non-linear, edge-preserving and noise-reducing smoothing filter. The intensity value at each pixel in an image is replaced by a weighted average of intensity values from nearby pixels. This preserves sharp edges by systematically looping through each pixel and adjusting weights to the adjacent pixel accordingly.

By configuring the accelerator with the correct parameters, it can produce exactly the same results as the region aggregation in the original DS3 ASIC. This means that the quality of the image is not affected in any way.

Although this functional unit requires a large number of clock cycles in order to produce an output, by processing in a pipeline manner, notable speedup should be obtained.

Version	Clock cycles	Speedup	Area
No accelerator	152,036,167	-	-
With accelerator	30,688,920	351%	+22%

Table 6.1: Effects of accelerating the region aggregation kernel

Table 6.1 shows the effects of accelerating the region aggregation kernel on both clock cycles and added extra area. The significant increase in area of the core is justified by the performance gained. In further chapters, the global effects on the speed of the algorithm are explored and described.

Another option to accelerate this part of the algorithm is to use the convolution unit that is currently being developed [40]. This unit is expected to be faster and more area effective and might prove to be a better alternative to the bilateral filter.

6.1.2 Hamming distance

The hamming distance calculation suffers from the following problems:

- For each individual pixel, a total number of 24 output bits are generated. This number can not be efficiently packed in vectors given that they are designed to contain 16 bits per element. Two vectors have to be used, which is both memory inefficient and computational inefficient as both vectors have to be processed.
- Since the reference image is compared with shifted versions of the target image, unaligned loads are required. These unaligned loads are emulated by combining information from two adjacent vectors
- Currently there is no operation to calculate directly the hamming distance between vectors. This operation is done by using a series of bit shifts, which requires a total of 13 clock cycles.

Following are a few methods that can help resolve the problems mentioned above. Some of them affect only one problem, whereas some affect multiple ones.

Sparse census transform The original census implementation does a number of 24 comparisons per pixel in a 7 x 7 window but other versions of the census transform that provide similar efficiency can be explored.

In [1], the sparse census transform is defined which uses fewer comparisons but obtains the same image quality. This idea is further refined in [2] where multiple census patterns are analysed and described. In these articles, it is shown that pixels that are close to the centre do not provide useful information and can be ignored.

Another more light version of the census transform is defined in [41] and is called the mini-census transform. This alternative uses only 6 pixels arranged in a distinct pattern within the selected window and manages to obtain good quality results. This is the most utilised adaptation of the transform in FPGA implementation but since no speedup would be gained by using this on the vector processor, it was decided not to use it.

The results mentioned in these articles can be easily verified by modifying the Matlab script that was used to test image quality in section 5.1.2. The patterns that use 16 bits suggested by [2] were implemented in this script and the resulting disparity images were sent to the Middlebury website for evaluation in their automatic framework.

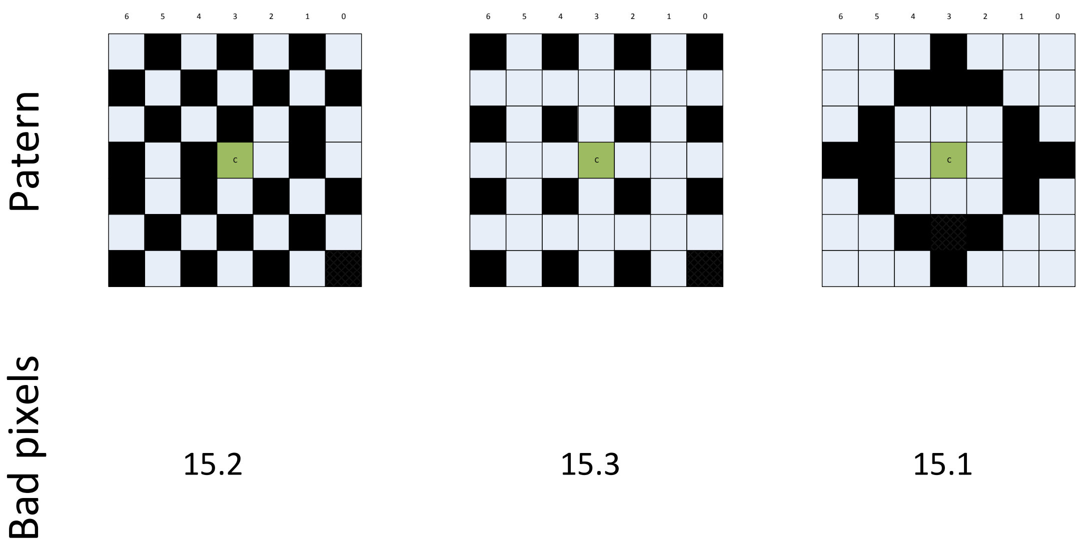


Figure 6.1: The average number of bad pixels in all 4 images on the Middlebury website. Left pattern-census transform used in the ASIC. Middle pattern - sparse census transform from [1]. Right pattern - modified census transform pattern from [2]

Figure 6.1 shows the average number of bad pixels in all 4 test images. Our results confirm that by using the sparse census transform the quality of the resulting images does not degrade.

In fact, by using the last census transform version, it can be seen that the quality slightly improves. Overall it is justifiable to use the last version of the census transform seeing that there is no downside to this approach.

All this changes offer the advantage that data transfer between the census transform and the hamming distance kernel is reduced to half and that the hamming distance and so is the number of computations that have to be performed.

Block access memory A significant problem is created by the unaligned loads and stores. A solution to this problem is to add memory that supports this type of operations. Similar to the BFA, the Block Access Memory (BAMEM) is a specialised hardware component that can only be found is some processors. Adding it to the processor will increase the size of the processor and should only be done if the extra performance obtained justifies it.

The BAMEM [28] can be considered an improved version of the of the vector memory which enables complex data access modes that significantly improve processing efficiency in image/video application, compared to standard vector memories. Those access modes are unaligned block/row, unaligned vector+slice and unaligned block+slice. Additionally, it supports general purpose features of vector memory subsystems, e.g. static array allocation, stacks and register spilling. In figure 6.2 examples of such access patterns can be noticed.

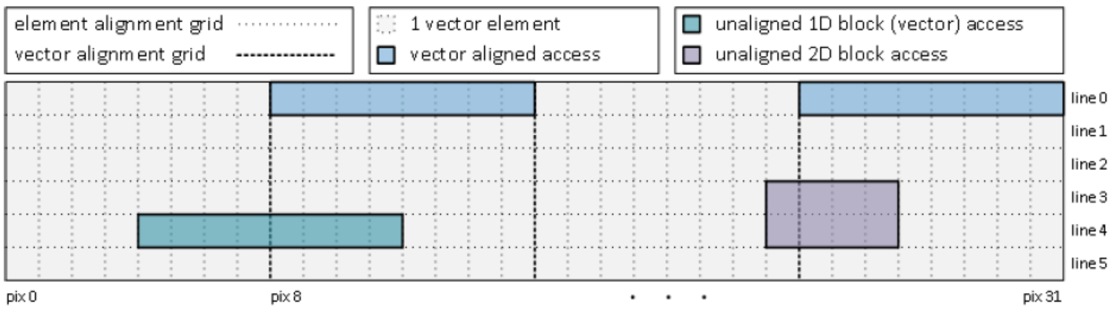


Figure 6.2: The multiple access patterns that the BAMEM supports.

Since now unaligned vector access is supported, it is no longer required to emulate these using multiple operations.

Hamming distance instruction The hamming distance is performed by applying a xor between two vectors and calculating the number of bits different from zero. The final result is then saturated to ignore values that are too large and unrealistic.

Even though the core does not contain an instruction to count the number of bits different from zero this can be found in other specialised processors. A new instruction can be created around this that performs all the operations necessary in order to perform the hamming distance calculation.

The new instruction needs as input the census transform of the left and right image and a bit mask to clip the result to a desired number of bits. Figure 6.3 shows the high-level diagram of the new instruction.

As can be observed, the new instruction is an extension of the **popu** logic. As a result, the required number of operation to perform hamming distance calculation has been reduced to a single instruction from the initial number of 13. Although the VLIW

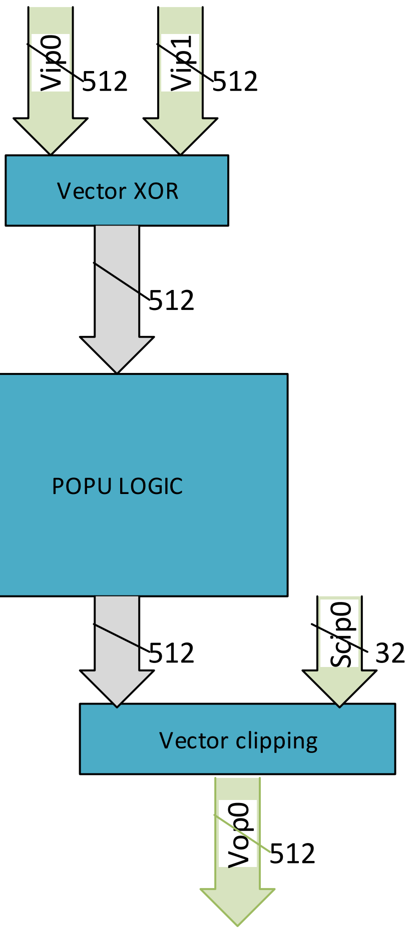


Figure 6.3: High level view of the hamming distance instruction

processor can calculate all of these instructions in parallel due to the high number of parallel functional units, it is clearly an advantage to free them up for other operations.

In table 6.2 the effects of improving each individual element of the hamming distance kernel are reported and the total speedup obtained is 778%.

Improvement	Clock cycles	Speedup	Area increase
Original	58,393,418	-	-
Using sparse census	42,316,456	37%	-
Adding vec_hamming instruction	27,806,841	109%	0.32%
Adding BAMEM	6,650,425	778%	67%

Table 6.2: Effects of accelerating the hamming distance

The BAMEM has the biggest impact on the total execution time of the kernel but also increases the area of the processor significantly. Since a single operation is now required, the reads and writes to memory can now be pipelined, which explained the significant speedup obtained.

6.1.3 Summary

The most computational intensive kernels of the algorithm have been accelerated by both software and hardware improvements. The added hardware blocks are the ones that have the highest impact on performance but at the same time increase the size of the core significantly. Table 6.3 shows the effects on overall performance of the applications and on the area.

Version	Total	Speedup	Area
Original	245,572,800	-	-
Improved kernels	72,482,559	238%	90.75%

Table 6.3: Kernel acceleration results

Real-time performance is not yet achieved, so more solutions need to be considered for this. In the remainder of this chapter. the parallel nature of the processor is exploited for more performance.

6.2 Parallelism improvements

VLIW processors achieve a very high level of parallelism by placing all the complexity of the schedule to the compiler but because run-time information is not available, the programmer needs to make changes to the code in order to obtain a better schedule. Loop unrolling and software pipelining are very common techniques that improve ILP, but in order for them to take effect, some modifications need to be made to the code.

6.2.1 Data storage or data recalculation

A solution for reducing data transfers between kernels is to merge them and keep the temporary data in the previously added BAMEM. Unfortunately, the limited amount of available memory prevents storing all the intermediate data. By summing up all the information in table 5.3 it can be seen that a total amount of $(5 * w * h + 2 * w * h * d) \div (vec\_elements)$ vector are required, which is clearly too high. The image can be divided into blocks but because of data dependencies, the size of this blocks is again large.

This problem can be solved by not storing everything and recalculating when necessary. Taking into account that the hamming distance kernel is the one that produces the most amount of data but is very fast, it seems that this is the best candidate for data recalculation.

Another factor that needs to be taken into account is that the bilateral filter processes 4 lines at a time so at least 4 new lines should be sent at each step for an efficient flow. The resulting structure can be seen in figure 6.4 where 4 input lines from the left line and 4 input lines from the right image are sent at a time and 4 processed output lines are received. Data dependencies force a delay between input and output of 3 time steps (or 12 lines).

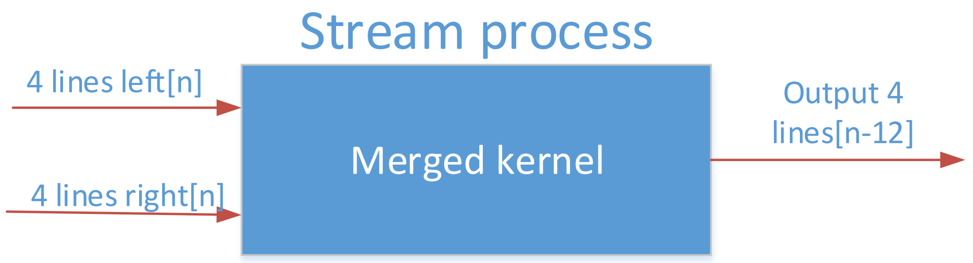


Figure 6.4: Merged flow of the algorithm. It receives 4 inputs of the left image and 4 inputs of the right image and outputs the disparity map corresponding to the previous 12 lines.

Algorithm version	Clock cycles	Speedup
Separate kernels	72,482,559	
Merged kernels	42,258,025	71%

Table 6.4: Separate kernel or merged version

Table 6.4 shows the clock cycles of the entire algorithm for the merged version and the separate kernel version. Even though data is now recalculated, the compiler can better schedule operations so higher ILP is obtained. The merged version does 2.5 more hamming distance but since now this is done in parallel, speedup is obtained.

6.2.2 Avoiding BAMEM writes

In the previous version of the implementation, in order to reuse as much data as possible, data is written to memory after each kernel finishes. This creates some false synchronisation points and efficient software pipelining can not be implemented. One technique to avoid this is to write the results of each step directly to the registers corresponding to the FU that will further process the pixels.

Similarly to what has been done to eliminate the use of the external memory, hamming distance elements are recalculated every time they are requested by the accelerator. Later on, experiments with different number of functional units that are able to calculate the hamming distance are made and the results are presented.

Figure 6.5 shows how data is now transmitted directly to the registers of the kernel that requires them, without any need of the BAMEM. This memory is accessed now only at the beginning for reads and at the end to store the data. In order not to put too

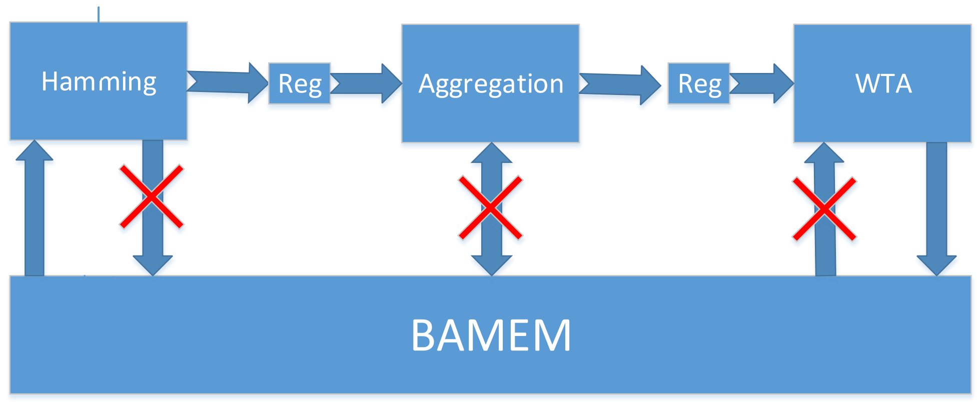


Figure 6.5: Avoiding the BAMEM to enable efficient software pipelining

much pressure on the registers corresponding to the accelerator, the hamming distance data can only be calculated when it is required.

Algorithm version	Clock cycles	Speedup	Total speedup
Separate kernels	42,258,025		71%
Merged kernels	28,749,562	50%	152%

Table 6.5: Speedup obtained from avoiding writes to BAMEM.

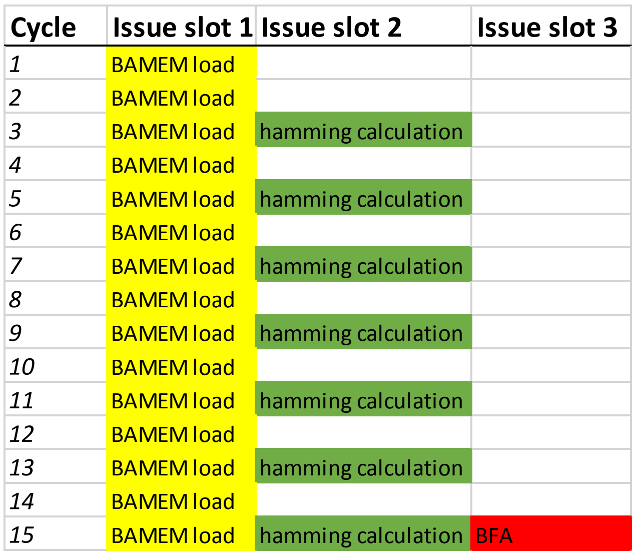
From table 6.5 the speedup obtained can be clearly observed. This can be explained by the fact that an efficient software pipeline is now crated that takes full advantage of the resources available. The compiler can now schedule operations in an efficient way and obtain a high scheduling density.

6.2.3 Using VMEM as BAMEM

From 6.6 shows a schedule of the algorithm with only the most important elements displayed. It is clear that the BAMEM load/store unit is used at full capacity. In order to further increase the processing speed, faster access to BAMEM is required. Unfortunately, adding a new load/store unit to this memory would be very expensive so an alternative is required.

For hamming distance calculation, two census transformed images are required: the census transform of the reference and of the target image. Unaligned loads are not required for the reference image. This immediately brings the idea to load the census transform of the reference image in the VMEM.

One problem that arises is that the bilateral filter only works with blocks of 4x8 while the VMEM can only store blocks of 1x32. The solution to this is to first load the data to the BAMEM and use it to transpose the data in the desired way. In order to do so, some extra clock cycles will be spent, more exactly $3 * (width * height) \div (vec\_elements)$,



Cycle	Issue slot 1	Issue slot 2	Issue slot 3
1	BAMEM load		
2	BAMEM load		
3	BAMEM load	hamming calculation	
4	BAMEM load		
5	BAMEM load	hamming calculation	
6	BAMEM load		
7	BAMEM load	hamming calculation	
8	BAMEM load		
9	BAMEM load	hamming calculation	
10	BAMEM load		
11	BAMEM load	hamming calculation	
12	BAMEM load		
13	BAMEM load	hamming calculation	
14	BAMEM load		
15	BAMEM load	hamming calculation	BFA

Figure 6.6: The schedule of the new version of the algorithm

which for a VGA image is a negligible amount of 38400. This equation derives from the fact that the bilateral filter requires 12 lines as input, and blocks of size 3.

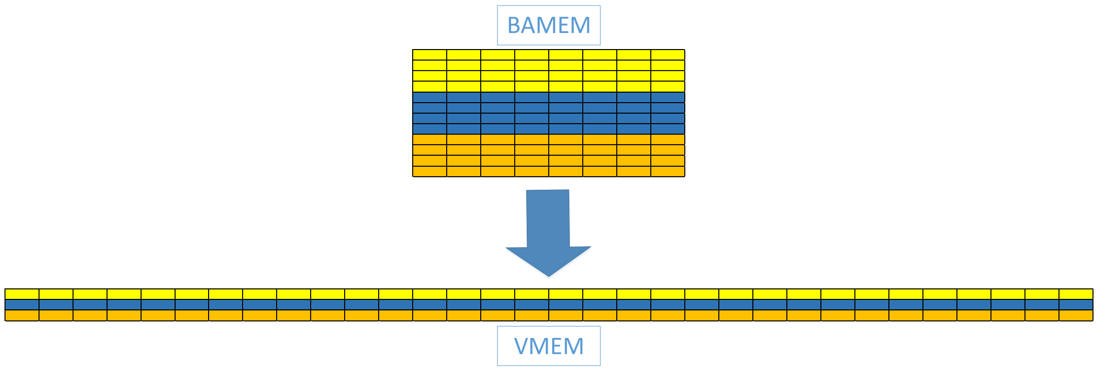


Figure 6.7: Transposing BAMEM data to the VMEM

In figure 6.7, the process of transposing blocks of data to the VMEM is displayed. Since only 12 lines of the census transformed image are required at a time, only a small part of the VMEM is used. For a VGA image, this represents less than 12% of the available memory but for larger images more memory might be required.

In table 6.6 the effects of loading data from multiple sources can be seen. Since there is no improvement, we can conclude that memory loads are not the bottleneck and that input to the hamming distance functional units and the accelerator is provided fast enough. further experiments can be done with increasing the number of these elements and observing the results.

Algorithm version	Clock cycles	Speedup	Total speedup
Only BAMEM	28,749,562		152%
BAMEM and VMEM	28,749,562	none	152%

Table 6.6: Effects of partitioning the data on two memories.

6.2.4 Summary

This section presented optimisations necessary to take full advantage of the parallelism offered by the VLIW processor. Figure 6.8 displays the data flow of the modified algorithm that uses both BAMEM and VMEM to load data and does not access any memory between stages.

Since the accelerator is the element with the highest latency, the main desire was to assure that it is not remain idle at any time. It can be considered that algorithm is now centred around the accelerator and all other elements are designed to supply it with data.

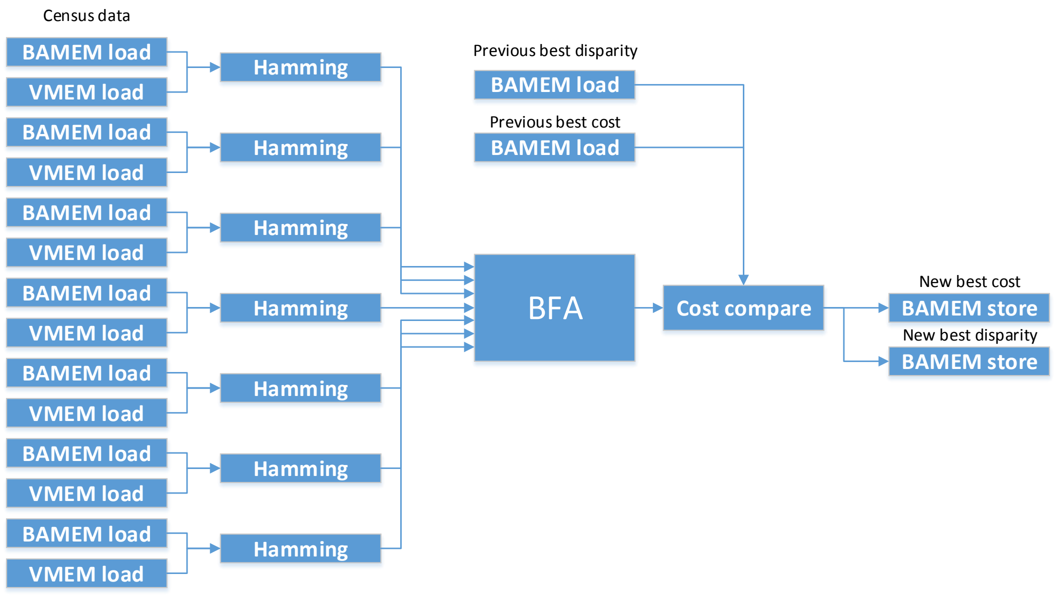


Figure 6.8: Final dataflow of the algorithm

The factor that contributed to the speedup of the algorithm the most was the option not to store intermediary results and recalculate data as much as possible. This decreased pressure from the memory system and placed in on the processing elements. Since the VLIW is capable of processing large amounts of data in parallel, this was clearly the better choice.

6.3 Conclusion

Table 6.7 shows a comparison between the original and the modified version of the algorithm. The cost of achieving this performance is paid by increased area. The accelerator and the block access memory are the primary elements that contribute to this increase in core size but without them, all the other optimisations would have been impossible.

Version	Total	Speedup	Area
Original version	245,572,800	-	-
Modified version	28,749,562	754%	+ 90.75%

Table 6.7: Final acceleration results

Based on the conclusions from the previous chapter, a solution for the most computational intensive elements was explored. The region aggregation was accelerated by using a dedicated hardware block. However, the algorithm for the hamming distance transform had to be first modified and only then an instruction to compute it could be created.

In order to take full advantage of the parallel nature of the VLIW core, the algorithm was restructured to achieve higher ILP. Since the accelerator is the slowest element of the new flow, it was necessary not to keep it idle at any time. Recalculating data and not storing any intermediary values proved to be the essential element to enable this.

7 Experimental results

This chapter analyses the impact on speed, area and image quality of all the changes done in the previous chapters and concludes with the optimal algorithmic and hardware configuration.

Section 7.1 describes the setup of the test environment and the test image that is going to be used. In section 7.2 the impact that the performed changes have on speed, area and data transfers are analysed while in section 7.3 the identified critical resources are increased and the effects reported. In section 7.4 we analyse how all changes affect image quality. Finally, 7.6 presents the conclusions and compares our solution to others .

7.1 Simulation Environment

A simulation environment based on the configuration from chapter 4 was created to test the speed of the implementation. The system elements are configured as seen in figure 4.1. This host processor reads both input images from the hard drive and writes 4 lines from each in the shared memory and then waits for them to be processed. This process is repeated until all lines have been processed.

Experiments with different numbers of functional units and accelerators were made possible by using the in-house The Incredible Machine (TIM) and Hive System Description (HSD) hardware description languages. The RTL code was generated to ensure that the core could be implemented in hardware, but for testing the performance of the application, only the simulator was utilised. The reason why the cycle accurate simulator was used is that it is much faster than the actual RTL simulation.

The total number of clock cycles is reported by the simulator and then added to the number of clock cycles required to transmit data. The schedule report is consulted every time to identify potential bottlenecks. Since the core can be synthesised at 500Mhz, this frequency is used to transform the clock cycles into frames per second.

For all tests, an image pair from the Middlebury test framework was used. The chosen algorithm is deterministic and its execution speed does not depend on input data, only the resolution of the image influences it. The images were grayscaled and resized to 640x480 in order to easily compare it to other implementations.

7.2 Modifications impact

This section analyses the impact on speed, area and data transfers of the modifications described in chapter 6. We start by looking at data transfers to and from the VLIW core.

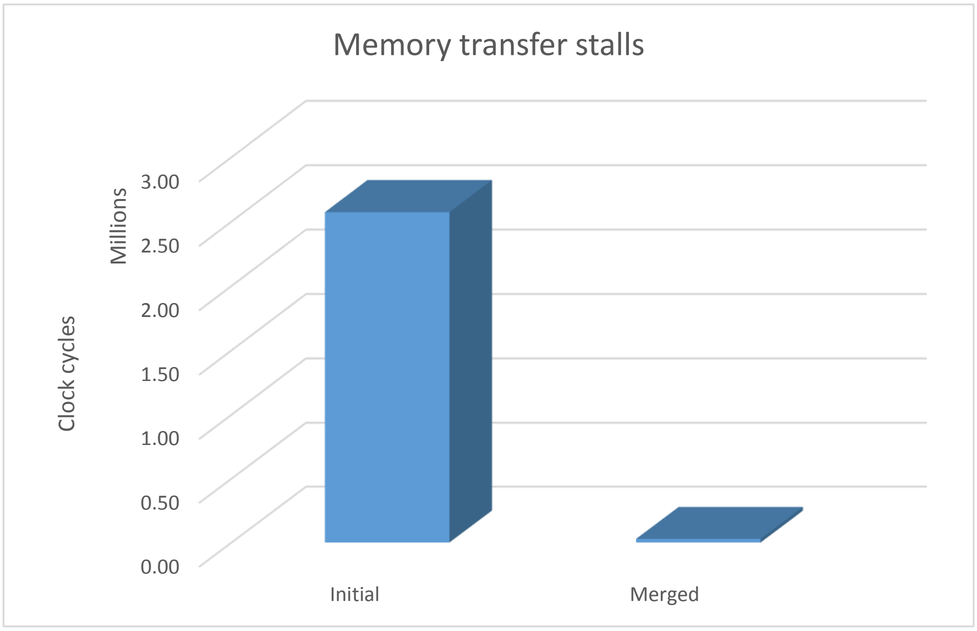


Figure 7.1: A comparison between stalls caused by memory transfers to the VLIW in the original version and the enhanced version

In figure 7.1 a comparison between the memory stalls in the original version and the new one is presented. By merging the kernels, this number decreases to a point where they are almost insignificant. This enhancement could not have been done without adding the BAMEM.

While these stalls do not have a severe impact on the performance of the algorithm, they might have a more detrimental on the final system. When other data transfers are present, the system bus might be occupied and transfer speed to the VLIW might be slowed down.

Table 7.1 shows the speedup and area of the core after each improvement has been done.

Version	Clock cycles	Fps	Speedup	Extra area
Original version	245,572,800	2.04	-	-
Added accelerator	124,225,553	4.02	97.68%	22.00%
Sparse census	108,148,591	4.62	127.07%	22.00%
Hamming instruction	93,638,976	5.34	162.25%	22.32%
Added BAMEM	72,482,559	6.90	238.80%	90.75%
Merged kernels	42,258,025	11.83	481.13%	90.75%
No BAMEM writes	28,749,562	17.39	754.18%	90.75%
VMEM as BAMEM	28,749,562	17.39	754.18%	90.75%

Table 7.1: A summary of the speedup and increase in core size after each improvement

We can notice that the accelerator and the block access memory are the ones that have the heaviest impact on the area. However, all other improvements to the algorithm could not have been done if these elements were not added. Adding the accelerator frees up issue slots so that other elements can be processed in parallel. The BAMEM allows fast loads and stores, without the need to emulate unaligned accesses and is able to provide data for accelerator at sufficiently high speeds.

In graph 7.2 a plot of the speedup and total area provides better information on the significance of each element. The biggest improvement is obtained by the methods that take advantage of the parallel nature of the VLIW core.

Merging the kernels and avoiding BAMEM writes has the greatest impact on performance. Both these modifications rely on the knowledge that recalculating data is faster than storing and loading it from memory. This proves that there is a significant difference between the speed of the memory system and that of the processing systems.

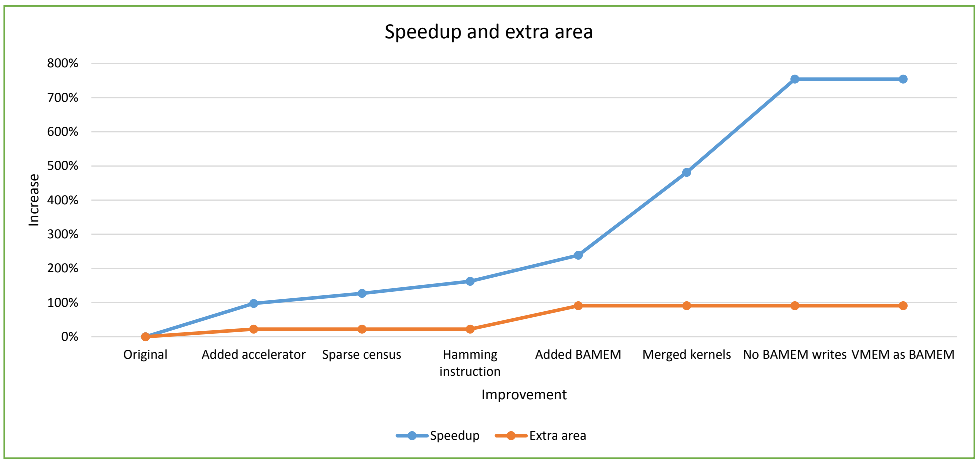


Figure 7.2: Speedup and area behaviour

The final part of the plot shows what happens when data is loaded from two sources, more specifically BAMEM and VMEM. At this point, memory accesses are no longer the bottleneck and more optimisations here will not further increase the speed. In order to go beyond this point, more processing elements need to be added to the core. Only when these elements will be added, loading data form two memories will be useful.

7.3 Design Space Exploration

Since the accelerator and the functional unit capable of performing the hamming distance operation are the critical resources for the algorithm, experiments are made with different numbers of such units in order to determine their effect. This has no impact on the quality of the resulting image since the compiler can efficiently adapt to the new hardware.

From table 7.2 it can be seen that it makes sense to have a higher number of hamming

Accelerator	Hamming FU	Area increase	Cycles	Fps
1	1	+90,75%	28,749,562	17.39
1	2	+91,07%	20,122,883	24.85
1	3	+91,39%	20,130,563	24.84
2	1	+113,73%	14,224,043	35.15
2	**2**	**+114,05%**	**13,604,245**	**36.75**
2	3	+114,37%	13,604,245	36.75
3	1	+136,71%	13,056,081	38.30
3	2	+137,04%	11,745,561	42.57
3	3	+137,36%	9,720,302	51.44
4	1	+159,70%	14,405,741	34.71
4	2	+160,02%	10,014,201	49.93
4	3	+160,34%	9,421,262	53.07

Table 7.2: Varying the number of accelerators and hamming distance functional units and their effects on performance and area

distance functional units to provide sufficient input to the bilateral filter accelerator. The impact on area that hamming distance units have is insignificant compared to the one that the bilateral filter accelerator has.

We can conclude that the Pareto Optimal design points are the ones that use at least two hamming distance functional units. The increased speed can be explained by the fact that this units decrease the initialisation time of the software pipeline, which is used multiple times throughout the algorithm.

Because our target is to reach real-time speed, a solution that is able to provide more then 25 frames per second is sufficient. The architecture with two accelerators and two hamming distance is preferred since it is optimal and does not have a high impact on area.

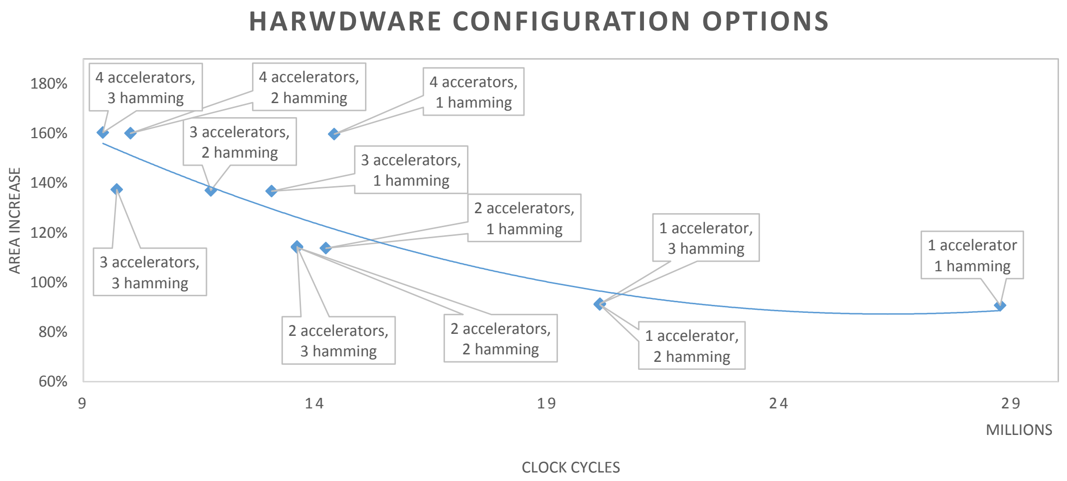
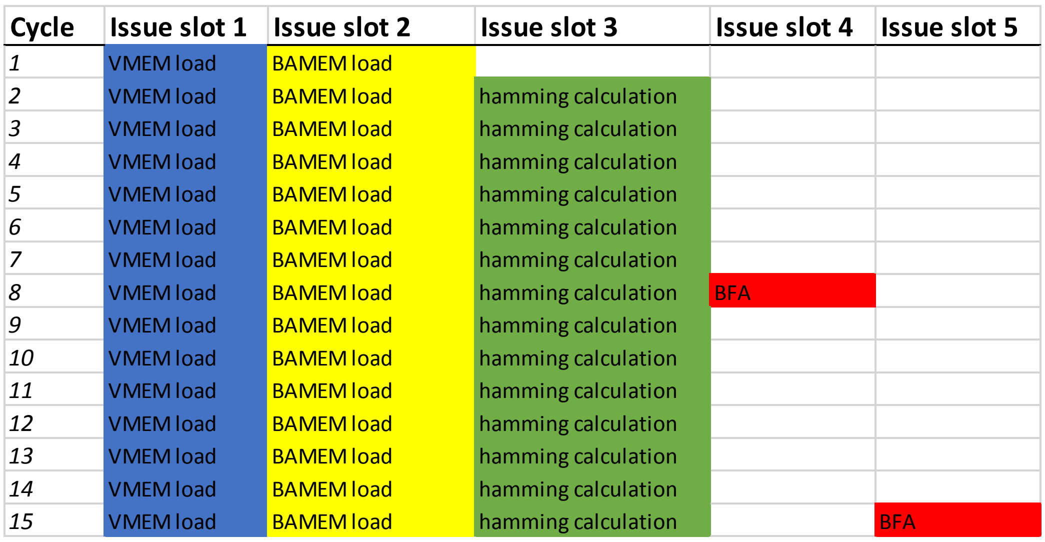


Figure 7.3: Hardware configurations based on critical resources

By plotting the data found in table 7.2 we can observe more easily the effects of increasing these resources. Graph 7.3 show that the maximum performance does not increase significantly bellow 10 million clock cycles. This proves that processing elements are no longer the bottleneck and that memory loads could again be the problem.



Cycle	Issue slot 1	Issue slot 2	Issue slot 3	Issue slot 4	Issue slot 5
1	VMEM load	BAMEM load			
2	VMEM load	BAMEM load	hamming calculation		
3	VMEM load	BAMEM load	hamming calculation		
4	VMEM load	BAMEM load	hamming calculation		
5	VMEM load	BAMEM load	hamming calculation		
6	VMEM load	BAMEM load	hamming calculation		
7	VMEM load	BAMEM load	hamming calculation		
8	VMEM load	BAMEM load	hamming calculation	BFA	
9	VMEM load	BAMEM load	hamming calculation		
10	VMEM load	BAMEM load	hamming calculation		
11	VMEM load	BAMEM load	hamming calculation		
12	VMEM load	BAMEM load	hamming calculation		
13	VMEM load	BAMEM load	hamming calculation		
14	VMEM load	BAMEM load	hamming calculation		
15	VMEM load	BAMEM load	hamming calculation		BFA

Figure 7.4: Schedule of the algorithm using two accelerators

By consulting the scheduling diagram seen in 7.4 we can easily conclude that this is the case. Even with only two accelerators, the memory system struggles to provide enough data for calculation. Since adding more memory would be too expensive, we can conclude that this is the maximum speed obtainable.

Graph 7.5 shows the effects that the increased number of resources has on speed and area. The extra hardware area is negligible compared to the gained performance.

7.4 Image quality evaluation

Before actually implementing the algorithm on the VLIW core, a bit-accurate program was created in Matlab. This was used to facilitate the evaluation of quality since this program is much faster than an actual simulation on the hardware simulator.

By using the Matlab script and sending the resulting images to the Middlebury website for evaluation, the reported number of pixels detected correctly is 85%. The resulting disparity map of all 4 images is shown in figure 7.6.

In figure 7.7 a more detailed analysis of the impact of each change is analysed. It is clear that the use of the sparse census transform instead of the original is the only elements that has any impact on actual image quality. The effect of this modification is small enough that it can be neglected.

Unfortunately, there is no precision information regarding the original DeepSea 3 implementation so no accurate comparison between the two can be performed. Since the

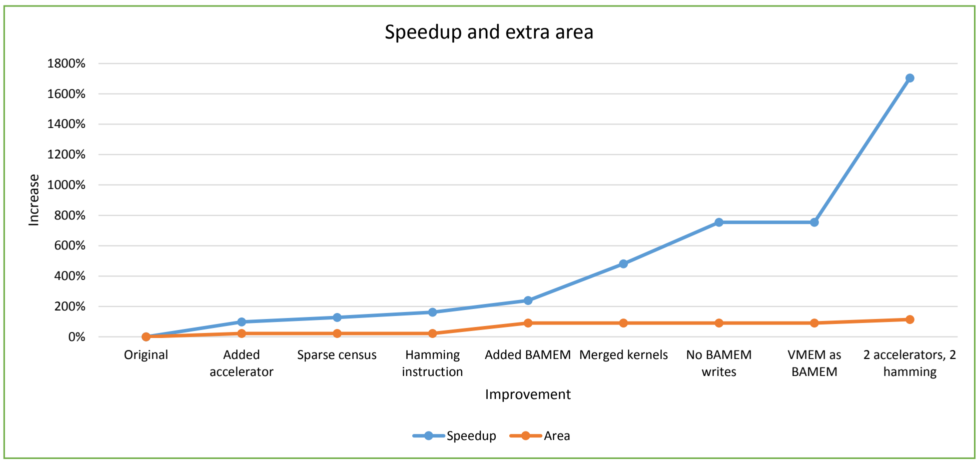


Figure 7.5: The final speedup and extra area

implementation follows the specifications of the ASIC, it is expected that the precision is similar.

7.5 Modified architecture

Figure 7.8 shows the structure of the core after all the previously described modifications have been made. No elements were removed and only the necessary functional units were added.

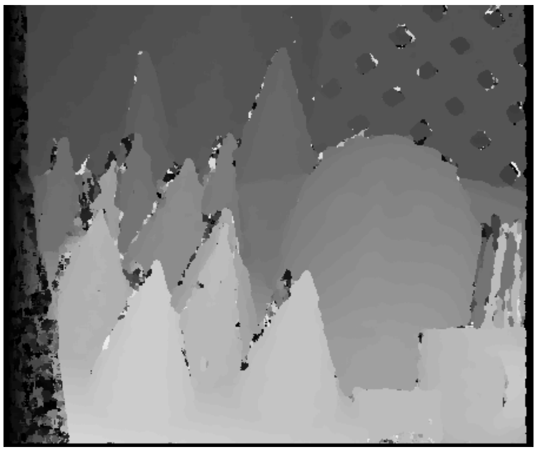
The newly added blocks added in the figure represent the following elements:

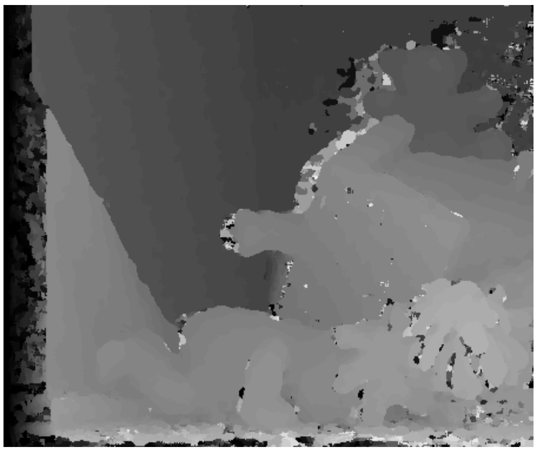
- **BAMEM** - Block access memory
- **BMLSU** - Block access memory load/store unit
- **H** - Hamming distance functional unit
- **ACC** - Accelerator

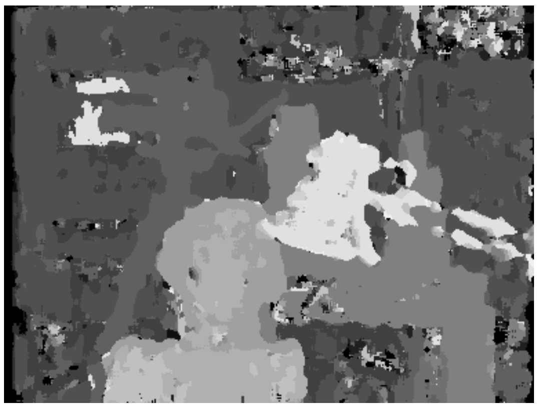
A load/store unit needs to be connected to the newly added memory in order to access the elements found here. The functional unit can be placed in any issue slot but will most likely keep it occupied since the unit will be heavily utilised.

Two more issue slots had to be added in order to accommodate the accelerators. Adding the accelerators in the already available issue slots would have generated too much register pressure.

The two new hamming distance functional units can be added in any issue slot present in the core. since they do not put too much pressure on the corresponding registers.


(a) Cones disparity map


(b) Teddy disparity map


(c) Tsukuba disparity map

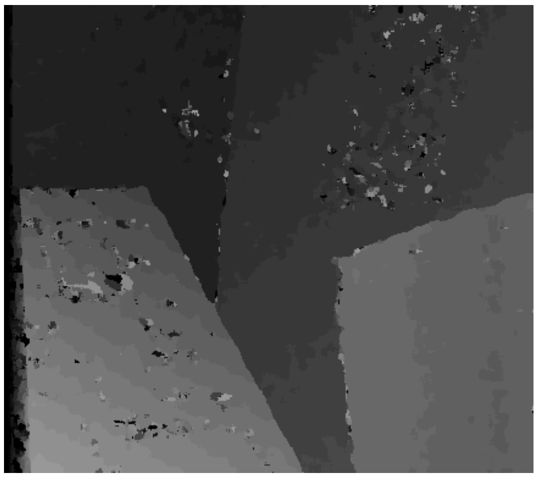

(d) Venus disparity map

Figure 7.6: The resulting disparity map of all 4 images on the Middlebury website

7.6 Conclusion

By adding extra hardware blocks and changing the algorithm so that it takes advantage of the increased hardware resources, real-time performance is reached while maintaining the image quality constant. Based on the requirements of the final application, a different hardware configuration can be chosen.

Using all the improvements and tradeoffs previously described, a single core of the system can reach a maximum of 36.75fps for a resolution of 640x480 with 64 disparities. This can be translated to 722 Mde/s using the metrics defined in chapter 2.

The cost for reaching this speed is paid by increasing the core area. In order to reach this speed,one block access memory, a specialised instruction and two bilateral filter accelerators need to be added. These improvements increase the total core area by +114,05%.

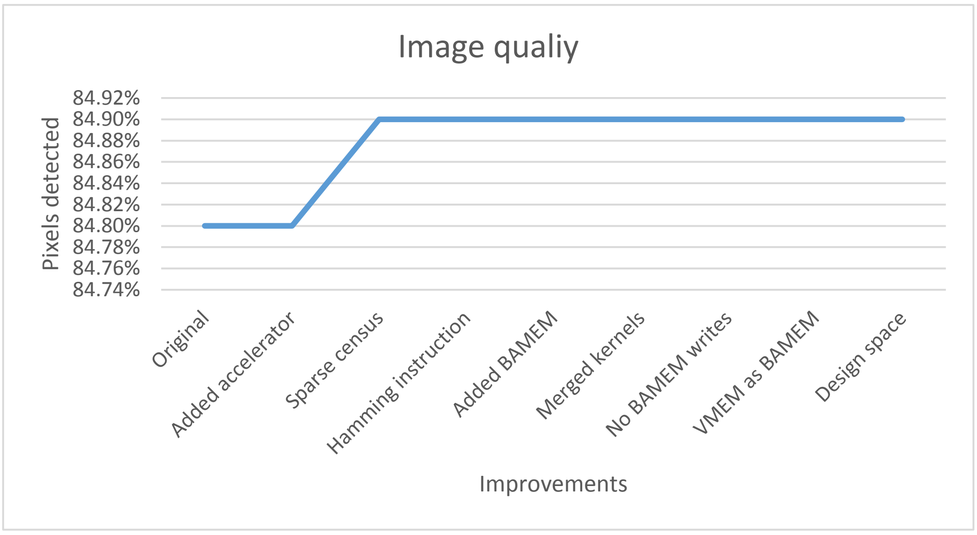


Figure 7.7: The effects on image quality of each modification

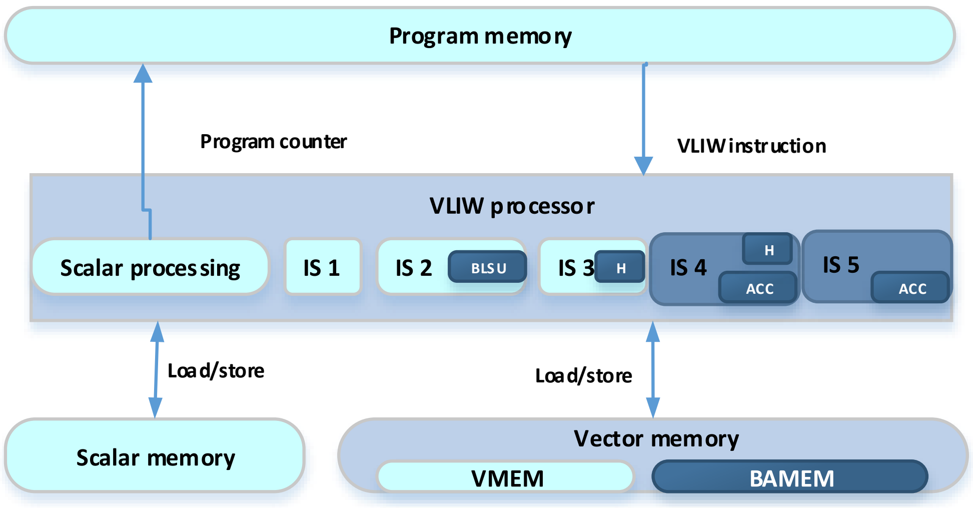


Figure 7.8: The modified version of the core.

Table 7.3 shows that compared to other available processor implementations, the Mde/s obtained using the solution proposed in this work achieves the best performance. Only two other VLIW cores were able to reach performance similar to the Intel IPU. The Tensilica LX2 and the Generic VLIW platform are the closest competitors to the implementation presented here.

Only one of the implementations here reports the accuracy obtained. This is the TI

Platform	Resolution	fps	Mde/s	Algorithm
Intel Pentium IV [23]	320x240 (32)	21	51	Census + Fixed RAU
Core 2 Duo [25]	320x240 (30)	42	96	Census + Fixed RAU
Generic VLIW [27]	640x480 (64)	30	589	Rank + SGM
Tensilica LX2 [26]	640x480 (64)	20	393	Rank + SGM
Freescale P4080 [22]	640x480 (91)	1.5	42	Rank + SGM
AMD Opteron [22]	225x187 (32)	26	79	Rank + SGM
TI C6416 [24]	640x480 (50)	3.8	58	Census + Fixed RAU
Intel IPU	**640x480** (64)	**36.75**	**722**	**Census + Fixed RAU**

Table 7.3: Available general purpose processor implementations compared to the Intel IPU

C6416 version, which reports detecting 90% of the pixels accurately.

The original Intel ASIC is able to reach 2.555 Mde/s. Since using a single core of the IPU a maximum speed of only 722/s can be obtained, the speed of the application is decreased 3.53 times. This decrease in performance is expected and is acceptable, taking into account the decrease in cost of the final system.

We can conclude that the maximum speed potential of the application on the VLIW core has been reached and modifying the memory system or changing the SIMD instructions to process more than 32 elements at a time are the only options for more speed.

Conclusions and future work 8

This chapter presents the conclusions of the work and highlights improvements that can be done in the future.

8.1 Conclusions

In this thesis, we addressed the acceleration of stereo vision on an Intel Image Signal Processor. The purpose of this was to investigate how real-time performance can be obtained, without using a dedicated ASIC and thus reduce the cost of the final product.

The Intel ISP is a suitable platform for implementing deep detection and can achieve considerable speed compared to other processors available at the moment. The SIMD instructions and the multiple functional units provide an unmatched level of parallelism which greatly benefits this type of computer vision algorithms.

The design of the DeepSea 3 ASIC was the initial starting point for the algorithm, but changes had to be made to adjust it to the platform. These changes provide considerable speed improvement, but the greatest performance enhancement is gained by using taking full advantage of the parallel nature of the VLIW core.

During the initial porting of the algorithm to the Intel Core, it was noticed that the SIMD resources of the algorithm were not used at full capacity. By adjusting the algorithm, the resulting accuracy of the images was only slightly modified.

Memory that supports unaligned loads had to be added, an instruction to calculate the similarity between pixels and an accelerator for the most computational intensive tasks were needed. Since these are the critical resources for this application, by experimenting with different such configurations, the tradeoff between area and speed was explored.

Because the algorithm produces a large amount of data, frequent memory accesses are required. Recalculating data and not storing intermediate results had a surprisingly positive influence on the speed of the final applications. Adding more processing power is less expensive than improving the memory system, which explains this result.

After all enhancements, a speed of 36.75 fps was reached by choosing a configuration out of all possible ones due to the fact that it is Pareto optimal and is able to reach real-time speed. The cost for this is a 114.05% increase in area. If this is not sufficient for the final application, another configuration can be chosen.

8.2 Future work

The speed of the enhanced core is unmatched when comparing to other similar solutions. Despite this, more work can be done to see how higher image accuracy can be obtained.

A few possible directions for obtaining this are the following:

- Adding **edge** information to the accelerator. By reconfiguring the accelerator to take into account edge information, smaller objects will be detected more accurately. This would reduce the blur effect caused by aggregating pixels that belong to different objects. A challenge that encountered here is finding a way to deal with the increased register pressure that this task demands.

- Add **intensity difference** information to the matching cost. Although the census transform is very robust, some implementation have managed to increase its effectiveness by combining with pixel intensity difference information. Images with low textures are the ones that are most likely to benefit from this.

- Add **SGM** to improve region aggregation. Semi-global matching is a reasonable compromise between local matching and global matching. By calculating it, alongside local matching, the resulting quality of the image can further improve. A problem with this technique is that it is very memory intensive, and the current system might not be powerful enough for it.

- Add **post processing** to fill in incorrectly detected pixels. Incorrectly matched pixels can be filled in by adding information from neighbouring ones. A median filter would be the simplest solution for this.

All this elements can be added alongside the already existing ones. It is expected that the increase in accuracy will come at a cost in number of frames processed.

Bibliography

[1] M. Humenberger, C. Zinner, M. Weber, W. Kubinger, and M. Vincze, "A fast stereo matching algorithm suitable for embedded real-time systems," *Computer Vision and Image Understanding*, vol. 114, no. 11, pp. 1180 – 1202, 2010, special issue on Embedded Vision.

[2] W. Fife and J. Archibald, "Improved census transforms for resource-optimized stereo vision," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 23, no. 1, pp. 60–73, Jan 2013.

[3] S. Nedevschi, S. Bota, and C. Tomiuc, "Stereo-based pedestrian detection for collision-avoidance applications," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 10, no. 3, pp. 380–391, Sept 2009.

[4] S. Battiato, G. Farinella, O. Giudice, S. Cafiso, and A. Di Graziano, "Vision based traffic conflict analysis," in *AEIT Annual Conference, 2013*, Oct 2013, pp. 1–6.

[5] G.-T. Michailidis, R. Pajarola, and I. Andreadis, "High performance stereo system for dense 3-d reconstruction," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 24, no. 6, pp. 929–941, June 2014.

[6] Intel. Intel real sense. [Online]. Available: http://www.intel.com/content/www/us/en/architecture-and-technology/realsense-overview.html

[7] G. Paya-Vaya, J. Martin-Langerwerf, P. Taptimthong, and P. Pirsch, "Design space exploration of media processors: A parameterized scheduler," in *Embedded Computer Systems: Architectures, Modeling and Simulation, 2007. IC-SAMOS 2007. International Conference on*, July 2007, pp. 41–49.

[8] B. Tippetts, D. Lee, K. Lillywhite, and J. Archibald, "Review of stereo vision algorithms and their suitability for resource-limited systems," *Journal of Real-Time Image Processing*, pp. 1–21, 2013.

[9] R. Szeliski, *Computer Vision: Algorithms and Applications*, 1st ed. New York, NY, USA: Springer-Verlag New York, Inc., 2010.

[10] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *Int. J. Comput. Vision*, vol. 47, no. 1-3, pp. 7–42, Apr. 2002.

[11] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, 1st ed. Norwell, MA, USA: Kluwer Academic Publishers, 1997.

[12] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library.* Cambridge, MA: O'Reilly, 2008.

[13] R. Zabih and J. Woodfill, “Non-parametric local transforms for computing visual correspondence,” in *Proceedings of the Third European Conference on Computer Vision (Vol. II)*, ser. ECCV ’94. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1994, pp. 151–158.

[14] H. Hirschmuller, “Stereo processing by semiglobal matching and mutual information,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 30, no. 2, pp. 328–341, Feb 2008.

[15] Middlebury university. Stereo video benckmark. [Online]. Available: http://vision.middlebury.edu/stereo/

[16] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets robotics: The kitti dataset,” *International Journal of Robotics Research (IJRR)*, 2013.

[17] M. Brown, D. Burschka, and G. Hager, “Advances in computational stereo,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 25, no. 8, pp. 993–1008, Aug 2003.

[18] F. Joseph, K. Francis, A. Hore, S. Roy, S. Josephine, and R. Paily, “An efficient hardware architecture for stereo disparity estimation,” in *VLSI Design and Test, 18th International Symposium on*, July 2014, pp. 1–6.

[19] K. Ambrosch, M. Humenberger, W. Kubinger, and A. Steininger, “Sad-based stereo matching using fpgas,” in *Embedded Computer Vision*, ser. Advances in Pattern Recognition, B. Kisaanin, S. Bhattacharyya, and S. Chai, Eds. Springer London, 2009, pp. 121–138.

[20] S. Thomas, K. Papadimitriou, and A. Dollas, “Architecture and implementation of real-time 3d stereo vision on a xilinx fpga,” in *Very Large Scale Integration (VLSI-SoC), 2013 IFIP/IEEE 21st International Conference on*, Oct 2013, pp. 186–191.

[21] S. Jin, J. Cho, X. D. Pham, K. M. Lee, S.-K. Park, M. Kim, and J. Jeon, “Fpga design and implementation of a real-time stereo vision system,” *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 20, no. 1, pp. 15–26, Jan 2010.

[22] O. Arndt, D. Becker, C. Banz, and H. Blume, “Parallel implementation of real-time semi-global matching on embedded multi-core architectures,” in *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIII), 2013 International Conference on*, July 2013, pp. 56–63.

[23] Y. K. Baik, J. H. Jo, and K. M. Lee, “Fast census transform-based stereo algorithm using SSE2,” *The 12th Korea-Japan Joint Workshop on Frontiers of Computer Vision*, pp. 305 – 309, 2006.

[24] M. Humenberger, C. Zinner, and W. Kubinger, “Performance evaluation of a census-based stereo matching algorithm on embedded and multi-core hardware,”

in *Image and Signal Processing and Analysis, 2009. ISPA 2009. Proceedings of 6th International Symposium on*, Sept 2009, pp. 388–393.

[25] C. Zinner, M. Humenberger, K. Ambrosch, and W. Kubinger, "An optimized software-based implementation of a census-based stereo matching algorithm," in *Advances in Visual Computing*, ser. Lecture Notes in Computer Science, G. Bebis, R. Boyle, B. Parvin, D. Koracin, P. Remagnino, F. Porikli, J. Peters, J. Klosowski, L. Arns, Y. Chun, T.-M. Rhyne, and L. Monroe, Eds. Springer Berlin Heidelberg, 2008, vol. 5358, pp. 216–227.

[26] C. Banz, C. Dolar, F. Cholewa, and H. Blume, "Instruction set extension for high throughput disparity estimation in stereo image processing," in *Application-Specific Systems, Architectures and Processors (ASAP), 2011 IEEE International Conference on*, Sept 2011, pp. 169–175.

[27] G. Paya-Vaya, J. Martin-Langerwerf, C. Banz, F. Giesemann, P. Pirsch, and H. Blume, "Vliw architecture optimization for an efficient computation of stereoscopic video applications," in *Green Circuits and Systems (ICGCS), 2010 International Conference on*, June 2010, pp. 457–462.

[28] R. Jakovljevic, "BAMEM2 user manual, Parallel Block-access MEMory subsystem of ISPs in Broxton IUNIT," in *Intel Internal document*, December 2013, revision 0.1.

[29] J. Woodfill, G. Gordon, D. Jurasek, T. Brown, and R. Buck, "The tyzx deepsea g2 vision system, ataskable, embedded stereo camera," in *Computer Vision and Pattern Recognition Workshop, 2006. CVPRW '06. Conference on*, June 2006, pp. 126–126.

[30] H. Hirschmuller and D. Scharstein, "Evaluation of cost functions for stereo matching," in *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, June 2007, pp. 1–8.

[31] ——, "Evaluation of stereo matching costs on images with radiometric differences," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 31, no. 9, pp. 1582–1599, Sept 2009.

[32] X. Mei, X. Sun, M. Zhou, shaohui Jiao, H. Wang, and X. Zhang, "On building an accurate stereo matching system on graphics hardware," in *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, Nov 2011, pp. 467–474.

[33] S. B. Kang, R. Szeliski, and J. Chai, "Handling occlusions in dense multi-view stereo," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1, 2001, pp. I–103–I–110 vol.1.

[34] O. Veksler, "Fast variable window for stereo correspondence using integral images," in *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, vol. 1, June 2003, pp. I–556–I–561 vol.1.

[35] J. Fang, A. Varbanescu, J. Shen, H. Sips, G. Saygili, and L. van der Maaten, "Accelerating cost aggregation for real-time stereo matching," in *Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on*, Dec 2012, pp. 472–481.

[36] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach*, 5th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[37] M. McDonnell, "Box-filtering techniques," *Computer Graphics and Image Processing*, vol. 17, no. 1, pp. 65 – 70, 1981.

[38] F. C. Crow, "Summed-area tables for texture mapping," in *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '84. New York, NY, USA: ACM, 1984, pp. 207–212.

[39] E. V. Dalen, "Bilateral Filter unit - Acceleration Specification," in *Intel Internal document*, October 2014, revision 1.1.

[40] J. V. D. Bor, "Flexible hardware accelerator for 2D Convolution and Census Transform," in *Master Thesis*, August 2015, revision 1.

[41] N.-C. Chang, T.-H. Tsai, B.-H. Hsu, Y.-C. Chen, and T.-S. Chang, "Algorithm and architecture of disparity estimation with mini-census adaptive support weight," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 20, no. 6, pp. 792–805, June 2010.