2016 IEEE 28th International Symposium on Computer Architecture and High Performance Computing

An Image Processing VLIW Architecture for Real-Time Depth Detection

Dan Iorga†, Razvan Nane†, Yi Lu*, Edwin Van Dalen*, Koen Bertels †
† Computer Engineering Research Group, Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands
Email: {D.Iorga, R.Nane, K.L.M.Bertels}@tudelft.nl
*Intel Benelux B.V., High Tech Campus 83, Eindhoven, 5656 AG, The Netherlands
Email: {edwin.van.dalen, yi1.lu}@intel.com

***Abstract*—Numerous applications for mobile devices require 3D vision capabilities, which in turn require depth detection since this enables the evaluation of an object's distance, position and shape. Despite the increasing popularity of depth detection algorithms, available solutions need expensive hardware and/or additional ASICs, which are not suitable for low-cost commodity hardware devices. In this paper, we propose a low-cost and low-power embedded solution to provide high speed depth detection. We extend an existing off-the-shelf VLIW image processor and perform algorithmic and architectural optimizations in order to achieve the requested real-time performance speed. Experimental results show that by adding different functional units and adjusting the algorithm to take full advantage of them, a 640x480 image pair with 64 disparities[1] can be processed at 36.75 fps on a single processor instance, which is an improvement of 23% compared to the best state-of-the-art image processor.**

I. Introduction

The depth at which objects are located can be estimated by using two or more cameras and finding corresponding pixels in each image. Driven by the rapid advance of technology, the price of video cameras is continuously decreasing making the addition of multiple cameras to entry-level hardware devices affordable, cost-effective. As a result, the gathering of depth information, which is used for advanced applications such as gesture control [1] or 3D reconstruction[2], becomes economically feasible. However, algorithms that extract depth information from video stereo camera systems have a high computational load. Therefore, in order to process captured images in real-time (i.e. 30 frames per second (fps)), they require expensive high-end platforms that are not suitable for low-cost and low-energy commodity hardware systems such as tablets.

There is a large number of articles that provide solutions based on Field-Programmable Gate Array (FPGA)s, Graphical Processing Unit (GPU)s, and Application-Specific Integrated Circuit (ASIC)s that can achieve high frame rates and excellent image quality. However, each of these devices have their limitations: FPGAs require substantial implementation time and are expensive, embedded GPUs, such as the NVIDIA Tegra K1 [3], require higher power consumption than most mobile devices can offer, while ASICs have a long and complex design process and therefore can not be easily modified. For example, Intel utilizes a high-performance ASIC [4] alongside their Image Processor Unit (IPU) in their Merrifield (Atom Z3460/Z3480) and Moorefield (Atom Z3560/Z3580) products. However, its high-cost and power consumption does not make it suitable for entry-level mobile devices. Very Long Instruction Word (VLIW) processors can be very efficient for multimedia applications [5]. Consequently, our aim is to port the algorithm utilised in the ASIC on the VLIW found inside the IPU. This would lead to a considerable reduction in terms of bill of material, excluding the cost of on-chip custom hardware integration and additional Printed Circuit Board (PCB) area that is required for the ASIC implementation.

In this paper, we propose an embedded VLIW architecture for real-time depth detection that will be used in future Intel IPUs. In order to reach a high frame rate, the reference algorithm used in the ASIC implementation needs to be adapted to the possibilities of the VLIW core while maintaining the same image quality. To this purpose, we propose architectural modifications to the VLIW core found on existing Intel IPUs.

Experimental results obtained after all the modifications have been performed show that a 640x480 image pair with 64 disparities can be processed at 36.75 fps on a single processor instance. The specific contributions of the paper are:

- We propose algorithmic modifications to the reference DeepSea algorithm [4], i.e., a recalculation scheme for the hamming distance with the usage of an appropriate census transform mask.
- We propose architectural extensions for the standard VLIW core found on the Intel IPU. These extensions are a specific Instruction Set Architecture (ISA) instruction, exploiting an optimized hamming distance unit, and a dedicated accelerator for performing region aggregation.

The paper is structured as follows. In section II we provide a brief overview of related work, with a focus on VLIW processors. Section III describes the VLIW core, the algorithm, and the development tools used in this work. Then, in section IV we iteratively optimize both the algorithm and the VLIW architecture by performing architectural and algorithmic optimizations. Finally, section V presents the

[1]Disparity is defined as the distance between a pixel value from the left image and the corresponding pixel value in the right image.

978-1-5090-6108-2/16 $31.00 © 2016 IEEE
DOI 10.1109/SBAC-PAD.2016.28

experimental results while section VI concludes the paper.

II. Related Work

Many ASIC, FPGA, and GPU implementations are able to produce excellent performance both in terms of speed and quality [6]. These are usually orders of magnitude faster than any Central Processing Unit (CPU)-based design; however, the high cost of such implementations makes it unattractive for embedded general-purpose applications. Therefore, in order to remain within the small power budget for embedded hardware devices, a CPU-based implementation is required. However, current CPU-based solutions are slow and cannot achieve real-time performance levels. Off-the-shelf processors provide low speed even if the algorithm is partitioned on multiple cores [7], [8], [9]. To improve on these results, more efficient solutions were proposed that are based on VLIW processors [10]. Finally, extending VLIW cores with custom ISA instructions can be considered the best approach as proven by [11] and [12]. This paper follows the last approach. In [6], an extensive survey of existing solutions can be found.

We observe that authors either use the *rank transform* together with *Semi-Global Matching* (*SGM*) [13] or the *census transform* with *fixed Region Aggregation* (*fRA*) [14]. Unfortunately, there is no information regarding the quality of the resulting images in the above papers, and as such, it is difficult to make an evaluation between the SGM-based implementations and the fRA-based implementations in terms of achieved accuracy. Therefore, in this work we assume a target quality of service of 85% correctly detected pixels. This requirement is derived from the solution offered by Woodfill et. al [4]. This ASIC is the reference point for our VLIW alternative design for which we require the same image quality.

III. Platform and Algorithm Overview

In this section, we provide background information regarding the hardware platform used. Furthermore, we describe the computational pipeline of the depth detection algorithm that was the starting point for our implementation.

A. Hardware Platform

The Intel IPU is responsible for processing images in commercially available systems such as the Merrifield and Moorefield range of processors. The VLIW core can be programmed to do tasks for which no dedicated block exists. Because the core is responsible for performing computationally intensive image processing tasks, it contains multiple issue slots designed for either vector processing or scalar processing. Figure 1 shows a high-level overview of the VLIW architecture. The vector issue slots can process 32 elements of 16 bits at a time, resulting in a vector element of 512 bits. The core includes separate data memory for the scalar issue slots and separate data memory for the vector issue slots (VMEM). The VMEM supports only aligned loads.

The architecture of the processor can be modified using the in-house architecture description languages. This process can be observed in figure 2, where a core based on the template

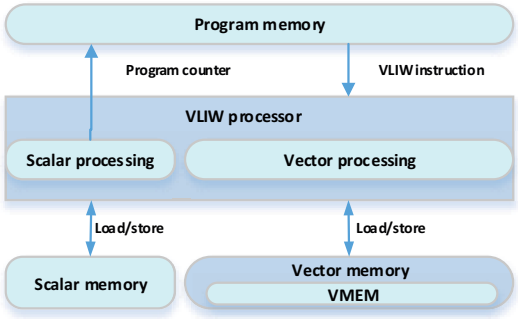

Fig. 1. The Architecture of the VLIW Core.

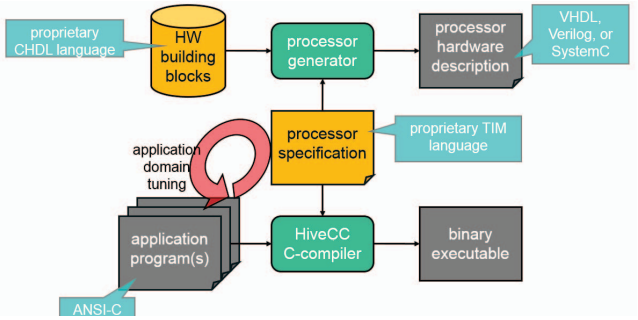

Fig. 2. The Process of Generating the Processor.

from 1 is generated using already implemented hardware blocks. This enables rapid experimentation with different configurations.

The number of issue slots, registers or memory blocks can be modified, or new dedicated functional units can be added to target specific applications. These functional units have to be previously expressed in a hardware description language such as VHDL or the in-house language known as CHDL. The interconnects between these elements needs to be clearly described such that the input/output ports receive the corresponding data.

The algorithm is compiled for the resulting architecture and simulated using the generated binary executable to obtain performance cycles and accuracy numbers. Furthermore, the generated processor hardware description files are synthesized to get the clock frequency.

B. Algorithm Description

There is an abundance of approaches that can be used to calculate the disparity map; however, most algorithms have a common structure that has been identified by Scharstein and Szeliski [15]. This methodology of structuring depth detection algorithms has been adopted by the majority of computer vision publications and consists of the following steps:

1) Preprocessing and matching cost computation (census & hamming distance)
2) Cost (support) aggregation (region aggregation)
3) Disparity computation & optimization (WTA)
4) Disparity refinement (postprocessing)

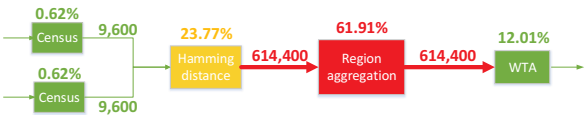

Fig. 3. Computational Pipeline Profile: Red represents Computational (% nrs. shown on boxes) or Data (data vector nrs. shown on edges) Intensive Elements, Yellow represents Medium, and Green represents Lightweight Elements.

The DeepSea3 ASIC [4] follows the previously described steps and runs at 200 frames per second for 512x480 images. It has been used successfully in many applications such as robotics. To obtain a real-time performance of this algorithm on the IPU's VLIW, we first create an equivalent straightforward implementation. However, profiling for an 640x480 image with 64 disparities on the standard configured VLIW processor revealed modest performance results (i.e., 2.04 fps).

Figure 3 shows the computational pipeline of the algorithm along with the computational and data intensive parts. The percentage of the total computational time is shown on top of each kernel and the number of 512 bits vector elements transmitted is shown on the arrows. The first step, preprocessing, is done using the census transform in order to reduce radiometric differences between the images. This is a form of non-parametric local transform, which relies on the relative ordering of pixels in the image. It maps the intensity values of pixels within a square window to a bit string, thereby capturing the structure of the image.

A cost of matching individual pixels is defined with the goal of attributing similar pixels a lower penalty in the matching process. Since the output of the previous kernel is a bit string, the hamming distance is used to establish a cost between the reference image and all possible candidates from the target image. As a result, a large amount of data is created at this step, proportional to the number of matching candidates. Therefore, this step makes the creation of an efficient pipeline difficult.

Pixels in proximity will likely have the same disparity because they belong to the same object. For more robust matching, window-based correlation is employed, which requires the sum of the hamming distances. The size of the window has been analysed in works such as [16] and based on this we have selected a window of size 7x7. The large amount of data created by the previous step needs to be processed in the region aggregation step. Our profile suggests that this is the most computational intensive kernel, requiring more than 60% of the total execution time.

Finally the best candidate is chosen by simply selecting the pixel with the smallest matching cost. When there is no clear "winner", matching can be considered unreliable and different techniques can be used for refinement. Incorrectly detected pixels can be replaced by using information from reliable matches. Because our target accuracy has already been reached, this step is not required.

IV. Depth detection optimizations

In this section, we describe the optimizations performed to obtain real-time depth detection speed. Our starting point is the implementation of the method proposed by Woodfill et al. [4], which we refer to as the original DeapSea3 algorithm. We perform both algorithmic and architectural optimizations to achieve the required performance. We start by improving the performance of each kernel and then optimize the code to reduce data transfers and increase parallelism.

A. Architectural Extensions

One of the main sources to improve the performance is to look at application kernels that occupy a large percentage of the computational pipeline. Based on profiling data shown in Figure 3, we identify *region aggregation* and *hamming distance* as candidates for acceleration. Software solutions are preferred due to the fact that they do not require any extra area. However, hardware solutions are used when no such option is available.

a) Region Aggregation: The region aggregation kernel receives a large amount of data that needs to be processed, making it the most computational intensive part of the algorithm. Software optimizations are based on box-filtering techniques [17] or on integral images techniques [18] and provide a significant speedup by taking advantage of already calculated partial sums. Despite their effectiveness, these techniques prove difficult to implement on an SIMD processor due to the irregular way in which data is processed. This motivates the addition of a dedicated **hardware accelerator** to the core. Using the previously described in-house architecture description languages, an existing bilateral filtering accelerator is added. This accelerator uses a sliding window technique by moving a square window across the image and performing calculations at each position. Internally, this filter uses a pipeline to process the data. Similar approaches have been used by Cadence in the Tensilica processor, where they also identify the potential of a bilateral filter unit for 3D depth filtering [19].

Figure 4 illustrates the input/output of the accelerator. Since the algorithm is based on a 7x7 window, a larger block of the image is required to provide information about the neighbours. This can be supplied by 6 vectors of size 4x8, where the last two lines of the block are not used. Spatial weight coefficients and intensity weight coefficients can be provided to preserve the sharpness of the image and not blur corners. As output, the filter returns the sum of pixels and the weight information, which can be multiplied to obtain the final result.

The 22% increase of the size of the core is compensated by the possibility of using this accelerator in other applications. Most noticeably, this can be used for de-noising or de-mosaicking [20].

b) Hamming distance: The number of operations required to perform the hamming distance operation depends on the size of the census mask. Since the core works with vector elements of 16 bits, the sparse census transform defined

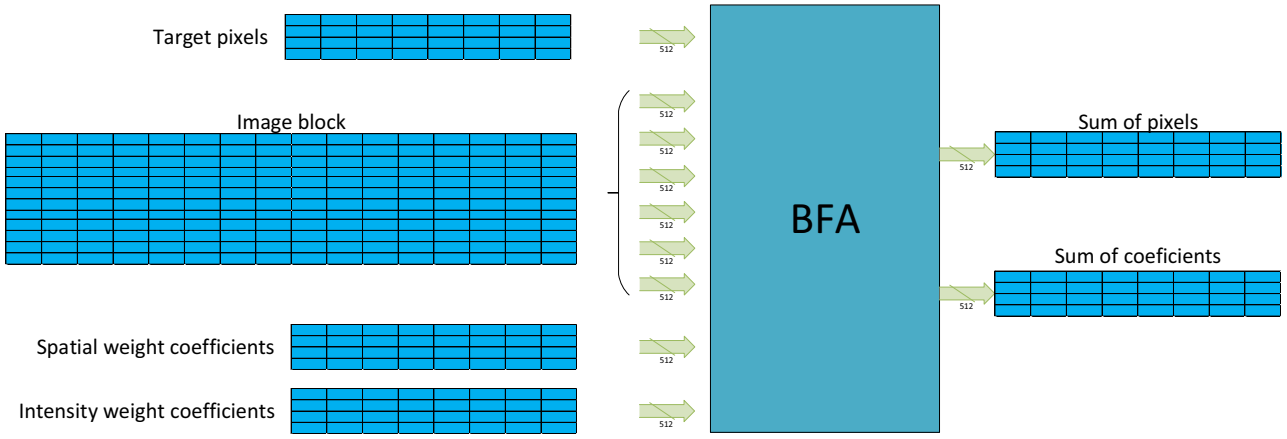

Fig. 4. The bilateral filter unit used to accelerate the region aggregation kernel. As input the accelerator takes the target pixels that need to be processed, an image block of size 16x10 for neighbouring information and the spatial and intensity weights chosen. As output, the accelerator returns the sum of pixels and the sum of coefficients for the target pixels.

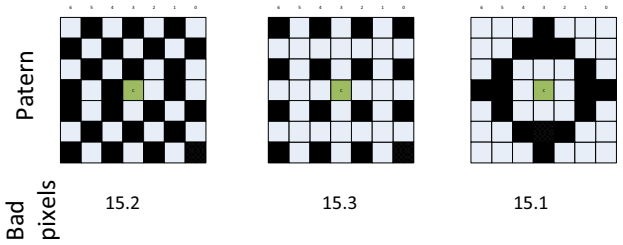

Fig. 5. Different census transform patterns and their effect on image quality. The middle one [16] and the right one [21], use a 16-bit pattern instead of 24. However, these do not increase the #bad_pixels in the final image.

in [16] and refined in [21] are considered. As shown in figure 5, the 16-bit census transform patterns, using fewer comparison points does not degrade the quality of the resulting image. All tests for this have been made by implementing the algorithm in Matlab and submitting the result to the Middlebury test framework [22]. The results from figure 5 show us that we can use any of the census patterns without any loss of quality and because the middle and right options require fewer comparisons, we prefer one of these options.

At this point, the output can be stored efficiently in vector data elements. However, to obtain the hamming distance, a series of operations have to be performed. Since these are simple operations that are repeated multiple times, we can obtain further performance increase by merging them. Therefore, the second optimization performed for this kernel is to create a **hamming distance instruction** as shown in figure 6. The new instruction performs a *XOR* on two vectors and then counts the number of ones in the result using an (m,k) counter [23]. The final number is clipped so that unreliable matches are discarded. This peephole optimization technique allows the replacement of a bundle of instructions with a single instruction.

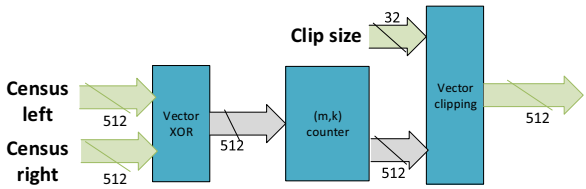

Fig. 6. The New Hamming Instruction

Since the operation requires unaligned loads from memory, a memory system capable of performing such operations is used. To this purpose, the block access memory described in [24] is added to the core.

B. Algorithmic Optimizations

Since the hamming distance kernel produces a large amount of data, stores to external memory have to be performed. This prevents the creation of an efficient pipeline. A solution is to merge the kernels and keep the generated data in registers. Because of the limited amount of registers available, it is impossible to store all intermediate data. This forces us to recalculate data whenever necessary and not to store any intermediate results. More explicitly, this is done by merging the loops of the hamming distance, region aggregation and winner takes it all kernel and calculating the data for the accelerator only when it is required. Since the accelerator is based on a sliding window technique, neighbouring hamming input data overlaps. To store this data would require a large amount of registers. Figure 7 shows the dataflow of the algorithm, where operations 2-14 will be repeated in the following iteration, despite the fact that they create the same data.

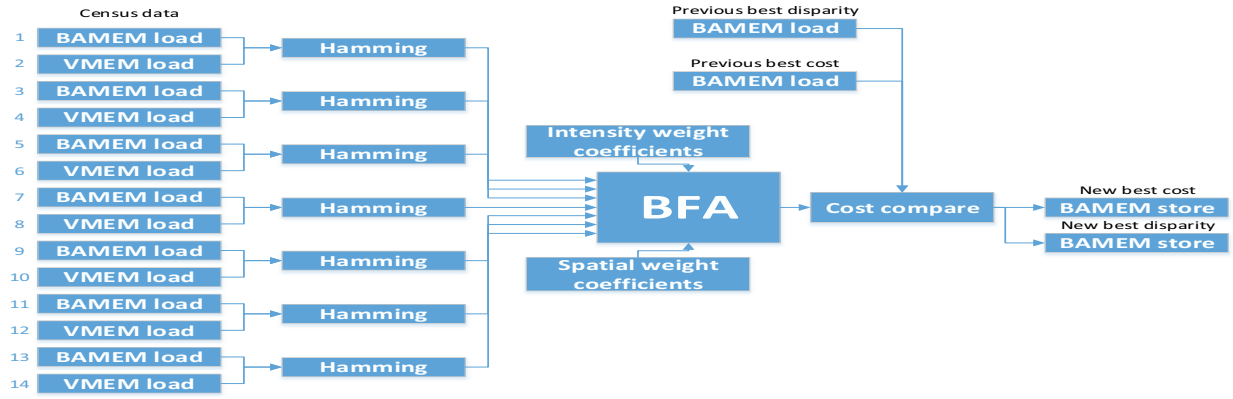

Fig. 7. The dataflow of the final algorithm. ”BAMEM loads” are LOADS from the block access memory and ”VMEM loads” are LOADS from the simple vector memory, followed by the newly hamming distance operations, and then processing in the bilateral filter accelerator.

This algorithmic modification centers the entire algorithm around the accelerator by removing any idle clock cycles in order to create an efficient software pipeline, without high register pressure. Even though the block access memory has to reload the same data, the fact that the load address are neighbouring allows for an efficient pipeline.

To ensure that we have not introduced a new bottleneck by using a large amount of block access memory loads, the final improvement is to store only the target image in this memory and to keep the reference image in the simple vector memory. By loading data from two sources instead of one, the speed of the algorithm is limited only by the performance of the bilateral filter accelerator and the hamming distance processing.

Figure 8 illustrates how memory accesses are avoided by writing data directly into the registers corresponding to the functional units that need the data. This data is sent on demand to the functional units, therefore no external memory access is required.

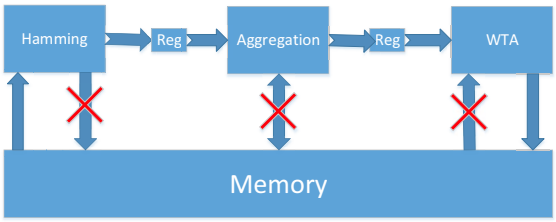

Fig. 8. Avoid Memory Writes by Recomputing Data.

V. Experimental results

In this section, we describe the experimental setup and subsequently perform an evaluation to study the impact on speed and area of the design options described in the previous section.

A. Experimental Setup

In order to test the application, we use a simple test environment. This consists of a host processor alongside the VLIW core connected by a system bus. The host processor is responsible for starting the VLIW core, sending raw image data to the core and reading the output data. We consider this a realistic scenario, where the VLIW core acts as a coprocessor.

Experiments with different numbers of functional units and accelerators were made possible by using the in-house hardware description languages. In order to measure the execution time, a cycle accurate simulator was used. The final version of the core was synthesized at a frequency of 500Mhz.

For all tests, an image pair from the Middlebury test framework[15] 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 gray-scaled and resized to 640x480 in order to easily compare it to other implementations.

B. Performance Evaluation

In figure 9, a plot of the speedup and total area[2] provides information on the significance of each element. The accelerator increases the speed of the application by 97% and the total size of the core by 22%.

The hamming distance instruction and the block access memory further increase the speed of the application to 238% but also have a significant impact on size. The hamming distance functional unit is very small and only contributes by 0.32%. However the newly added memory for unaligned loads is expensive and contributes with 67%.

The biggest improvement is obtained by taking advantage of the parallel nature of the VLIW core and avoiding memory accesses when computing the distances. Recalculating the hamming distance is faster than storing and loading it. The issue slots that have been freed by using the accelerator

[2]Due to confidentiality reasons we can not report the exact area of the core

can be used for these calculations. 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 elements to determine their effect on performance.

C. Image quality

To test the quality of the resulting depth map, the framework proposed in [15] is used. This average percentage of bad pixels is measured by comparing the result of four test images to the ground truth; this has become a standard for determining the accuracy of stereo vision algorithms.

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

Table I shows the average number of badly corrected pixels is 14.9%. Although the quality of the image is not high compared to other algorithms, it is more than enough for a low-power, low-cost embedded solution. The average quality of the resulting disparity map can be explained by the fixed region aggregation used in our implementation. The quality of the algorithm can be increased but at a cost of a reduced number of frames per second.

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

TABLE I
QUALITY METRICS OF OUR ALGORITHM

D. Area-Performance Trade-off Exploration

Due to the previously added (separate) vector memories, we are able to load multiple data at the same time. As a result, incorporating more processing elements on the core is meaningful because these will be able to access new data and perform different operations in parallel. To understand the potential of adding more functional units, we explore the area-performance trade-off. Concretely, the region aggregation accelerator and the hamming distance functional units are the critical resources, and as such these resources can be increased for more processing power. This has no impact on the quality of the resulting image since the retargetable compilation flow (shown in figure 2) can automatically adapt to the new hardware architecture. The experimental results related to this design space exploration are shown in figure 10. It is clear that configurations with more than one hamming distance function unit are Pareto optimal and are the ones that should be considered.

We have shown that using more than two accelerators does not have a considerable impact on the performance of the application. By checking the schedule of the system, we notice that the memory system is not able to provide data fast enough for the accelerators to work at full capacity. Therefore, using only two accelerators and two hamming distance functional units seem to be the best configuration.

To further increase the speed of the application, another memory can be added to the system for more parallel loads. This configuration already satisfies the target real-time requirements and adding additional memory would unnecessarily increase the cost. By using this configuration and synthesizing the core at 500Mhz, a frame rate of 36.75 can be obtained for VGA images with 64 disparities.

In order to compare different depth detection solutions, we use a metric based on the image size and the number of disparities performed. This metric was proposed in [6] and defines the millions of disparity evaluations per

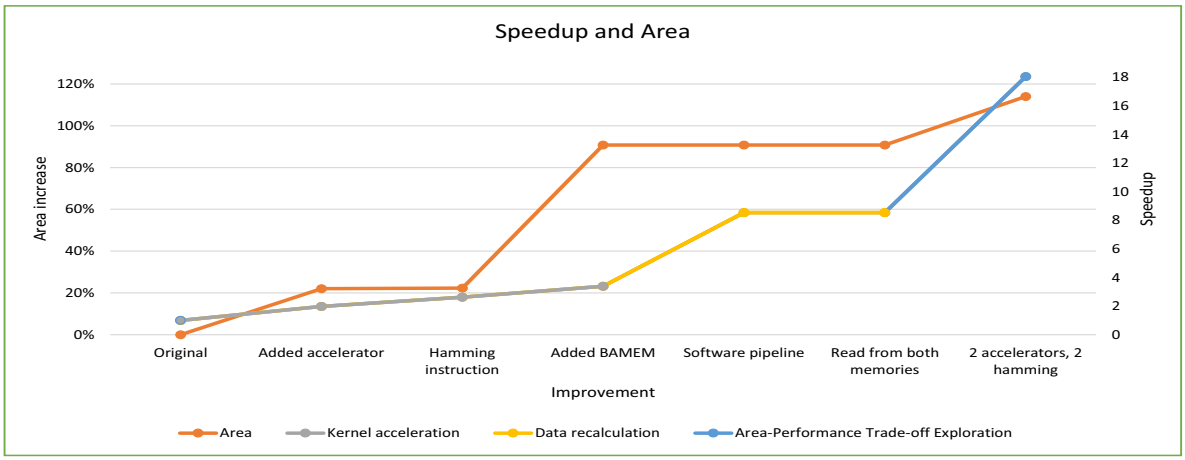

Fig. 9. The Speedup and Area after each Modification.

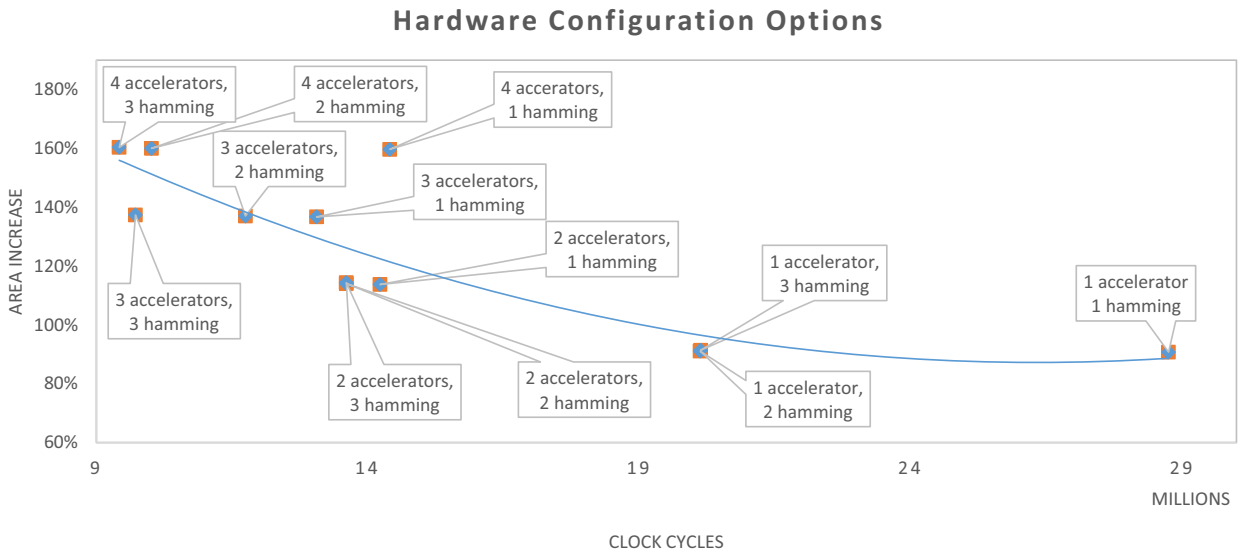

Fig. 10. Hardware Configurations based on Critical Resources.

second (Mde/s):

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

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

Table II lists the available CPU-based implementations found in literature alongside their performance results. Multi-core off-the-shelf processors provide modest speed results regardless of the algorithm that they use. Modified VLIW cores are able to achieve the highest performance [12], [11] by using dedicated instructions and enhancing the memory system. Compared to other available implementations, our solution is able to obtain the highest Mde/s.

Platform	Resolution	fps	Mde/s	Algorithm
Freescale P4080 [7]	640x480 (91)	1.5	42	Rank + SGM
Intel Pentium IV [8]	320x240 (32)	21	51	Cens + fRA
AMD Opteron [7]	225x187 (32)	26	79	Rank + SGM
Core 2 Duo [9]	320x240 (30)	42	96	Cens + fRA
TI C6416 [10]	640x480 (50)	3.8	58	Cens + fRA
Tensilica LX2 [12]	640x480 (64)	20	393	Rank + SGM
Generic VLIW [11]	640x480 (64)	30	589	Rank + SGM
Intel IPU's VLIW	**640x480 (64)**	**36.75**	**722**	**Cens + fRA**

TABLE II
AVAILABLE GENERAL PURPOSE PROCESSOR IMPLEMENTATIONS

A comparison with FPGAs and GPUs can also be made but since our goal is to target only low-power solutions, we have not included it. Our solution requires less than 1W of power (due to confidentiality reasons we can not report the exact number) while embedded GPUs such as the NVIDIA Tegra require orders of magnitude more than that. The work described in [12], involving the Tensilica LX2 core does not offer exact numbers but based on [25] we can expect that their power consumption is also under 1W.

VI. Conclusion

In this paper, we presented both algorithmic and architectural changes to provide a low-power low-cost extension of the existing IPU's VLIW core. These modifications will lead to a considerable reduction in terms of bill of material, excluding the cost of on-chip custom hardware integration and additional PCB area that is required for the ASIC implementation.

We proposed the use of a dedicated bilateral filter accelerator in order to perform region aggregation as well as a fast hamming distance instruction. The changes in the algorithm consist of using a newer version of the census mask and recalculating the hamming distance information in order to avoid expensive memory operations. These methods allowed us to obtain significant speedup compared to other available solutions. After all enhancements, a real-time speed of 36.75 fps was achieved at the cost of a 114.05% increase in processor area, which is only a fraction of the entire SoC.

REFERENCES

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

[2] 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.

[3] NVIDIA. Nvidia tegra k1. [Online]. Available: http://www.nvidia.com/object/tegra-k1-processor.html

[4] 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.

[5] G. Paya-Vaya, J. Martin-Langerwerf, P. Taptimthong, and P. Pirsch, "Design space exploration of media processors: A parameterized scheduler," 2007.

[6] 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.

[7] 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.

[8] 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.

[9] C. Zinner, M. Humenberger, K. Ambrosch, and W. Kubinger, "An optimized software-based implementation of a census-based stereo matching algorithm," 2008.

[10] M. Humenberger, C. Zinner, and W. Kubinger, "Performance evaluation of a census-based stereo matching algorithm on embedded and multi-core hardware," 2009.

[11] 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," 2010.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

[18] 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.

[19] Cadence. Tensilica image vision processing. [Online]. Available: http://ip.cadence.com/ipportfolio/tensilica-ip/image-vision-processing

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

[21] 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.

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

[23] O. Spaniol, *Computer Arithmetic: Logic and Design*, 28th ed. New York, NY, USA: John Wiley & Sons, Inc., 1981.

[24] R. Jakovljevi, A. Beri, E. van Dalen, and D. Miliev, "New access modes of parallel memory subsystem for sub-pixel motion estimation," *Journal of Real-Time Image Processing*, pp. 1–18, 2014. [Online]. Available: http://dx.doi.org/10.1007/s11554-014-0481-3

[25] Cadence. Tensilica power consumption. [Online]. Available: $http://ip.cadence.com/uploads/pdf/xtensa\_LX2\_April07.pdf$