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

David Langerman, Sebastian Sabogal†, Dr. Barath Ramesh‡ and Dr. Alan George§
Department of Electrical and Computer Engineering, University of Pittsburgh
NSF Center for Space, High-performance, and Resilient Computing (SHREC)
Pittsburgh, PA
Email: *dal181@pitt.edu , †ses220@pitt.edu , ‡barath.ramesh@pitt.edu , §alan.george@pitt.edu

***Abstract*—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 depth sensor resolution and quality. These advances in time-of-flight sensors provide a platform for new research into real-time, depth-upsampling algorithms targeted at high-resolution video systems with low-latency requirements. This paper describes a case study in which a previously developed bilateral-filter-style upsampling algorithm is profiled, parallelized, and accelerated on an FPGA using high-level synthesis tools from Xilinx. We show that our accelerated algorithm can effectively upsample the resolution and reduce the noise of time-of-flight sensors. We also demonstrate that this algorithm exceeds the real-time requirements of 90 frames per second necessitated by mixed-reality hardware, achieving a lower-bound speedup of 40 times over the fastest CPU-only version.**

Keywords—time-of-flight sensor, depth upsampling, FPGA acceleration, high level synthesis, image processing

I. Introduction

Mixed-reality technology has grown popular in recent years and is expected to become a 1.65-billion-dollar industry by 2024 [1]. The advent of this technology introduces many challenges from both the hardware and software perspectives. Advanced head-mounted displays have native resolutions greater than 2160×1200 (2.5 million pixels) displaying at refresh rates of 90 Hz, which translates to a frame time of roughly 11 ms per image [2-3]. 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 mixed-reality 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 [4]. 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 fail to meet the frame time demands of mixed-reality systems [5].

National Science Foundation Grant No. CNS-1738783

In this paper, a case study involving a mixed-reality headset assembly is detailed. This headset consists of a high-resolution color camera as well as a low-resolution, time-of-flight (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 a display. This case is a typical scenario for augmented reality applications 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 [4]. 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 [5]. To mitigate these inaccuracies, methods have been developed to fuse data from both the ToF sensors and the high-resolution cameras [4]; however, many of these methods can be too computationally intensive to meet the real-time constraints of modern apps on conventional hardware. 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 applications. Specifically, this paper describes a case study in which the Noise-Aware Filter for Real-Time Depth Upsampling (NAFDU) [6] is tested on a CPU platform, then accelerated using Vivado HLS and evaluated on the Xilinx Zynq UltraScale+ MPSoC device on the ZCU102 board.

II. Related Work

In [5] and [7], 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. Additionally, the algorithm's iterative behavior imposes severe restrictions for real-time apps.

Yuan, M. details a framework for temporal upsampling using a hybrid camera [8]. 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 [9], 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 [6], 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 [6]. 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 inconsistent with energy-minimization techniques like the ones used in [7], [10], and [11]. 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.

III. Background

A. Noise-Aware Filter for Depth Upsampling (NAFDU)

As previously stated, NAFDU was chosen for hardware acceleration. This method is based on a traditional bilateral filter, which is expressed in (1).

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

In (1), I_q is the pixel value in the original image at p, Ω is the neighborhood centered at pixel p. Generally, both f and g are gaussian functions; f is referred to as the domain term, and g is referred to as the range term. k_p 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 [12], 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 aforementioned method of using a second image to calculate the range term of the bilateral filter can lead to the phenomenon known as "texture-copying" [6]. 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) [6]. 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 made 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 (2).

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

In (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 [6].

B. Time-of-Flight (ToF) Sensor

ToF sensors are growing in popularity and are shrinking in cost. This newfound demand is derived from the fact that they produce high-resolution depth maps at high frame rates. These sensors have two main components: a light source, usually a diffused near-infrared laser, and an image sensor. The time that it takes for each photon to leave the light source and arrive at the sensor is measured and used to infer the depth at that point in space.

C. High-Level Synthesis (HLS)

As HLS tools continue to mature, complex hardware designs can be synthesized using few lines of code and run at speeds comparable to those composed from low-level hardware description languages such as VHDL or Verilog. Vivado HLS was selected for this study. This tool facilitates the development and simulation of hardware using high-level C and C++ programming languages. Synthesizable functions that describe hardware can be translated to register-transfer language (RTL) for synthesis. Vivado HLS also includes a co-simulator to simulate the translated functions using a testbench written in C or C++. The ability to use higher-level languages can significantly reduce the overall development cycle for hardware. The tradeoff for this increase in productivity is less flexibility

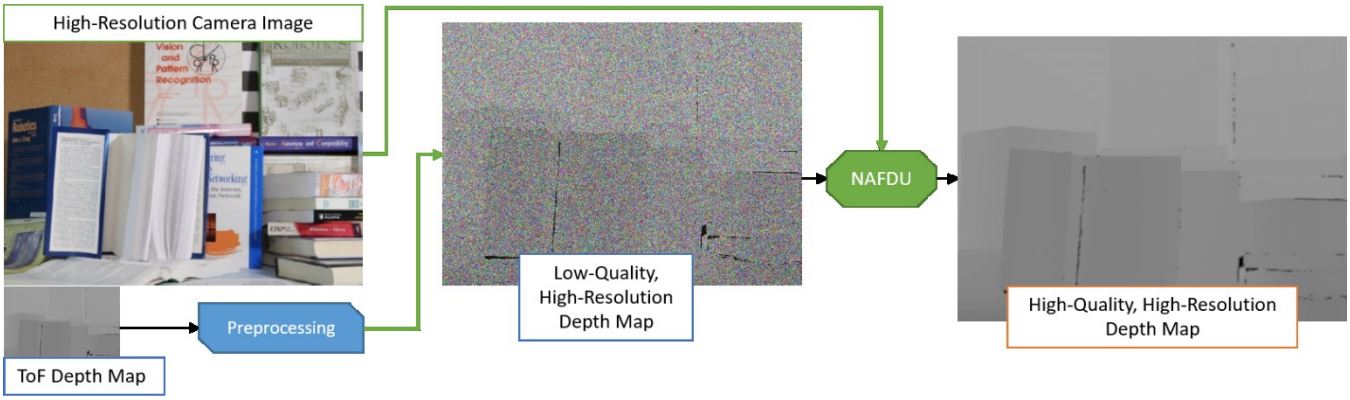

Fig. 1. NAFDU System Diagram, "Books" from Middlebury Dataset Used as an Example [10]

and control of the design in synthesis and implementation on hardware.

D. Hardware Testbed

The hardware testbed used in this case study is a 2048×1536 resolution color camera coupled to a 320×240 resolution ToF. A diagram of the processing flow of this testbed is shown in Fig. 1. The target maximum latency of the accelerated algorithm is 11 ms, matching the frame display time of popular mixed-reality headsets [2-3]. 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.

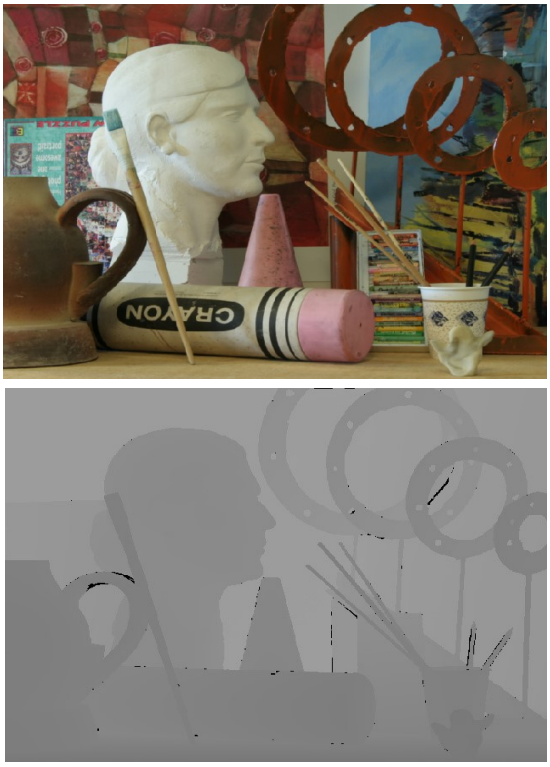

Fig. 2 "Art" from Middlebury Dataset [13]

IV. 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 board.

A. NAFDU on Desktop PC

The following steps were used to validate the algorithm of the original CPU codebase: a single depth map and color image was 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 V.A. The Middlebury image dataset was used throughout this study for evaluating the quality of upsampled depth maps [13-14]. Example images from this dataset can be seen in Fig. 1 and Fig. 2.

B. NAFDU Optimization and Parallelization

After evaluating the serial baseline, a series of optimizations were applied. The goal of these optimizations was twofold: first, to ensure that the desktop version of NAFDU could be compared fairly to the FPGA implementation in terms of speed and accuracy; second, to accelerate the algorithm to stream live video for demonstration purposes. OpenMP was used to parallelize the codebase to evaluate NAFDU for parallelizability and scalability.

NAFDU follows a structure typical of most convolution-style computer-vision algorithms, where two outer loops iterate through all pixels and two inner loops iterate through the spatial neighborhood around the pixel specified by the outer loop. OpenMP pragmas were applied to the outer loop to optimize the number of threads created and thus minimize parallel overhead. Three thread-scheduling schemes were tested: static, dynamic, and guided. Guided scheduling proved to yield the

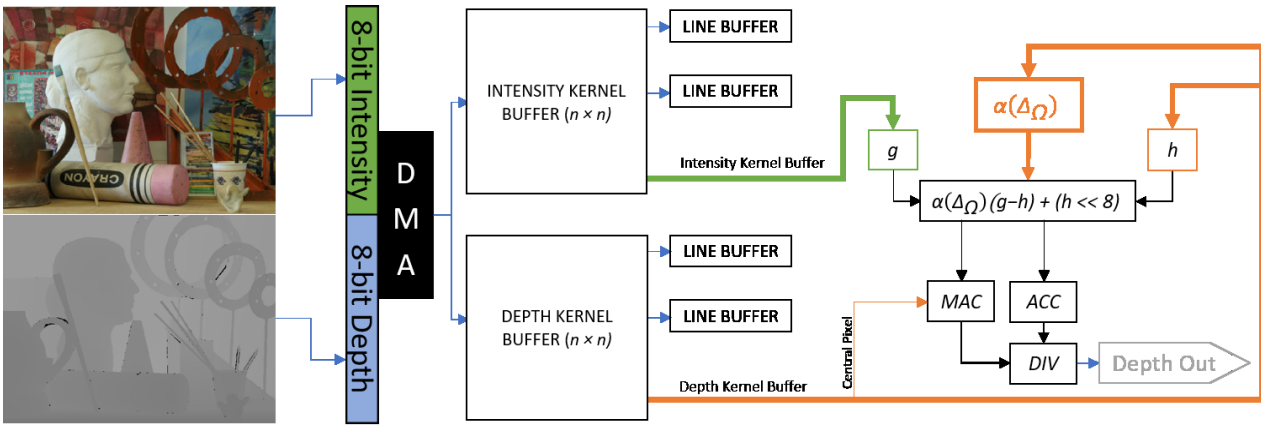

Fig. 3. FPGA NAFDU Accelerator Diagram for Kernel Width *n*

best performance of the three for all cases, with static scheduling as 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 [15]. 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.

After assessing the parallelizability of NAFDU, the initial codebase was transformed from floating-point arithmetic to fixed-point arithmetic, which led to a dramatic speedup which is outlined in Section V.A. 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 times speedup for all tested kernel widths.

C. 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, the authors desired to provide another reference to the academic community of HLS being used to accelerate a real-time application. Vivado HLS was chosen due to the authors' familiarity with Xilinx tools.

A detailed diagram of the accelerator developed in the final design is shown in Fig. 3. This architecture is beneficial for scalability when applied 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 [16], which restricts the image resolutions that they can support. Our FIFO architecture does not share this limitation, because the resource usage scales with kernel size, not image resolution.

V. 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 MPSoC board.

A. NAFDU on Desktop PC

In Fig. 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 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, in practice, due

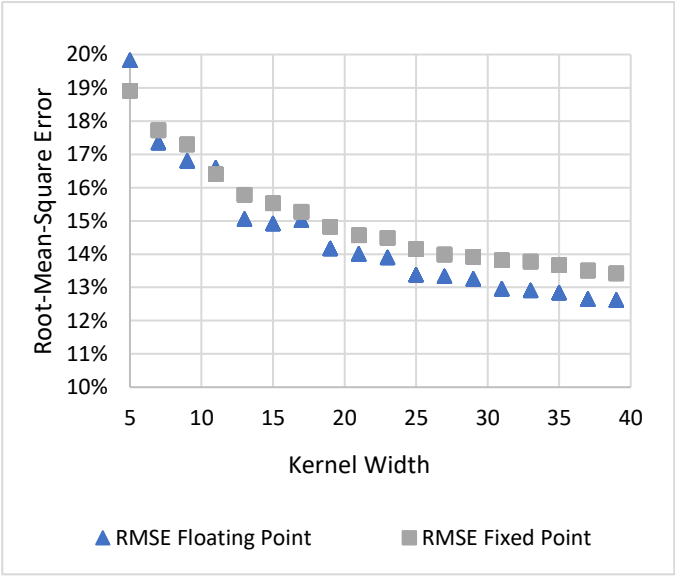

Fig. 4. RMSE vs Kernel Width

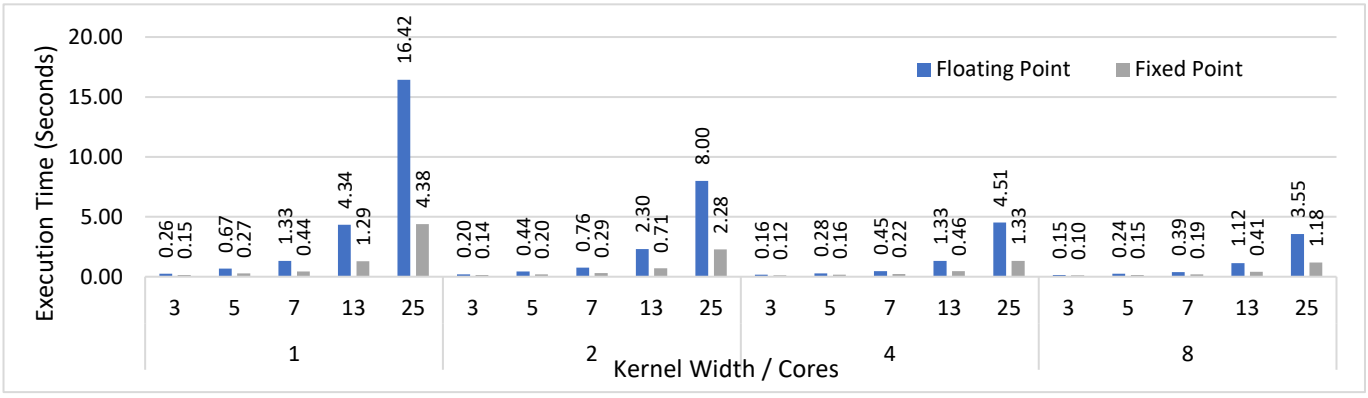

Fig. 5. NAFDU CPU Execution Times

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 Fig. 5. Both floating- and fixed-point results are included in the graph. The graph demonstrates that for the floating-point version, we achieved a 4.62× speedup with 8 cores in the 25 px-kernel width case. The fixed-point version for the same kernel, conversely, only achieved a 3.66× speedup with 8 cores 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, 8, is larger than the number of physical cores, in this case 4.

B. 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 estimates for 2048×1536 images. The development board used was the Xilinx ZCU102 for selected kernel widths up to 13 px.

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 measured clock rate of 300 MHz was slower than the simulated clock period of 500 MHz. The total throughput of the test system was limited by a slower clock rate and the DMA-based system architecture, which passes data to the accelerator from an image loaded by software into DDR. These 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 frame time 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 4.7× improvement over the original GPU implementation. All kernels tested had identical frame times, showcasing the scalability of our streaming architecture.

2) Resource Usage

The final resource utilization percentages for all kernel sizes tested can be found in Fig. 6. The kernel size 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 automatically-generated adder trees use a significant number of LUTs to handle pipelined integer multiply-accumulations.

The performance per watt for each kernel width appears to scale quadratically with kernel size, as shown in Fig. 7. We see

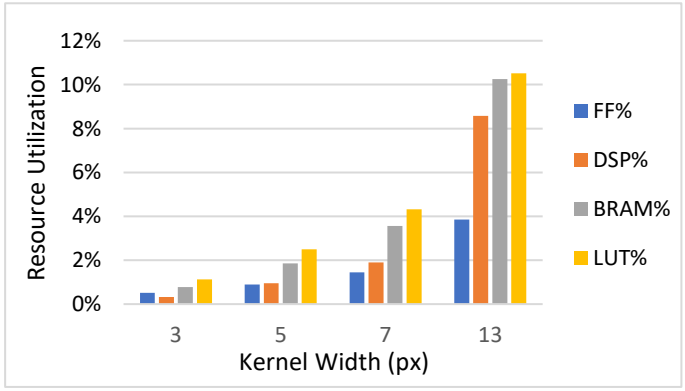

Fig. 6 Zynq UltraScale+ MPSoC Resource Utilization

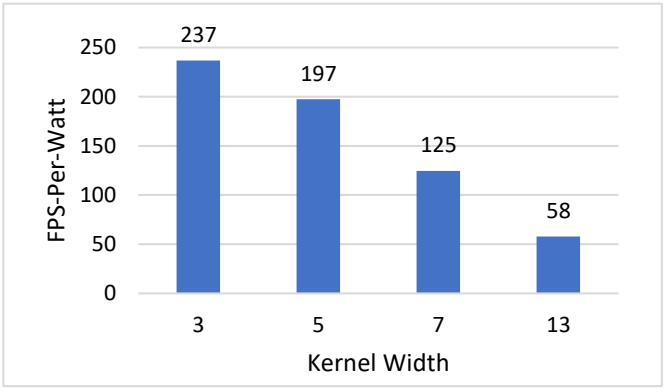

Fig. 7 Zynq UltraScale+ MPSoC Performance-per Watt

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 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.

VI. Conclusions

In summary, the goal of this research was to select a depth-upsampling algorithm suitable for hardware acceleration to meet the high-resolution, real-time constraints imposed by mixed-reality apps. NAFDU was chosen for this purpose due to its low algorithmic complexity, deterministic execution time, and high accuracy compared to more complex methods. We parallelized NAFDU on a desktop PC to demonstrate its scalability and asses the error discrepancy between floating- and fixed-point data representations. 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. This execution time represents a 40× speedup from 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 has met the real-time, high-resolution constraints imposed by mixed-reality apps and is a prime example of how HLS tools have matured in terms of producing hardware that is both scalable and capable of real-time performance.

Acknowledgment

This work was supported by the NSF SHREC Center industry and agency members and by the IUCRC Program of the National Science Foundation under Grant No. CNS-1738783.

VII. References

[1] "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].

[2] A. Binstock, "Powering the Rift," oculus.com, 15 May 2015. [Online]. Available: https://www.oculus.com/blog/powering-the-rift/. [Accessed 17 August 2018].

[3] HTC, "VIVE Specs," 2018. [Online]. Available: https://www.vive.com/us/product/vive-virtual-reality-system/. [Accessed 17 August 2018].

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

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

[14] D. Scharstein and R. Szeliski., "High-accuracy stereo depth maps using structured light," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Madison, WI, 2003.

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

[16] 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.