Contents lists available at ScienceDirect

Remote Sensing of Environment

journal homepage: www.elsevier.com/locate/rse

# Near-real-time focusing of ENVISAT ASAR Stripmap and Sentinel-1 TOPS imagery exploiting OpenCL GPGPU technology

Achille Peternier, John Peter Merryman Boncori *, Paolo Pasquali

*Sarmap SA, Purasca, Switzerland*

ARTICLE INFO

*Article history:*
Received 1 July 2016
Received in revised form 13 February 2017
Accepted 11 April 2017
Available online 14 April 2017

*Keywords:*
Synthetic Aperture Radar
Image focusing
GPGPU
OpenCL

ABSTRACT

This paper describes a SAR image focuser application exploiting General-purpose Computing On Graphics Processing Units (GPGPU), developed within the European Space Agency (ESA) funded SARIPA project. Instead of relying on distributed technologies, such as clustering or High-performance Computing (HPC), the SARIPA processor is designed to run on a single computer equipped with multiple GPUs. To exploit the computational power of the latter, while retaining a high level of hardware portability, SARIPA is written using the Open Computing Language (OpenCL) framework rather than the more widespread Compute Unified Device Architecture (CUDA). This allows the application to exploit both GPUs and CPUs without requiring any code modification or duplication. A further level of optimization is achieved thanks to a software architecture, which mimics a distributed computing environment, although implemented on a single machine. SARIPA's performance is demonstrated on ENVISAT ASAR Stripmap imagery, for which a real-time performance of 8.5 s is achieved, and on Sentinel-1 Interferometric Wideswath (IW) raw data products, for which a near-real time processing time of about 1 min is required. Such a performance has the potential of significantly reducing the storage requirements for wide-area monitoring applications, by avoiding the need of maintaining large permanent archives of Level 1 (focused) imagery, in favor of lighter Level 0 (raw) products, which can be focused on-the-fly within the user's application processing pipelines at almost no overhead.

## 1. Introduction

With the successful launch in the last two years of the first four Sentinel satellites, two of which (Sentinel-1A and 1B) carrying a Synthetic Aperture Radar (SAR) payload, users and service providers in the remote sensing field are more and more faced with Big Data handling problems. The Sentinel routine acquisition plans, their free and open data policy, and the commitment of the European Copernicus Programme to ensure mission continuity up to 2030 is unprecedented and has already significantly increased the number of Earth Observation (EO) data users. This increased data availability, combined with the maturity level of several data processing techniques, which have been developed since the early '90s, has the potential of boosting research activities and commercial services based on satellite data, and also represents a prerequisite for the development of cost-effective, and potentially also near-real-time, monitoring services.

On the other hand, it is well established that new technological solutions are required to handle Big Data. An effective approach for some users and applications is to run computationally-intensive algorithms on supercomputers or distributed computing systems, which consist of a large number of physical machines (worker nodes), located in central processing facilities, such as the ESA G-POD environment (e.g., De Luca et al., 2015), or virtualized through cloud computing (e.g., Zinno et al., 2015). At the same time, several research institutions and SAR-data service providers currently rely on small to medium size in-house processing facilities (e.g., local clusters or just powerful workstations). Both scenarios benefit from solutions which improve the degree to which the computational resources of single machines are exploited. In particular General-purpose computing on Graphics Processing Units (GPGPU) has received lots of attention in recent years, since potentially it can provide access to massively-parallel processing capabilities (up to several thousands of cores) on single workstations, enabling these to perform as "personal supercomputers". Furthermore, the availability of frameworks including programming APIs for standard languages (like C/C++) allows applications with a high level of portability across different hardware platforms (ranging from laptops to multiple-GPU servers) to be built, although an effort is required in terms of software architecture design and code refactoring.

This paper describes a performance-optimized SAR image focusing software, developed from scratch within the ESA SARscape Image Processor Accelerator project (SARIPA), and which is not related to the operational processor used by ESA for the generation of Level 1 SAR

* Corresponding author at: Sarmap SA, Via Cascine di Barico 10, 6989 Purasca, Switzerland.
*E-mail address:* jmerryman@sarmap.ch (J.P. Merryman Boncori).

http://dx.doi.org/10.1016/j.rse.2017.04.006
0034-4257/© 2017 Elsevier Inc. All rights reserved.

products. The main goal of SARIPA is to explore the potential of GPGPU technology to reduce processing time on a single machine, thus tackling the data deluge problem by providing a faster and portable local processing, requiring less computational resources compared to High-performance Computing (HPC) and cloud computing. ENVISAT ASAR Image Mode (IM) and Sentinel-1 Interferometric Wideswath (IW) raw images are used as a test-case, with the goal of achieving near-real-time focusing performance on a single computer with multiple GPUs.

Near-real-time SAR image focusing has been the topic of several recent papers, which have addressed the exploitation of multi-core CPUs (Imperatore et al., 2016), or GPUs, using the Compute Unified Device Architecture technology (NVIDIA, 2016a) proprietary to NVIDIA (Zhang et al., 2016; Trittico et al., 2014; Di Bisceglie et al., 2010). SARIPA is based on the more portable Open Computing Language technology (OpenCL, 2016), which unlike CUDA allows the generated software to be executed on a wider range of devices, including computers with non-NVIDIA GPUs or without any GPU at all.

This paper shares the experience gathered through the design and development of SARIPA, whose outcome is not limited to SAR focusing, but could be reused for several other computationally intensive SAR processing algorithms (e.g., interferometry, measurement of deformation time series). The strengths and weaknesses of basing SARIPA on the more liberal OpenCL are discussed in Section 2, whereas the software architecture is described in Section 3. Section 4 details the processor's performance and its potential impact on the Big Data problem in a concrete application scenario. Discussions and conclusions are provided in Sections 5 and 6 respectively.

## 2. Frameworks for multi-core CPU and GPU exploitation

Several frameworks have emerged in recent years to leverage the computational power provided by modern GPUs. These range from dedicated libraries and Software Development Kits (SDKs) to extensions directly embedded into compilers such as Microsoft's C++AMP and OpenACC. Concerning GPU computing, the two most widespread frameworks are the Compute Unified Device Architecture (NVIDIA, 2016a), and the Open Computing Language (OpenCL, 2016). CUDA and OpenCL are also frameworks that provide developers with the finest control over code implementation and performance, unlike C++AMP and OpenACC, which focus on making the GPU-side aspects as transparent and automatic as possible through high-level abstractions (Hoshino et al., 2013).

CUDA is an NVIDIA proprietary parallel computing platform and programming model, supporting various languages (e.g., C, C++, FORTRAN), and providing optimized libraries for standard mathematical algorithms, like Basic Linear Algebra Subprograms (BLAS) and the Fast Fourier Transform (FFT). CUDA only works on GPUs that are produced by NVIDIA, which on one hand limits code portability to other hardware platforms (i.e., GPUs produced by AMD or Intel and GPU-less computers), on the other it provides a simpler and high efficiency framework, since NVIDIA-specific optimizations can be automatically performed and new features can be added without requiring the consensus of other hardware manufacturers. Furthermore, NVIDIA is also the producer of the most widespread cards for GPU computing, namely the TESLA series (NVIDIA, 2016b), which are often used in HPC.

OpenCL is a framework for writing applications that can be executed across a series of heterogeneous computational devices that include not only GPUs, but also CPUs, Digital Signal Processors (DSPs), Field-Programmable Gate Arrays (FPGAs) and ARM processors. OpenCL is an open standard maintained and supported by the nonprofit Khronos Group consortium (Khronos, 2016). Compared to CUDA, OpenCL provides a more abstract framework, allowing direct portability of the code between hardware solutions of different vendors (NVIDIA, AMD and multi-core CPUs), at the expense of a slightly steeper learning curve and less versatile implementation of hardware-specific optimizations, e.g., concerning on-board memory and data communication with the CPU and with other GPU cards.

From the programming point of view, the CUDA and OpenCL frameworks show many similarities. In both a distinction is made between two logical parts of the code, namely a host part, to be executed on the CPU of the host machine, and a so-called device part, to be executed by many parallel threads (kernels) on the selected GPGPU device(s). Similarly, both frameworks distinguish between host and device memory, and provide functions to handle allocation and data-transfer between these. Concerning the API, this is unique in OpenCL, whereas for CUDA two APIs providing the same performance are available: the CUDA Driver API and the CUDA Runtime API. The former is also very similar to OpenCL, with a high correspondence between functions of the two frameworks.

Concerning performance, it is expected for GPU-based implementations to outperform CPU-based ones, as the workload, i.e., number of floating-point operations per second (FLOPS), increases (Lee et al., 2010). Due to its higher abstraction level and portability, OpenCL implementations have been found in the past to be slower than CUDA ones (Fang et al., 2010), although recently this gap has been reduced significantly (Kim et al., 2015) and mainly depends on the quality of the runtime implementation.

In addition, a common limitation for a generalized GPGPU-based approach is related to the size of the workload. The GPU is a kind of secondary computer within the main computer, featuring its own processor (the GPU), its own memory (the video RAM, or simply VRAM), its own conventions (instruction set, caching, data alignment, timing, etc.) and its own connectivity (usually a fast PCI-Express bus connecting the graphics card with the rest of the machine). This means that for a given task, the GPU-side execution performs a series of additional steps that are not required by its conventional CPU counterpart. A typical GPU-side computation consists in: 1) copying relevant information from system to device memory; 2) the compilation, parameterization and execution of a specific series of instructions; 3) waiting for the asynchronous execution to terminate; 4) copying the results back from device to system memory. Depending on the size and complexity of the problem, the overhead incurred by these four operations can be higher than the computational speedup provided through GPGPU.

These considerations make the choice of when and how to use GPGPU in SAR processing a more delicate matter, since typically image-wide operations (in principle a perfect match for GPU-side execution) are interleaved by smaller operations (e.g., setting up filtering parameters, sampling sub-image portions, etc.) which may introduce a severe overhead.

To gauge the impact of the aforementioned aspects and to guide the design and optimization of SARIPA, several tests, detailed in Peternier et al. (2013), were performed to analyze the behavior of common SAR-related algorithms carried out via GPGPU. In this section we discuss a performance test concerning 1D FFT calculation, since this is a core algorithm, which is heavily used within many SAR processing applications, including image focusing. The tests were carried out on Machine-A (Table 1).

We compare the highly-optimized FFT provided in the Intel's Math Kernel Library (MKL), taken as a reference FFT implementation for CPU-only performance, with GPU processing using CUDA (using

**Table 1**
Platform used for the FFT performance tests.

| | Machine-A (FFT performance test) |
|---|---|
| OS | Windows 7 |
| CPUs | 1× Intel Core i7-930 @2.8GHz (4 cores) |
| RAM | 12 GB |
| GPU | 1× NVIDIA TESLA M2050 Fermi (3 GB of VRAM) |
| Storage | Not relevant for the test |

cuFFT, released as component of the NVIDIA CUDA SDK) and OpenCL (using the FFT implementation released by Apple). For each power of two between $2^1$ and $2^{23}$, we carry out 7 iterations, each time computing an FFT of size $2^N$. The first two iterations are discarded, to let CUDA and OpenCL load all the information and allocate all their internal structures, since several of them are only initialized at the first real usage (lazy-loading). Timing results, measured using native counters such as the Read Time-stamp Counter (RDTSC) and performance counters, are based on the average of the last 5 runs (the observed variance between independent runs is <5%). Power saving, dynamic CPU frequency scaling (including Intel's Turbo Boost) and Simultaneous MultiThreading (SMT) are disabled to provide more stable and repeatable results. For the same reason, we used the single-CPU Machine-A instead of the more powerful Machine-B (used later for the image focusing tests) to avoid having to account for latencies introduced by the Non-Uniform Memory Architecture (NUMA) of the latter.

The results are reported in Fig. 1 and provide a tangible evidence of the behaviors previously explained. For large workloads (i.e., sample sizes $N > 2^{13}$), both GPU-based implementations outperform the MKL CPU-only FFT implementation. For smaller FFT lengths ($N < 2^{14}$), the GPU performance is degraded because of the overhead introduced by the data communication between host and device-memory and by the preparation of the kernels prior to their execution. In such cases, performance becomes comparable or even greater than the time required for the calculations. The impact of the overhead is clearly seen on the left side of the chart, where CUDA and OpenCL require almost the same constant number of seconds despite the growing number of samples. Concerning the two GPU implementations, the CUDA cuFFT version is slightly faster than the OpenCL-based one, mainly for larger sample sizes ($N > 2^{18}$). This speed difference confirms a measurable performance advantage of CUDA over OpenCL thanks to its closed ecosystem, which allows NVIDIA to avoid dealing with several issues that must be handled by the portable and abstract layer available through OpenCL. This aspect is also highlighted by the lower overhead required to start a cuFFT compared to an OpenCL-based one, although both the CUDA and OpenCL implementations have been executed on exactly the same hardware. Fig. 1 also confirms that GPGPU in general is only efficient when a problem reaches the critical mass required to compensate the overhead incurred for setting up and igniting the GPU-side processing. When the problem is small (in our case, $N < 2^{14}$), CPUs are a faster option than GPUs. As detailed in Section 3.2, the critical mass for efficient GPU-processing is easily reached for SAR image focusing.

The software presented in the following section is written considering all these aspects to maximize efficiency by offloading data-intensive tasks to the GPU and by keeping CPU-side computations mainly for I/O operation balancing, algorithm parametrization and multiple-process orchestration.

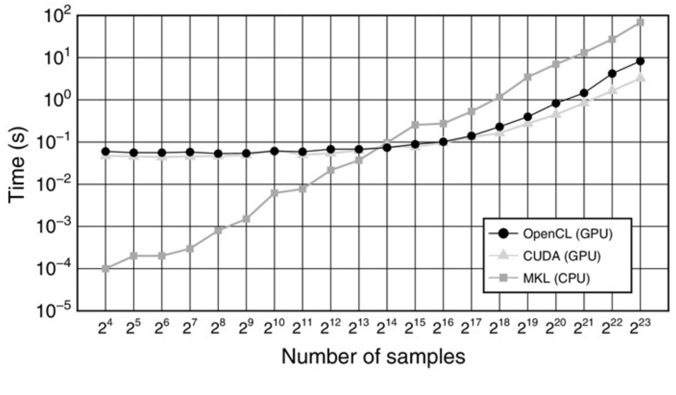

**Fig. 1.** 1D FFT performance test comparing MKL (CPU), CUDA (GPU) and OpenCL (GPU).

## 3. Methods

### 3.1. Processor architecture

The SARIPA processor is designed to work on a single high-end machine, which could indifferently be a supercomputer node, a workstation or a server computer, by deploying a locally-distributed series of computational processes that communicate through shared memory and shared message queues. This provides a very fast, zero-copy mechanism that allows processes to access memory areas that are shared with other processes. In this way, any modification is immediately visible from the other processes without incurring into the typical latency of network distributed systems.

The processor is based on the client-server architecture shown in Fig. 2. Its input consists of a list of user-specified operations to be carried out on a set of input raw data files. The output is the corresponding set of focused images. A *console application* (client) allows users to specify a list of task requests, which are registered and queued by the *SARIPA daemon* (server), and scheduled for execution on one or more *worker processes*. Each of these is associated with a detected hardware device (e.g., a GPGPU-enabled device or a CPU, depending on the selected OpenCL runtime). The server periodically monitors the overall system resources (e.g., memory use, I/O levels, CPU load) and the status of the worker processes (pending tasks) to prevent saturation of the system and to identify where to queue new tasks (load-balancing). Once all the scheduled tasks have been executed, the user receives a notification from the server.

Each worker process consists of a *processing pipeline*, which implements a sequence of operations, including signal processing algorithms (e.g., FFT, convolution, etc.), represented by a set of *filter* components, and of an instance of a SAR acceleration engine (*SARX engine* component). The latter is a library created on top of OpenCL featuring a compact API to simplify and abstract the implementation and usage of GPGPU-specific elements (such as device memory and OpenCL kernels). In the following, such a worker process instance, containing a processing pipeline, shall be referred to as *pipeline worker process*.

For tasks which have a processing duration comparable to that of the data read/write operations, a dedicated worker process containing multiple *I/O unit* components is used. This stand-alone, singleton worker process is responsible for optimizing multiple concurrent I/O requests, by internally prioritizing reading requests with respect to focused image writing operations, in order not to let the pipeline worker processes starve due to the lack of new input data. When new input information is required by one of the pipeline worker processes, it quickly creates a copy of the disk data into the shared memory allocated by the caller to immediately release disk usage. This shared memory segment (instead of the disk) is then used by the pipeline worker process that owns it for further processing. The pipeline worker processes internally rely on these shared memory segments to transfer information to/from the OpenCL buffers used by the SARX engine. Once all the pipeline operations have been executed on a specific shared memory segment, the I/O unit takes its ownership and writes the output to the disk when other pipeline worker processes are not waiting for new input data. This reduces contention of the disk, which is shared by all the processes running on the machine, by minimizing the number of disk-related operations and by dynamically changing their priority as required. In the following, such a singleton worker process containing I/O units is referred to as *I/O worker process*.

### 3.2. Focusing algorithm

The focusing algorithm implemented in SARIPA is shown in Fig. 3, and is suitable for Stripmap (e.g., ASAR IM) as well as for TOPS mode data (e.g., Sentinel-1 IW). Firstly, some relevant parameters are extracted from the raw data (e.g., changes in the Sample Window Start Time and/or in the range and azimuth sampling rates), and from the sensor

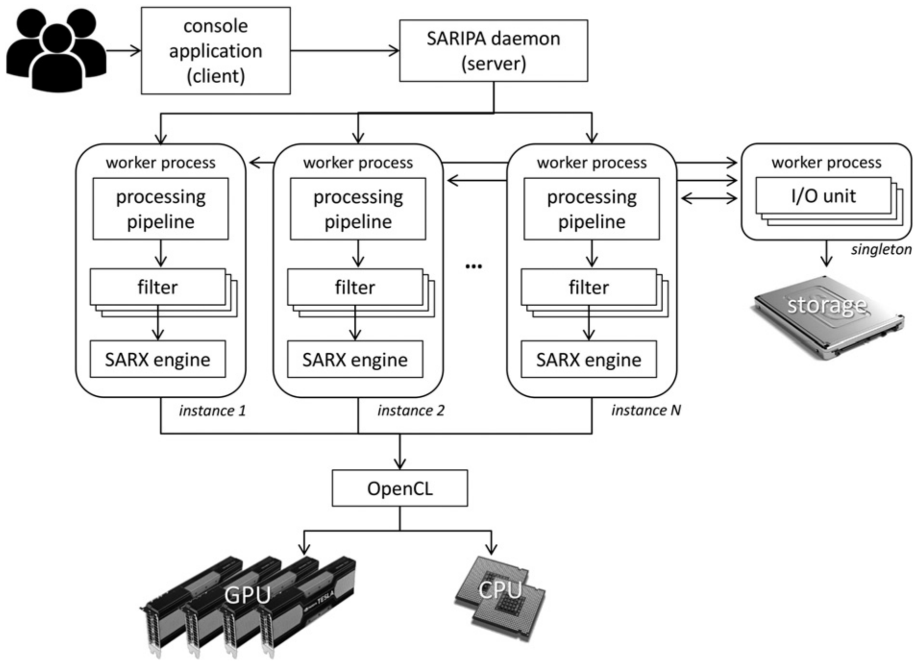

**Fig. 2.** SARIPA processor architecture.

auxiliary files, namely the instrument and calibration auxiliary files and optionally a precise state-vector file. At this stage, the azimuth frequency modulation rate is also estimated with the geometric method of Cumming and Wong (2005), since this parameter is required for the computation of some image-wide parameters for TOPS data (CLS, 2016). The steps enclosed in the dashed rectangle in Fig. 3 are then applied to the whole image for the Stripmap case, and to each burst for the TOPS case. For both acquisition modes the raw data is first decoded and then range compressed, carrying out frequency domain matched filtering for each azimuth line. Subsequently, the Doppler centroid is estimated with the method of Madsen (1989), and the data is transformed to the range-Doppler domain by applying 1D azimuth FFTs to each column of range compressed data. In this domain, autofocus is optionally carried out for the Stripmap case and azimuth compression is efficiently performed using a variant of the original Omega-k algorithm (Cafforio et al., 1991), also known as monochromatic Omega-k (Bamler, 1992). The first step of the latter consists in the so called Stolt interpolation, which is approximated by a complex exponential multiplication for each range column. Azimuth compression is then completed by transforming the data to the 2D frequency domain by applying a 1D range FFT to each data column, multiplying by a 2D complex reference function and by taking a single 2D IFFT (Cumming and Wong, 2005).

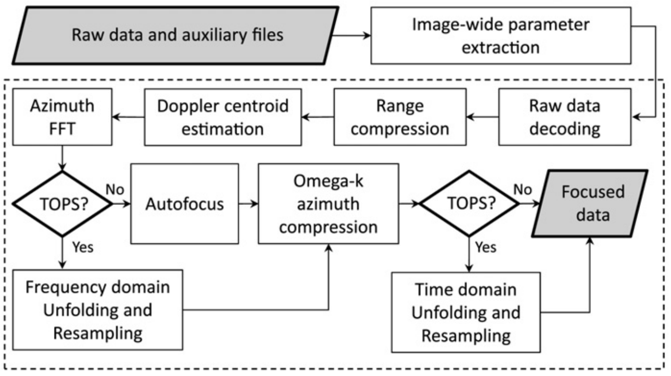
Raw data and auxiliary files
Image-wide parameter extraction
Azimuth FFT
Doppler centroid estimation
Range compression
Raw data decoding
TOPS?
No
Autofocus
Omega-k azimuth compression
TOPS?
No
Focused data
Yes
Frequency domain Unfolding and Resampling
Yes
Time domain Unfolding and Resampling

**Fig. 3.** Stripmap and TOPS focusing algorithm.

For TOPS mode acquisitions, the azimuth antenna steering requires the azimuth compression processing block to be respectively preceded and followed by a Doppler frequency domain and by a time domain unfolding and resampling step (De Zan et al., 2006). The former is required to assign the correct (un-aliased) Doppler frequency to each data line prior to azimuth compression, since the Doppler centroid variation in the azimuth dimension exceeds the Pulse Repetition Frequency of the radar. Time-domain unfolding is instead required to assign the azimuth-compressed returns to their correct zero-Doppler azimuth positions, since some of these will lie outside the burst acquisition start and stop times, due to the squinted viewing geometry associated with the azimuth antenna steering. Each unfolding and resampling block is a 1D step applied in turn to each data column, and consists of a mosaicking sub-step (i.e., a *n*-fold replication of the input data column), a deramping sub-step (i.e., a multiplication by a complex exponential), a low-pass filtering and decimation (or resampling) sub-step (e.g., carried out with a FIR filter), and a re-ramping sub-step (i.e., a multiplication by a complex exponential which reinstates the phase modulation compensated during de-ramping). A more detailed description is considered to be outside the scope of this paper and is provided in CLS (2016).

To relate to the efficiency considerations discussed in Section 2, it is useful to gauge the sizes of the data buffers and FFTs involved in the above-described algorithm. For ENVISAT ASAR IM, the input data size is roughly in the order of 30,000 lines (azimuth dimension) × 6000 columns (range dimension). For Sentinel-1 IW data, a raw burst typically contains about 1500 lines × 20,000 columns. For the processing steps requiring FFT algorithms, it is convenient to pad the data to a size, which can be expressed as any combination of powers of small integers (e.g., 2, 3, 5). An overview of the FFT sizes required for each acquisition mode is provided in Table 2. Since the execution of N FFTs of size M results in an overall workload batch size of N × M, the critical mass required for efficient GPU processing (Fig. 1) is easily reached for each of the processing steps.

## 3.3. Processing pipelines

In this section we discuss the application of the proposed architecture (Fig. 2) to the focusing algorithm described in Section 3.2. In the ENVISAT ASAR IM case, a single pipeline worker process is used to

**Table 2**
Number and size of the FFTs required by each focusing algorithm step.

| | Range compression | | Azimuth FFT | | Omega-k azimuth compression | | |
|---|---|---|---|---|---|---|---|
| | Num. of 1D FFTs | 1D FFT size | Num. of 1D FFTs | 1D FFT size | Num. of 1D FFTs | 1D FFT size | 2D IFFT size |
| ASAR IM | 30,000 | $2^{13}$ | 6000 | $2^{15}$ | $2^{15}$ | $2^{13}$ | $2^{15} \times 2^{13}$ |
| Sentinel-1 IW | 1500 | $2^{15}$ | 20,000 | $2^{11}$ | $2^{11}$ | $2^{15}$ | $2^{11} \times 2^{15}$ |

focus one image. There is also always one single instance of the I/O worker process. The same instance can be used by more pipeline worker processes when a batch of multiple images to focus is queued through the server. Each processing pipeline implements the steps shown in Fig. 3. Depending on the available hardware resources, multiple worker process instances featuring the same pipeline can be spawned by the server to allow focusing of several raw data strips in parallel. The server is responsible for spawning and shutting down all the worker processes. When the server receives a request to focus a set of raw data products, a first batch of decoding and focusing tasks is sent to the available pipeline worker processes. Each of these in turn sends a request to the I/O worker process to load the raw data and decode it within its own shared memory segment using one of the available I/O units. When I/O is finished, the pipeline worker process is notified and can carry out the focusing steps, while the I/O worker process carries out similar requests for other pipeline worker processes. When focusing is terminated, the pipeline worker process instructs the I/O worker process to write the output data to disk, and becomes available to receive the next raw data focusing task.

For the Sentinel-1 IW case, two processing pipelines have been implemented. The first one carries out raw data decoding, whereas the second one performs the remaining focusing steps in Fig. 3. Given the raw data of a single IW product, the server queues the decoding task to a single pipeline worker process and leaves the other instances available for focusing. These are used to focus several IW raw data bursts in parallel. More in detail, when the server receives the request to focus an IW raw data product, the whole contents of the data file are loaded into the system memory by the pipeline worker process where the decoding task has been queued. This process identifies and extracts the raw data bursts from the stream of space packets, decodes them, and sends one focusing request for each decoded raw burst back to the server, which in turn queues the request to the next available pipeline worker process.

For the Sentinel-1 IW case, the I/O operations are directly carried out within each pipeline worker process, rather than passing through the singleton I/O worker process. Since the ratio between the duration of the I/O operations and focusing is higher for an ENVISAT ASAR raw data file, compared to a Sentinel-1 IW burst, which is smaller in size and requires additional processing time (e.g., for the unfolding and resampling processing steps in Fig. 3).

### 3.4. Performance optimization

The proposed architecture optimizes the workload execution along the two axes shown in Fig. 2: horizontally, it improves the throughput by splitting the workload processing into several parallel worker processes; vertically, the latency required by each worker process is reduced by taking advantage of the additional computational power offered by multicore CPUs and GPGPU (using GPUs when available or multicore CPUs through the OpenCL CPU-only runtime).

Performance optimization is carried out at several levels. Firstly, the signal processing algorithms implemented in the filter components in Fig. 2 are written in OpenCL, allowing the pipeline worker processes to be executed on CPU (mainly small operations to avoid the overhead mentioned in Section 2), as well as on GPU (to offload large workloads). As mentioned in Section 1, the latter features a significantly higher level of parallelism. For instance, a modern multi-core processor with 8 cores and 2 SMT units, when optimally used, can execute a maximum of 16 threads simultaneously. A standard GPU features typically at least 512 cores that can process 512 (and more) threads at the same time.

A second level of optimization is given by tuning specific OpenCL parameters, such as the workgroup sizes and kind of host-device memory transfer, according to the underlying hardware. From this perspective, optimal settings are platform-specific and are typically documented in hardware-manufacturer guidelines and best-practice recommendations. While on the one hand OpenCL features code-level portability across heterogeneous platforms, on the other hand performance is not guaranteed. In our approach, the SARX engine component (Fig. 2) identifies, whenever possible, the characteristics of the underlying OpenCL platform and automatically tunes dynamic parameters to have a better matching between the software and the underlying hardware architecture. For example, NVIDIA provides a faster memory-copy mechanism using "memory pinning", which is automatically activated by the SARX engine when the NVIDIA OpenCL runtime is used. The size of internal OpenCL memory buffers is also dynamically allocated in function of the detected device memory. In this way, larger portions of data can be processed in one single execution on devices with larger amount of memory.

A third level of optimization is provided by the SARIPA daemon (server) component, which acts as load-balancer among the available worker processes. In this way, pipeline worker processes that are already busy are not overloaded by receiving new tasks when other worker processes are idle. The server daemon is also responsible for inspecting the underlying hardware and allocating the ideal number of worker processes according to the amount of detected resources, which allows automatic portability and adaptation. In the ENVISAT ASAR case, the I/O worker process also rearranges data read/write operations to make sure that pipeline worker processes always have fresh information to ingest. In this way, all the physical resources provided by the computer, i.e., CPU and GPU computational power, system and device memory, and disk storage are balanced, maximizing their exploitation.

## 4. Results

### 4.1. Testing environment

The test-runs described in this section are obtained with the server-class Machine-B, whose properties are listed in Table 3. The machine is configured with SMT enabled, dynamic frequency scaling (including power saving and Turbo Boost) disabled, and by assigning each worker process to one of the two NUMA nodes (one for each CPU and 32 GB of RAM). By using a fast SSD disk we can reduce I/O latencies due to data loading from disk to memory (and vice versa). Workload acquisition from external sources such as network streams of data is not considered in this work, although the proposed design is compatible with a

**Table 3**
Platform used for the focusing performance tests.

| | Machine-B (focusing performance test) |
|---|---|
| OS | Windows 7 |
| CPUs | 2× Intel Xeon E5-2670 @2.6GHz (8 cores each) |
| RAM | 64 GB |
| GPU | 4× NVIDIA Tesla K20 (5 GB of VRAM each) |
| Storage | 2× standard SSD HDs (500 MB/s); 1× high performance SSD HD (2 GB/s) |

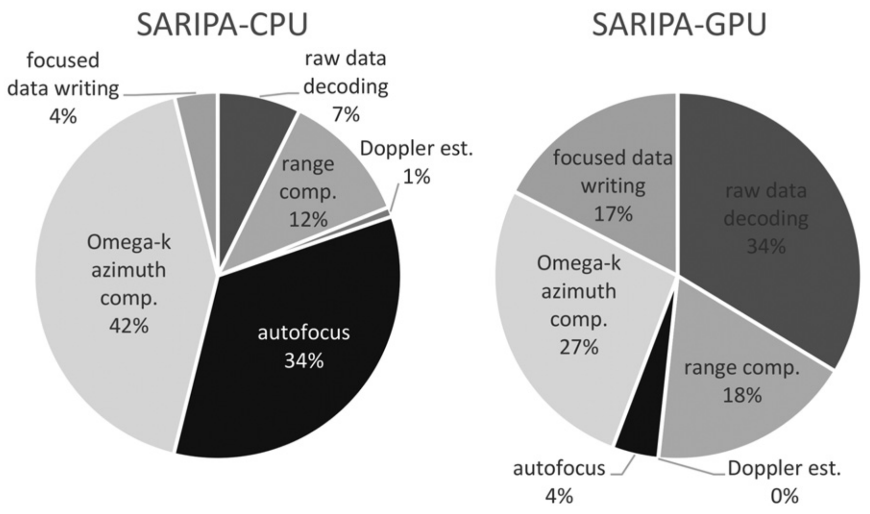

**Fig. 4.** ENVISAT ASAR IM focusing. Timing breakdown for OpenCL CPU (left) and GPU implementation (right).

potential future extension in this direction, maximizing the power of each computational node in a HPC context.

OpenCL on CPU is executed using the Intel CPU-only runtime version 15.1, while the runtime provided by the NVIDIA GPU driver is used for GPGPU.

### 4.2. ENVISAT ASAR IM

A standard raw data frame acquired by ENVISAT ASAR IM is first focused with an external software (SARscape®) to obtain a reference in terms of performance and image quality. The same frame is then focused with SARIPA, using OpenCL on CPU, as well as OpenCL on GPU acceleration. The two processing pipelines are referred to in the following as SARIPA-CPU and SARIPA-GPU respectively. The latter requires about 8.5 s, which is about half of the ~16 s Stripmap mode acquisition time, whereas SARscape and SARIPA-CPU required 83 s and 44 s respectively. The fact that SARIPA-CPU outperforms SARscape by almost 50% indicates that the on-the-fly code generation performed by OpenCL (using available hardware extensions such as SSE and AVX registers) is more efficient than standard (parallel) C++ code in exploiting the high-end dual-CPU used for the experiments.

Concerning the SARIPA-CPU and SARIPA-GPU performance, Fig. 4 (left) shows that for SARIPA-CPU the two most time-consuming steps are autofocus and Omega-k azimuth compression, which, as expected, are computationally intensive, whereas I/O operations (raw data decoding and focused data writing) take up only 11% of the 44 s required to process the image. The SARIPA-GPU case, Fig. 4 (right), is completely different, since although the azimuth compression is still one of the most time-consuming tasks, the I/O operations (which take exactly the same time as for the previous SARIPA-CPU case) amount to 51% of the 8.5 s needed to focus the image.

As mentioned in Section 3, although only one pipeline worker process is used for focusing one ENVISAT ASAR IM image, more worker processes can be used to focus several images in parallel. The SARIPA-CPU and SARIPA-GPU performance on the test machine as a function of the number of worker processes is shown in Fig. 5 (this number does not include the I/O worker process, which is always one). The end-to-end processing time per image is used as a quality factor (the lower the better). SARIPA-GPU exploits the 4 NVIDIA TESLA GPUs available on the test machine, allocating one GPU to each pipeline worker process. With 4 GPUs and 4 pipeline worker processes, 11.6 s are required to focus 4 images (i.e., 2.9 s/image). Compared to the single-GPU case, this corresponds to a speedup factor of 3, and is therefore 25% less than the ideal performance (a speedup factor of 4). This is a reasonable achievement, since while on one hand the hardware resources (number of GPUs) are increased fourfold, on the other the machine memory busses and number of hard-disks is constant, creating contention and reducing scalability.

The problem of a finite amount of resources shared by a growing number of worker processes is clearly visible in the SARIPA-CPU performance scalability (Fig. 5). In this case, the total number of cores provided by the two CPUs is shared among the available pipeline worker processes, so that the allocation of more worker processes only marginally improves the scalability of the system, to the point (5 worker processes) where more pipelines are only consuming resources without

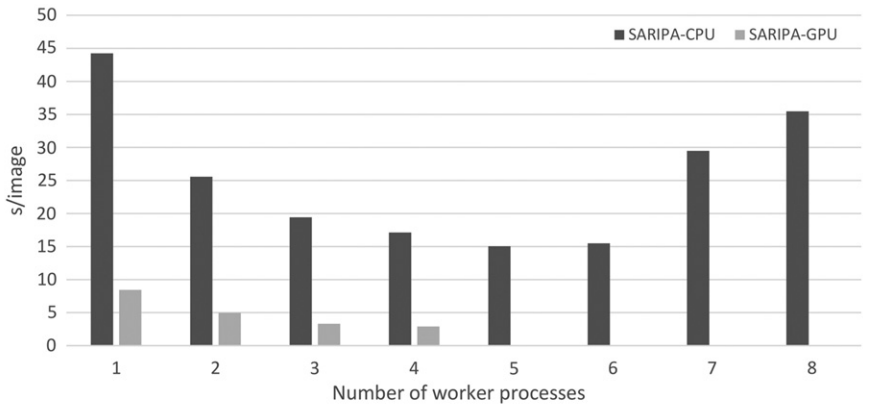

**Fig. 5.** ENVISAT ASAR IM focusing. Per-image processing times as a function of the available pipelines for the OpenCL CPU and GPU implementations.

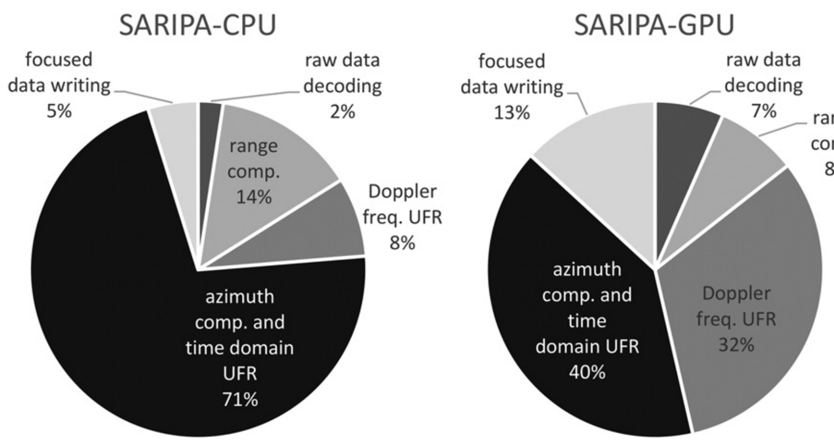

**Fig. 6.** Sentinel-1 IW (TOPS) focusing. Timing breakdown for OpenCL CPU (left) and GPU implementation (right).

bringing any additional speedup. This problem is also due to the system memory consumed by each worker process. When the CPU-only runtime of OpenCL is used, standard system memory is used instead of device memory. When >5 worker processes are used, all the available system memory is consumed and disk swapping dramatically reduces performance.

## 4.3. Sentinel-1 IW

In the Sentinel-1 IW case, there is no external non-OpenCL reference implementation to compare with, since information concerning the performance of ESA's operational processor and its hardware configuration is not publicly available. Therefore, only an inter-comparison between the SARIPA-CPU and SARIPA-GPU is presented.

The wall-time required to decode and focus a full frame (250 km, 35 bursts) on the target machine is 211 s for SARIPA-CPU, using an optimal number of 6 pipeline worker processes (Fig. 7), and 65 s for SARIPA-GPU, using 4 worker processes (again, one per available GPU). This is a near-real-time performance, considered the ~30 s TOPS mode acquisition time: in principle, the alternation of two computers like the target machine could be enough to feed a stream of raw-to-focused images for one of the Sentinel-1 satellites.

As in the ENVISAT ASAR IM case, Omega-k azimuth compression is the most time-consuming task (Fig. 6). Unlike ENVISAT ASAR IM though, I/O is less of a concern, since for each burst about 250 MB of data enters the focusing pipeline and almost the same size is written on output. For the ENVISAT ASAR IM case, the input and output data volumes were 150 MB and 1.1 GB respectively. There is an interesting difference in the CPU vs GPU performance of the Doppler frequency-domain Unfolding and Resampling (UFR) step. In this case (unique in the whole SARIPA project), CPU is actually faster than GPU. The reason is technical and linked to the hardware nature of CPUs: the Doppler UFR uses a lot of memory accesses to neighbor memory areas. Since GPUs have a limited cache support, this information is immediately lost and data must be recovered from (distant) VRAM memory each time, while on CPUs the information is more likely available in the CPU cache, making the processing faster.

Concerning scalability, with respect to an increase in worker processes, a result comparable to the ENVISAT ASAR IM case is achieved (Fig. 7).

## 4.4. Big data implications

To highlight the potential advantages of SARIPA in the Big Data context, we consider the case of a user, who requires focused imagery on a continental scale for his/her application. We assume the area of interest to be the whole of Europe (latitude 34 N to 71 N and longitude 26 W to 43E), and we consider two SAR datasets, namely the entire ENVISAT ASAR IM archive (from year 2002 to 2012) and the current Sentinel-1 IW dual-polarization image archive (from year 2014 to 31 Jan. 2017).

In the case of ENVISAT ASAR IM, the total Level 0 data (raw segments) corresponds to about 121,000 standard frames (15 s duration), with an average size of 150 MB, yielding a total data volume of 17.3 TB. The total Level 1 (IMS) product volume for the same dataset amounts to about 69.2 TB (see Table 4).

By applying the performance scored by SARIPA in Section 4.2, and considering four parallel pipelines (one per GPU), which are thus capable of processing four ASAR IM Level 0 standard frames concurrently (with a throughput of about 1 image each 3 s), the focusing of the whole dataset would require roughly 100 h to complete (i.e., four days and five hours).

In the ASAR case it is hard to quantify the time required to obtain the Level 0 products, since these are not provided through a web service, but on request. ASAR IM Level 1 products are instead available through

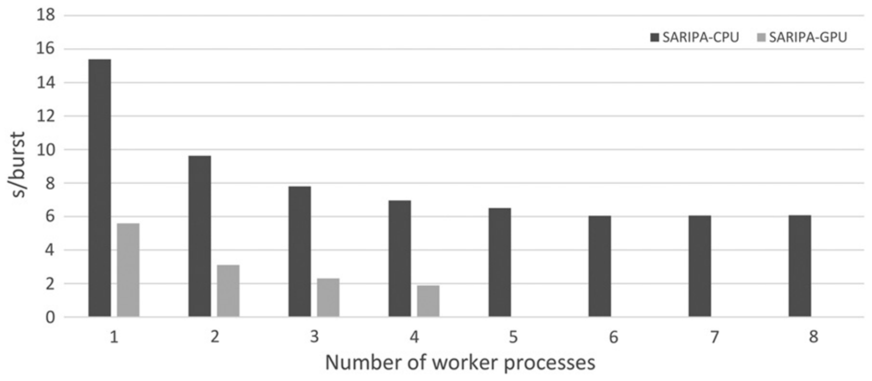

**Fig. 7.** Sentinel-1 IW (TOPS) focusing. Per-burst processing times as a function of the available pipelines for the OpenCL CPU and GPU implementations.

**Table 4**
ENVISAT ASAR IM archive over Europe (2002 to 2012).

| | Num. of frames | Size of one frame | Total size |
|---|---|---|---|
| Level 0 (RAW) | 121,000 | 150 MB | 17.3 TB |
| Level 1 (IMS) | | 600 MB | 69.2 TB |

the On The Fly (OTF) dissemination service. A user is entitled to a daily quota of up to 30 products (although more can be downloaded if there are no concurrent requests). The download of a standard Level 1 frame, in our experience, starts a few minutes after the user submits a request and takes approximatively 30 s to complete, although the exact figures depend on network connectivity and number of concurrent requests.

For the Sentinel-1 scenario, the dataset we consider consists of all the dual polarization (VV + VH) IW Level 0 (RAW) products acquired by the Sentinel-1A and Sentinel-1B sensors since the start of routine observations (Oct. 2014 for Sentinel-1A and Oct. 2016 for Sentinel-1B), until January 2017. This amounts to 65,456 files with an average size of 1422 GB for a total of about 88.8 TB of data. The same information corresponds to about 284 TB of Level 1 (SLC) products with an average size of 4550 MB (see Table 5). For both kinds of product, the sizes we report refer to the compressed files, as downloaded from ESA's Sentinels Scientific Data Hub portal. As shown in Table 5, the volume of the Level 1 products is about 3.2 times larger than for the Level 0 ones.

The SARIPA processor takes 65 s to focus a Sentinel-1 IW Level 0 image. In the case of dual polarization data, twice this time, i.e., 130 s, is required to focus the whole Level 0 product. By applying these figures to the data volumes in Table 5, about 2364 h (98.5 days) would be required to focus the entire dataset.

Concerning data access, in Table 6 we report the estimated times required to acquire the Level 0 and Level 1 products through ESA's Sentinels Scientific Data Hub for different downlink data rates. We consider one download at a time (the Scientific Data Hub policies restrict concurrent downloads to a maximum of two per user), and neglect search, service interruption, partial download resuming, and data validation times. In our experience, the average download bandwidth is currently closer to 1 MB/s than 10 MB/s or 100 MB/s.

## 5. Discussion

The near-real time performance of SARIPA, achieved at a relatively low hardware cost (the target machine in Table 3 costs about 20,000 USD at the time of writing), makes several interesting options available concerning Remote Sensing Big Data scenarios:

1) Data could be downloaded and stored in Level 0 format, the volume of which is considerably smaller compared to the Level 1 case by a factor 4 in the ENVISAT ASAR IM case (Table 5) and by a factor 3 in the Sentinel-1 IW case (Table 6).
2) For data users and service providers, it would be a viable option to carry out focusing on-the-fly, since the processing time is either negligible (in the ENVISAT ASAR IM case) or anyhow a small fraction of the time currently required to generate SAR amplitude- or phase-based value added products. This would eliminate the need of keeping a permanent local archive of focused imagery. When required, the raw images could be focused and kept in a temporary storage, which is freed once the value-added products have been generated.
3) For the many thousands of data users, which access SAR products through web-based services (OTF service for ENVISAT ASAR IM and Sentinels Scientific Data Hub for the Sentinel-1 IW case), the time which is currently required to download the Level 1 products is typically larger than that required to download the corresponding Level 0 products and focus them (see Section 4.3 and Table 6). Also, the time required for some archive management tasks (e.g., MD5 consistency checks and decompression) scales with file size, and is therefore reduced in the case of Level 0 products.

**Table 5**
Sentinel-1 IW current product archive (Oct. 2014–Jan. 2017) over the whole Europe.

| | Num. of frames | Size of one frame | Total size |
|---|---|---|---|
| Level 0 (RAW) | 65,456 | 1422 MB | 88.8 TB |
| Level 1 (SLC) | | 4550 MB | 284 TB |

**Table 6**
Download time for the Sentinel-1 IW data volumes reported in Table 5.

| | 1 MB/s | 10 MB/s | 100 MB/s |
|---|---|---|---|
| Level 0 (RAW) | almost 3 years | ~108 days | ~11 days |
| Level 1 (SLC) | 9 years and a half | almost 1 year | ~35 days |

From the performance point of view, when SARIPA-GPU is used to focus the ENVISAT ASAR IM data, the bottleneck is no longer CPU-bound, but I/O-bound. The time required to read/decode the raw data file and write the focused image takes >50% of the wall-time required to complete the process. This is due on one side to the speed of the focusing pipeline and on the other to the nature of the ASAR Level 0 format which, given a 150 MB raw data file input, generates about 1.1 GB of SLC focused data after 8 s. I/O is less of a problem for Sentinel-1 IW, since each pipeline worker process needs about 6 s to focus 250 MB of input data (one raw data burst), and generates about the same size of output (SLC focused) information. Further performance improvements would require optimizing SARIPA not only in terms of computational power, but also in terms of I/O bandwidth with regards to the available storage hardware.

Another aspect which needs to be considered concerns the memory requirements. Available GPU memory is typically a fraction of the system one. For SAR image focusing and other algorithms using FFT calculations, this limitation needs to be considered, since available OpenCL FFT routines currently require the number of samples to be aligned to some specific value. For large images, such as the ones chosen as test-cases, this might require the allocation of significantly bigger, zero-padded buffers. If these exceed the available device memory, images must be split into partially overlapping sub-tiles, decreasing efficiency.

Regardless of performance considerations, one of the most valuable results of the OpenCL approach presented in this paper is software portability. Although tests have been performed on a high-end server machine equipped with multiple GPUs, the availability of CPU-only OpenCL runtimes makes it possible to execute the same application on computers with old or even without any GPU at all (such as many HPC/cluster nodes or laptops). For example, SARIPA-CPU can focus a Sentinel-1 IW raw file in <15 min on an off-the-shelf quad-core laptop with 16 GB of memory.

## 6. Conclusions

The SARIPA project showed that focusing of ENVISAT ASAR IM and Sentinel-1 IW raw data products can be achieved in real- and near-real-time on one single machine, exploiting the proposed software architecture and GPGPU via the OpenCL framework. This improved performance opens a series of new interesting options to mitigate the Big Data problems related to current and near-future SAR processing applications, enabling the user to maintain permanent local archives only of Level 0 (raw) data products instead of bulkier Level 1 (focused) imagery. Generating the latter from the Level 0 data using the approach presented in this paper is also more time efficient for typical data access scenarios, compared to downloading the Level 1 data directly.

The target of SARIPA was to optimize the performance using a single machine, which could be a stand-alone workstation, as well as a single node of a larger HPC/cluster system. One of the key advantages of our approach is that it is not linked to a specific hardware device or vendor, allowing the exploitation of available GPUs and/or CPUs without any code duplication. Compared to more hardware- and vendor-specific solutions (e.g., CUDA), a small performance penalty can be expected but the implementation of the guidelines discussed in Section 2, in particular concerning workload sizing and memory management, can significantly reduce this drawback.

From an IT perspective, SAR processing is a complex problem, which puts all the hardware resources under stress, due to its significant memory and computational requirements. The software architecture presented in this work implements a strategy to maximize hardware resource exploitation by balancing the load placed on each system component. As such, it could be further extended and adapted also to other computationally intensive SAR processing algorithms, applied for instance to interferometric and/or amplitude data stacks.

## Acknowledgements

This work was funded by the SARscape Image Processor Accelerator (SARIPA) project, under ESA/ESTEC contract no. 3-13467/12/NL/FF/fk.

## References

Bamler, R., 1992. A comparison of range-Doppler and wavenumber domain SAR focusing algorithms. IEEE Trans. Geosci. Remote Sens. 30, 706–713.

Cafforio, C., Prati, C., Rocca, F., 1991. SAR data focusing using seismic migration techniques. IEEE Trans. Aerosp. Electron. Syst. 27, 194–207.

CLS, 2016. Sentinel-1 level 1 detailed algorithm definition. European Space Agency S1-TN-52-7445, Issue 2.0, 29. Feb. 2016 (155 pages).

Cumming, I.G., Wong, F.H., 2005. Digital Processing of Synthetic Aperture Radar Data. Artech House, Norwood (Chapter 8).

De Luca, C., Cuccu, R., Elefante, S., Zinno, I., Manunta, M., Casola, V., Rivolta, G., Lanari, R., Casu, F., 2015. An on-demand web tool for the unsupervised retrieval of Earth's surface deformation from SAR data: the P-SBAS service within the ESA G-POD environment. Remote Sens. 7 (11):15630–15650. http://dx.doi.org/10.3390/rs71115630.

De Zan, F., Guarnieri, Monti, A., 2006. TOPSAR: terrain observation by progressive scans. IEEE Trans. Geosci. Remote Sens. 44, 2352–2360.

Di Bisceglie, M., Di Santo, M., Galdi, C., Lanari, R., Ranaldo, N., 2010. Synthetic aperture radar processing with GPGPU. IEEE Signal Process. Mag. 27, 69–78.

Fang, J., Varbanescu, A.L., Sips, H., 2010. A Comprehensive Performance Comparison of CUDA and OpenCL. Proc. of IEEE International Conference on Parallel Processing (ICPP'11), Taipei (Taiwan), 13-16 Sep. 2011.

Hoshino, T., Maruyama, N., Matsuoka, S., Takaki, R., 2013. CUDA vs OpenACC: Performance Case Studies with Kernel Benchmarks and a Memory-Bound CFD Application. Proc. of IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'13), Delft (The Netherlands), 13-16 May 2013.

Imperatore, P., Pepe, A., Lanari, R., 2016. Spaceborne synthetic aperture radar data focusing on multicore-based architectures. IEEE Trans. Geosci. Remote Sens. 54, 4712–4731.

Khronos, 2016. Khronos Group. https://www.khronos.org (accessed 27.06.2016).

Kim, J., Dao, T.T., Jung, J., Joo, J., Lee, J., 2015. Bridging OpenCL and CUDA: A comparative analysis and translation. Proc. of ACM International Conference for High Performance Computing, Networking, Storage and Analysis (SC'15). New York (USA), 15-20 Nov. 2015.

Lee, V.W., Kim, C., Chhugani, J., Deisher, M., Kim, D., Nguyen, A., Satish, N., Smelyanskiy, M., Chennupaty, S., Hammarlund, P., Singhal, R., Dubey, P., 2010. Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU. Proc. (2010) of ACM International Symposium on Computer Architecture (ISCA'10), Saint-Malo (France), 19-23 Jun. 2010.

Madsen, S.N., 1989. Estimating the Doppler centroid of SAR data. IEEE Trans. Aerosp. Electron. Syst. AES-25, 134–140.

NVIDIA, 2016a. CUDA parallel computing platform. http://www.nvidia.com/object/cuda_home_new.html (accessed 27.06.2016).

NVIDIA, 2016b. Tesla supercomputing solutions. http://www.nvidia.com/object/tesla-supercomputing-solutions.html (accessed 27.06.2016).

OpenCL, 2016. The open standard for parallel programming of heterogeneous systems. https://www.khronos.org/opencl/ (accessed 27.06.2016).

Peternier, A., Defilippi, M., Pasquali, P., Cantone, A., Krause, R., Vitulli, R., Ogushi, F., Meroni, A., 2013. Performance Analysis of GPU-based SAR and Interferometric SAR Image Processing. Proc. of the 4th Asia-Pacific Conference on Synthetic Aperture Radar (APSAR'13), Tsukuba (Japan), 23-27 Sep. 2013.

Trittico, D., Fratarcangeli, M., Ferrara, R., Marra, S., 2014. Near-real-time Multi-GPU ωk Algorithm for SAR Processing. Proc. 2014 conf. on Big Data from space (BiDS'14), ESA/ESRIN, Frascati (Italy), 12–14 Nov. 2014.

Zhang, F., Li, G., Li, W., Hu, W., Hu, Y., 2016. Accelerating spaceborne SAR imaging using multiple CPU/GPU deep collaborative computing. Sensors 16, 494–513.

Zinno, I., Elefante, S., Mossucca, L., De Luca, C., Manunta, M., Terzo, O., Lanari, R., Casu, F., 2015. A first assessment of the P-SBAS DInSAR algorithmic performances within a cloud computing environment. IEEE Journal of Selected Topics in Applied Earth Obs. And Rem. Sensing 8, 4675–4686.