Accelerating Real-Time, High-Resolution Depth Upsampling on FPGAs

by

David Langerman

B.S. Computer Engineering, South Dakota School of Mines & Technology, 2017

Submitted to the Graduate Faculty of

the Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Master of Science in Electrical Engineering

University of Pittsburgh

2019

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This thesis was presented

by

David Langerman

It was defended on

March 22, 2019

and approved by

Jingtong Hu, Ph.D., Assistant Professor
Department of Electrical and Computer Engineering

Alex Jones, Ph.D., Professor
Department of Electrical and Computer Engineering

Thesis Advisor: Alan D. George, Ph.D., Mickle Chair Professor
Department of Electrical and Computer Engineering

Accelerating Real-Time, High-Resolution Depth Upsampling on FPGAs

David Langerman, M.S.

University of Pittsburgh, 2019

While the popularity of high-resolution, computer-vision applications (e.g. mixed reality, autonomous vehicles) is increasing, there have been complementary advances in time-of-flight (ToF) depth-sensor resolution and quality. These advances in ToF sensors provide a platform that can enable real-time, depth-upsampling algorithms targeted for high-resolution video systems with low-latency requirements. This thesis demonstrates that filter-based upsampling algorithms are feasible for real-time, low-power scenarios, such as those on HMDs. Specifically, the author profiled, parallelized, and accelerated a filter-based depth-upsampling algorithm on an FPGA using high-level synthesis tools from Xilinx. We show that our accelerated algorithm can accurately upsample the resolution and reduce the noise of ToF sensors. We also demonstrate that this algorithm exceeds the real-time requirements of 90 frames-per-second (FPS) and 11 ms latency of mixed-reality hardware, achieving a lower-bound speedup of 40 times over the fastest CPU-only version and a 4.7 times speedup over the original GPU implementation [1].

Table of Contents

List of Figures

1.0 Introduction

Mixed-reality (XR) technology has grown more popular in recent years and is expected to become a 1.65-billion-dollar industry by 2024 [2]. The advent of this technology introduces many challenges from both the hardware and software perspectives. It is well known that the de facto standard for "real-time" image processing was 30 FPS. Augmented reality technology completely redefines "real-time" computer vision. Advanced head-mounted displays (HMDs) for mixed-reality applications have native resolutions greater than 2160×1200 (2.5 million) pixels displaying at 90 frames per second (FPS), which translates to roughly 11 ms [3] [4] from scene capture to render. This new real-time requirement is well beyond the previously defined "real-time" benchmark of 33 ms. Any dropped frames can result in a loss of realism at best, and motion sickness at worst [5]. Considering these requirements, performing complex image-processing algorithms, such as image-based depth inference, on this high-resolution data in real time is a challenging task. In XR apps, it is often essential to be able to infer depth from images and/or sensors to display images on surfaces perceived by the user. Traditional algorithms to infer depth using camera sources alone, such as disparity mapping, fail in non-textured regions, and high-accuracy algorithms based on Markov-random-field models do not satisfy the accuracy and performance requirements of mixed-reality systems [6]. Methods based on the fusion of both color and depth information often achieve high accuracy; but, even on systems accelerated with a GPU, methods that claim real-time performance often fail to meet the strict demands of mixed-reality systems [7].

In this thesis, we demonstrate that filter-based approaches for depth upsampling can achieve XR real time on low-power devices that would be present in mixed-reality headsets. Our

headset consists of a high-resolution color camera as well as a low-resolution ToF depth sensor. The goal of this research is to perform real-time upsampling of the low-resolution ToF data to match the resolution of the color image, which will then be rendered to an HMD. This case is a typical scenario for augmented reality apps in which real-time scene depth is necessary for virtual objects to interact with real-world surfaces.

Upsampling depth data in real time from a ToF sensor is challenging due to several factors. First, ToF sensors are notoriously noisy and suffer from the "flying pixel" problem [6]. This phenomenon can introduce noise which cannot be easily modeled, making it difficult to filter with traditional methods [8]. Second, ToF sensors tend to have inadequately small resolutions, requiring them to be upsampled by a large factor to match typical rendering resolutions. This situation can be computationally intensive and tends to result in noisy, inaccurate depth images [7]. To improve accuracy, methods have been developed to fuse data from both ToF sensors and high-resolution cameras [6]; however, many of these methods are too computationally intensive to meet the real-time constraints of modern XR apps on mainstream desktop CPUs and GPUs. We propose that previously developed methods can leverage hardware acceleration using modern high-level synthesis (HLS) tools for field-programmable gate arrays (FPGAs) to achieve high-resolution, low-latency depth upsampling for mixed-reality apps. Specifically, this thesis describes a case study in which the Noise-Aware Filter for Real-Time Depth Upsampling (NAFDU) [9] is tested on a CPU platform, then accelerated using Vivado HLS and evaluated on the Xilinx Zynq UltraScale+ MPSoC device on the ZCU102 board.

2.0 Related Work

In [7] and [10], low-resolution depth data from a ToF sensor is fused with a high-resolution color image using a novel region-growing, energy-minimization algorithm. The authors show that their method achieves high accuracy on the industry-standard Middlebury dataset compared to many other methods. However, the algorithm's execution time is dependent on image content. A particularly noisy depth image could cause execution time to be very long depending on its distance from the final output. Additionally, the algorithm's iterative, nondeterministic behavior imposes severe restrictions for real-time apps.

Yuan, M. details a framework for temporal upsampling using a hybrid camera [11]. This approach achieves high accuracy, but it was shown to be unable to handle *rapid* scene-changes, such as those occurring in a typical head-mounted display scenario when a user quickly turns his or her head.

A unique approach to the problem of running augmented-reality apps on mobile devices was proposed by Shea in [12], where scene data from a mobile device was uploaded to an external cloud server with a high-powered GPU. This methodology was successful in reducing the overall power consumption of the system while also maintaining a high degree of image quality compared to processing entirely on the mobile device. However, the interaction delay of their system was 55 ms, which does not fall within the 11 ms constraint of a typical, real-time application.

In [9], Chan details a straightforward sensor-fusion algorithm, which applies a modified Joint-Bilateral Upsampling (JBU) filter to low-resolution depth data and high-resolution color data [9]. This method shows high accuracy relative to other more computationally intensive methods. Further, the algorithm itself is tightly bounded in execution time with respect to input, which is

favorable compared to energy-minimization techniques like the ones used in [10], [13], and [14]. Though the execution time cited in the original paper of 49 ms falls outside of the 11 ms constraint of real-time apps, due to the desirable characteristics of the algorithm, NAFDU was chosen to be accelerated on hardware for this study.

3.0 Background

In this section, background information regarding the NAFDU will be outlined. Subsequently, the tools, hardware, and acceleration frameworks that were used in this study will be summarized.

3.1 Noise-Aware Filter for Depth Upsampling (NAFDU)

As previously stated, NAFDU was selected for hardware acceleration. This method is based on a traditional bilateral filter, which is expressed in Equation 3-1.

$$P = \frac{1}{k_p}\sum_{q\in\,\Omega} I_q\, f(\|p-q\|)\, g\big(\|I_p - I_q\|\big) \qquad 3\text{-}1$$

Bilateral Filter Kernel

In Equation 3-1, q and p are pixel coordinates in the original image. I_q is the pixel value in the at pixel p, Ω is the neighborhood centered at p. Usually, both f and g are gaussian functions; f is referred to as the domain term, and g is referred to as the range term. The k_p term is the normalization term and is equal to the sum of the domain and range terms over the current neighborhood.

NAFDU is based on the JBU filter proposed by Kopf [15], in which the bilateral filter is used for upsampling by taking the range term from a separate, high-resolution image. The

assumption made by the JBU is that the high-resolution image will have similar, more detailed structural information about the scene, which can be referenced for the low-resolution image. Chan illustrates that the method of using a second image to calculate the range term of the bilateral filter can lead to the phenomenon known as "texture-copying" [9]. Texture-copying in the JBU filter is due to the erroneous assumption that the color image will exhibit similar structural characteristics in its gradient as the depth data (i.e. observing a color-varying surface that may, in fact, be physically flat) [9]. NAFDU addresses this assumption by introducing a sigmoid blending function, α, which is used to make the range term dependent on both the color image and the depth image. The assumption is that if a given area has a small depth gradient, it is likely flat, and therefore the range term from the depth data should be used for upsampling, even if a large color gradient is observed. Similarly, if a high color gradient is observed in conjunction with a high depth gradient, the range term from the high-resolution data is preferred because it is assumed to be more accurate. This blending function has the effect of preserving edges during upsampling, while also avoiding the texture-copy phenomenon in flat areas that may have high color variations. The adjusted range term for the NAFDU upsampling filter is expressed in Equation 3-2.

$$r_p = \alpha(\Delta_\Omega)\, g\left(\left\|\tilde{I}_p - \tilde{I}_q\right\|\right) + \left(1-\alpha(\Delta_\Omega)\right) h\left(\left\|I_{p\downarrow} - I_{q\downarrow}\right\|\right) \qquad 3\text{-}2$$

NAFDU Kernel

In Equation 3-2, $\tilde{I}_p$ and $\tilde{I}_q$ are the pixel values in the color image in the neighborhood, Ω, centered at p; Δ_Ω is the absolute difference between the maximum- and minimum-valued pixels in

the current neighborhood; h and g are gaussian functions; $\alpha(\Delta_\Omega)$ is the sigmoid blending function. A more thorough explanation of the NAFDU algorithm is presented in [9].

3.2 Time-of-Flight (ToF) Sensors

ToF sensors are growing in popularity and are shrinking in cost. This newfound demand is driven by their ability to produce relatively high-resolution depth maps at high frame rates compared with other depth sensor types such as structured light sensors. ToF sensors have two main components: a light source, usually a diffused near-infrared laser, and an image sensor. At each frame, the laser sends out a series of modulated pulses. The number of pulses and the wavelength of the pulse is dependent on the sensor. When the photons from the pulse are reflected back at the sensor, the time that it took for each photon to leave the light source and arrive back at the sensor is measured. The time-of-flight for each photon is proportional to the distance from the sensor and used to infer the depth at that point in space.

Another popular type of depth sensor is called a structured-light sensor. These sensors also use near-infrared emitters, but instead of modulated pulses, a pattern is projected onto the scene at each frame. Based on the distortion of this pattern, the scene structure can be inferred using simple computer-vision techniques. These sensors can also produce high-quality, high-resolution depth maps. The disadvantage compared to ToF sensors is the real-time image processing that necessitates extra device logic to process the collected pattern to infer scene depth. ToF sensors do not require complex logic, and therefore they can scale to higher resolutions more easily.

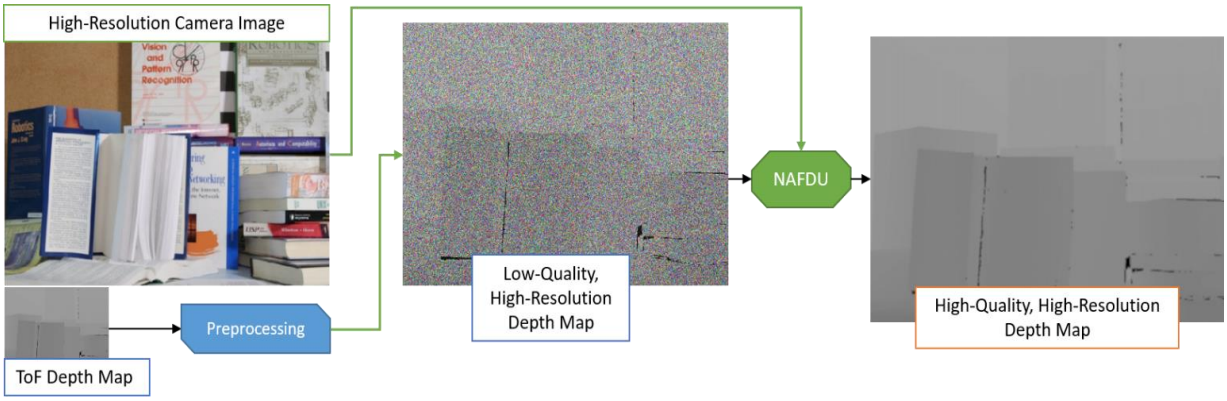

Figure 1. NAFDU System Diagram, "Books" from Middlebury Dataset Used [16]

3.3 Hardware Testbed

The hardware testbed used in this case study is a 2048×1536 resolution color camera coupled to a 320×240 resolution ToF sensor. A diagram of the processing flow of this testbed is shown in Figure 1. A frame is simultaneously collected from the color camera and the depth sensor. The depth map is then upsampled to the target resolution using the nearest-neighbor method, which uses a negligible amount of processing power, but produces a low-quality upsampled depth map. This low-quality depth map is then streamed to the NAFDU alongside the color image. It is notable that the pixel correspondence between these two images must be calibrated beforehand since the NAFDU algorithm relies on similarity between two images at the same pixel coordinate. The output of the NAFDU block is the high-quality, high-resolution depth map which can be used for later rendering in an augmented reality app.

The target maximum latency of the accelerated algorithm is 11 ms, matching the frame time of popular mixed-reality headsets [3] [4]. We acknowledge that the latency must be lower in

practice to account for other processing in the rendering pipeline; however, 11 ms was judged to be a reasonable initial target for a proof-of-concept experiment.

4.0 Approach

In this section, the approach in each stage of the research is outlined. The first stage includes the steps taken to optimize and parallelize NAFDU on a desktop PC. These initial steps were taken to assess the scalability of the algorithm. In the final stage of this research, NAFDU was accelerated using Vivado HLS and run on the Xilinx ZCU102 MPSoC.

4.1 NAFDU on High-Performance Desktop PC

Figure 2. "Art" from Middlebury Dataset [16]

The following steps were used to validate the algorithm of the original CPU codebase: a single depth-map and color-image were read as input, the depth-map resolution was reduced using the resize function in OpenCV, and then the NAFDU algorithm was used to restore the depth map to its original size. The new image was then compared to the original using root-mean-square error (RMSE) as a metric. Multiple kernel window sizes were tested to assess their effect on RMSE, which are shown later in Section 5.1. The Middlebury image dataset was used throughout this

study for evaluating the quality of upsampled depth maps [16] [17]. Example images from this dataset can be seen in Figure 1 and Figure 2.

4.2 NAFDU Optimization and Parallelization

After evaluating the serial baseline, a series of optimizations were applied. The goal of these optimizations was to ensure that the desktop version of NAFDU could be compared fairly to the FPGA implementation in terms of speed and accuracy. The Open Specification for Multiprocessing (OpenMP) framework offers a set of compiler directives integrated into the GNU C Compiler (GCC) to efficiently parallelize code written in C/C++. OpenMP was used in this study to parallelize the NAFDU algorithm on CPU in order to evaluate it for scalability.

NAFDU operates in a manner typical of most convolution-style computer-vision algorithms: two outer loops iterate through all pixels (line-by-line; pixel-by-pixel) and two inner loops iterate through the spatial neighborhood around the pixel specified by the outer loops. OpenMP directives were applied to the outer loop to reduce the number of total thread forks and thus minimize parallel overhead. Three thread-scheduling schemes were tested: static, dynamic, and guided. Static scheduling does not provide any load-balancing, and simply partitions the problem into fixed block sizes at compile time. Dynamic scheduling provides load balancing of threads at runtime but does so at the cost of performance. Guided scheduling is akin to dynamic scheduling, but it adjusts the concurrent block sizes from large chunks to small as the program executes. Guided scheduling was proven to yield the best performance of the three for all cases, with static scheduling a close second. It is well known that dynamic scheduling tends to yield the largest parallel overhead due to the additional requirement of runtime load balancing [18]. The

code was benchmarked for runtime and parallel efficiency for one through eight threads on a desktop PC with an Intel Core i7 4790k @ 4.6 GHz and 16 GB of RAM. Runtime was calculated with microsecond precision by calling `omp_get_wtime()` immediately before and after the NAFDU function call, and subtracting the two values. Reported values are averages of 100 runs.

After assessing the parallelizability of NAFDU, the initial codebase was optimized by translating floating-point arithmetic to fixed-point arithmetic, which led to a dramatic speedup. Due to the insignificant difference in speed but notable increase in accuracy, 64-bit integers were used in place of 32-bit integers, using 23 bits of fractional precision. This design optimization removed all floating-point instructions from the codebase and resulted in less than a 1% decrease in accuracy but provided a 1.5 to 4.0 times speedup for all tested kernel widths.

4.3 Design for High-Level Synthesis

HLS was selected over a hardware description language in this research for two main reasons. First, the increased productivity afforded by HLS allowed for rapid experimentation with multiple architectures. Second, given that HLS is a relatively new and constantly evolving technology, there is desire in the academic community to see HLS being used in different application scenarios, in this case, a real-time XR app. Vivado HLS was chosen due to the authors' familiarity with Xilinx tools.

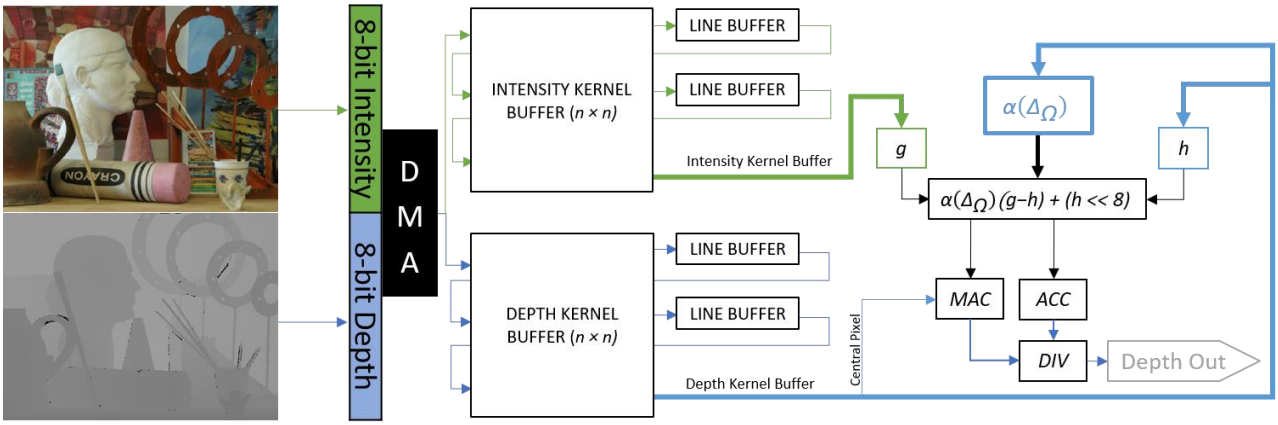

Figure 3. FPGA NAFDU Accelerator Diagram for Kernel Width *n*

A detailed diagram of the accelerator developed in the final design is shown in Figure 3. This streaming architecture benefits scalability when applied to high-resolution images, such as those in our testbed (2048×1536). It has been shown that frame-based accelerators, in which the entire video frame must be kept in memory for computation, are limited by the amount of BRAM (block RAM) on the device [19], which restricts the image resolutions that can be supported. Frame-based designs that do not rely on BRAM are forced to fetch data from off-chip memory, which can add significant latency and jitter to the design. The stream-based architecture used in this study does not have this limitation because the resource utilization increases only with kernel-size, and by extension adder-tree depth, not image resolution, which only increases the length of the line buffers.

In this design, video data is streamed in as a 16-bit stream. The upper byte is the high-resolution guidance image pixel and the lower byte is the depth pixel. It is notable that the color image must be converted to grayscale before it is streamed to this accelerator. After the kernel buffers are filled, the domain and range weights are calculated for each pixel in the depth kernel. All terms are calculated in parallel and the weights are accumulated. This accumulation causes a large adder tree to be produced when using large kernel sizes. After all terms are accumulated, the

output is divided by the sum of the weights to maintain the intensity of the original depth image. This design results in minimal latency and consistent performance at high clock frequencies, but it does so at the cost of resource usage, though with small kernels (3 px, 5 px, 7 px) resource usage on our test platform is minimal.

Both execution time and power efficiency were gathered for the FPGA design. Execution time was calculated by recording the time between when the first pixel is sent to the accelerator and when the last pixel is processed. Memory transfer times are ignored in our study. This is justified because this accelerator would be used to stream pixels directly from a camera and depth sensor and would not need to fetch data from DDR in an actual system. System power was recorded with a WhatsApp™ power meter attached to the ZCU102. Average readings were recorded with and without the accelerator present in the design. Reported FPS-per-watt are based on the maximum power that the accelerator adds to the system, which accounts for both dynamic and static power of the accelerator. We reason that since the development board contains many unnecessary processing elements and peripherals, overall system power was not relevant and would artificially decrease the reported efficiency of the accelerator.

5.0 Experimental Results

In this section, detailed results will be presented from both the CPU and FPGA stages of the case study. The CPU results include error reduction and execution times for selected kernel widths, and the FPGA results include execution times and resource utilization on the Xilinx ZCU102 board.

5.1 NAFDU on Desktop PC

In Figure 4, we show the expected error reductions when NAFDU is used to correct a depth map which has been injected with noise. As expected, as the spatial kernels increase in size, the error decreases in the resulting image. Notably, the final RMSE stabilizes at ~12% for floating point and ~14% for fixed point regardless of increasing the kernel radius. The authors therefore

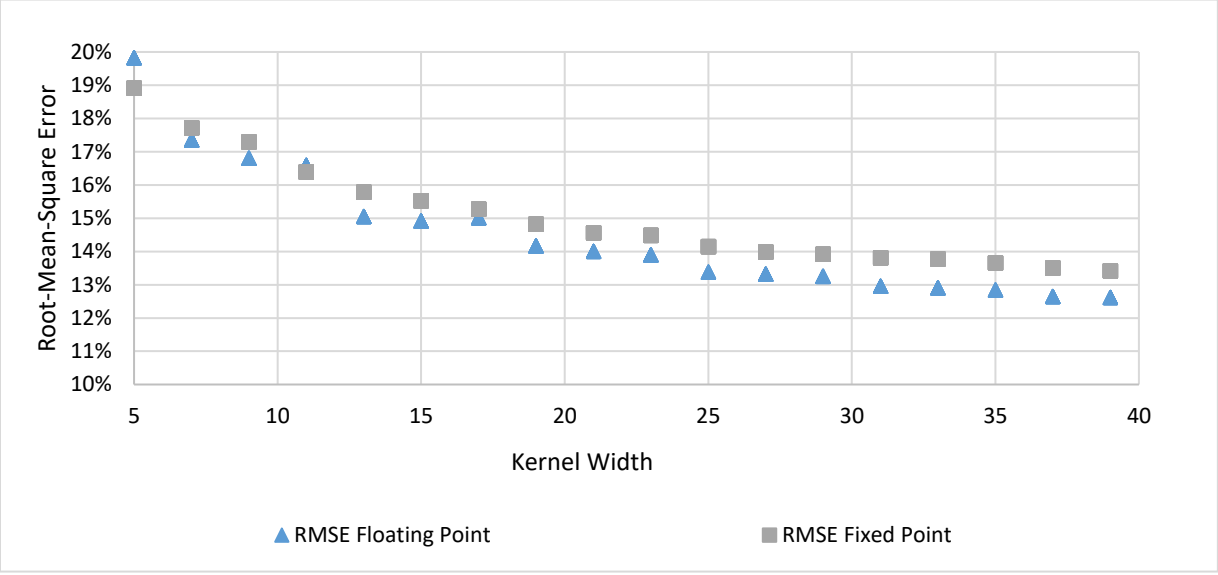

Figure 4. RMSE vs. Kernel Width

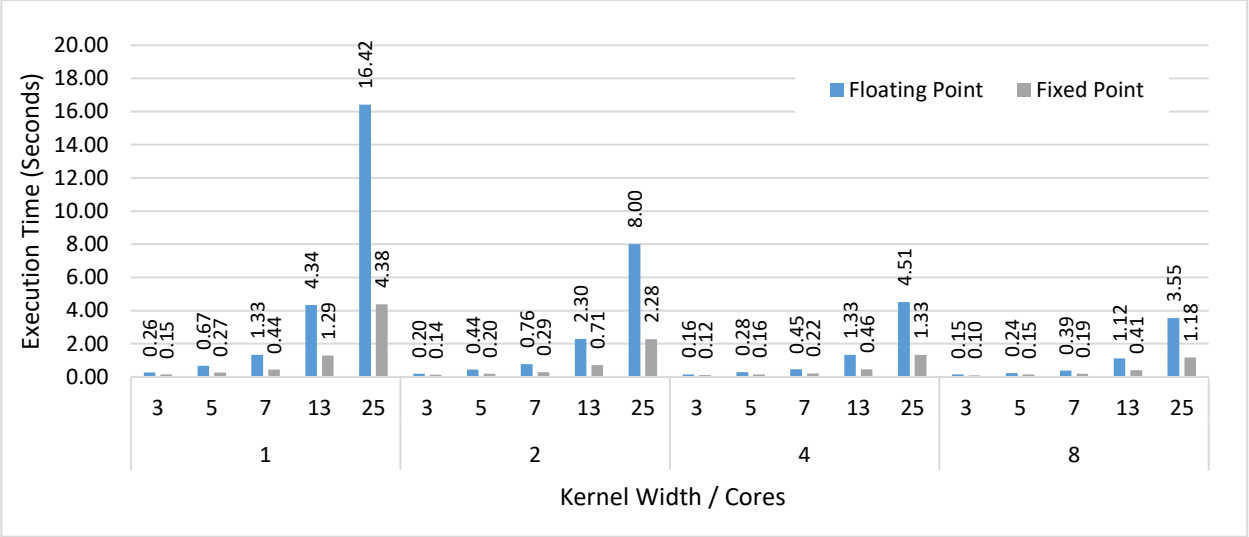

Figure 5. NAFDU CPU Execution Times

judged a 25 px kernel width to be a reasonable maximum target for our hardware designs, as it is the smallest kernel which achieves this accuracy. However, due to long compilation times, the 13 px kernel was the maximum kernel size that was able to be implemented in hardware for this study. Simulation results indicate that performance of the 25 px kernel would be similar to others tested.

Execution times for the parallelized version of NAFDU on a CPU are shown in Figure 5. Results for floating-point and fixed-point versions of NAFDU are included in the graph. The graph demonstrates that for the floating-point version, we achieved a speedup of 4.62 when using eight threads in the 25 px kernel case. The fixed-point version for the same kernel, conversely, only achieved a speedup of 3.66 when using eight threads over the serial baseline. We reason that the lower speedup is due to resource contention of the integer math units when the number of threads, eight, is larger than the number of physical cores, in this case four.

5.2 NAFDU on FPGA

The results shown for the accelerated NAFDU were gathered using Vivado HLS version 2018.2. Results include pixel latency and theoretical framerates as well as resource utilization for 2048×1536 images. The development board used was the Xilinx ZCU102 for selected kernel widths up to 13 px.

5.2.1 Performance Results

The target frame time for real-time, high-resolution image processing was previously defined as 11 ms. In our tests, for all kernel sizes, we report that the performance on the Zynq UltraScale+ MPSoC is comparable to expected results from simulations. The clock rate on hardware was 300 MHz. The total throughput of the test system was limited by the host software, which was a simple direct memory access (DMA) system running on top of Petalinux. This system passes data to the accelerator from an image loaded by software into DDR. The performance results are a lower bound on what could be expected with a production-level system, which could interface directly with camera data as a stream instead of passing data to the accelerator from DDR. Even considering this lower-bound performance, our design achieves a frame time of 10.5 ms for all kernels, which meets the target of 11 ms. This frame time also represents a speedup of 40 compared to the fastest CPU version for the 13 px kernel and a factor of 4.7 times improvement over the original GPU implementation. All kernels tested had identical frame times, showcasing the scalability of our streaming architecture with respect to image resolution. However, increasing the kernel size of the design significantly impacts resource utilization, and is therefore the limiting factor in this design.

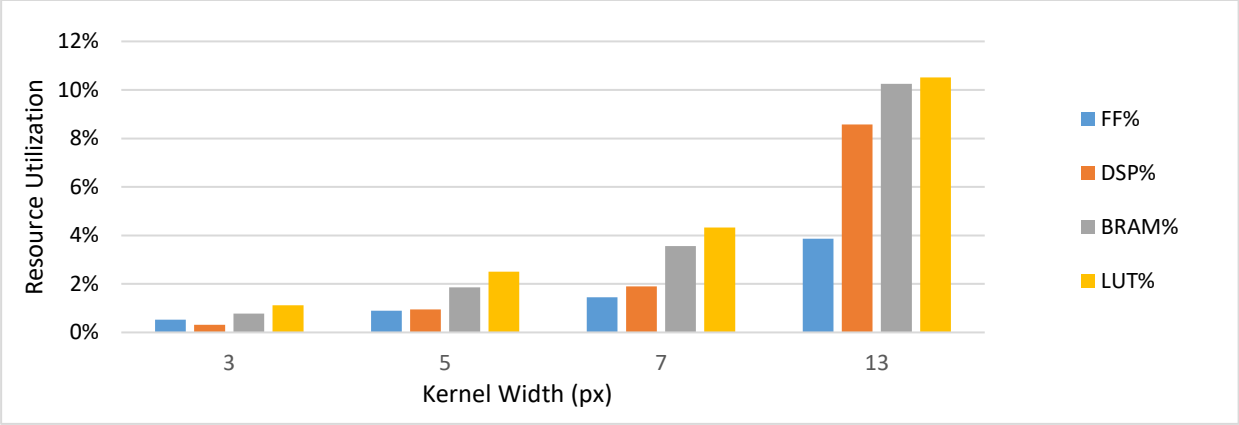

Figure 6. Zynq UltraScale+ MPSoC Resource Utilization

5.2.2 Resource Utilization

The final resource utilization percentages for all kernel sizes tested can be found in Figure 6. The kernel size, and by extension the line buffers and adder trees, significantly impacts the amount of resources used in the design. Adder trees are automatically generated by the HLS synthesizer to handle the two-dimensional multiply-accumulate reduction over the entire kernel. Upon deeper analysis, these adder trees use a significant number of LUTs to handle fully-pipelined integer multiply-accumulations. This could be reduced by increasing the initialization interval of the NAFDU pipeline from 1 clock cycle, which would slow down the design but require less resources to implement the adder trees.

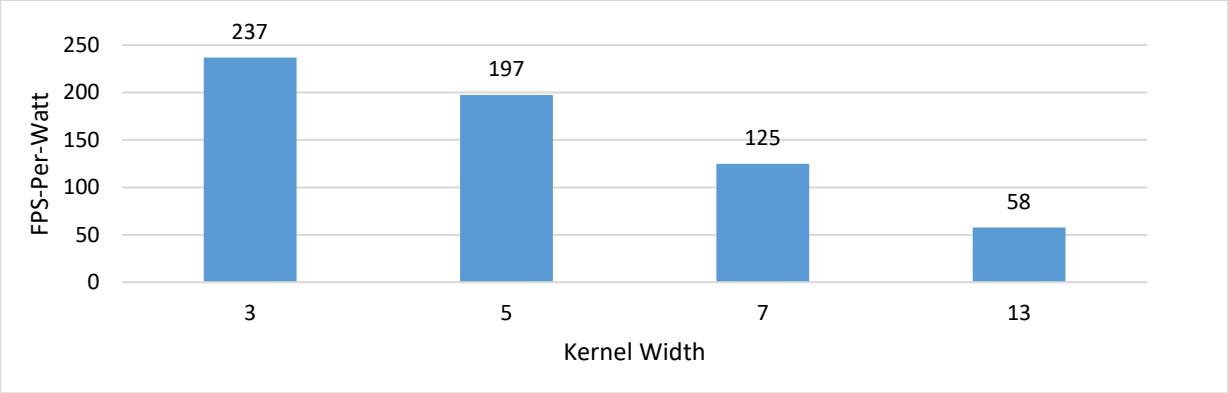

Figure 7. Zynq UltraScale+ MPSoC Performance-per Watt

The performance per watt for each kernel width appears to scale quadratically with kernel size, as shown in Figure 7. We see that the smallest kernel width yields the highest performance per watt because all kernel sizes maintained the same framerate while the resource utilization increased. FPS-per-watt results for designs between 3 px and 7 px kernel widths also suggest that this device can achieve real-time performance at sub-watt power, a common target for embedded-system designs.

6.0 Discussion

As shown in the results above, the NAFDU algorithm performs well on FPGAs, and outperforms both the original GPU implementation and the parallelized CPU version. Additionally, the target framerate of 90 FPS and target latency of 11 ms is met. Further, the authors show that the FPGA used also has a comparatively low power requirement, making it feasible to install directly on an HMD, which would not be the case for the high-performance CPU or GPU. While the resource utilization scales poorly with kernel size, as shown in Figure 6, kernel sizes generally are not larger than 5 px wide. In fact, the original implementation on a GPU only used a 3 px kernel [9]. Granted, this was on a 640 × 480 image, but since higher resolution images are targeted in this study, it was concluded that a 3 px kernel is not large enough to reduce noise in the output depth map. The area of the kernel covered a much smaller area of the total image and was more sensitive to local noise as a result.

Filter based approaches like the NAFDU are not generally considered state-of-the art in terms of accuracy. It could be argued that filtering approaches will never be as accurate as energy minimization (EM) approaches like Markov Random Fields, however, there are many reasons why these algorithms are infeasible for mixed reality designs. HMDs are increasing in screen resolution, so algorithms used in depth-mapping pipelines must be highly scalable with respect to input and output resolution. The authors therefore deem filter-based approaches to depth-upsampling to generally be more viable for real-time applications than EM approaches. EM algorithms generally scale poorly with large output resolutions, because adding pixels to the output depth map serves to increase the output solution space, which EM approaches attempt to fit to some probabilistic model. Additionally, from a hardware acceleration perspective, EM approaches introduce huge

amounts of data dependencies, since each iteration of the minimization depends on the result of the last. Thus, the inherent serialization in these algorithms makes massive parallelism difficult to achieve, which is necessary in the case of GPU implementations. EM algorithms also are generally not bound in execution time, and some inputs may process much slower than others. This nondeterministic execution pattern makes EM algorithms difficult to map to a dataflow architecture, such as an FPGA, where design blocks with fixed execution times are necessary. Fixed execution time is necessary in FPGA designs because pipelining a dataflow architecture with variable execution time is nearly impossible, since a physical bound must be set on how deep the pipeline must be. The only solution for these real-time, iterative algorithms is to short circuit the processing pipeline after some number of iterations, and either produce an incomplete output, or force execution of the entire pipeline to stall while the output is computed. The depth and frequency at which these execution stalls occur is application dependent and would likely result in an extremely inefficient design.

7.0 Conclusions

In summary, the goal of this research was to demonstrate that filter-based depth-upsampling algorithms, as opposed to energy minimization algorithms, are suitable for hardware acceleration to meet the high-resolution, real-time constraints imposed by mixed-reality apps. The NAFDU algorithm was selected for this purpose due to its relatively low algorithmic complexity, deterministic execution time, and high upsampling accuracy compared to more complex methods. We parallelized NAFDU on a desktop PC to demonstrate its scalability and assess the error introduced by quantizing floating-point arithmetic to fixed-point. Subsequently, we designed and developed a NAFDU accelerator using Vivado HLS which showed significant performance gains when executed on the Zynq UltraScale+ MPSoC platform. The final design achieved a 10.5 ms frame time for the largest kernel width tested, 13 px, and therefore meets the real-time requirement for mixed-reality apps of 11 ms. This execution time achieves a 40× speedup over the fastest CPU version and a 4.7× improvement over the original GPU implementation of NAFDU. Kernel widths smaller than 13 px also achieve high performance-per-watt, demonstrating that low-power, embedded systems could leverage the NAFDU accelerator and still maintain high framerates. The accelerated version of NAFDU meets the real-time, high-resolution constraints imposed by mixed-reality apps and is a demonstration of how filter-based depth-upsampling algorithms can be applied in XR apps to enhance overall depth-map quality.

Future work on this topic involves further benchmarking and parallelization of the NAFDU on new embedded GPUs, which could be used in an HMD. Specifically, the NVIDIA Tegra AGX will be used to compare the performance-per-watt of new embedded GPUs to FPGAS. Given the difficulty of programming and optimizing designs for FPGAs, a useful study would be to

investigate if an embedded GPU can closely match the processing power and energy-efficiency of the FPGA design. The results from this study would indicate whether the GPU is a more attractive option for XR real-time processing on HMDs, given the relative ease of programming as well as cost.

Bibliography

[1] D. Langerman, S. Sabogal, B. Ramesh and A. George, "Accelerating Real-Time, High-Resolution Depth Upsampling on FPGAs," in *IEEE International Conference on Image Processing Applications and Systems*, Nice, France, 2018.

[2] "Augmented Reality Market Size By Component," Global Market Insights, December 2017. [Online]. Available: https://www.gminsights.com/industry-analysis/augmented-reality-ar-market. [Accessed 17 August 2018].

[3] A. Binstock, "Powering the Rift," Oculus.com, [Online]. Available: https://www.oculus.com/blog/powering-the-rift/.

[4] HTC, "VIVE Specs," HTC, 2018. [Online]. Available: https://www.vive.com/us/product/vive-virtual-reality-system/. [Accessed 2018].

[5] D. J. Zielinski, H. M. Rao, M. A. Sommer and R. Kopper, "Exploring the effects of image persistence in low frame rate virtual environments," in *IEEE Virtual Reality*, Arles, France, 2015.

[6] I. Eichhardt, D. Chetverikov and Z. Jank´o, "Image-guided ToF depth upsampling: a survey," *Machine Vision and Applications,* Vols. 3-4, no. 28, pp. 267-282, 2017.

[7] V. Gandhi, J. Cech and R. Horaud, "High-resolution depth maps based on TOF-stereo fusion," in *Robotics and Automation IEEE International Conference*, 2012.

[8] A. Belhedi, A. Bartoli, S. Bourgeois, V. Gay-Bellile, K. Hamrouni and P. Sayd, "Noise modelling in time-of-flight sensors with application to depth noise removal and uncertainty estimation in three-dimensional measurement," *IET Computer Vision,* vol. 9, no. 6, pp. 967-977, 2015.

[9] D. Chan, H. Buisman, C. Theobalt and S. Thrun, "A noise-aware filter for realtime depth upsampling," in *Workshop on Multi-camera and Multi-modal Sensor Fusion Algorithms and Applications*, 2008.

[10] G. D. Evangelidis, M. Hansard and R. Horaud, "Fusion of range and stereo data for high-resolution scene-modeling," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 37, no. 11, pp. 2178-2192, 2015.

[11] M.-Z. Yuan, L. Gao, H. Fu and S. Xia, "Temporal Upsampling of Depth Maps Using a Hybrid Camera," *IEEE Transactions on Visualization and Computer Graphics,* 2018.

[12] R. Shea, A. Sun, S. Fu and J. Liu, "Towards Fully Offloaded Cloud-based AR: Design, Implementation and Experience," in *Proceedings of the 8th ACM on Multimedia Systems Conference*, Taipei, Taiwan, 2017.

[13] C. Pal and D. Scharstein, "Learning conditional random fields for stereo," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Minneapolis, MN, 2007.

[14] L. Yuan, X. Jin, Y. Li and C. Yuan, "Depth map super-resolution via low-resolution depth guided joint trilateral up-sampling," *Journal of Visual Communication and Image Representation,* no. 46, pp. 280-291, 2017.

[15] J. Kopf, M. F. Cohen, D. Lischinski and M. Uyttendaele, "Joint bilateral upsampling," *ACM Transactions on Graphics,* vol. 3, no. 26, p. Article 96, 2007.

[16] S. D and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence," *International Journal of Computer Vision,* vol. 1/2/3, no. 47, pp. 7-42, 2002.

[17] R. S. D. Scharstein, "High-accuracy stereo depth maps using structured light," in *IEE Computer Society Conferences on Computer Vision and Pattern Recognition*, Madison, WI, 2003.

[18] J. M. Bull, "Measuring Synchronisation and Scheduling Overheads in OpenMP," *Proceedings of First European Workshop on OpenMP,* vol. 8, p. 49, 1999.

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