GPU-Based Nonlocal Filtering for Large Scale SAR Processing

Gerald Baier[1], Xiao Xiang Zhu[1,2]

(1) *Remote Sensing Technology Institute (IMF), German Aerospace Center (DLR), Oberpfaffenhofen, 82234 Wessling, Germany*
(2) *Signal Processing in Earth Observation (SiPEO), Technical University of Munich (TUM), Arcisstr. 21, 80333 Munich, Germany*

November 30, 2016

Abstract

In the past few years nonlocal filters have emerged as a serious contender for denoising synthetic aperture radar (SAR) images, offering superior noise reduction and detail preservation compared to many other filters. In this manuscript we analyze how nonlocal filters, whose computational costs were so far prohibitive for large scale processing, can be implemented efficiently on graphics processing units (GPU). As a case study NL-SAR, a state of the art SAR filter, is implemented to run on a NVIDIA Tesla K40. We describe the appeal of GPUs, or any other coprocessor, for nonlocal filters. Nonlocal filtering of TanDEM-X interferograms for generating digital elevation models with a higher resolution and accuracy is given as an application that benefits from efficient and fast nonlocal filtering.

Index terms— synthetic aperture radar, denoising, nonlocal, graphics processing units (GPU)

1 Introduction

Starting with NVIDIA's Compute Unified Device Architecture (CUDA) and later the Open Computing Language (OpenCL) coprocessors that accelerate numerical algorithms via general purpose computing on graphics processing units (GPGPU) have become ubiquitous in high performance computing. With their throughput-oriented architecture coprocessors are far better suited for tasks with abundant parallelism, such as image processing, as general purpose central processing units (CPU). These qualities make them attractive for large scale processing of SAR data, which was our prime motivation to investigate their usage for nonlocal filtering.

Nonlocal filters have emerged in the past years as the state of the art filtering concept, outperforming traditional filters in terms of noise reduction and detail preservation. With regard to DLR's TanDEM-X mission, nonlocal filters permit the generation of a severely less noisy digital elevation model with far more details, compared to legacy local filters [1]. Currently their excessive computational cost keep them from widespread use. In this paper we evaluate how coprocessors, in our case a NVIDIA Tesla K40, can be used for nonlocal filtering, where we use NL-SAR [2] as an example. The paper is structured as follows. Section 2 gives a brief introduction to the nonlocal filtering concept and NL-SAR. Section 3 describes, on a high level of abstraction, the architecture of modern GPUs affecting how NL-SAR is to be implemented, which is described in Section 4. Section 5 gives performance results and shows the processing of a complete TanDEM-X interferogram as an example.

2 Nonlocal Filtering and NL-SAR

Nonlocal filters, which were first introduced in [3], indiscriminately use measurements in a far larger area for estimating the noise free parameters of each target pixel than conventional filters. Inside this area, termed search window or search area, a nonlocal filter assigns each pixel a weight, depending on its similarity to the target pixel, before estimating its value through weighted means:

$$\hat{u}(\mathbf{x}) = \sum_{\mathbf{y}\in\partial_{\mathbf{x}}} w(\mathbf{x},\mathbf{y})u(\mathbf{y}), \tag{1}$$

where $\hat{u}$ is the denoised estimate, $\partial_{\mathbf{x}}$ is the search window centered around the pixel position $\mathbf{x}$, w are the weights and u are the original pixel values.

The similarities between pixels are not only a function of the respective pixel values themselves, but also take into account their local neighborhoods, called patches. By comparing patches instead of pixels local structures are considered, leading to improved filtering results. The price to pay, compared to traditional filters, is a severely increased computational complexity, which for a naïvely implemented nonlocal filter is $O(NWP)$, where N is the number of pixels in the image, W the search window size and P the size of a patch.

There exist several filters that adapted the nonlocal filtering concept to SAR, InSAR and PolSAR image statistics. NL-SAR [2] is used as a case study to show the potential of coprocessors for nonlocal filtering of SAR data. Only a very brief and incomplete introduction to NL-SAR is given here, for a complete treatment the interested reader is referred to the original paper.

NL-SAR operates on covariance matrices and is therefore capable of filtering simple SAR amplitude images as well as InSAR or PolSAR data. The similarity for two pixels is defined as a likelihood-ratio test based on the hypothesis that their two Wishart distributed covariance matrices are equal:

$$\mathcal{L}\left(C_1, C_2\right) = \frac{|C_1||C_2|}{|\frac{1}{2}\left(C_1 + C_2\right)|^2} \tag{2}$$

where C_1 and C_2 are the empirical covariance matrices of the two pixels and $|.|$ is the determinant. To ensure that the estimation of C_1 and C_2 is robust and has full rank, off-diagonal elements are rescaled and convolution with a Gaussian kernel is performed as a preestimation step. The patch dissimilarities are computed by summing up the negative logarithm of their respective pixel similarities:

$$\Delta(\mathbf{x}, \mathbf{y}) = -\sum_{\mathbf{p} \in \mathbb{P}} \log \mathcal{L}\left(C(\mathbf{x} + \mathbf{p}), C(\mathbf{y} + \mathbf{p})\right), \tag{3}$$

where $\mathbb{P}$ contains all the indices' offsets in a patch.

The weighting kernel is trained on a homogeneous area and stores the patch dissimilarities' corresponding empirical cumulative distribution function (ECDF). During filtering patch dissimilarities are mapped into weights depending on their quantile in the ECDF. Fig. 1 shows the mapping kernel.

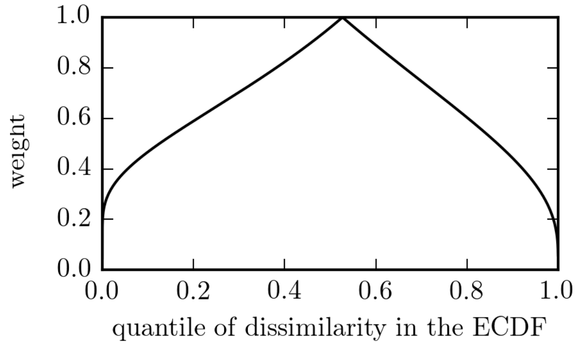


Figure 1: Mapping of dissimilarities into weights

NL-SAR also features a parameter selection which, depending on the equivalent number of looks, picks the best search window size, patch size and Gaussian kernel scale. Bias reduction further ensures a closer adherence to the original data in case the nonlocal estimate is oversmoothed.

3 GPU Architecture

In order to implement NL-SAR to run efficiently on GPUs, their underlying architecture has to be considered. Fig. 2 shows the general architecture of a GPU computing device and how it works in tandem with the host computer.

The host transfers data to the GPU's global memory and then schedules a function, called kernel, which executes in parallel on the compute units and manipulates data stored in global memory.

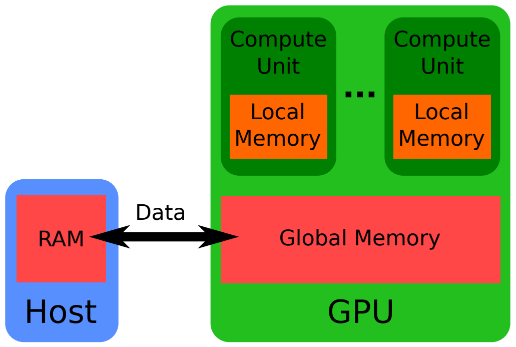


Figure 2: High-level view of a GPU and the host computer

Each compute unit launches several threads which perform the actual computation of the kernel and should have identical workloads for optimal performance.

All threads of a compute unit share access to the global and local memory, where the access to local memory is much faster and access to the global memory should be coalesced, meaning the threads access consecutive memory addresses. Global memory is furthermore accessible by threads from all compute units, however synchronization of threads is only possible for threads inside one compute unit and not among them. The architecture imposes a certain paradigm for implementing efficient GPGPU code:

- communication between host and device should be avoided
- it is mandatory that the problem can be split up in order to be processed separately and independently by the compute units
- as access to local memory is much faster, copying data from global to local memory speeds up the program if the data in local memory is accessed multiple times

In the following section we will describe how the architecture applies to and can be exploited by nonlocal filters.

4 Nonlocal Filters on GPUs

Nonlocal filters are extremely compute-intensive so that the overhead caused by exchanging data with the host is negligible. As every pixel can be processed independently, filtering an image can be easily split up into tiles, which are then processed in parallel by the compute units. Due to the large overlapping search windows nonlocal filters extensively reuse data so that many kernel functions can easily be sped up by making use of local memory. Taking Eq. (1) as an example, the pixel data $u(\mathbf{y})$ can be copied to local memory, where it is accessible for all threads of a compute unit.

All of the just mentioned facts show the potential of GPUs for accelerating nonlocal filtering. In the following paragraphs we describe in greater details some of the parts of nonlocal filters and especially NL-SAR that are less trivial to efficiently implement.

The most computationally expensive task for any nonlocal filter is calculating the weights, which requires both the patch and pixel similarities. As Fig. 3 shows, the pixel similarities computed for two pixels can be reused for many patches, and in the case of NL-SAR even for patches with different sizes. By exploiting this redundancy as described in [4], the computational cost is in essence independent of the chosen patch size.

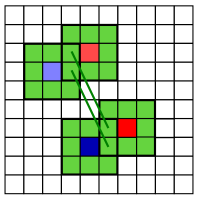


Figure 3: Reusing pixel similarities

The pixel similarities are stored in a four dimensional array, where the first two dimensions define the offset of the target pixel to the center pixel and the latter two are the image dimensions plus the overhead caused by the search window and the patch size. Fig. 4 visualizes how pixel similarities with the same offset to the center pixels are stored consecutively in memory. Storing the pixel similarities and later the patch similarities and weights in such a manner allows for efficient coalesced memory access when multiple pixels are processed in parallel.

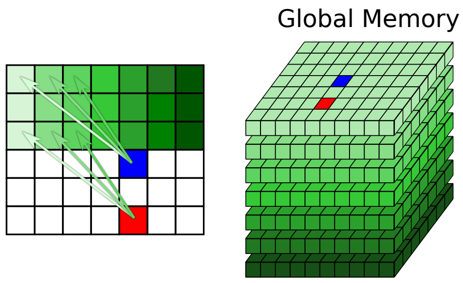


Figure 4: Memory Layout for coalesced Access to Pixel Similarities with identical Offset

A 2D convolution over the last two dimensions with a rectangular window produces the patch dissimilarities (see Eq. (3)). For best performance the 2D convolution is performed as two separate 1D convolutions and makes use of local memory as described in [5]. Separation of the convolution is crucial to avoid idle threads after caching values in local memory.

Number of Compute Units	15
Number of CUDA cores	2880
CUDA cores per Compute Unit	192
Graphics Clock	745MHz
Memory Clock	3000MHz
Global Memory	12 GB
Local Memory Size (set at runtime)	16/32/48 KB
Peak single precision Flops	4.29 TF/s
Peak double precision Flops	1.43 TF/s

Table 1: Tesla K40 technical specification

Computing the weights from the patch dissimilarities requires some adaption of the scheme that is used in the original paper, which relies on a binary search to find the quantile of the patch similarities. We eliminate the binary search by resampling the ECDF on a uniform grid and storing the respective quantiles in a lookup table.

An additional optimization is exploiting the symmetry of weights, i.e. $w(\mathbf{x}, \mathbf{y}) = w(\mathbf{y}, \mathbf{x})$.

5 Experiments

NL-SAR was implemented in OpenCL following the aforementioned general implementation guidelines and optimizations. Development and benchmarking was conducted on a NVIDIA Tesla K40, where Table 1 lists the most pertinent technical specifications. Single precision floating point numbers proved to be sufficient for nonlocal InSAR filtering.

For an InSAR image of size 2048×2048 Fig. 5 lists the processing time with a search window of size 21×21 for different patch sizes $\mathcal{P}$ and Gaussian kernel scales $\mathcal{S}$. The same code is run for comparison on all cores of an Intel i7-4770.

One goal of employing coprocessors is large scale reprocessing of TanDEM-X interferograms to provide a DEM product of higher resolution and accuracy [1]. Fig. 6 shows a DEM of Weihai, China with an area of roughly 2700km^2 and compares the filtering result of a conventional boxcar filter and NL-SAR. For this image a 27962×15240 pixels interferogram had to be processed, which with $\mathcal{S} = [1, 3, 5]$ and $\mathcal{P} = [3, 5, 7, 9, 11]$ took 35min on the NVIDIA Tesla K40.

6 Conclusion

An initial feasibility study of implementing a nonlocal InSAR filter on a GPU was presented showing that nonlocal filtering is ideally suited to be offloaded to a dedicated coprocessor due to its high compute intensity. Our performance numbers and example show, that large scale nonlocal filtering of TanDEM-X interferograms is practical.

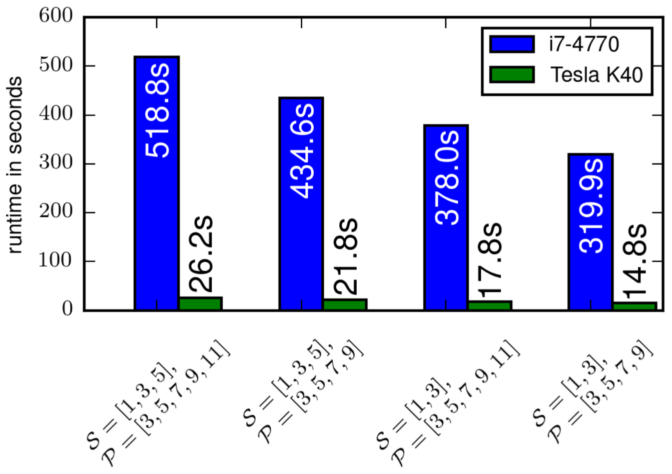


Figure 5: Runtime of NL-SAR for an InSAR image of size 2048×2048 for different parameter sets

Aknowledgement

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Tesla K40 GPU.

References

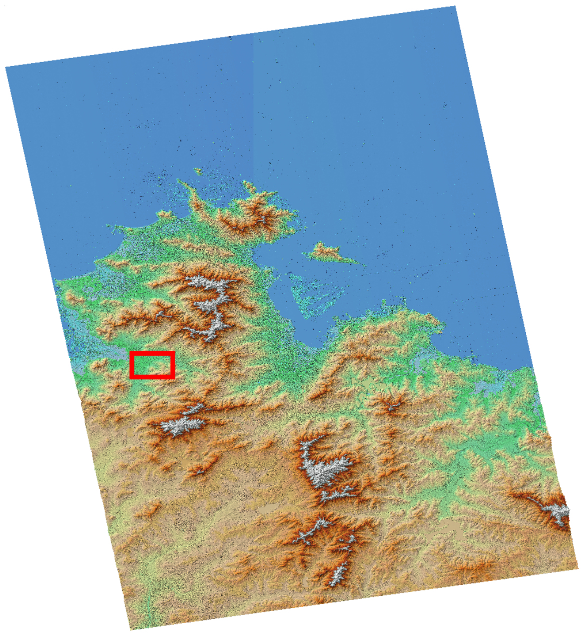
[1] Xiao Xiang Zhu, Richard Bamler, Marie Lachaise, Fathalrahman Adam, Yilei Shi, and Michael Eineder, "Improving TanDEM-X DEMs by Non-local InSAR Filtering," in *EUSAR 2014; 10th European Conference on Synthetic Aperture Radar; Proceedings of*, June 2014, pp. 1–4.

[2] C.-A. Deledalle, L. Denis, F. Tupin, A. Reigber, and M. Jager, "NL-SAR: A Unified Nonlocal Framework for Resolution-Preserving (Pol)(In)SAR Denoising," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 53, no. 4, pp. 2021–2038, April 2015.

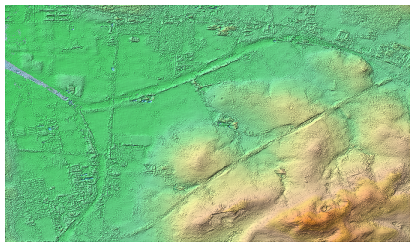
[3] Antoni Buades and Bartomeu Coll, "A non-local algorithm for image denoising," in *In CVPR*, 2005, pp. 60–65.

[4] J. Darbon, A. Cunha, T.F. Chan, S. Osher, and G.J. Jensen, "Fast nonlocal filtering applied to electron cryomicroscopy," in *Biomedical Imaging: From Nano to Macro, 2008. ISBI 2008. 5th IEEE International Symposium on*, May 2008, pp. 1331–1334.

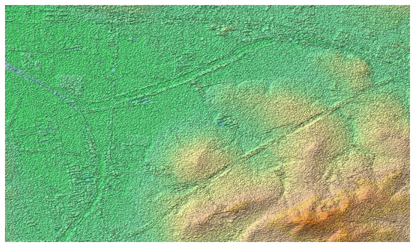
[5] Victor Podlozhnyuk, "Image Convolution with CUDA," `http://docs.nvidia.com/cuda/samples/3_Imaging/convolutionSeparable/doc/convolutionSeparable.pdf`, July 2012.



(a) Complete DEM



(b) NL-SAR



(c) Boxcar

Figure 6: Test site Weihai and results for NL-SAR and a boxcar filter