Real-time Airborne SAR Imaging. Motion compensation and Autofocus issues.

Hubert M.J. Cantalloube
Office National d’Études et Recherches Aérospatiales, Chemin de la Hunière 91120 Palaiseau, France
tél +33 1 69 93 62 12, fax +33 1 69 93 62 69, http://phare.normalesup.org/~cantallo/index.html, cantallo@onera.fr
Carole E. Nahum
Direction Générale de l’Armement, 5 rue des Mathurins 92221 Bagneux, France

Abstract

The $\omega - k$ SAR algorithm used for processing the RAMSES/Sethi airborne radar signals was recently experimented on massively parallel graphic processor. It showed the capability to process a full resolution image (10 cm in both range and azimuth) at about 1/2 the acquisition rate on a single Nvidia M2050 card. Thus the possibility to achieve a SAR processing rate above the acquisition rate should be possible on a reasonable multi-card system. However, this would not be enough to effectively achieve the producing an effectively focused image as soon as it is acquired. Indeed, unlike the satellite case, the motion compensation is highly critical for airborne SAR, and the effective trajectory may differ significantly from the planed one. Furthermore, the present SAR processor assumes the knowledge of the full acquisition trajectory before choosing an optimal nominal trajectory, computing the motion compensation for the whole acquisition and then only starting the signal processing. Here are detailed the modifications in the processing chain to be done for having the real-time capability, this requires use a fixed nominal trajectory (from flight planning), widening the mocomp capabilities to cope with the increased trajectory deviation, compute the motion compensation with raw trajectory on-the-fly, prepare the autofocus during processing, perform autofocus from navigation system improved trajectory and eventually refocus the image.

1 Context

Besides its use as a SAR technology test-bed and for measuring targets and infrastructure signatures for military use, ONERA SAR systems RAMSES & Sethi are also used for scientific investigations in sometimes remote area. As of today, SAR signal is recorded on board with merely a visual examination on a scope, and the processing is done off-line after landing and dismounting the hard disk bay from the aircraft. Unavoidable incidents in acquisition (such as unexpected side-wind that offsets the illumination area, excessive tracking errors, external radio frequency interferences, system or operator malfunctioning) are coped with by redundancy in acquisition runs during a flight and redundant and/or spare flights. But since scientific acquisition campaigns are more cost-constrained and/or time-limited, there is a strong incentive to include on board a computation capability for synthesising the image in real-time thus assessing the pointing to the area on interest, the final image quality and possibly re-target the acquisitions during the flight.

As multi-cores CPUs and parallel graphic processors (GPU) become common, the SAR image synthesis software used at ONERA is being ported to both using Native Posix Threads for multi-cores CPUs (coarse grained systems) and OpenCL for GPU (fine grained systems). First implementations of SAR processing on GPU were aimed at reducing the computation time of the slower back-propagation (time-domain) algorithms [1], recent near real-time demonstrations [2] implement the range-Doppler algorithm, an approximate algorithm not appropriated to wide-band and/or airborne SAR imaging such as ours, but of which algorithmic complexity is only a small factor (about 2) lower than that of the $\omega - k$ algorithm we use. Therefore, real-time implementation of our algorithm is potentially within the reach of today’s “off-the-shelf” technology by using GPUs (multi-core CPU would be dissuasively expensive as of today).

2 Real-time airborne SAR image synthesis difficulties

SAR image synthesis requires a knowledge of the antenna trajectory with an accuracy around 1/10 of the wavelength. Of course, an offset error as a minor impact (it changes the terrain elevation dependency of the focusing when the offset is significant with respect to terrain elevation variations) and a constant velocity bias a smaller impact on focusing (it distorts the image at first order) but nevertheless, in the above example (10 cm resolution at X-band with 5 km range) a velocity bias of 2 cm/s has a non negli-
gible impact on resolution.

Such an accuracy on the trajectory is beyond the capability of the navigation system, but even worse, the real-time trajectory output is less accurate than the actual trajectory output with a few tens of second delay (the reason is that the real-time trajectory output uses only prior GPS ranging and satellite-broadcast atmospheric corrections, while the final trajectory uses later measurements and atmospheric information thus achieving a much better trajectory estimation).

This implies we need an autofocus with two options: Either we wait until we have the final trajectory estimate from navigation system, while collecting data for autofocus using the real-time trajectory, compute an updated trajectory from final navigation trajectory and autofocus data, and then start the synthesis. Or alternatively, we may start the synthesis from the real-time trajectory, while simultaneously collecting autofocus data, and then compute an updated trajectory from final navigation trajectory and autofocus data, and eventually refocus the synthesised image.

The drawback of the first method (which can use the present software) is that the image appears about one or two acquisition times after the end of the acquisition, which is -in practical cases- while the aircraft is already manoeuvring to line up for the next acquisition. The second method has the advantage of producing, with a few tens of second lapse, an image which -though not fully focused- can be used to assert the image quality and the targeted area illumination. Furthermore, the focusing can be locally improved interactively around a point-like echo to assert the achievable azimuth resolution, even before the final trajectory output is available (fig. 1). The difficulty with the second method is that it requires changes in the SAR synthesis program and improvements on existing autofocus and refocusing post-processing software that are detailed below:

2.1 Improvement of the SAR synthesis program

There are a few minor changes necessary that are, first, the capability to use an a priori nominal trajectory (given the flight-plan) without knowledge of the final trajectory. Second, the autofocus data gathering should be possible simultaneously while synthesising an image (to date it is an “auxiliary output” of the time-domain processor which is abortive -for autofocus algorithm which does not require a prior low-resolution image-). Third, the motion compensation should be calculated on-the-fly and not before starting the synthesis (because trajectory is unknown before acquisition). This is not a difficult point because it was done that way in earlier versions of the software (it was, later on, moved to the beginning because this allowed to assert the correctness of the processing parameters -and possibly update them- before starting the full synthesis).

More important change is for the capability of motion compensate with a larger disturbance between nominal and actual trajectory. This is needed because the disturbance is now the distance between the planned trajectory and the actual trajectory unlike in the off-line case the distance between the actual trajectory and a linear regression of it. One solution (which is derived from the bistatic version of the processor) is to use the nominal trajectory for pre-summing/velocity uniforming stage and the processing block sequencing, but use an intermediate ad-hoc “nominal trajectory for the block” during the processing of each individual block. The extra cost is the mapping of the synthesis coordinate system (that of the processing block trajectory) to the final coordinate system (that of the full image, hence the targeted trajectory). Although both coordinate systems are cylindrical coordinates, the mapping is neither uniform, nor mono-dimensional (as is the Stolt-transform) because of the non uniform terrain altitude (on which the focusing depends for the non-linear trajectories typical of airborne SAR). However, as in the bistatic processing case, the mapping can be performed by two successive one-dimensional (azimuth and range) mappings hence the Fourier domain interpolation requires only zero padding one line (or a few lines -typically 32- on a graphic processor) at a time, hence the memory impact can keep minimal.

2.2 Paralleling the SAR synthesis program

At present, our off-line SAR processor is fully functional on multi-CPU/core environment (NTP version), but only the monostatic frequency domain processing (faster $\omega - k$ algorithm) is coded on GPU (some of the RFI filtering for older waveform types is not functional yet) because it is much faster than the time-domain processing (even when factored [3]) and requires less memory (at least a factor of two for 50% block overlap) in the present implementation (though this can be changed by interleaving factoring and integration).

Multi-CPU/core paralleling (using NTP interface) [4] share the computation of each row/column steps of the processing by blocks of consecutive rows/columns, the threads are synchronised at the end of each stage using mutex locks. OpenCL GPU implementation [5] makes the computation by blocks of consecutive rows/columns matching the elementary scheduling vector of the GPU architecture (typically 32 or 64 at present). In order to make graphic memory accesses parallel, the column order steps are preceded and followed by a block-transpose operation (each $32 \times 32$ blocks are transposed in place). This block transposition can be efficiently done in parallel using shared memory (a small but fast memory that can be accessed out of order through a crossbar device). To cope with the smaller memory available on today GPU, the two main buffers of the $\omega - k$ algorithm are fused when possible (and when that reduces memory footprint) and the synthesised row of image are transmitted to the GPU (and later output to disk) not by full processing blocks, but by smaller GPU block
slices (typically 32 or 64 consecutive rows).

One important point in coding is to pipeline the reading of raw signal from recorder RAID disk array, the transmission to the GPU (through a relatively slow PCIe bus) and the decoding & range compression kernel on the GPU. Similarly, the last kernel (range band remodulation and illumination computation), the transmission back to the CPU and the writing of the result image to disk are pipelined to hide the slow PCIe and disk transfers.

Since disk write are much slower than disk read and last illumination & formatting stage is not computationally intensive, we further interlace the writing of synthesised lines with reading & range compressing the signal for next processing block.

For example, we tested on a typical high-resolution image (fig. 2), with 10 cm resolution at X-band, a swath of 1 km at 5 km range, and a stripe length of 10 km. It corresponds to 2 minutes of signal acquisition of the RAMSES radar on downtown Toulouse (France). The time series of the processing on GPU is given on fig. 3. The image was synthesised in 4 minutes and 29 seconds elapsed time (9 seconds of CPU-only preparation and 4 minutes 20 seconds of GPU computation) on a server with a single Nvidia M2050 GPU. The GPU was “warmed” (hence the kernel already compiled) prior to run. Raw signal is read from an 8-disk RAID5 with 2500 Mb/s sustained rate specification (close to 3 Gb/s measured), and resulting image is written to the same RAID array. Disk transfers latency is almost completely hidden (but the writing of the last two output blocks) by pipelining and double-buffering. PCIe transfers between CPU and GPU memories (18.57 s) are mostly (98.6 %) paralled with kernel execution. Kernel overhead accounts for 1.7 % of elapsed time (4.42 s).

There is a provision for about 10 s improvement by coding for the GPU the remaining CPU-executed code for prior motion compensation parameters evaluation, and around 10 to 20 s by better coding two kernels that use non-coalescent memory accesses (hence serialised execution on the GPU). GPU memory footprint (which is critical for high resolution imaging on present generation of GPU, limited to 6 GB graphic RAM) can be further reduced by pre-integrating in the Fourier domain (hence serialised execution on the GPU). GPU memory footprint (which is critical for high resolution imaging on present generation of GPU, limited to 6 GB graphic RAM) can be further reduced by pre-integrating in the Fourier domain (hence serialised execution on the GPU).

For comparison, the same image (8.6% RMS relative difference) is computed on the two Xeon X5650 2.67 GHz clocked 6-core CPU’s (12 cores total) of the same server in 24 minutes elapsed time. This indicates that the synthesis of such an high-resolution image should be possible in a time comparable to that of acquisition with the present algorithm on a similar high-end desktop computer using GPUs (typically 2 Nvidia 2050). Though technically possible, attaining real-time performance with multi-core CPUs that way is still an expensive solution (typically 11 Xeon 2.66 GHz X5650 chips of 6 core each would be needed for our example assuming perfect scalability).

### 2.3 Parallelling the autofocus program

As the autofocus is currently performed [6], it can only be performed once the full trajectory is acquired and once the autofocus data have been pre-processed (as a by-product of the image synthesis program).

The autofocus data is high resolution profiles on a small neighbourhood of points on the ground that are used for multilateration. The profiles are narrowband filtered in Doppler around the ground frequency computed from the initial (real-time) trajectory and reduced in repetition rate. The autofocus program is a coarse to fine loop between the synthesis of small images around the ground-points (using Polar Format Algorithm) for successive apertures, the measure of the apparent motion between these images (using fast correlation of detected images), and the computation of the corresponding trajectory errors (using a Singular Value Decomposition for robust inversion of the error-to-displacement formula). Algorithm is started with short apertures (hence low azimuth resolution, by high time resolution) are progressively tuned to longer aperture (hence high azimuth resolution, thus yielding accurate velocity corrections but at a lower time-resolution).

Both PFA and fast correlation are easy to parallel on either multi-processors/core or on GPU; SVD can most probably using sub-optimal algorithms of which the convergence is slower, but that can be parallellised [7].

### 2.4 Parallelisation & improvement of the re-focusing post-processor

Our SAR image post-processor is an extensive tool-kit, of which one of the tool is used to refocus an image, typically for a near-range error, a velocity bias, trajectory datation error and/or a bandwidth equalisation (it also allows windowing/resolution change and thermal noise injection because in was designed to simulate lower resolution satellite SAR image).

It performs the refocusing by overlapping square blocks (this requires that the defocusing is moderate, but in our case the real-time trajectory error is few cm/s typical). On each block, a Fourier transform is done and the aperture is computed (from initial trajectory and terrain elevation) for each point in the aperture a correction in bandwidth equalisation and range is applied. The range correction is computed using the initial (real-time) and the final (autofocus) trajectories.

Parallelling this code is probably easy on multi-processor/core by using one thread per overlapping block. On GPU, the Fourier transform can be done in parallel within each block (and probably several blocks in parallel too, in order to occupy the large number of available elementary processors). Aperture computation and phase corrections can probably be efficiently parallellled, but even if not, the refocusing computation is largely dominated by the back-and-forth two-dimensional Fourier transforms.
3 Perspectives

Fig. 4 summarises the architecture of a real-time SAR processor using massively parallel processors. Compared to existing software, the ground-point neighbourhood extraction and block Fourier filtering is not available in frequency domains processor, nominal trajectory and motion compensation are fully computed on start-up (here azimuth migration is computed at once, range migration & mocomp is computed per processing block), non-linear remapping to global image coordinates (similar to bistatic remapping operation) is needed. Eventually, autofocus and refocusing needs being accelerated to near-real-time speeds.

We have described here the software adaptation needed for real-time SAR image synthesis on-board an airborne system such as RAMSES/Sethi. It should be possible with an high-end desktop computer equipped with multiple GPUs with the today available performances (with a typical 3U volume and about 1 kW power). The design could also be adapted to UAV sized projects as the BuSARd system if implemented on an aeronautic grade module (typically 2 CPU cores and 2 smaller GPUs on a 1 l/300 W air-cooled module) with smaller power requirement due to both higher frequency (hence lower integration time, hence smaller memory footprint) and narrower swath (due to lower flight altitude).

Figure 1: Refocusing example (screen-copies of the actual analysis tool) on a corner reflector. Left=image, right=aperture (yellow line) on phase-colour-coded spectrum. A: Image synthesised with trajectory offset by 0.03 s (focus-blurred). B: Same image interactively refocused with the genuine trajectory. C: Image directly synthesised with the genuine trajectory (for comparison), D: Resolution analysis on the refocused image.

Figure 2: Airborne SAR image of 10 cm resolution at X-band acquired in 2 minutes and synthesised off-line in 4 minutes and 29 seconds elapsed time on a single Nvidia M2050 GPU server with a RAID disk array.

Figure 3: Time series of the processing for fig. 2. Green=signal write to GPU, red=execution on GPU, blue=image & trajectory read from GPU, black=number of kernel (grids) waiting for execution.

Figure 4: Synoptic of the proposed real-time SAR processor. $\omega - k$ SAR processor representation is simplified (only steps relevant to required improvements are figured).

References


