GPU Processing for UAS-Based LFM-CW Stripmap SAR

Craig Stringham and David G. Long

Abstract

Unmanned air systems (UAS) provide an excellent platform for synthetic aperture radar (SAR), enabling surveillance and research over areas too difficult, dangerous, or costly to reach using manned aircraft. However, the nimble nature of the small UAS makes them more susceptible to external forces, thus requiring significant motion compensation in order for SAR images to focus properly. SAR backprojection has been found to improve the focusing of low-altitude stripmap SAR images compared to frequency domain algorithms. In this paper we describe the development and implementation of SAR backprojection appropriate for UAS based stripmap SAR that utilizes the unique architecture of a GPU in order to produce high-quality imagery in real-time.

Introduction

Unmanned air systems (UAS) carrying synthetic aperture radar (SAR) can obtain high-quality high-resolution information over areas too difficult, dangerous, or costly to reach using manned aircraft. SAR systems are active radars that transmit and receive microwave signals. The received signals are used to create images of the surface and are able to operate regardless of illumination or weather conditions. The nimble nature of a small UAS makes it much more mobile but also more susceptible to external forces, thus requiring significant motion compensation in order for SAR images to focus properly. Low-altitude operation further complicates motion compensation of stripmap SAR images, due to the large range of incidence angles and increased range cell migration. SAR backprojection inherently handles arbitrary aircraft motion and low-altitude geometry and can form images directly along known topography making it a particularly effective algorithm for UAS-based SAR. However, backprojection is much more computationally demanding than frequency domain algorithms (Melvin and Scheer, 2012). Fortunately, backprojection processing is easily parallelized and computed efficiently on graphics processing units (GPU). Several studies have been conducted on implementing backprojection on GPUs, most notably Fasih and Hartley (2010), Benson *et al.*, (2012), Capozzoli *et al.* (2013), and Nguyen *et al.* (2004), but these papers focus on the simplest form of spotlight mode SAR backprojection and are not directly applicable to stripmap SAR. In this paper we present the development and implementation of a highly efficient GPU-based SAR backprojection processor for stripmap imaging. In particular, we develop a stripmap processor for linear frequency-modulated continuous-wave (LFM-CW) SAR systems operated on a UAS.

This paper is organized as follows. We begin with a background discussing the LFM-CW signal, stripmap SAR, and a brief introduction to the NVIDIA GPU architecture and Compute Unified Device Architecture (CUDA). Then, we develop a SAR backprojection method that accounts for motion during the pulse and the moving antenna pattern, which is suitable for UAS based stripmap SAR. This is followed by a discussion of the implementation of the SAR processor on a GPU. Finally, we use SAR data from CASIE 2009 (Long *et al.*, 2010) to analyze the performance of the implementation and present the resulting imagery.

Background

LFM-CW Signal

Modern SAR systems can be very small, low-power, and lightweight such that they can be used on a small UAS. This reduction in size has greatly been made possible by technology advancements and the use of LFM-CW technology. LFM-CW radars maximize the signal to noise ratio (SNR) achievable for a given peak transmit power by continuously transmitting, maximizing the energy of the received signal. To achieve high-resolution, the transmit signal is modulated over a wide range of frequencies.

The transmit signal of an LFM-CW radar can be described as the complex exponential:

$$s_t(\eta,t) = \exp\left\{-j\left(2\pi f_0 t + \pi k_r t^2 + \varphi\right)\right\} \tag{1}$$

where f_0 is the carrier frequency, k_r is the chirp rate, φ is the initial phase of the system, and η and τ are respectively "slow-time" and "fast-time" which are typical SAR notation. Slow-time changes discretely with each pulse while fast-time ranges over the length of the pulse, T_p. The received signal from a single scatterer with position, $\vec{x}$, can be described as an attenuated, time-delayed copy of the transmit signal. The received signal can be written as:

$$\begin{aligned} r_{\vec{x}}(\eta,t) &= A_{\vec{x}}(\eta,t)\sigma_{\vec{x}} s_t\left(t-\tau_{\vec{x}}(\eta,t)\right) \\ &= A_{\vec{x}}(\eta,t)\sigma_{\vec{x}}\exp\left\{-j\left(2\pi f_0(t-\tau_{\vec{x}}(\eta,t)) + \pi k_r(t-\tau_{\vec{x}}(\eta,t))^2 + \varphi\right)\right\} \end{aligned} \tag{2}$$

where $A_{\vec{x}}$ is an amplitude function due to the antenna pattern, incidence angle, the distance to the scatterer, and the backscatter from the target is σ_x, and $\tau_{\vec{x}}(\eta,t)$ is the round-trip propagation time of the radar signal, which consists of the time for the pulse to travel from the transmit antenna to the position $\vec{x}$ plus the time for the backscatter to radiate back to the receive

Brigham Young University, 459 Clyde Building, Provo UT 84602 (stringham@ieee.org).

Photogrammetric Engineering & Remote Sensing
Vol. 80, No. 12, December 2014, pp. 1107–1115.
0099-1112/14/8012–1107

doi: 10.14358/PERS.80.12.1107

antenna on the aircraft. Because the motion of the aircraft is many orders of magnitude slower than the speed of light c_0 and the transmit and receive antennae are frequently only separated by a small distance the propagation time is approximated as:

$$\tau_{\vec{x}}(\eta,t) = \frac{2 \parallel \vec{p}(\eta,t) - \vec{x} \parallel}{c_0} \tag{3}$$

where $\vec{p}$ is the position of the radar. The recieved signal is mixed with the transmit signal and low-pass filtered, which is mathematically equivalent to mixing the received signal with the complex conjugate of the transmit signal. The resulting "dechirped" signal is given by:

$$\begin{aligned} &s_{\vec{x}}(\eta,t) = \\ &= A_{\vec{x}}(\eta,t)\sigma_{\vec{x}}\exp\{j\left(2\pi k_r \tau_{\vec{x}}(\eta,t)t - \pi k_r \tau_{\vec{x}}(\eta,t)^2 + 2\pi f_0 \tau_{\vec{x}}(\eta,t)\right)\}. \end{aligned} \tag{4}$$

Because LFM-CW SAR is typically operated from low altitude, the imaging scene is narrow enough that the dechirped signal occupies much less bandwidth than the transmit signal. This enables a significant reduction in the data storage requirements and the required speed of the instrument's analog to digital converter (ADC).

Stripmap SAR

The most notable SAR operational modes are spotlight and stripmap (Curlander and McDonough, 1991; Melvin and Scheer, 2012). In spotlight SAR the antenna is controlled to point at the same patch of ground during the flight; whereas with stripmap SAR the side-looking antenna has fixed pointing. In spotlight mode, processing objects in the imaging scene are visible during the entire data collection (with the exception of shadowed objects obstructed by other objects in the imaging area), whereas in stripmap mode the imaged scene is constantly progressing. This leads to fundamental differences in the processing algorithms for the two operation modes.

Traditionally LFM-CW stripmap SAR image processing is performed with frequency domain algorithms such as the range Doppler algorithm (RDA), the frequency scaling algorithm (FSA), and the Ω-*k* algorithm (Cumming and Wong, 2005; Melvin and Scheer, 2012; Wu *et al.*, 2012; Pfitzner *et al.*, 2013). Frequency domain algorithms use the assumption that the aircraft is travelling a straight line with constant velocity in order to process image sections in batches using FFTs. There are a number of modifications to these algorithms that compensate for non-linear motion (Moreira and Huang, 1994; Stevens *et al.*, 1995; Zaugg and Long, 2007), but these methods complicate the original algorithms and their performance has limitations (Jakowatz *et al.*, 1996). In contrast, the SAR backprojection algorithm described below is a time-domain algorithm that inherently compensates for non-linear flight paths and surface topography. The improved focusing capability of backprojection has been noted in a number of studies (Nguyen *et al.*, 2004; Frey *et al.*, 2009; Stringham and Long, 2011). Backprojection increases the computational burden of generating SAR images. While there are methods for accelerating backprojection with some assumptions as noted by Ulander *et al.* (2003) and Moon and Long (2013) and others. As described in this paper, there are many cases in which SAR backprojection can be performed in real-time using the dramatic increase of signal processing power available on GPUs.

GPU Architecture

GPUs have anywhere from tens to thousands of processing cores, enabling the fast and efficient processing of billions of math operations per second on a single device. While there are several different GPU architectures, the key aspects important to our discussion can be found in nearly all recent GPUs. As an illustration we use an NVIDIA GTX 285 GPU shown in Figure 1.

An NVIDIA GPU is broken into groups of processing cores which NVIDIA calls multiprocessors. The number of multiprocessors is dependent on the particular GPU. Multiprocessors contain three types of sub-processors including single precision units (SPU), special function units (SFU), and double precision units (DPU). Floating point operations including multiply, add, and fused multiply-add are performed by the SPUs and DPUs for 32-bit and 64-bit values, respectively. The SFUs perform 32-bit transcendental operations such as inverse, inverse square root, sine, cosine. The 64-bit transcendental operations are performed by software. The number of each type of processing units is also dependent on the GPU. Each multiprocessor includes an instruction unit which controls which processing units operate at a given time. All of the operating units execute the same instruction on different data, much like a single instruction multiple data (SIMD) architecture, but because the instruction unit controls which units are operating, the processing streams, or "threads," are allowed to branch independently.

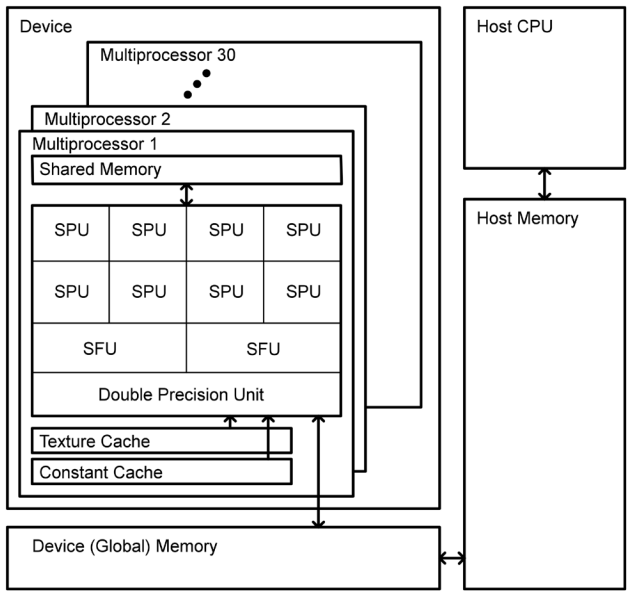

Figure 1. Illustration of the NVIDIA GTX 285 GPU architecture.

Table 1. Latency of GPU Operations in Number of Clock Cycles for the NVIDIA T10 Architecture (GTX 285) (Wong et al. 2010)

Operation	Latency (cycles)	Throughput (ops/cycle)
multiply	24	11.2
add,sub,max,min,mad	24	7.9
divide	137	1.5
square-root	56	2.0
sine,cosine	48	2.0
shared memory access	38	N/A
global memory access	436-443	N/A

In terms of memory, the GPU device has a large amount of dedicated off-chip memory, termed global memory. Each multiprocessor has a large bank of 32-bit registers, a block of shared memory, and constant and texture caches. These resources are shared among the processors within a multiprocessor. It is important to understand the types of memory available because with the vast amount of computing power of the GPU,

the achievable performance gains are frequently determined by the memory accesses of the threads. This is illustrated by the latency of different operations as shown in Table 1.

The texture cache is a unique GPU feature. It provides read-only access to global memory, and is optimized for spatial locality for one-, two-, or three-dimensional arrays. It has hardware built in to provide linearly-interpolated values with virtually no performance cost. The linear interpolation is performed using only 8 bits of fractional precision, but in many cases the reduction of memory access time warrants the lower accuracy. Later, we explain how the hardware linear interpolation can be used to greatly accelerate backprojection processing.

In order to utilize the computing capabilities of the GPU, NVIDIA provides the Compute Unified Device Architecture (CUDA) which augments the C programming language. CUDA adds a kernel construct which ties a groups of threads to multiprocessors, where each thread executes the same function virtually in parallel. Each thread maintains its own program counter, and branches can diverge in the execution. If the branches diverge, the independent branches are executed serially. In this case the processor is not fully utilized. Thus it is important to avoid branch divergence to achieve optimal performance.

In the following section we develop SAR backprojection for LFM-CW stripmap SAR. Then we describe the real-time implementation of the backprojection algorithm using the GPU's processing capabilities described in this section.

Derivation of Stripmap SAR Backprojection

In LFM-CW SAR, the motion of the aircraft during a pulse can be significant as noted by Ribalta (2011) and Meta *et al.* (2007). Ribalta (2011) derives a backprojection method for focusing spotlight-mode LFM-CW SAR data; following a similar methodology we develop a backprojection method for stripmap LFM-CW SAR that includes an extension for ultra-wide-band (UWB) SAR. The received signal can be written as the integral of the dechirped signal, Equation 4, from a single target over the imaging area:

$$s(\eta,t)=\int_R I(\vec{x})A_{\vec{x}}(\eta,t)e^{j\left(2\pi k_r t\tau_{\vec{x}}(\eta,t)+\tau_{\vec{x}}(\eta,t)2\pi f_0-\pi k_r t\tau_{\vec{x}}^2(\eta,t)\right)}dx, \tag{5}$$

where R is the imaging area, k_r is the chirp rate, and I is the imaged area. The simplest method for reconstructing the SAR image, albeit highly computationally demanding, is discrete time-domain correlation, described as:

$$\hat{I}(\vec{x})=\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}W_{\vec{x}}[m,n]s[m,n]s_{\vec{x}}^{*}[m,n], \tag{6}$$

where $\hat{I}(\vec{x})$ is the reconstructed SAR image, $s_{\vec{x}}$ is the reference signal for a target at pixel location $\vec{x}$, W is an apodization window, N is the number of samples in a pulse, and M is the number of pulses over which the backprojection is calculated. Square braces are used to indicate discrete samples such that:

$$s[m,n]=s(\eta_m,t_n)=s(mT_p,n/f_s) \tag{7}$$

where T_p is the pulse length, and f_s is the ADC sampling frequency. To simplify we use a phase-only reference signal yielding

$$\hat{I}(\vec{x})=\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}W_{\vec{x}}[m,n]s[m,n]]\cdot \exp\{-j\left(2\pi k_r t_n\tau_{\vec{x}}[m,n]+2\pi f_0\tau_{\vec{x}}[m,n]-\pi k_r\tau_{\vec{x}}^2[m,n]\right)\} \tag{8}$$

Using no approximations Equation 8 generates the ideal reconstructed image; however it is computationally intensive, on the order of $O(MNL_I)$ where L_I is the number of pixels in the image, M is the number of pulses, and N is the number of samples in each pulse.

To reduce computation we first range-compress the samples. In order to perform range compression, the range from the radar to the target is traditionally approximated as stationary during a pulse. However, because LFM-CW SAR uses a much longer pulse we approximate τ_x as having linear motion during the pulse, i.e.:

$$\tau_{\vec{x}}[m,n]\approx\tau_{\vec{x}}[m,0]+v_{\vec{x}}[m]t_n \tag{9}$$

where

$$v_{\vec{x}}[m]=\frac{d\tau_{\vec{x}}(\eta_m,0)}{dt}=\frac{2<\vec{v},\vec{p}(\eta_m,0)-\vec{x}>}{c_0\,\|\,\vec{p}(\eta_m,0)-\vec{x}\,\|} \tag{10}$$

where $\vec{v}$ is a vector of the component velocities. By selecting an apodization window that can be separated into range and azimuth windows W^a and W^r, Equation 8 can be written as:

$$\hat{I}(\vec{x})\approx\sum_{m=0}^{M-1}W_{\vec{x}}^{a}[m]\exp\{-j\phi_\alpha[m]\}\cdot\sum_{n=0}^{N-1}W_{\vec{x}}^{r}[n]s[m,n]\cdot\exp\{-j\phi_r[n]\} \tag{11}$$

where

$$\phi_\alpha[m]=2\pi f_0\tau_{\vec{x}}[m,0]-\pi k_r\tau_{\vec{x}}^2[m,0] \tag{12}$$

and

$$\phi_r[n]=2\pi k_r t_n\tau_{\vec{x}}[m,0]+2\pi k_r v_{\vec{x}}[m]t_n^2+2\pi f_0 v_{\vec{x}}[m]t_n-\pi k_r(2\tau_{\vec{x}}[m,0]v_{\vec{x}}[m]t_n+v_{\vec{x}}^2[m]t_n^2) \tag{13}$$

The t_n^2 terms in Equation 13 prevent us from replacing the second summation with the range-compressed data. Ribalta's derivation accomplishes this task by noting that the linear terms $2\pi k_r t_n\tau_{\vec{x}}$ and $2\pi f_0 v_{\vec{x}}t_n$ dominate the summation in many imaging scenarios, and discards the other terms. However, to improve upon this approximation we instead use the least squares approximation

$$t_n^2\approx T_p t_n-\frac{T_p^2}{6} \tag{14}$$

The error induced by the approximation in Equation 14 is minimal for low-altitude and typical aircraft speeds. Using Equation 14 in Equation 11 yields:

$$\hat{I}(\vec{x})\approx\sum_{m=0}^{M-1}W_{\vec{x}}^{a}[m]\exp\left\{-j\left(\phi_\alpha[m]-\pi k_r v_{\vec{x}}^2[m]\frac{T_p^2}{6}+2\pi k_r v_{\vec{x}}[m]\frac{T_p^2}{6}\right)\right\}\cdot S_m\left(\frac{N}{f_s}\left(k_r\tau_{\vec{x}}[m,0]+f_0 v_{\vec{x}}[m]+k_r T_p v_{\vec{x}}[m]-\frac{k_r}{2}v_{\vec{x}}[m]\left(2\tau_{\vec{x}}[m,0]+T_p v_{\vec{x}}[m]\right)\right)\right) \tag{15}$$

where S_m is the Fourier transform of $W_r[n]s[m,n]$, which is the range-compressed data. Range compression is accomplished using a Fourier transform of the dechirped data because each object is represented by a single frequency related to the time-delay using the stop-and-hop assumption in the dechirped data, Equation 4. Using the range-compressed data instead of the dechirped data greatly reduces the computational complexity to the order of $O(ML_1 + MN\log_2(N))$.

By analyzing a point at the far end of the range swath for a particular radar's imaging parameters, further simplifications can be made. For example the fourth term phase is only beneficial for extremely high-bandwidth and long pulses which

are difficult to realize in practice, and the third phase term and the last term on the range index can be neglected because $v_x << 1$, which is a ratio of the aircraft speed and the speed of light. The third range index is only beneficial in for extreme ranges that are not typical in LFM-CW imaging. With this small approximation Equation 15 simplifies to:

$$\hat{I}(\vec{x}) \approx \sum_{m=0}^{M-1} W_{\vec{x}}^{a}[m] \exp\left\{-j\phi_{\alpha}[m]\right\} \\ S_m\left(\frac{N}{f_s}\left(k_r \tau_{\vec{x}}[m,0] + f_0 v_{\vec{x}}[m] + k_r T_p v_{\vec{x}}[m]\right)\right) \tag{16}$$

The first term in indexing S_m is found in traditional backprojection methods, and the second term partially compensates for the motion during the pulse and was identified by Ribalta (Ribalta, 2011). The third term, which we call the UWB correction, was not included in Ribalta's study; however, we show that in ultra-wide-bandwidth operation the third term can be significant.

Figure 2 illustrates the different indexing terms for the range compressed data for an UHF radar with 500 MHz bandwidth and target at a 45° squint. We note that in a practical imaging scenario, the transmit signal at this frequency would be notched to avoid conflicting bands; however, the notching does not significantly alter the results of the simulation. The range-compressed signal received from a single point target with the aircraft moving during the pulse is represented by the thick plot. The thin plot represents the signal received as if the aircraft were stationary as assumed with "stop-and-hop." The dashed vertical lines represent the three indexing calculations discussed: from right to left the lines describe the indexing term using the stop-and-hop assumption, Ribalta's correction, and the UWB correction described in Equation 16.

In this scenario the UWB correction described in Equation 16 is needed in order to correctly index the peak of the range-compressed data and properly focus the radar image. The ratio of Ribalta's correction and the additional UWB correction is the ratio of the carrier frequency and the bandwidth of the radar. So for narrow band SAR imaging where the bandwidth is much smaller than the carrier frequency, Ribalta's correction is sufficient for focusing the SAR image.

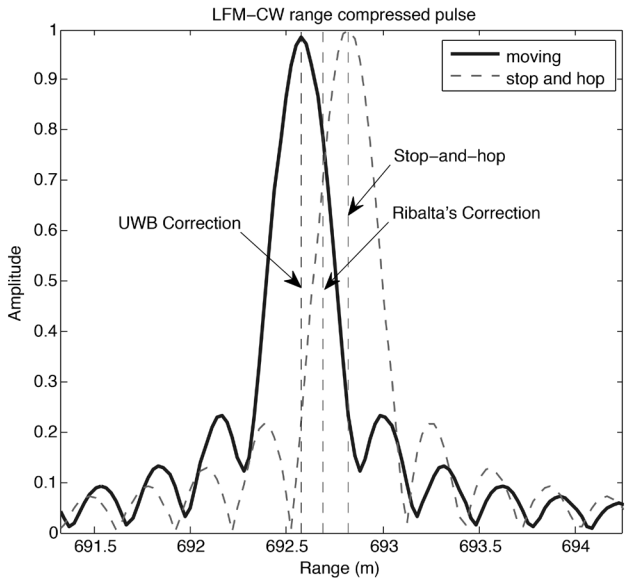

Figure 2. An illustration of range-compressed data for a point target observed by a moving aircraft (thick plot) and a stationary aircraft (dashed plot) and the available indexing terms. The indexing terms are represented by the vertical dashed lines. From right to left, the lines respectively represent the indexing term using the stop-and-hop assumption, Ribalta's correction, and the indexing term given in Equation 16. The simulation is for an UHF radar with a 500 MHz bandwidth and a target at a 45° squint.

Compensated Backprojection

The phase-only backprojection described in Equation 15 works well for data sets collected on platforms where the antenna attitude is fairly constant, such as satellites and large aircraft; however, a small UAS is very susceptible to external forces often resulting in undulating motion. Thus, some pixels are calculated using more samples and different antenna gains than others resulting in SAR images with varying intensity. In this discussion we do not address the effects of incidence angle which are addressed in detail in Frey *et al.* (2013).

In order to remove the variation of intensity due to the motion and attitude of the aircraft we examine the backprojection filter. The complex gain $A_{\vec{x}}$ from Equation 4 is given by the radar equation, which can be written as:

$$A_{\vec{x}} = \left(B \frac{G_{\vec{x}}^2(\eta)}{\| \vec{p}(\eta) - \vec{x} \|^4} \sigma_{\vec{x}}^0\right)^{1/2} \tag{17}$$

where B is a constant that accounts for all the radar parameters that are independent of flight and position parameters such as the transmit power, receiver gain, system losses, etc., $G_{\vec{x}}(\eta)$ is the antenna gain (assuming mono-static operation), and $\sigma_{\vec{x}}^0$ is the normalized backscatter. Using Equation 17 we relate the magnitude of Equation 16 to the normalized backscatter as:

$$\sigma^0 \approx \frac{|\hat{I}(\vec{x})|^2}{B\left(\sum_m W_a(m) \frac{G_{\vec{x}}^2(\eta)}{\| \vec{p}(\eta) - \vec{x} \|^4}\right)} \tag{18}$$

If all of the system parameters are precisely known this derivation produces fully calibrated images and even when the full parameters are not fully known most of the variation can still be compensated for within a scale factor.

Interpolation

Note that Equation 8 uses continuous values of the range compressed data; however, the range-compressed data is discrete, requiring that the range-compressed data be interpolated. The selection of the interpolation method can dramatically affect both the speed and accuracy of the processor. In order to produce high-quality images, backprojection implementations use some form of interpolation; although it is not always explicitly stated. Ideal interpolation is achieved using the DTFT of the dechirped data or Dirichlet interpolation of the discrete range-compressed values. Both of these approaches are computationally demanding. Because of the simplicity of implementation, some backprojection methods use linear interpolation (Fasih and Hartley, 2010) while others significantly zero-pad the range-compression FFT to approximately apply Dirichlet interpolation at discrete points (Stringham and Long, 2011; Benson *et al.*, 2012). Combining zero-padding and polynomial interpolation methods increases the accuracy; however, the non-equispaced result FFT (NERFFT) provides even better performance. The NERFFT is related to the non-uniform FFT (NFFT or NUFFT). The NERFFT applies a window to the dechirped data prior to using the FFT such that the DTFT results can be obtained using only a small number of range-compressed samples (Fourmont, 2003; Capozzoli *et al.*, 2013). For completeness, details of the NERFFT are given in the Appendix.

Sub-aperture Processing

Another important function of a SAR processor is to provide multi-looking to reduce speckle. In multilook processing, sub-aperture images are created using subsets of the data partitioned by the azimuth angle or Doppler frequency. The subsets can be created using azimuth filters, but, as discussed in the next section, in backprojection processing it is more effective to apply azimuth windows. The final multi-looked image is created by power summing the sub-aperture images. Each sub-aperture image is effectively created with a narrower beamwidth than the physical antenna beamwidth resulting in a lower-resolution image. With the separate sub-aperture images formed from different portions of the phase history, the speckle statistics are independent for non-overlapping sup-apertures. Averaging the magnitude of the sub-aperture images reduces the image noise and makes it easier for users to interpret the images (Jakowatz *et al.*, 1996).

GPU Implementation

As shown in Equation 16, backprojection image formation can be independently computed for individual pixels. This makes it easy to parallelize the computation; however, simple parallelization does not lead to a real-time processor in most imaging scenarios. Because the UAS is constantly moving and the antenna is not steered, different areas are illuminated during each pulse. As a result, each pixel of the backprojection image only needs a subset of the collected pulses in its calculation, but each pixel needs a unique subset. Therefore, if not carefully implemented, the backprojection calculations are performed for pulses that do not contribute to a pixel's final value. The ability to compute the backprojection image in real-time is largely achieved by reducing unnecessary computations.

Overall Implementation

The structure of the backprojection processor is outlined in the pseudo-code shown in Algorithm 1 (see Appendix). Stripmap SAR collections frequently result in very long strip images. It is neither reasonable nor desirable to process a flight's collection as a single image. Therefore, the backprojection processing is broken into multiple images. To begin, we take a section of data of reasonable size resulting in an image that fits in the GPU memory. The image is oriented along the flight direction allowing efficient use of the GPU thread structure to break up the backprojection calculation.

Each section is broken into small groups or batches of pulses. A window is applied to the dechirped data to provide the desired range apodization and the NERFFT window. Then the FFT is computed using the NVIDIA CUFFT library on the GPU. Concurrently, the CPU is used to calculate the three-axis heading of the aircraft, simplifying the calculation of the azimuth angle that is used for the apodization window and multi-look processing. The azimuth bins that are visible during the batch are determined using the first and last antenna positions. This information is used in the backprojection kernel to reduce the number of calculations performed for pixels outside of the radar beamwidth during a batch. The size of the batch has considerable effect on the performance of the algorithm. A large batch reduces the number of global memory accesses, but a small batch reduces the number of pixels included in the backprojection calculations that receive negligible contribution from a pulse. Finding the optimal batch size for a given velocity, PRF, and beam-width is found with some experimentation. The backprojection kernel is then executed for each batch of pulses.

At the beginning of the backprojection kernel, the antenna positions are copied to the shared memory so that each block of threads only accesses the global memory for the radar positions once. Using shared memory for variables accessed by multiple threads can greatly reduce the kernel runtime. The backprojection kernel calculates the distance and angle from the pixel to each antenna position. The angle is used to calculate azimuth apodization windows for the full aperture and the sub-apertures. The distance is used to calculate the expected phase of each pulse's contribution and to interpolate the range-compressed data. To further reduce the runtime, local variables are used to accumulate the pixel contribution so that writes to global memory only occur once during each batch.

In the following we describe key portions of the proposed backprojection algorithm, including the interpolation and sub-aperture processing steps.

Interpolation

In the backprojection kernel the range-compressed data is interpolated for every contribution to the pixel. Thus, the performance of the interpolator is critical to the backprojection computation. Our criteria for selecting an interpolator is based on the accuracy and runtime performance. The selection of the interpolator affects other implementation tradeoffs, which can have dramatic effect on its overall runtime performance. In this section we compare the error of the interpolation methods directly, but only discuss the runtime performance in terms of computational complexity. We also present a method for accelerating nearly any interpolation method using the hardware linear interpolation on a GPU.

Accelerating Interpolation Using GPU Texture Cache

Sigg and Hadwiger (2005) demonstrate a method to accelerate cubic interpolation by casting linear interpolation as part of the cubic interpolation. We use this same concept to show how linear interpolation can be used within any convolutional interpolation scheme to accelerate the processing. The finite fractional precision of the GPU hardware linear interpolation incurs some accuracy loss; however, when used to accelerate the NERFFT, in many cases the resulting accuracy is greater than other common interpolation methods, including nearest neighbor, linear, and cubic interpolation.

We define a convolutional interpolation scheme by:

$$S_I(x) = \sum_{m=-K+1}^{K} S\lfloor x \rfloor + m\rfloor c_m(h) \tag{19}$$

where $h = x - \lfloor x \rfloor$, and the functions $c_m(h)$ are not necessarily linear. In the form of Equation 19, traditional linear interpolation has the coefficient functions

$$\begin{aligned} c_0(h) &= (1-h) \\ c_1(h) &= h \end{aligned} \tag{20}$$

Cubic spline interpolation can also be described in this form. In order to take advantage of the GPU hardware linear interpolation we rewrite Equation 19 with linear interpolation as an intermediate step. We begin by describing the summation in pairs as follows:

$$\begin{aligned} S_I(x) &= \sum_{m=-K+1}^{K} S[x+m]c_m(h) \\ &= \sum_{r=0}^{K-1} S[x+\nu]c_\nu(h) + S[x+\nu+1]c_{\nu+1}(h) \end{aligned} \tag{21}$$

where $\nu = 2r - K + 1$. With some manipulation, we then describe the pairs as a linear interpolation yielding:

$$S_I(x) = \sum_{r=0}^{K-1} C(h) S_L\left(\lfloor x \rfloor + \nu + B(h)\right) \tag{22}$$

where $B(h) = \frac{c_{v+1}(h)}{c_v(h) + c_{v+1}(h)}$ and $C(h) = c_v(h) + c_{v+1}(h)$. Equation 22 effectively applies the window weightings c_m for two memory accesses by shifting the h of the linear interpolation. Using the texture cache's linear interpolation, a pair of memory accesses is aggregated into one, effectively cutting the memory accesses in half and reducing the overall processing times.

Error Comparison

In order to create a SAR backprojection processor that appropriately balances the tradeoff of speed and accuracy, we need to select an interpolator that balances computational accuracy and complexity. Up to this point we have generally discussed the error of the interpolators. In this section we quantify the interpolation error of several interpolators for use on a GPU. In order to capture the effects of the GPU hardware interpolation and single precision arithmetic accurately, we use a Monte Carlo approach to compare the RMS error of the interpolation schemes previously discussed.

The goal of the interpolator is to provide both magnitude and phase accurate range-compressed samples at non-integer indices. The data is simulated as a band-limited signal using a Gaussian random number generator. The points at which the range-compressed data is interpolated are simulated with a uniform random number generator, which closely matches the distribution of range samples. We then pass the data and the points to each of the interpolators and calculate the RMS error compared to the DTFT of the data at the points. We average the results of multiple realizations so that the error statistics do not significantly change with more realizations.

Figure 3 shows the RMS interpolation error for the interpolators on a log-log plot with the RMS error in dB on the y-axis and the zero-padding factor on the x-axis. The error of the nearest neighbor, linear, and cubic spline interpolation methods drops log-linearly with the zero-padding factor each with slightly steeper slope, which equates to lower error. The reduction in error for these methods is due to higher zero-padded data being smoother. In contrast the NERFFT error drops quickly with small zero-padding factors and then trends to leveling out. Thus, the motivation for higher zero-padding factors is greatly reduced when using the NERFFT.

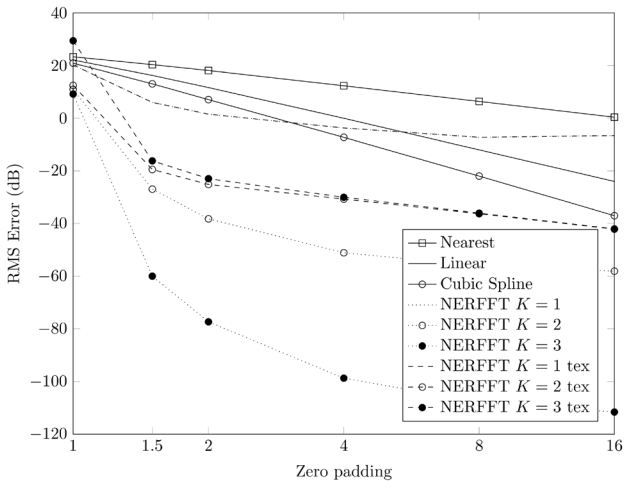

Figure 3. RMS error for different range-compressed data interpolation methods with several zero-padding factors. There are two versions of the NERFFT used; the ones denoted "tex" are accelerated using hardware linear interpolation.

With wider windows (higher K), the NERFFT is considerably more accurate. We note that the number of memory accesses for $K = 1$ and $K = 2$ is equivalent to linear and cubic interpolation, respectively. The $K = 1$ NERFFT has lower error than linear at zero-padding factors less than 4; however, the NERFFT window applied to the raw data increases the quantization noise introduced during the FFT which limits the achievable accuracy gains for the NERFFT with higher zero padding.

Two versions of the NERFFT are used in Figure 3, the latter of them being accelerated using the hardware linear interpolation. The accelerated versions are denoted "tex" in the legend. For the $K = 1$ NERFFT accelerated results are identical. The accelerated $K = 2$ NERFFT only has slightly increased error compared to the non-accelerated version, but the accelerated $K = 3$ NERFFT has slightly higher error than the $K = 2$ NERFFT. This demonstrates the limited accuracy available from the texture cache's linear interpolation. Note that the NERFFT $K = 2$ case is of the same computational complexity as the cubic spline and has dramatically lower error. Using the NERFFT both reduces the error and lowers the memory storage requirements.

NERFFT Implementation

Using the results of the error analysis given in Figure 3, we select the accelerated NERFFT $K = 2$ for the range compression because of its high accuracy, lower memory usage, and potentially lower computational cost. In order to optimally take advantage of the GPU resources described in the background, transcendental functions and memory accesses need to be reduced when possible. In the GPU implementation of the NERFFT described in (Capozzoli *et al.*, 2013) each NERFFT access requires $2K$ memory accesses and $2K$ evaluations of the hyperbolic sine and square root operations. We reduce the required computations and increase the speed by approximating the combined Bessel functions $C(h)$ and $B(h)$ shown in Equation 22 using polynomials.

Sub-aperture Processing

Fasih and Hartley (2010) suggested that sub-aperture processing can be achieved using the implementation used for full-resolution processing by running the backprojection kernel multiple times with different sections of the data. While such a method keeps the code cleaner, it is more efficient to calculate all of the sub-aperture images within the same thread. Calculating the sub-apertures within the same thread avoids duplicate memory accesses and recalculations for overlapping sub-apertures. The inclusion of the sub-apertures in the processing adds another memory layout choice. The sub-aperture image pixels can either be assigned as discrete images or one conglomerate image with the sub-apertures for each pixel occupying a contiguous slot in memory. In our experiments, we find that with a significantly large batch size and a small number of sub-apertures, either layout performs equally well because the writing of the image pixels only occurs once during each batch.

Results

To demonstrate the performance of the GPU backprojection implementation, we use SAR data collected as part of the Characterization of Arctic Sea Ice Experiment (CASIE) (Long *et al.*, 2010). The CASIE SAR data was collected using the microASAR, a small, LFM-CW, C-band SAR system on board the NASA SIERRA UAS (Zaugg *et al.*, 2010; Stringham *et al.*, 2011). During the CASIE mission the microASAR operated with a 160 MHz bandwidth and an 11 degree beamwidth resulting in approximately 1 m × 15 cm single-look resolution. To improve the visual quality of the image we reduce the speckle using seven sub-apertures with 50 percent overlap. The resulting images have 50 cm × 50cm pixel spacing. For details about the CASIE imagery

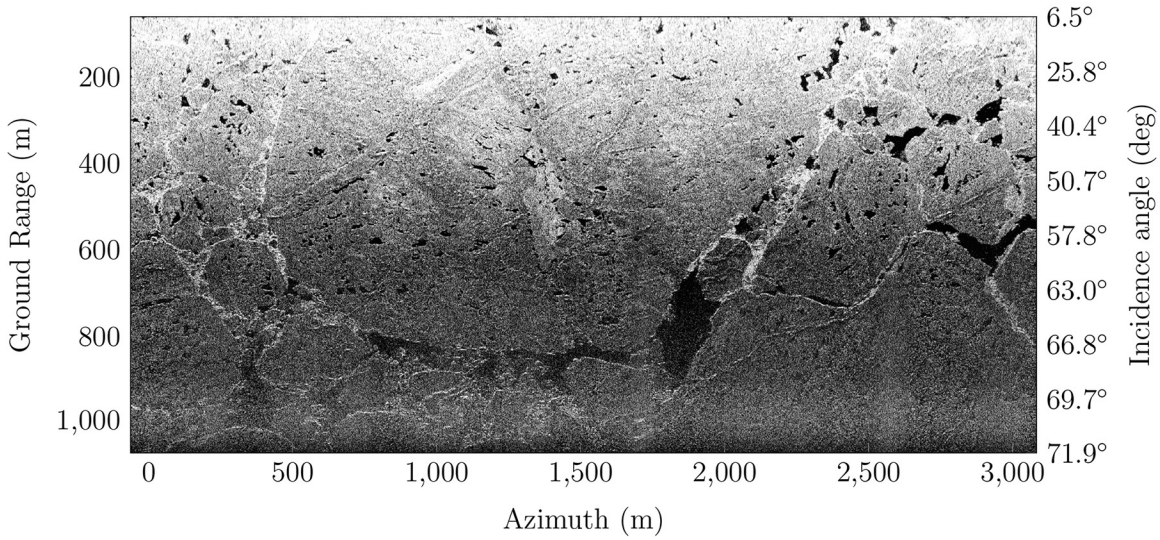

Figure 4. An example image of sea ice processed with the GPU SAR backprojection. The left y-axis specifies the ground range in meters, and the right y-axis is the incidence angle.

(a) (b) (c)

Figure 5. An example of the improvement made using the compensated backprojection: (a) shows the uncompensated image, (b) shows the weighting function, and (c) shows the compensated image.

and for sample data see Long *et al.* (2010), Zaugg *et al.* (2010), Stringham and Long (2011), and Long and Stringham (2012).

Figure 4 is an example of the imagery produced using the backprojection implementation described in this paper. The data was collected over sea ice in the Arctic Ocean north of Svalbard Norway. The UAS travels along the top edge from left to right. The low-altitude of this image results in a very wide range of incidence angles as illustrated on the right side of the image, and the backscatter roll-off due to incidence angle can be seen in the far range.

To illustrate the effectiveness of the compensated backprojection, a portion of an image created during a maneuver is shown in Figure 5. In this image the UAS is finishing a climb from an altitude of 270 m to 350 m. As with any maneuver, there are significant attitude changes which are typically unaccounted with traditional SAR processing methods. In contrast to Figure 5a, the compensated image in Figure 5c has only a slight residual variation.

The image shown in Figure 4 was processed using a 2008 Mac Pro desktop equipped with 2–3.2 GHz quad-core Intel Xeon processors, 16 GB of memory, and a NVIDIA GTX 285 Video card. The image collection time is 100 sec. Including the time to load the data and write the image to disk, the backprojection processing takes 85 sec using nearest neighbor interpolation without the texture cache, 70 sec for the $K = 2$ NERFFT implementation, and 60 sec using the $K = 1$ NERFFT implementation. Using the GPU's texture cache to accelerate the interpolation, the NERFFT requires only one memory access in the $K = 1$ case and two in the $K = 2$ case. The runtime for both NERFFT implementations is better than the nearest neighbor implementation which only has one memory access. The improved runtime results from using the texture cache, which significantly reduces the latency of the memory accesses. We note that the GPU implementation of the processing is faster than real-time.

Conclusions

Using small LFM-CW SAR on unmanned air systems can provide unprecedented information for scientific and military missions. The low-altitude geometry and significant motion typical of small UAS operation requires significant motion compensation to generate high quality imagery. This work

presents an efficient backprojection processor that provides excellent motion compensation for UAS-based stripmap SAR using consumer-grade graphics processing units. We developed the stripmap backprojection algorithm from spatial coordinates accounting for the un-steered antenna and for the motion of aircraft during the long duration of an LFM-CW pulse. The processor takes advantage of the unique processing hardware of the GPU to produce quality images faster than real-time for the CASIE mission. This work shows that SAR backprojection can be used effectively in real-time scenarios from a UAS.

For those interested in working with UAS SAR data, we provide a small sample of the CASIE data set with basic processing scripts on-line (Long and Stringham, 2012).

Acknowledgments

This work was supported, in part, under the Marginal Ice Zone Observations and Processes Experiment (MIZOPEX) by NASA.

APPENDIX

NERFFT

The NERFFT reconstructs a non-equispaced Fourier transform using a windowed and zero-padded discrete Fourier transform (Fourmont, 2003; Capozzoli *et al.*, 2013). The key principle of the NERFFT is that the complex exponential in the DFT summation can be rewritten as:

$$e^{-ix\omega} = \frac{1}{\sqrt{2\pi}\phi(x)} \sum_{m\in\mathcal{Z}} \hat{\phi}(\omega - m) e^{-im\omega} \tag{A1}$$

where ϕ and $\hat{\phi}$ are a window function and its Fourier transform, respectively, and $\mathcal{Z}$ is the support of $\hat{\phi}$. Using Equation A1, the interpolated range-compressed data can be described as:

$$\begin{aligned} S(\omega) &= \frac{1}{\sqrt{2\pi}} \sum_{m\in\mathcal{Z}} \hat{\phi}\left(\omega - \frac{2\pi m}{cN}\right) \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} e^{-im\frac{2\pi k}{cN}} \frac{s[k]}{\phi\left(\frac{2\pi k}{cN}\right)} \\ &= \frac{1}{\sqrt{2\pi}} \sum_{m\in\mathcal{Z}} \hat{\phi}\left(\omega - \frac{2\pi m}{cN}\right) S_c\left[\frac{\omega cN}{2\pi} + m\right] \end{aligned} \tag{A2}$$

where S_c is the FFT of $\frac{s[k]}{\phi(\frac{2\pi k}{cN})}$ with a zero-padding factor c. A proof of Equation A2 is given in (Fourmont, 2003). To illustrate, let ϕ be a rect window so that $\hat{\phi}$ is the Dirichlet kernel. Equation A2 then yields Dirichlet interpolation. Note that the summation over k in Equation A2 is centered around 0 denoting a centered FFT. This both simplifies the development and results in both ϕ and $\hat{\phi}$ being real, which reduces the number of multiplies required; however, most software routines for the FFT perform the uncentered FFT. The uncentered FFT can be performed by circularly shifting the input data.

In Equation A2, a Kaiser-Bessel window is used to concentrate the energy of the signal into a short summation over m. The Kaiser-Bessel window and its Fourier transform are given by:

$$\begin{aligned} \phi(x) &= \frac{I_0(K\sqrt{\alpha^2 - x^2})}{I_0(K\alpha)}, \\ \hat{\phi}(\omega) &= \begin{cases} \sqrt{\frac{2}{\pi}} \frac{\sinh\left(\alpha\sqrt{K^2 - \omega^2}\right)}{\sqrt{K^2 - \omega^2} I_0(K\alpha)}, & \omega \le \alpha 0, \quad \omega > \alpha, \end{cases} \end{aligned} \tag{A3}$$

where

$$\alpha = \pi(2 - 1/c) - 0.01, \tag{A4}$$

and $2K$ is the length of the support of $\hat{\phi}$. It has been shown by Capozzoli *et al.* (2013) and Fourmont (2003) that in the case $c = 2$, a six point window, $K = 3$, can reduce the interpolation error down to the precision of single floating point arithmetic. In the paper, we compare several interpolation methods with a range of zero-padding factors in order to navigate the tradeoffs of memory, speed, and accuracy of the interpolation stage.

Algorithms

Algorithm 1: Pseudo-code for GPU stripmap SAR backprojection

```
Break data collection into sections
For each section
  Allocate memory for the image section
  Allocate memory for a batch of pulses
  For each batch of pulses
    Copy batch of pulses to GPU memory
    Apply apodization window
    Apply NERFFT window
    Compute FFT
    Rotate position data
    For each thread group
      Copy position and attitude to shared memory
      Call backprojection kernel
      For each thread
        For each pulse in batch
          Calculate distance and angle
          Calculate expected phase
          Interpolate range-compressed
          For each subaperture
            Calculate azimuth apodization
            Multiply sample by apodization and phase
            Accumulate the results
            Accumulate the apodization and antenna weights
  Copy the image and apodization weights from the GPU
  Save the complex image
  Average the power of the pixels
  Divide by weightings
  Save the multi-looked image
```

References

Benson, T., D. Campbell, and D. Cook, 2012. Gigapixel spotlight synthetic aperture radar backprojection using clusters of GPUs and CUDA, *Proceedings of the Radar Conference (RADAR), 2012 IEEE*, IEEE, pp. 0853–0858.

Capozzoli, A., C. Curcio, and A. Liseno, 2013. Fast GPU-based interpolation for SAR backprojection, *Progress In Electromagnetics Research*, 133:259–283.

Cumming, I., and F. Wong, 2005. *Digital Processing of SAR Data*, Artech House, Norwood, Massachusetts.

Curlander, J.C., and R.N. McDonough, 1991. *Synthetic Aperture Radar: Systems and Signal Processing*, Wiley, New York.

Fasih, A., and T. Hartley, 2010. GPU-accelerated synthetic aperture radar backprojection in CUDA, *Proceedings of the Radar Conference, 2010 IEEE*, IEEE.

Fourmont, K., 2003. Non-equispaced fast Fourier transforms with applications to tomography, *Journal of Fourier Analysis and Applications*, 9, 431–450.

Frey, O., C. Magnard, M. Ruegg, and E. Meier, 2009. Focusing of airborne synthetic aperture radar data from highly nonlinear flight tracks, *Transactions on Geoscience and Remote Sensing*, IEEE, 47:1844–1858.

Frey, O., M. Santoro, C.L. Werner, and U. Wegmuller, 2013. DEM-based SAR pixel-area estimation for enhanced geocoding refinement and radiometric normalization, *Geoscience and Remote Sensing Letters*, IEEE 10:48–52.

Jakowatz, C.V., D.E. Wahl, P.H. Eichel, D.C. Ghiglia, and P.A. Thompson, 1996. *Spotlight-Mode Synthetic Aperture Radar: A Signal Processing Approach*, Springer.

Long, D.G., and C. Stringham, 2012. Sample BYU microASAR data, URL: *http://www.mers.byu.edu/microASAR/CASIE_sample/* (last date accessed: 30 September 2014).

Long, D.G., E. Zaugg, M. Edwards, and J. Maslanik, 2010. The microASAR experiment on CASIE-09, *Proceedings of the Geoscience and Remote Sensing Symposium (IGARSS), 2010*, IEEE International, pp. 3466–3469.

Melvin, W., and J. Scheer, 2012. Principles of modern radar: Advanced techniques, *Principles of Modern Radar*, SciTech Publishing.

Meta, A., P. Hoogeboom, and L. Ligthart, 2007. Signal processing for FMCW SAR, *Transactions on Geoscience and Remote Sensing*, IEEE, 45:3519–3532.

Moon, K., and D.G. Long, 2013. A new factorized backprojection algorithm for stripmap synthetic aperture radar, *Positioning 4*.

Moreira, A., and Y. Huang, 1994. Airborne SAR processing of highly squinted data using a chirp scaling approach with integrated motion compensation, *Transactions on Geoscience and Remote Sensing*, IEEE, 32:1029–1040.

Nguyen, L., M. Ressler, D. Wong, and M. Soumekh, 2004. Enhancement of backprojection SAR imagery using digital spotlighting preprocessing, *Proceedings of the Radar Conference*, IEEE, pp. 53–58.

Pfitzner, M., F. Cholewa, P. Pirsch, and H. Blume, 2013. FPGA based architecture for real-time SAR processing with integrated motion compensation, *Proceedings of the 2013 Asia-Pacific Synthetic Aperture Radar Conference (APSAR)*, 2013, pp. 521–524.

Ribalta, A., 2011. Time-domain reconstruction algorithms for FMCW-SAR, *Geoscience and Remote Sensing Letters*, IEEE 8:396–400.

Sigg, C., and M. Hadwiger, 2005. Fast third-order texture filtering, GPU Gems, 2:313–329.

Stevens, D., I. Cumming, and A. Gray, 1995. Options for airborne interferometric SAR motion compensation, *Transactions on Geoscience and Remote Sensing*, IEEE, 33:409–420.

Stringham, C. D. Long, B. Wicks, and G. Ramsey, 2011. Digital receiver design for an offset IF LFM-CW SAR, *Proceedings of the 2011 Radar Conference*, IEEE.

Stringham, C., and D.G. Long, 2011. Improved processing of the CASIE SAR data, *Proceedings of the 2011 Geoscience and Remote Sensing Symposium (IGARSS)*, IEEE International.

Ulander, L.M., H. Hellsten, and G. Stenstrom, 2003. Synthetic-aperture radar processing using fast factorized back-projection, *Transactions on Aerospace and Electronic Systems*, IEEE, pp. 760–776.

Wong, H., M.-M. Papadopoulou, M. Sadooghi-Alvandi, and A. Moshovos, 2010. Demystifying GPU microarchitecture through microbenchmarking, *Proceedings of the International Symposium On Performance Analysis of Systems & Software (ISPASS)*, IEEE, pp. 235–246.

Wu, Y., J. Chen, and H. Zhang, 2012. A real-time SAR imaging system based on CPUGPU heterogeneous platform, *Proceedings of the2012 IEEE 11th International Conference on Signal Processing (ICSP)*, pp. 461–464.

Zaugg, E., and D. Long, 2007. Full motion compensation for LFM-CW synthetic aperture radar, *Proceedings of the 2007 Geoscience and Remote Sensing Symposium, IGARSS*, IEEE International, pp. 5198–5201.

Zaugg, E., D. Long, M. Edwards, M. Fladeland, R., Kolyer, I. Crocker, J. Maslanik, U. Herzfeld, and B. Wallin, 2010. Using the microASAR on the NASA SIERRA UAS in the characterization of Arctic sea ice experiment, Proceedings of the 2010 Radar Conference, IEEE, pp. 271–276.