EURASIP Journal on Advances in Signal Processing
a SpringerOpen Journal

RESEARCH Open Access

Exploring super-resolution implementations across multiple platforms

Brian Leung and Seda Ogrenci Memik*

Abstract

The performance of many applications, such as video streaming, webcam conferencing, and aerial surveillance, all greatly depend on video quality. A major issue with higher quality video is that either more data bandwidth or storage resources must be dedicated for transferring or storing the video. However, if the low-resolution video is transferred or stored in order to conserve data bandwidth and storage space, super-resolution is a viable solution that can be applied afterwards on the receiving end to rectify the poor quality of the low-resolution video. Super-resolution is an imaging technique that leverages motion blur and multiple low-resolution frames to construct a high-resolution frame. In our paper, we implement and analyze a super-resolution algorithm across multiple platforms ranging from purely hardware to purely software and even a mix of both hardware and software. More specifically, we examine the performance for a field-programmable gate array (FPGA) implementation on two different FPGAs, a software/hardware solution on a FPGA with a soft core processor, a general purpose graphics processing unit (GPGPU) implementation, and a MATLAB implementation. Overall, we found that the GPGPU provides the best overall performance with up to 29 FPS with 35 iterations of the super-resolution algorithm. A high-performance FPGA can have comparable performance and rival the GPGPU in some cases. One of the interesting results was that the hardware/software FPGA combination performed worse than the pure software implementation.

Keywords: Super-resolution algorithm, Video processing, High-resolution frame, Streaming video, Field programmable gate arrays, Graphics processor

1. Introduction

In the recent years, there has been a considerable rise in the use of internet video streaming applications, such as Hulu, YouTube, and Netflix. A study has shown that these sites accounted for 42.7% of the United States' internet traffic in 2010 [1]. In addition, findings have shown that during primetime (8 to 10 pm) television hours, Netflix customers account for about 20% of the United States' internet bandwidth consumption [1]. Rising usage in internet bandwidth has become a pressing issue - so much so that many internet service providers are starting to impose bandwidth caps on their customers [2,3].

Part of the problem is the fact that transmitting decent streaming video quality requires a large amount of bandwidth. Video compression techniques do exist to help alleviate the problem, but certain techniques are susceptible to packet losses in transmitted compressed video streams, which could damage video quality [4,5].

A possible alternative solution is to stream low-quality video but then implement a super-resolution algorithm on the receiving end to upscale the streaming video. Super-resolution is an imaging technique that uses multiple low-resolution frames to construct a high-resolution frame. It leverages on the fact that the frames are all unique and provide different information of the same scene in the video. Thus, packet losses or a dropped frame will not have as much of an impact on image quality compared to some video compression techniques.

In our paper, we examine an implementation of a super-resolution algorithm that can be used to upscale video on a number of different platforms ranging from pure hardware to pure software implementations. We also examine a mixed hardware and software implementation in an attempt to determine the feasibility of a typical consumer utilizing such implementations.

* Correspondence: seda@eecs.northwestern.edu
Electrical Engineering and Computer Science Department, Northwestern University, Evanston, IL 60208, USA

In addition to having applications for streaming video, super-resolution has many applications in numerous fields. In some cases where poor video quality is due to hardware limitations, namely the complementary metal-oxide semiconductor (CMOS) sensor resolution or a lens on a camera, super-resolution can be used to help rectify or improve video quality. Examples of hardware limitations include the CMOS sensor on a video graphics array (VGA) camera where the resolution is already fixed or the zoom length of the lens of a digital camera that is fixed. Note that although the process technology for a sensor can be changed to improve resolution, it can also introduce new problems [6]. If the video is not up to par, super-resolution can be used to boost image quality and resolution.

Super-resolution also has applications in military surveillance for unmanned aerial vehicles (UAV). Aerial video feeds are often used for intelligence, surveillance, close air support for ground troops, and reconnaissance missions. However, since the camera is often attached to the belly or wing of the airplane, the recorded video is subject to flight vibrations and disturbances from the elements. Video quality from UAVs is often marred by noise, jitters, and blurry frames [7,8]. Again, super-resolution can be utilized in this case to improve video quality so that the UAV can make better decisions based on the video feeds. Note that the jitters and blur can actually improve the super-resolution results and help the algorithm produce a more detailed and stabilized frame.

The purpose of this paper is to examine several different implementations across multiple platforms of the super-resolution algorithm in terms of performance. More specifically, we chose to examine a field-programmable gate array (FPGA) hardware solution with a budget FPGA and high performance FPGA, a mixed hardware/software FPGA implementation, a general purpose graphics processing unit (GPGPU) implementation, and a strictly software solution in MATLAB. We compare each of these solutions in terms performance. We made an effort to choose equipment with a reasonable price to performance ratio as we are not looking to tweak for ultrahigh performance nor spend exorbitant amounts of money on hardware. The prices for the hardware for the implementations range from US$495 to US$717. We are simply examining the options a consumer or small company with limited funds may consider.

Our experimental results lead to the following conclusions. When comparing the FPGA implementation on the Stratix III with the GPGPU implementation, for a low number of iterations, the Stratix III FPGA outperforms the GPGPU implementation. However, for ten iterations or more, the GPGPU is able to outperform the FPGA. The number of iterations required typically depends on the type of application. For improving web chatting video or a video stream, it may be sufficient to run the algorithm for less than ten iterations on the FPGA; however, if the application is to analyze the details of a video stream, more iterations may be required, and a GPGPU would be preferable in this case.

The rest of the paper is organized as follows. We discuss related work in Section 2. We present the details of the super-resolution algorithm we implemented in Section 3. In Section 4, we detail the architecture of each of our hardware, software, and hardware/software techniques. We present our results in Section 5. Finally, we conclude in Section 6.

2. Related work

The concept of super-resolution first emerged in work published by Tsai and Huang [9] in which they utilized frequency domain methods to improve image quality. Their initial research inspired a series of other works in this field. Irani and Peleg followed in the subsequent years and developed an iterative feedback super-resolution algorithm [10,11], which is the same algorithm we implement in this paper across multiple platforms. More recent works focused on different computational methods for interpolation [12,13] to accelerate the fusion of multiple samples during image super-resolution. These methods are specifically optimized for the case of nonuniformly sampled image frames. While these approaches provide more sophisticated optimization methods, their implementation in resource-constrained hardware platforms such as FPGAs is challenging. Therefore, we opted to study the trade-offs associated with different platforms (FPGA, mixed HW/SW, GPUs) with an algorithm of less complexity, where hardware implementation in resource-constrained environments is still feasible.

Along another track, several works have considered accelerating super-resolution for real-time operation on FPGAs. Angelopoulou et al. proposed combining an adaptive image sensor in conjunction with an FPGA for spatial resolution enhancement [14]. In a following paper, they focused on the developing of the spatial resolution enhancement implementation on the FPGA - namely an iterative back projection method in which low-resolution frames are loaded to memory and read for processing to generate a high-resolution frame [15]. Bowen and Bouganis have examined implementing their own weighted mean super-resolution algorithm combined with an existing multiframe super-resolution algorithm [16]. Our FPGA-based techniques implement the original algorithm by Irani and Peleg [10,11]. Unlike the previous works, we do not focus only on a single FPGA hardware platform but also on hardware/software FPGA implementation, MATLAB implementation, FPGA implementation on a budget FPGA and performance FPGA, and GPGPU implementation.

There has also been some research in the realm of using GPGPUs for super-resolution. Cluff et al. had developed a

super-resolution algorithm on a GPGPU as part of a hierarchical dense correspondence algorithm [17]. Our work is different in that we aim to compare the trade-offs between the multiple implementations, hardware, hardware/software, and software, in terms of performance.

3. Super-resolution algorithm

For the various hardware and software architectures examined in this paper, we implemented the super-resolution algorithm developed by Irani and Peleg [9,10]. Although this particular super-resolution algorithm is older, we have seen in our own experience that it is still used in some industry applications, such as UAVs. This is an iterative feedback algorithm in which the pixel information of several low-resolution frames is combined to form a single high-resolution frame. Each of the individual frames provides slightly different and valuable information in creating the super-resolution frame. Note that none of the low-resolution frames are exactly identical; they are all different, and thus, the algorithm leverages on this fact to extract the details from each low-resolution frame and combine them to form a single high-resolution frame. As a byproduct, this algorithm also aids in image stabilization.

3.1. Algorithm flow

An enlarged hypothesis frame, *H*, is first created from several sequential low-resolution frames. The initial hypothesis frame can either be a simple bicubic interpolation of one of the frames or a bicubic interpolation of the average of all the low-resolution frames. Afterwards, this hypothesis frame is iteratively altered and adjusted with the information from all of the low-resolution frames.

The super-resolution algorithm can be described by Equation 1 below:

$$\sum_{0}^{N}(H-\sum_{0}^{n-1}F_{n}^{-1}\left[F_{n}\left(H,O_{n}\right)-O_{n}\right])$$

Super_resolution ***(input:Original Low Resolution Frames [O_1,…, O_k]; output:SR_Frame)***

```
H = Bicubic_Interpolation(Avr(O1, …,Ok));
For i=1 to N
    For j= 1 to k
        SumFrame = SumFrame + Fk^-1(Fk(H, Ok)-Ok);
    End loop;
    H = H – SumFrame;
End loop;
SR_Frame = H;
Return SR_Frame;
```

Pseudo Code for the SuperResolution Algorithm.

Algorithm 1 runs for *N* iterations specified by the user. The hypothesis frame, *H*, is adjusted in each of the iterations of the algorithm. F_n is a function that is used to align the hypothesis frame with the original low-resolution frames. These are then subtracted with each of the original, individual low-resolution frames, O_n. F_n^{-1} is a function that reverses the alignment and enlarges the error frame. These are summed over the number, *n*, of low-resolution frames so that it can be used to adjust the hypothesis frame, *H*.

The pseudocode shown above summarizes the main steps, and Figure 1 depicts a block diagram of the computation. As shown, the original enlarged hypothesis frame enters a loop that runs for *N* iterations, which is user determined. During the first stage, the high-resolution hypothesis frame is scaled down to the same size as the low-resolution frames. Then, image registration and alignment of the low-resolution hypothesis frame to the original low-resolution frames is performed.

In the second stage, differences of the scaled down aligned hypothesis frame and the original low-resolution frames are taken. This generates multiple error/difference frames between the hypothesis and each of the low-resolution frames. Finally, all of these difference frames are combined and enlarged back to the high-resolution frame size. This creates the enlarged combined error frame that will be used to adjust the hypothesis frame.

A difference is now taken between the enlarged combined error frame and the enlarged hypothesis frame. This final step helps to adjust the hypothesis frame to be used in the next iteration. When the algorithm has completed the specified *N* iterations, it outputs the super-resolution frame.

3.2. Super-resolution example

Figure 2 shows a comparison using bicubic interpolation and the super-resolution algorithm to generate a high-resolution frame. The original low-resolution frame sizes are 240 × 320 pixels. The high-resolution frame is 480 × 640 pixels. Note that this example of the super-resolution algorithm is performed using five low-resolution frames with ten iterations.

We have zoomed in specifically to the sign, which has 'TIMSON DR' printed on it. In this particular example, visible advantages of the super-resolution algorithm are in improved color contrast and decreased overexposure in comparison to the bicubic interpolation technique.

4. Architecture

We have developed four different implementations of the super-resolution algorithm across a variety of platforms ranging from pure software to pure hardware and some with a mix of both hardware and software. This section will provide details of each of the implementations -

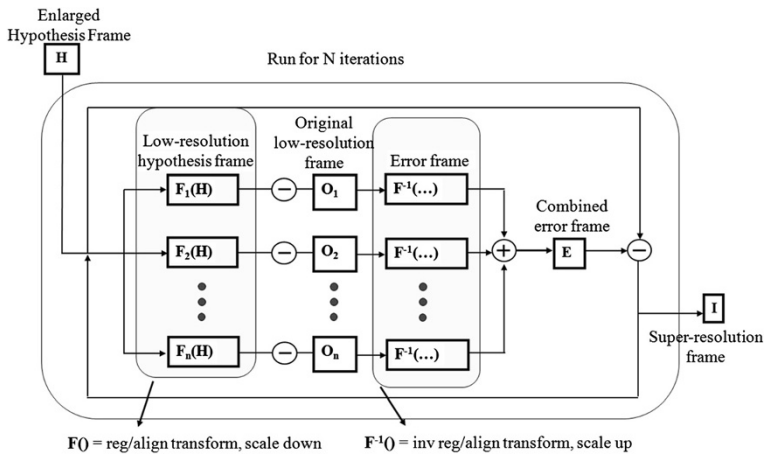

Figure 1 Super-resolution algorithm overview.

FPGA, FPGA with custom instructions (CI), GPGPU, and MATLAB.

4.1. FPGA

As shown in Figure 3, the FPGA implementation contains three main parts - the M4k memory blocks to store the low-resolution frames and the resulting high-resolution frame, the control logic, and the floating point pixel calculation module constructed from single precision, 32-bit floating point operators. The following optimization steps have been implemented to achieve a high performance implementation:

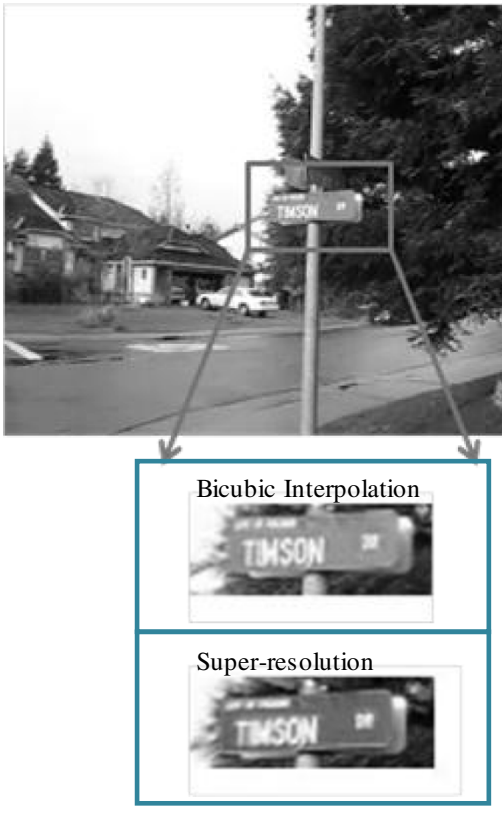

Figure 2 Original low-resolution (240 × 320) image and cropped high-resolution image comparing bicubic interpolation and super-resolution.

4.1.1. Memory layout optimization

The M4k memory blocks are initially preloaded with the low-resolution frames. Multiple M4k blocks were necessary, and the frames had to be divided among these blocks. Vectors that contain information about the necessary pixel shifts are also provided to align the low-resolution frames that are stored. The pixel shifts correspond to offsets in memory; thus, an additional shifting module did not need to be implemented.

4.1.2. Timing optimization of functional blocks through application-specific pipelining

The control logic takes care of multiplexing, the correct data pixels from the memory locations to the floating point pixel calculation module. In addition, this module includes a clock cycle counter. The following floating point pixel calculation module and memory access stages of the design are all pipelined. Each block in Figure 3 -

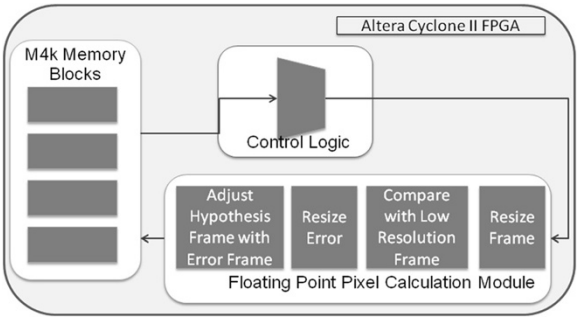

Figure 3 FPGA implementation overview.

memory accesses and writes, the resizing frame, compare with low-resolution frame, resize error, and adjust hypothesis frame with error frame - are stages of the pipeline. We found that the longest latency of all the pipelined stages is 12 clock cycles at 103.15 and 122.25 MHz for the Cyclone II FPGA and Stratix III FPGA, respectively. Thus, every stage of the pipeline is 12 clock cycles. The counter helps ensure that the correct pixel data from the M4k memory blocks are flopped into the floating point pixel calculation module at the correct time.

The floating point pixel calculation module performs the algorithm outlined in Figure 1 and writes the resulting high-resolution pixel back to the portion of the M4k memory block reserved for the final high-resolution frame. This module mainly consists of floating point multiplication and floating point addition modules for the resizing stages, floating point subtraction and floating point addition modules for the comparison stage to generate the error frame, and floating point subtraction modules for the hypothesis adjustment stage. Note that we used a combined floating point addition and subtraction unit, in which a single bit was used to designate the module as either a subtract operation or an add operation. All of the floating point operations are single precision (32-bit) floating point operations. The shared add and subtract unit requires seven clock cycles to complete, and the multiplication unit requires five clock cycles to complete.

4.2. FPGA with custom instructions

This implementation is a combination of hardware and software. The general overview of the architecture is shown in Figure 4. A NIOS II soft core processor (Altera Corporation, San Jose, CA, USA) is instantiated and used in conjunction with dedicated single precision, 32-bit floating point custom instructions (floating point multiply, floating point subtract, and floating point addition).

The NIOS II soft core processor is a RISC processor constructed using the logic resources on the FPGA board and is capable of compiling and running C code.

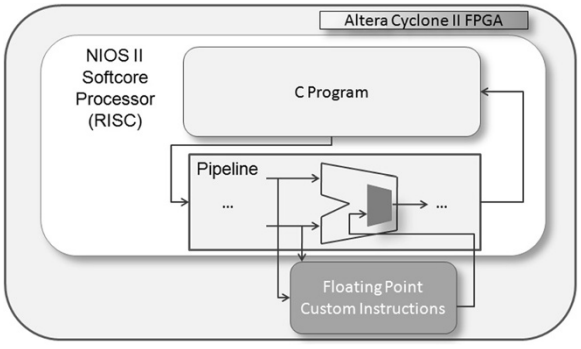

Figure 4 FPGA with custom instructions (CI) overview.

The dedicated floating point custom instructions are directly connected to the arithmetic logic unit (ALU) pipeline stage of the NIOS II processor. Within the ALU, there is a MUX selecting between output from the floating point custom instructions and regular ALU operations, which is controlled by the C code.

All floating point operations in the super-resolution algorithm are off-loaded to the dedicated floating point custom instructions. Thus, the general algorithm and control logic are written in C and run on the NIOS II processor. However, the actual floating point calculations are accelerated and performed by the dedicated hardware implemented as custom instructions on the FPGA.

This method is particularly friendly to software developers in that the data transfer and floating point computations are transparent to the software programmer. The only requirement would be to specify that the operation is to be off-loaded to the custom instructions with a simple command.

5. Results

In this section, we will present our results and an analysis of the performance. For each of our individual platforms, we used five consecutive low-resolution frames from a 240 × 320 video stream to generate a single 480 × 640 frame. The performance metric is frames per second (FPS), which is calculated by measuring the time that it takes to process a single super-resolution frame.

5.1. Platform setup

For the FPGA implementation, we targeted two different FPGA boards, a Cyclone II FPGA and a Stratix III FPGA, to compare the performance between a budget, low-end FPGA and a high-performance FPGA. We used the Quartus II 9.0 environment to develop our hardware design. We used the floating point addition, floating point subtraction, floating point multiplication, and M4K RAM memory modules included in the Quartus II software within the Megawizard tool to implement the super-resolution algorithm. These are single precision, 32-bit modules.

As for the FPGA CI implementation, we used the Quartus II 9.0 software with SOPC builder to instantiate the NIOS II processor. We used the NIOS II 9.0 software environment to develop and compile our C code and to enable support for off-loading floating point calculations to the floating point custom instructions on

Table 1 FPGA logic usage

FPGA Cyclone II	FPGA Stratix III	FPGA CI
Percent logic used	Percent logic used	Percent logic used
95%	93%	35%

Table 2 Performance comparison

	FPGA Cyclone II		FPGA Stratix III		FPGA CI		GPGPU		MATLAB	
Iterations	Time (s)	FPS	Time (s)	FPS	Time (s)	FPS	Time (s)	FPS	Time (s)	FPS
5	0.045	22.38	0.017	59.692	3.51	0.29	0.021	47.18	3.15	0.32
10	0.089	11.19	0.034	29.846	6.43	0.16	0.023	42.73	4.91	0.20
15	0.134	7.46	0.050	19.897	9.30	0.11	0.026	39.04	6.69	0.15
20	0.179	5.60	0.067	14.923	12.30	0.08	0.028	35.94	8.43	0.12
25	0.223	4.48	0.084	11.938	15.21	0.07	0.030	33.29	10.65	0.09
30	0.268	3.73	0.101	9.949	18.10	0.06	0.033	30.40	11.99	0.08
35	0.313	3.20	0.117	8.527	21.10	0.05	0.034	29.02	13.82	0.07

the FPGA. Note that this implementation was only implemented on the Altera DE2 board containing the Cyclone II FPGA.

For the GPGPU implementation, we ran the super-resolution algorithm on a NVIDIA 8800 GT graphics card with CUDA v2.0. The GPGPU version of the super-resolution algorithm involves writing C code with CUDA extensions. The main portions of the code consist of transferring the low-resolution frames and resulting high-resolution frame between the host (computer) to the device (GPGPU), writing the kernel to do the frame resizing, writing the kernel to compare and generate the error frame, and writing a kernel to adjust the hypothesis with the error frame data.

As a general purpose alternative, we also implemented the super-resolution algorithm in MATLAB. This is a tool often used by scientists because of the ease of use and ability to easily manipulate matrices and images. The super-resolution algorithm depicted by the pseudo-code in Section 3.1 and Figure 1 is implemented. All functions were custom-written except for the bicubic interpolation function, which was taken from MATLAB. We have not utilized any toolboxes to off-load computation to special units, such as GPU. We found the toolbox for GPU utilization to be quite buggy at the time of our development and decided to avoid any instability. We have also relied on the fact that we present a pure GPU implementation for comparison anyway. Our MATLAB implementation was on a desktop machine that had an Intel 2.53 GHz Core 2 Duo processor with 3 GB of RAM.

5.2. FPGA resource usage

Table 1 provides the percentages of logic used for the FPGA Cyclone II implementation, the FPGA Stratix III implementation, and the FPGA CI implementation. The reported resource usage for the FPGA Cyclone II implementation and FPGA CI implementation are after mapping the design to the Altera DE2 development board while the FPGA Stratix III results are based on reports obtained after the place and route tool in Quartus II. As it is evident from the results, the FPGA CI implementation requires much less logic resources because it only requires the instantiation of the NIOS II processor and the dedicated floating point hardware.

5.3. Performance analysis

Table 2 provides the computation times and FPS with varying iterations of the super-resolution algorithm across all of our implementations.

Figure 5 shows a graphical representation of the FPS results for all of our implementations. As can be seen, the FPGA CI implementation provided the worst overall performance in terms of FPS among the five implementations - even worse than the pure software version in MATLAB. The subpar performance of this implementation can be attributed to the fact that the algorithm is implemented in C code running on the 50-MHz NIOS II processor. Although there are benefits of off-loading the floating point operations to dedicated hardware, the overhead of transferring the data and slow processor outweigh the benefits of the faster hardware-based floating point calculation modules. In addition, there is no way to increase the number of parallel floating point operations as the C code is run sequentially whereas the pure FPGA implementation could have multiple pixel processing units running in parallel. The C code is sequentially executed, and operations

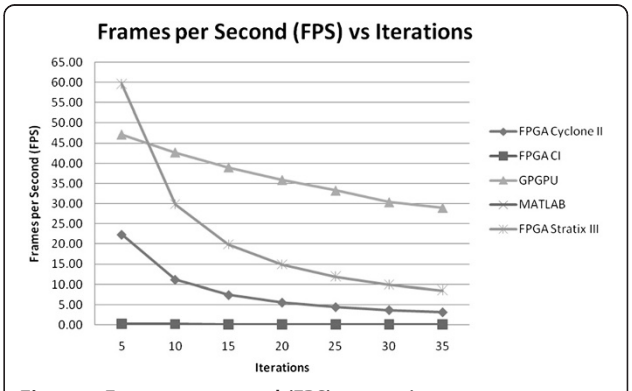

Figure 5 Frames per second (FPS) comparison.

are sequentially passed off to the optimized hardware-based floating point custom instructions.

As can be seen, the pure software implementation on MATLAB marginally outperforms the FPGA CI implementation by brute force of the performance capability of the processor.

The hardware-based FPGA Cyclone II implementation has mediocre performance but is markedly better than the MATLAB and FPGA CI implementations. However, it is unable to outperform or even reach the levels of computational power of the GPGPU. This is due to the fact that the Cyclone II FPGA is a lower end FPGA board and has limited logic resources.

When using the high-end Stratix III FPGA, the performance is much better than the Cyclone II FPGA. The Stratix III FPGA is based on the 65-nm technology and contains much more available logic elements in comparison to the 90-nm technology of the Cyclone II FPGA. With more available densely packed logic, we were able to incorporate more pixel processing units to increase parallel pixel processing. We fit up to four floating point pixel calculation modules on the Cyclone II FPGA as opposed to nine on the Stratix III FPGA. In addition, the compiled design had a higher clock speed for the Stratix III FPGA at 122.25 MHz compared to the Cyclone II FPGA at 103.15 MHz. With a higher clock speed and logic resources available, the Stratix III FPGA was able to provide much better performance when compared to the Cyclone II FPGA.

When comparing the FPGA implementation on the Stratix III with the GPGPU implementation, it can be seen that for a low number of iterations, the Stratix III FPGA outperforms the GPGPU implementation. However, for ten iterations or more, the GPGPU is able to outperform the FPGA. The number of iterations required typically depends on the type of application. For improving web chatting video or video stream, it may be sufficient to run the algorithm for less than ten iterations on the FPGA; however, if the application is to analyze the details of a video stream, more iterations may be required, and a GPGPU would be preferable in this case. There also seems to be a steep drop-off in performance as the number of iterations increase for the Stratix III FPGA implementation. This can be attributed to the fact that the GPGPU is a more specialized hardware and is designed to deal with floating point graphics and matrix operations that works well with the super-resolution application.

The GPGPU implementation provides the best overall performance. It is able to achieve better than 29 FPS for even up to 35 iterations.

6. Conclusions

In our analysis of the super-resolution algorithm across multiple platforms - FPGA implementation (Cyclone II and Stratix III), FPGA CI implementation, GPGPU implementation, and MATLAB implementation - we found that the GPGPU fared best in terms of overall performance. The GPGPU implementation is able to provide up to 29 FPS for 35 iterations of the super-resolution algorithm. The worst overall performance came from the FPGA CI implementation due to the much slower clock speed and data transfer times to the dedicated floating point modules. Although those individual floating point modules are designed and tweaked for maximum performance, they were not able to overcome the overhead of the much slower NIOS II processor and data transfer rates. As expected, the MATLAB software implementation also performed near the bottom of the group. The FPGA Cyclone II implementation had mediocre performance, and the FPGA Stratix III implementation rivaled the GPGPU performance in some cases. Choosing between the Stratix III and GPGPU would strongly depend on the application.

Competing interests
The authors declare that they have no competing interests.

Received: 31 October 2012 Accepted: 16 May 2013
Published: 5 June 2013

References

1. Sandvine, *Fall 2010 Global Internet Pheonomena (2010)*. http://www.sandvine.com/news/global_broadband_trends.asp. Accessed 26 March 2013
2. R Singel, *Comcast Rolls Out Broadband Meters Coast to Coast. (Wired, 2010)*. http://www.wired.com/business/2010/04/comcast-broadband-meters/. Accessed 26 March 2013
3. R Singel, *AT&T Puts Broadband Users on Monthly Allowance. (Wired, 2011)*. http://www.wired.com/business/2011/03/att-dsl-cap/. Accessed 26 March 2013
4. D Wu, YT Hou, Zhu W, Zhang Y, Peha JM. Streaming video over the internet: approaches and directions. IEEE Transactions on Circuits and Systems for Video Technology **11**(3), 282–300 (2001)
5. E Setton, T Yoo, X Zhu, A Goldsmith, Cross-layer design of ad hoc networks for real-time video streaming. IEEE Wireless Communications **12**(4), 59–65 (2005)
6. SC Park, MK Park, MG Kang, Super-resolution image reconstruction: a technical overview. IEEE Signal Processing Magazine **20**(3), 21–36 (2003)
7. Y Wang, R fevig, RR Schultz, Super-resolution mosaicking of UAV surveillance video, in *15th IEEE International Conference on Image Processing* (, San Diego, 2008)
8. AH Yousef, Z Rahman, Super-resolution reconstruction of images captured from airborne unmanned vehicles, in *Proceedings of SPIE*, vol. 7701 (Orlando, 2010)
9. RY Tsai, TS Huang, Multiframe image restoration and registration. Advances in Computer Vision and Image Processing **1**, 317–339 (1984)
10. D Keren, S Peleg, R Brada, Image sequence enhancement using sub-pixel displacements, in *Computer Society Conference on Computer Vision and Pattern Recognition, Ann Arbor*, 1988
11. M Irani, S Peleg, Super resolution from image sequences. ICPR **2**, 115–120 (1990)
12. A Gilman, DG Bailey, SR Marsland, Least-squares optimal interpolation for fast image super-resolution, in *IEEE International Symposium on Electronic Design, Test and Application* (, Ho Chi Minh City, 2010)
13. A Gilman, *Least-squares optimal interpolation for direct image super-resolution*. PhD Thesis, School of Engineering and Advanced Technology (Massey University, Palmerston North, 2009)
14. ME Angelopoulou, CS Bouganis, P Cheung, G Constantinides, Robust real-time super-resolution on FPGA and an application to video enhancement. ACM Trans. Reconfigurable Technol. Syst **2**(4) (2009). 22:1-22:29
15. ME Angelopoulou, CS Bouganis, PYK Cheung, GA Constantinides, FPGA-based real-time super-resolution on an adaptive image sensor. Applied Reconfigurable Computing **4**, 125–136 (2008)

16. O Bowen, C Bouganis, Real-time image super resolution using an FPGA, in *International Conference on Field Programmable Logic and Applications* (Heidelberg, 2008), pp. 89–94
17. S Cluff, BS Morse, M Duchaineau, JD Cohen, GPU-accelerated hierarchical dense correspondence for real-time aerial video processing, in *Workshop on Motion and Video Computing*. Snowbird **8–9**, 1–8 (Dec 2010)

doi:10.1186/1687-6180-2013-116
Cite this article as: Leung and Memik: **Exploring super-resolution implementations across multiple platforms.** *EURASIP Journal on Advances in Signal Processing* 2013 **2013**:116.