High Performance SAR Focusing Algorithm and Implementation

Claudio Passerone, Claudio Sansoè, Riccardo Maggiora
Department of Electronics and Telecommunications
Politecnico di Torino
Torino, Italy

***Abstract*—Efficient and fast processing of spaceborne Synthetic Aperture Radar (SAR) data is becoming increasingly important in many applications, such as surveillance and disaster recovery. Moreover, constellations of satellites generate a huge amount of data, and standard toolchains may not cope well with them. Dedicated architectures have been used in the past, but they are hard to maintain from both the hardware and software points of view. In this paper, we present SAR focusing algorithms for the COSMO-SkyMed constellation of satellites, based on established techniques, implemented using commercial massively parallel architectures: they allow an easy development cycle, while at the same time enjoying better performance with new technology advancements.**

TABLE OF CONTENTS

1. INTRODUCTION

The space borne Synthetic Aperture Radar (SAR) system provides an active all-weather, day and night, penetrating, remote sensing mean and produces high resolution complex (amplitude and phase) images of the land under illumination of radar beam. Unlike optical sensors, the SAR system needs a post-processing procedure on the data acquired to form the final image [1].

For decades, Synthetic Aperture Radar (SAR) imaging from satellites techniques have been used to represent radar data in a way that is meaningful to a human analyst. The calculations necessary for these imaging techniques are computationally complex, and since the advent of SAR, computational throughput has always been a critical factor in the development, implementation, and use of processing algorithms.

In a few words SAR processing is a computationally intensive technique which is used for creating images from radar signals gathered by a moving platform such as an aircraft. By combining signals gathered from multiple points in space (multiple angles of azimuth and elevation), higher resolution images can be constructed without needing a larger physical antenna. The computational load increases with the image size and the amount of data, and so techniques for accelerating the processing of the input radar signals and generating the output images are necessary to be able to process the large amount of data in real time [2]. Our goal is real time processing of Cosmo-SkyMed satellites data which means that the volume of data which SAR processors have to deliver necessitates very fast and very efficient processing mainly to enable fast decision making.

With the continuous advance of computational power provided by CPUs and DSPs, imaging routines have become faster and faster over the years. Chip makers, however, have reached a clock-speed plateau, as processors with increasingly high transistor densities are no longer able to dissipate the heat associated with increasingly high clock speeds. This heat dissipation challenge has led to the recent trend away from ever-increasing clock speeds and instead has resulted in an explosion of multi-core, commercial-grade processors. Multicore processors offer the potential for extremely high throughput, with the ability to achieve such results depending greatly on the nature of the computations being performed [3].

Due to the rapid growth of the computational capacity of Graphics Processing Units (GPUs) over the past decade, researchers are increasingly using these emerging architectures to accelerate high performance applications. GPUs are characterized by hundreds of self-sufficient but simple processors, each with several execution units, and support high computational throughput for data-parallel algorithms. Following high-quality software interfaces for general purpose graphics processor programming, such as NVIDIA CUDA released in 2007, numerous studies have demonstrated the tremendous benefit of using GPUs as accelerators for tasks decomposable to the same operations performed on different chunks of data [4].

2. RELATED WORK

To the best of authors knowledge, interesting preliminary works on SAR data from satellites processors based on GPU platform equal or very similar to the one here adopted can be found in [5], [6] and [7]. These implementations have been tested on relatively small sample images and, if scaled, demonstrate comparable computational time with respect to the presented solution.

It is worth noting that our solution, as stated in the previous chapter, is dedicated to the processing of COSMO-SkyMed satellites data and has been tested on real COSMO-SkyMed sample data. Another important peculiarity of our solution

978-1-4799-1622-1/14/$31.00

resides in the Doppler parameters estimation directly from the received SAR data that resulted to be crucial for obtaining even an acceptable image quality.

3. COSMO-SkyMed and focusing algorithms

COSMO-SkyMed (COnstellation of small Satellites for Mediterranean basin Observation) is the largest Italian investment in Space Systems for Earth Observation, commissioned and funded by Italian Space Agency (ASI) and Italian Ministry of Defense (MoD), and it is *natively* conceived as a Dual-Use (Civilian and Defence) end-to-end Earth Observation System aimed to establish a global service supplying provision of data, products and services compliant with well-established international standards and relevant to a wide range of applications, such as Risk Management, Scientific and Commercial Applications and Defence/Intelligence Applications. The system consists of a constellation of four Low Earth Orbit mid-sized satellites, each equipped with a multi-mode high-resolution Synthetic Aperture Radar (SAR) operating at X-band and fitted with particularly flexible and innovative data acquisition and transmission equipment [8]. The constellation average daily acquisition capability of 1800 images acquired in a 24 hours moving window (75 Spotlight plus 375 Stripmap or ScanSAR for each satellite) drives the necessity to drastically improve the efficiency of SAR image formation especially in view of the commissioning of the second COSMO-SkyMed generation satellites. The COSMO-SkyMed sensor imaging operating modes considered in this work are the stripmap and the spotlight.

The Stripmap mode, which is the most common imaging mode, is obtained by pointing the antenna along a fixed direction with respect to the satellite flight path. The antenna footprint covers a strip on the illuminated surfaces as the satellite moves and the system operates. The acquisition is virtually unlimited in the azimuth direction, except for the limitations deriving from the SAR instrument duty cycle (about 600 s, allowing a strip length of 4500+ km). In this mode, the radar Tx/Rx configurations are time invariant, allowing receiving from each ground scatterer the full Doppler bandwidth allowed by the azimuth aperture of the antenna beamwidth. The stripmap mode is characterized by a swath width of about 40 km, an azimuth extension for the standard product (square frame) of about 40 Km (corresponding to an acquisition of about 6.5 s), Pulse Repetition Frequency (PRF) values ranging from a minimum of 2905.9 Hz to a maximum of 3874.5 Hz, a chirp duration in the range [35, 40] μs, a chirp bandwidth accommodated along range on the basis of the required ground resolution (typically equal to 3 m), spanning from 65,64 MHz at the far range (with a sampling rate of 82.50 MHz) to 138,60 MHz at the far range (with a sampling rate of 176.25 MHz).

The spotlight mode, in which the antenna is steered (both in the azimuth and the elevation plane) during the overall acquisition time in order to illuminate the required scene for a time period longer than the one of the standard stripmap, increasing the length of the synthetic antenna and therefore the azimuth resolution (at expense of the azimuth coverage), is characterized by an azimuth frame extension of about 11 km, a range swath extension of about 11 Km, PRF values ranging from a minimum of 3148.1 Hz to a maximum of 4116.7 Hz, allowed chirp duration in the range [70, 80] μs (depending on specific access area), a chirp bandwidth accommodated along range on the basis of the required ground resolution (typically equal to 1 m) ranging from 185.2 MHz to 400.0 MHz and finally a sampling frequency equal to 187.5 MHz.

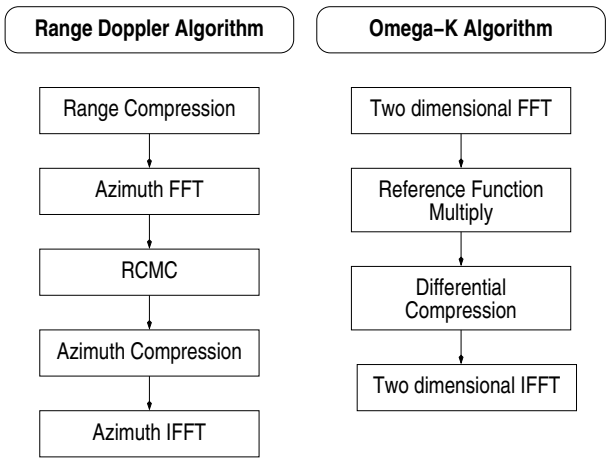


Figure 1. RDA and ω-K flows

Several algorithms can be used to process COSMO-SkyMed Stripmap and Spotlight data. In this work, the Range-Doppler Algorithm (RDA) and the ω-k algorithm have been implemented. They are both based on the compression of linearly frequency modulated signals (chirp) along the range and azimuth directions, using a matched filter in the frequency domain. While the chirp in the range direction is generated by the radar antenna itself, the chirp along the azimuth direction appears due to the Doppler effect (synthetic aperture), caused by the change of relative speed between the radar and the targets. As illustrated in Figure 1, the two algorithms differ in the sequence of operations, with the RDA needing an additional Range Cell Migration Correction (RCMC) step, that is taken care by the reference function mutliplication in the ω-k algorithm. Section 5 gives a more in-depth description of the algorithms from the point of view of the implementation; for more on details theoretical aspects, see [1].

4. GPUs and CUDA

A modern Graphic Processing Unit (GPU) can be used as a General Purpose computing device that acts as an accelerator for certain applications, with respect to the normal Central Processing Unit, i.e. a microprocessor core (or multi-core). The accelerator looks like an Input/Output unit to the rest of the system, so it communicates using I/O commands and DMA memory transfers.

Internally a GPU consists of many parallel processor cores, in the order of several hundreds, or even up to some thousands, of a (usually large) memory, and wide interconnection busses to improve data transfer performance. In our project we used an nVidia Tesla C1060 GPU ([9]), which has 240 parallel cores and 4 GByte of own memory (called the *device* memory, to distinguish it from the main memory of the host computer).

The parallel cores are grouped into several *multiprocessors*, with each of them having also a small internal SRAM, shared among all cores belonging to it, used to cache the large device memory or to enable fast communication between cores. This memory is called *shared* memory, and efficiently exploiting it

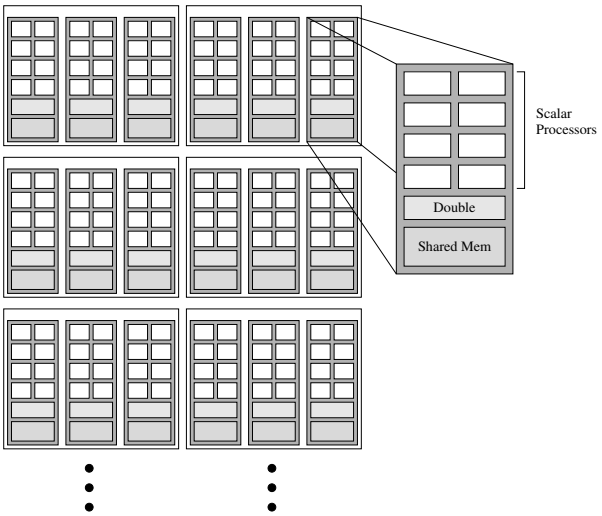


Figure 2. Tesla C1060 internal block diagram

often has a huge favorable impact on performance. Figure 2 shows a block diagram of a GPU architecture: in a Tesla C1060 each multiprocessor contains 8 parallel cores and an additional double precision unit, while newer architectures have 16 cores, that can be combined in pairs to perform 64-bit floating point operations.

The execution model is based on threads, that are run by each core. Within a multiprocessor, all cores execute the same thread on different data, much like a Single Instruction Multiple Data (SIMD) parallel machine, but data dependent execution is handled in a slightly different way. Threads can be suspended, maybe because they make a slow memory access, and other thread executed, with the context switch taking basically no time. Thus, by having multiple concurrent threads, the computational resources can be effectively exploited.

The programming model, called CUDA (Compute Unified Device Architecture), is based on the execution of *kernels*, each divided into a number of threads, which are further grouped into blocks within a user definable grid. A programmer has to identify those parts of a program that could be best implemented on the accelerator, by exposing the parallelism to efficiently exploit the architecture. Kernels are written in C language, but the same C description cannot be compiled as is for the host processor, so there is an additional effort in porting an application to use a GPU accelerator.

Fortunately, GPU manufacturers often provide a set of high level library functions that are carefully optimized for the hardware accelerator, and implement many common algorithms found in computing intensive applications. For instance, in our case we took advantage of the CUDA cuFFT library, that provides Fast Fourier Transform (FFT) functions for 1-, 2- and 3-dimensional arrays. However, for application specific computation, dedicated parallel kernels were developed.

5. IMPLEMENTED TOOLS

Two kind of processor were implemented, one using the Range-Doppler algorithm, the other with the ω-k algorithm. Both algorithms were briefly presented in Section 3 from a theoretical point of view; here their parallel implementation on the GPU hardware is analysed, with additional details on the low level choices in algorithm implementation.

All tools have been applied to raw data from the Cosmo-SkyMed constellation, and the output is in the Slant-range Complex Single-look (SCS) format. Ground projection, as well as higher level products were also implemented, but are outside the scope of this paper. Both the raw data and the outputs are in the Hierarchical Data Format, Version 5 (HDF5, see [10] for details), and a freely available library has been used for reading and writing files.

The tools also support data in CEOS raw format ([11]), with only small differences in the loading and writing phases with respect to the HDF5 version. Simple procedures for reading and writing files have been implemented, but details are omitted here.

All the following discussions assume that enough memory is available on the GPU hardware to hold an entire image and the necessary temporary buffers. If this is not the case, slicing of the image in the range and/or azimuth directions is required, with decreased performance due to the overhead for data transfers between the host and device memories. The image is also zero-padded in both the range and azimuth directions, to make the two lengths multiples of the four prime numbers 2, 3, 5 and 7; this is to ensure that the FFT functions used operate at the best of their performance.

Range-Doppler Algorithm (RDA)

As discussed previously, the Range-Doppler algorithm performs two separate compressions in the range direction and in the azimuth direction, with the additional Range Cell Migration Correction performed in the doppler domain. Key to the accuracy and resolution of the final result is the correct estimation of several parameters, like the doppler frequency and the azimuth FM rate. As these values are not constant within an image, polynomial models have been used to characterize their variations, and the model parameters are directly derived from the image data itself.

The input to the algorithm is a $W \times H$ image with complex numbers in 32-bit floating point format (i.e. 8 bytes for each pixel, 4 for the real part, and 4 for the imaginary part). This image corresponds to the samples received by the SAR antenna for each of the H transmitted pulses, with the high frequency carrier already removed on-board the spacecraft. Many additional ancillary data are also present in the image file, and some of them are extracted for the subsequent processing steps.

Among the available data are biasing and antenna pattern values, that allow to correct the received samples to get a higher signal to noise ratio. Both corrections are carried out using dedicated parallel kernels that operate on pixel values directly.

Range Compression—The first step of the algorithm consists in the convolution of each range row with a matched filter, that depends upon the chirp used during signal transmission. Chirp samples are contained in the raw image data, and are extracted during the loading phase and stored into a one dimensional array. Convolution is performed in the frequency domain, as the complex product of the spectrum of each range scan line with the conjugate of the spectrum of the chirp. So for each line, an FFT, a complex multiplication, and an inverse FFT are required.

The Nvidia CUDA CuFFT library already provides highly optimized functions to perform Fast Fourier Transforms. The library, which is extensively used throughout the remaining of the algorithm, requires the initial definition of a plan, which contains all the relevant information for the transformation, such as the number of samples and the direction of the transformation. An additional parameter is the number of concurrent FFTs to perform, called `batch`: as computation is parallelized over hundreds of cores, increasing the opportunities to exploit parallelism is very important to improve performance. In practice, it turns out that an 8192 sample FFT takes roughly the same amount of time as a batch of 8 concurrent 8192 sample FFTs; higher values for the batch parameter lead to a linear increase in the running time.

Therefore, in our implementation three plans were defined: two of them have the batch parameter equal to H, and differ only for the direction of the transformation (forward and inverse, respectively). The third one is a forward FFT of a single line, which is applied to a zero padded version of the chirp, extended to the same size of a range scan line.

The entire image is initially forward transformed with a single call of the FFT library function. As the raw data is not needed anymore, the transformation is in-place to save memory. Then the complex multiplication takes place: a dedicated pointwise kernel has been developed to perform it in parallel. Finally, a second in-place FFT, this time in the inverse direction, is applied to the whole image to return to the spatial domain. The original image is thus lost, and only the range compressed image is kept for the subsequent processing steps.

Doppler parameters estimation—The synthetic aperture radar concept is based on the Doppler effect that changes the frequency of the transmitted signals as the spacecraft moves with respect to ground targets. A good knowledge of doppler paramteres is therefore of paramount importance to achieve the highest possible resolution along the azimuth direction.

In principle, doppler parameters can be computed from the relative geometry of the radar and target system. However, while this is a common practice for airborne SARs, the resulting accuracy for a satellite is often not adequate enough, as not only the satellite speed and attituted should be precisely known, but also Earth rotation and curvature should be taken into account. Hence, estimation of the doppler parameters from the received data is required.

Two main parameters are estimated, using three distinct procedures, that share many similar processing steps:

- Doppler centroid: the doppler frequency for a ground target when the satellite is at closest approach. This value is 0 for a SAR with an antenna perfectly orthogonal to the flight path, and it is non-zero otherwise. The estimation is divided in two phases: first, the fractional part of the doppler frequency with respect to the Pulse Repetition Frequency (PRF) is computed, then the integer part, also known as the *ambiguity* number M_{amb} is estimated.
- Azimuth FM rate: the rate of change of the doppler frequency along the azimuth direction. This value is necessary to derive the correct matched filter for the subsequent azimuth compression step.

In all cases, the estimated values vary slowly with the image location. Therefore, for the estimation process the image itself is divided into several contiguous small blocks, parameters are computed for each individual block, and a smooth polynomial fitting surface is then obtained using the Nelder-Mead method ([12]), initialized with data obtained from geometical and orbital parameters.. This procedure has the added benefit that any error in the estimation process on a block is averaged with the other blocks, resulting in better accuracy.

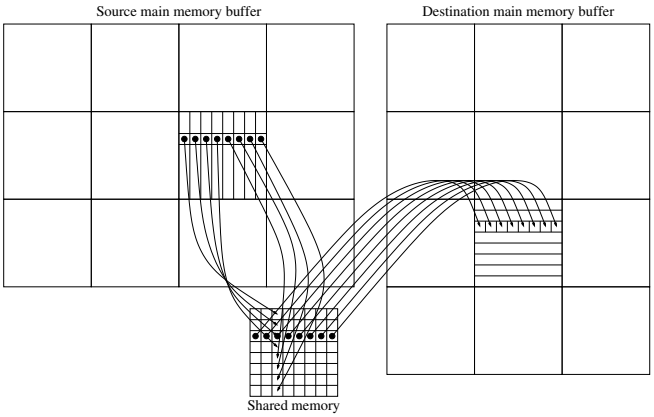


Figure 3. Matrix transposition with memory coalescing in reading and writing

Most computations in the estimation processes are carried out in the range-doppler domain, i.e. in the spatial domain for the range direction, and in the frequency domain for the azimuth direction. Unfortunately, FFT functions in the CUDA library do not provide a *stride* parameter, so the samples of the arrays to be transformed need to be contiguous in memory. Transposition of individual blocks, or of the entire image, is hence needed before applying the FFT.

In order to be efficient, parallel matrix transposition should exploit the memory access capabilities of the GPU hardware. In particular, many main memory transactions should be grouped, or *coalesced*, in a single wider transaction to optimize access time. In a straightforward parallel implementation of transposition, each thread reads one piece of data from the source matrix, and writes it to the destination buffer. However, due to the matrix memory layout, coalescing is possible either while reading data, or while writing it, but not in both cases at the same time.

An alternative implementation uses shared memory to accomplish the same task, while exploiting coalescing in main memory in both reading and writing. Threads are divided in an $n \times n$ matrix: each thread reads a value from an $n \times n$ block in the source image, and stores it transposed in a temporary buffer in shared memory. Shared memory is much faster than main memory, and coalescing is not needed, while reading from main memory enjoys coalescing because threads reads contiguous data. Once the temporary buffer is completely filled in, the destination main memory image is written. The destination block is in a transposed location with respect to the source block, and threads are organized to store data contiguous in memory, thus achieving coalescing also in writing. The trick is that a thread reads a value, but writes another one read by the transposed thread in the matrix. Figure 3 shows an example for the block at location (2,1) with $n = 8$, and with the details for the threads of the third row: they all read contiguous data, and then write contiguous data as well.

Having covered some of the common functions used in the doppler parameters estimation, the three individual procedures can be analysed in more details:

Fractional Doppler centroid estimation: the range compressed image is divided into a number of blocks, with a user defined size. For each block, the Doppler centroid fractional part is estimated as the peak of the Doppler magnitude spectrum. As the spectrum for a single azimuth line in a block is very noisy, the magnitude is averaged over all lines of the block; then, its maximum is found by convolving it with a power filter with a sinusoidal shape and by looking for the zero crossing. The detailed steps for each block are the following:

1. The power filter is generated and its Fourier transform computed[2].
2. The block is transposed to get the azimuth direction in the rows, and then rows are transformed to the frequency (doppler) domain using FFTs in batches equal to the block height.
3. The magnitude of each azimuth line is computed, and the average of all lines gives the doppler power specturm. In both cases, pointwise parallel kernels were developed to increase performance.
4. The power spectrum is further transformed using a forward FFT, multiplied with the power filter transform to perform the convolution, and finally inverse transformed to get back to the doppler domain.
5. The zero crossing of the convolution is determined on the host, as this computation is not time critical because it involves a single line.

Once all blocks are completed, a first order polynomial surface is fitted to the fractional centroid values, such that the final outcome of the procedure is given by:

$$d_{\mathrm{frac}} = d_0 + d_1 r + d_2 a + d_3 r a$$

where r and a are coordinates on the range and azimuth directions, respectively. A second order polynomial model can also be used, with very few modifications and a slightly higher running time. Before fitting, the signal to noise ratio of the average power spectrums are also computed, and potentially inaccurate values are removed.

Azimuth FM rate estimation: again, the range compressed image is divided into user defined blocks, possibly with a different size with respect to the fractional doppler centroid estimation procedure. Each block is transformed in the doppler domain, and decomposed into two *looks* with non-overlapping spectrum. In turn, each look is separately compressed along the azimuth axis using a nominal FM rate K_a derived from orbital data. In case the nominal FM rate is not precise enough, it results in a mis-registration of the two compressed looks, and the amount of mis-registration gives the error on the FM rate. Averaging over all range lines in the block allows to reduce noise and achieve a better accuracy. The implemented steps are thus as follows:

1. The block is transposed to get the azimuth direction in the rows, and then rows are transformed to the frequency (doppler) domain using FFTs in batches equal to the block height[3].
2. The block is compressed along the azimuth direction, using a matched filter derived from the nominal FM rate. Compression is obtained by the complex multiplication with the transformed matched filter, and the result is kept in the doppler domain at this stage. Notice that the matched filter is computed for each azimuth line separately, as the coefficients change from one scan line to another.
3. The Doppler spectrum is divided in two looks, that are separately inverse transformed back to the spatial domain. The magnitude of each look is also computed.
4. The mis-registration along the range direction of the two magnitude looks is computed using their mutual cross-correlation. To implement this step, looks are transposed, transformed, complex multiplied, and transformed back. The average of all range lines is computed to reduce noise.
5. The maximum of the cross-correlation is computed on the host processor. Its position gives the amount of FM rate error ΔK_a for the block.

A first order polynomial model is used also in this case to get a smoothly varying FM rate over the entire image. The Nelder-Mead algorithm is started with the nominal FM rate, with the average FM rate error from all blocks already added to improve convergence.

Integer Doppler centroid estimation: the last doppler parameter that is estimated is the integer part M_{amb} of the Doppler centroid. The procedure is similar to the FM rate estimation, as it is based on detecting the mis-registration in the range direction of multiple looks. However, contrary to the technique described above, it starts from the range-doppler image, rather than the range compressed one. Therefore, the entire image needs to be transformed along the azimuth direction. To avoid allocating memory for the transposed image, the operation is carried out in vertical slices: a slice is first transposed in a temporary buffer, a batch of FFTs equal to the number of lines in the transposed buffer is applied, and then the slice is written back in its original position by transposing it again. Afterwards, the integer Doppler centroid is estimated using the following steps:

1. The estimated fractional part of the Doppler frequency is used to determine the vertical location of the maximum of the Doppler power spectrum.
2. The image is divided in slices[4], and for each slice a set of pair of looks, symmetric with respect to the maximum of the Doppler power spectrum, are defined at increasing distances from the estimated maximum.
3. Each look is compressed in the azimuth direction by multiplying it with the frequency domain matched filter and transforming it back to spatial domain. The magnitude of each look is computed.
4. Magnitudes of compressed looks are compared in pairs to determine the range mis-registration. The displacement is computed by finding the maximum of the cross-correlation, averaged over all the lines of the looks to reduce noise. The cross-correlation is upscaled by a factor of 4 to get a higher accuracy. Pairs that are further away with respect to the maximum of the Doppler power spectrum show a linearly increasing displacement, where the rate of increase is related to the doppler frequency.
5. A fitting first order surface without the mutual $r \cdot a$ term is computed, and the integer part of the Doppler frequency is determined as the coefficient along the range direction, rounded to the lower integer value.

A graphical representation of this procedure is illustrated in Figure 4, for a single slice of the image, with the detected displacements shown on the graph on the right-hand side. When all slices are considered, a plane rather than a approximating straight line is computed.

[2]The power filter is identical for all blocks, and it is computed only once.

[3]If the block size is the same as the one used for the fractional Doppler centroid estimation, this step can be shared with step 2 of the previous procedure, with a considerable saving in running time.

[4]These are different, and typically larger, than slices used to transform the image in the Range-Doppler domain.

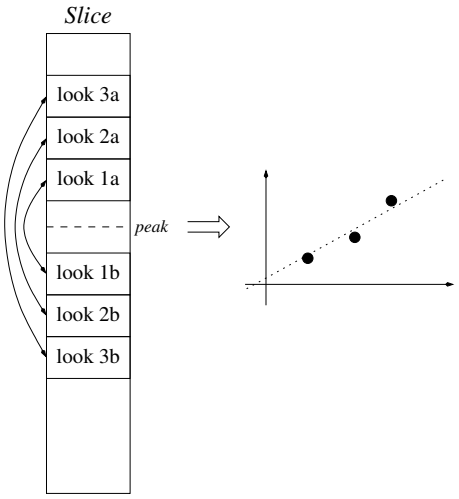


Figure 4. Integer Doppler frequency determination

For all three estimation procedures, a magnitude based method has been adopted. The literature also contains several phase based methods, which were in some cases also implemented. However, the ones presented here gave the best results from both an accuracy and a performance point of view, so the others are not analysed.

After estimations are over, the image is in the Range-Doppler domain, as it was transformed during the evaluation of the doppler ambiguity number. If estimations are skipped, the azimuth FFT using coalesced transposition and slicing is carried out anyway, as it is necessary for the next steps in the algorithm.

Range Cell Migration Correction (RCMC)—Range cell migration occurs because the distance (range) between the antenna and a target changes along the flight path. This effect must be compensated, or otherwise severe smearings appear in the the final focussed image. This is best achieved in the Doppler domain, as the spectrum for a target is also skewed, with the amount of skew depending on the effective doppler frequency. Hence, RCMS usually consists in a remapping of samples in the range-doppler domain, using some sort of interpolation mechanism.

Key to an accurate RCMC is the precise knowledge of the doppler centroid, given by both the integer and the fractional parts estimated in the previous steps. The correction is hence computed using the available models. Remapping is along the range direction only, and has been implemented in two user-selectable ways:

- Using a nearest neighborhood method, to achieve high performance with a slightly less accurate correction.
- Using a weighted sinc interpolator, for maximum accuracy.

In both cases, a GPU kernel performs the correction one range line at a time, concurrently for all pixels of each line. The result oevrwirtes the initial image, so that at the end of processing a corrected range-doppler array is available.

Azimuth compression—The last step in the algorithm performs the final compression along the azimuth direction. Contrary to the matched filter in range, that is extracted from the chirp data, for azimuth compression the matched filter must be properly computed, and it depends on both the accurate estimations of the doppler centroid and of the azimuth FM rate. Moreover, it changes with range, so different filters are used for each azimuth scan line.

As the image is already in the doppler domain, compression corresponds to a complex multiplication, followed by a last inverse FFT. To reduce the number of transposition, the matched filter is computed directly in the frequency domain along range lines, and complex multiplication directly applied. This is performed using two successive parallel kernels, but they can in principle be grouped into a single one.

The final IFFT uses the usual coalesced transposition, followed by a batch of IFFTs, and terminated with a last coalesced transposition that overwrites the initial data. The resulting image is therefore in Slant-range projection, and all pixels carry a complex data. To visualize it, the magnitude for each pixel must be computed.

ω-k Algorithm

Contrary to the RDA, the ω-k algorithm performs all compressions and corrections within the frequency domain in both the range and azimuth directions. and it is able to achieve better accuracy for large antenna squint angles. The complexity is comparable to the Range-Doppler algorithm, and many of the processing steps are shared with it, so they will simply be referenced here, rather than repeated.

In principle, as described in Section 3, the ω-k algorithm consists in a two dimensional FFT, the product with the reference function followed by the differential Stolt interpolation, and a concluding two dimensional inverse FFT. However, our implementation of fractional and integer doppler centroid parameter estimations requires a range compressed and a range-doppler image respectively. While in RDA these images are readinly available along the standard processing flow, they must be appropriately created in the ω-k flow. Hence, the sequence of steps is different whether estimation of doppler parameters from received data is performed or not; in the following, the flow that includes estimations is considered, with the other one being faster because it needs less domain transformations.

For the reasons outlined above, the ω-k algorithm starts with range compression, i.e. a range FFT, the chirp matched filter multiplication, and a range IFFT. Fractional doppler centroid estimation follows, as the image is now in the range compressed state. The integer doppler centroid estimation requires the image to be in the range-doppler domain, so a coalesced transposition and an azimuth FFT are executed before invoking the estimation procedure. Up to this stage, the algorithm is actually identical to the RDA, except that the doppler rate estimation is not performed, as it is not needed in the following steps.

The RDA and ω-k algorithm diverge at this point. As the image is currently in the range-doppler domain, a forward range FFT is needed to transform it from range time to range frequency. RDA does not need it, as both RCMC and Azimuth compression are performed in the range-doppler domain. The reference multiply and differential Stolt interpolation are then executed, as described in the following paragraphs.

Reference function multiply—The complex reference function is computed for each range scan line with a parallel pointwise kernel, using the estimated doppler centroid two dimensional

model at a reference range, corresponding to the middle of the image swath. This function consists of a phase term only (the magnitude is 1), and its full formulation includes a square root; to improve performance, the square root is actually implemented using its second order Taylor approximation, with all intermediate computations carried out in double precision, and the final value converted to a single precision floating point complex value.

After computing the reference function, a second pointwise kernel performs the actual complex multiplication, and this is repeated for all range lines in the image. The final output is an image in both the range and azimuth frequency domain, which is correctly focused at midrange, but still has a residual error along the range frequency direction, due to how the reference function has been computed.

Note that the reference function multiplication is applied to an already range compressed image, rather than the original one transformed in the two dimensional frequency domain. The reason is that parameter estimations were carried out before this step. However, this is acceptable, as this step is invariant with respect to range compression, and allows us to save the memory required to store the original image, while keeping the compressed one only.

Stolt interpolation—As the resiudal error after the reference function multiplication depends on range frequency, rather than range time, it can be eliminated with a mapping interpolation along the range frequency dimension. This step, called Stolt interpolation, finally focuses the entire image.

Initially, a vector containing the value of range frequencies is computed. This is useful, as the interpolation is azimuth dependent, but range frequencies are not, so they can be stored and reused for all lines without modifications.

Then, looping on all lines of the image, a parallel kernel computes the amount of displacement of each range frequency, as a floating point number, again using an approximation of the exact function, that includes a square root. All displacements are stored in a vector in the device memory, which is used as the input of an interpolation kernel, that generates the new line using a nearest neightborhood approach. A linear interpolation has also been implemented, but it did not show an appreciable better accuracy in the final result.

Following the Stolt interpolation, the image is focused but still in the two dimensional frequency domain. A range inverse FFT, followed by a transposition and an azimuth inverse FFT brings the image back in the spatial domain, where each pixel is a complex value in the Slant-range projection. Similarly to the RDA, a visible image is obtained by computing the magnitude at each pixel.

6. Experimental Results

The entire system for focusing using both the Range-Doppler as well as the ω-K algorithm has been implemented on a Tesla C1060 GPU with 4 Gbyte of main memory, on an Intel i7 host processor with 8 Gbyte of memory, running a 64 bit Linux operating system. The Cuda toolkit that was used was version 3.5.

An example of focusing of a Stripmap image using the ω-k algorithm is presented in Figure 5, with the input and output images, as well as an intermediate result, transformed into the spatial domain to show various processing levels. The source raw data is at the top of the figure; then, a range compressed image, used for parameter estimation, follows. The final focused image is at the bottom. Figure 6 shows an enlarged area after bulk compression but before differential Stolt interpolation on the top, and after final focusing on the bottom; as the area has been selected away from mid-range, the latter presents a better resolution

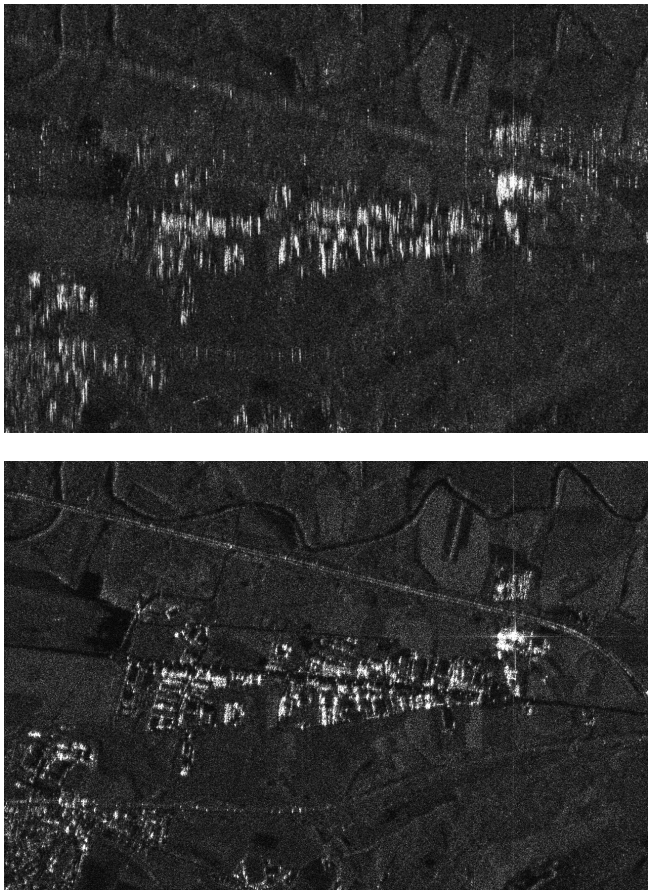
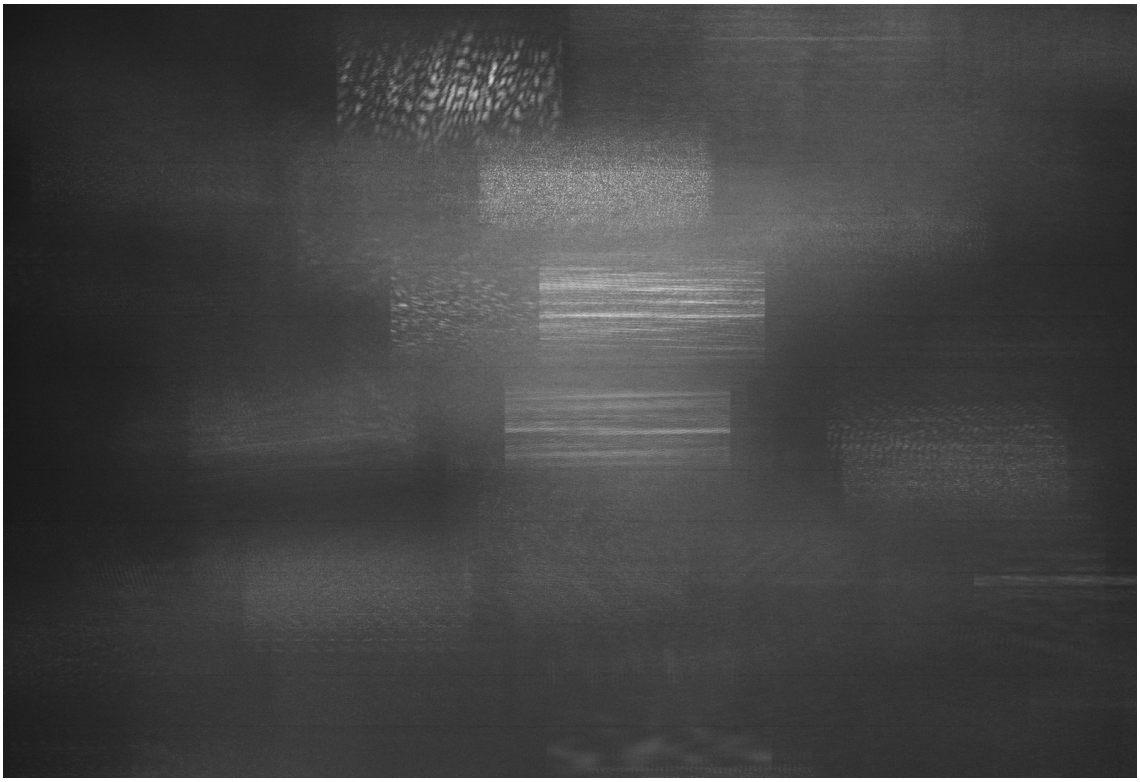


Figure 6. Bulk compression without and with differential Stolt interpolation

The performance of some of the main steps are listed in Table 1. The transfer of a 1 GByte image from host to device memory (or viceversa) takes around 270 ms, with a throughput of more than 3.5 GByte/s. The next two lines illustrate the performance of Fourier transforms and transpositions, which are among the most used kernels in the system. The range compression of a 16384×8192 image takes 330 ms (including forward FFT, complex multiplication and inverse FFT), while compression along the azimuth direction takes 3.9 s, with an initial transposition, forward FFT, matched filter multiplication, inverse FFT and final transposition. In general, the FFT library achieves around 140 Gflops performance.

The last line of the table shows the overall processing time for a 400 Mpixel input image: with less than 20 s, the equivalent throughput is around 20 Mpixel/s. Timing performance are very similar for both the Range Doppler and ω-k algorithms. The algorithms were applied to either Stripmap or Spotlight raw images from the Cosmo-SkyMed satellites, with a resolution of 1 meter for Spotlight and 3 meters for Stripmap. The running time also includes Doppler parameter estimations in



(a) Raw Image

(b) Range compressed image

Figure 5. Processing steps for a Cosmo-SkyMed Stripmap image

(c) Focused image

Figure 5. Processing steps for a Cosmo-SkyMed Stripmap image (cont.)

Table 1. Performance figures for various algorithm steps

Description	**Time**
1 Gbyte transfer	270 ms
Range compression (16384×8192)	330 ms
Azimuth compression (16384×8192)	3.9 s
Overall algorithm	< 20 s

all cases. The running time is measured from the instant that the source image is available in host memory to the instant in which the output image is transferred to the host, i.e. it ignores the I/O time for image transfers to and from the disk. The image is also supposed to be already decrypted and decompressed. Support for immediate display of the image using OpenGL directives without transferring the final image to the host is also provided.

The achieved focusing accuracy is remarkable, as the results are virtually indistinguishable with respect to the Cosmo-SkyMed official processing toolchain, but with a fraction of the running time. Although not covered in this paper, our system also supports other features provided by Cosmo-SkyMed, such as multi-looking, as well as the generation of higher level products (ground projected, geo-coded, ...), again enjoying a substantial speed-up due to the use of the accelerating GPU.

The Tesla C1060 and the CUDA Toolikt 3.5 are now old versions. Performance could easily be improved with more recent hardware and firmware, that takes advantage of a better architecture with more processing cores, and expanded libraries; the new cuFFT, for instance, supports input arrays with stride, which allows to perform the azimuth transformations without transposing the data. As these are used quite extensively and constitute our main bottleneck at the moment, we expect a substantial decrease in running time by adopting them, with minor changes to our software tool-chain. This is in fact one of the advantages of relying on commercial hardware with a very large user base.

All algorithms were developed using 32-bit floating point data, for better accuracy compared to a fixed point approach. Some of the estimation procedures even use double precision floating arithmetic for some intermediate results. However, Tesla performance in double precision are around ten times slower than the equivalent processing in single precision, therefore we didn't explore implementing the entire algorithm using 64-bit doubles. On the other hand, in newer GPU architecture there is only a $2\times$ penalty for double precision, so one could trade-off running time to achieve even higher accuracy, at the cost of a higher memory cost.

7. CONCLUSIONS

In this paper, a GPU based implementation of the Range-Doppler and ω-k algorithms for Synthetic Aperture Radar focusing has been presented. The highly parallel architecture allows to reach near real-time performance, which is necessary in some specific SAR applications, and also needed to cope with the large amount of data produced by a constellation of satellites.

The technique has been applied to actual data acquired by the COSMO-SkyMed satellites, and the flow includes all required radiometric corrections and Doppler parameter es-

timations to obtain a perfectly focused image, comparable in quality to the standard processing chain currently used in the COSMO-SkyMed ground stations. The obtained throughput is similar to other state-of-the-art works in the literature.

ACKNOWLEDGMENTS

The authors wish to thanks the entire team at e-Geos, Rome, Italy, for the support in understanding Cosmo-SkyMed specifications and data formats, and the useful suggestions during the development of the focusing processors.

REFERENCES

[1] I. Cumming and F. Wong, *Digital Signal Processing of Synthetic Aperture Radar Data: Algorithms and Implementation*. Artech House, Incorporated, 2005.

[2] T. Hartley, A. Fasih, C. Berdanier, F. Ozguner, and U. Catalyurek, "Investigating the use of gpu-accelerated nodes for sar image formation," in *IEEE International Conference on in Cluster Computing and Workshops*, Sept. 2009, pp. 1–8.

[3] D. B. G. Rubin, E.V. Sager, "GPU acceleration of SAR/ISAR imaging algorithms," in *Proc. of the Antenna Measur. Tech. Ass.*, Oct. 2010, pp. 1–5.

[4] A. Fasih and T. Hartley, "GPU-accelerated synthetic aperture radar backprojection in CUDA," in *IEEE Radar Conference*, May 2010, pp. 1408–1413.

[5] B. Liu, K. Wang, X. Liu, and W. Yu, "An efficient SAR processor based on GPU via CUDA," in *2nd International Congress on Image and Signal Processing*, Oct. 2009, pp. 1–5.

[6] N. R. C. Clemente, M. D. Bisceglie and M. Spinelli, "Processing of synthetic aperture radar data with GPGPU," in *IEEE Workshop on Signal Processing Systems*, Oct. 2009, pp. 309–314.

[7] X. Jin and S.-B. Ko, "GPU-based parallel implementation of SAR imaging," in *International Symposium on Electronic System Design*, 2012, pp. 125–129.

[8] "UGS COSMO-Sky-Med," http://www.cosmo-skymed.it.

[9] nVIDIA Corporation, "Tesla C1060 Datasheet," 2010. [Online]: http://www.nvidia.com/docs/IO/43395/\-NV_DS_Tesla_C1060_US_Jan10_%lores_r1.pdf

[10] HDF Group, "Hierarchical Data Format, Version 5," 2013. [Online]: http://www.hdfgroup.org/HDF5

[11] CEOS WGD on SAR Data Standards, "CEOS SAR Data Products Format Standard," 1989. [Online]: http://wgiss.ceos.org/archive/archive.pdf/sardata.pdf

[12] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The Computer Journal*, vol. 7, no. 4, pp. 308–313, 1965.

BIOGRAPHY

Claudio Passerone *received the M.S. degree in Electrical Engineering from Politecnico di Torino, Italy, and the Ph.D. degree in Electrical Engineering and Communication from the same university, in 1994 and in 1998, respectively. He is currently associate professor within the Electronic and Telecommunication Department of Politenico di Torino. His research interests include system-level design of embedded systems, high performance parallel architectures and electronics for small satellites. Prof. Passerone is a co-author of two books and has published over 60 international journal and conference papers. He has served on the techinical committees of several IEEE conferences and has received the best paper award at the 9th International Conference on Electronics, Circuits and Systems, in 2002. He is co-founder and member of the board of the company 4C Polito Space.*

Claudio Sansoè *obtained the master degree in Electronics from Politecnico di Torino in 1984. From 1984 to 1985 he was with ESA (European Space Agency). In 1987 he was cofounder of ByProg s.r.l., a firm specialized in design of digital electronic systems and software for automation and process control. In 1990 he became Aggregate Professor at Politecnico di Torino, where he is currently Associate Professor. During the years, his research activities involved many fields of Electronics and Computer Science. His main past research interests were design of integrated high-speed parallel processing architectures, design and VLSI implementation of high-speed decoders for convolutional and turbo codes for space applications, and design of node controllers for high capacity WDM optical packet networks. Since 2004 he is deeply involved in the design and implementation of nanosatellites.*

Riccardo Maggiora *received the M.S. degree in Telecommunications Engineering from Politecnico di Torino, Italy, and the Ph.D. degree in Electrical Engineering and Communication from the same university, in 1995 and in 1999, respectively. He is currently Aggregate Professor within the Department of Electronics and Telecommunications of Politenico di Torino, Italy where he is member of the Laboratory for Antennas and Electromagnetic Compatibility Group and group leader of the Plasma Facing Antenna Group. His research interests include HW and SW design of radar systems, antenna design for radars and communications, and plasma facing antennas design and analysis. Prof. Maggiora is co-author of more than 100 international journal and conference papers. He is co-founder and member of the board of the company 4C Polito Space.*