Performance Analysis of GPU-based SAR and Interferometric SAR image processing

Achille Peternier*, Marco Defilippi†, Paolo Pasquali†, Alessio Cantone†,
Rolf Krause*, Raffaele Vitulli‡, Fumitaka Ogushi§ and Alberto Meroni§
* Institute of Computational Science (ICS), University of Lugano (USI),
Via G. Buffi 13, CH-6904 Lugano, Switzerland
{first.last}@usi.ch
† Sarmap SA, Cascine di Barico, CH-6989 Purasca, Switzerland
{first.last}@sarmap.ch
‡ ESA-ESTEC, Keplerlaan 1, 2201 AZ Noordwijk, The Netherlands
raffaele.vitulli@esa.int
§ Exelis VIS K.K., Nakayama Bldg. 3F, 1-20-3 Hongo Bunkyo-ku, Tokyo, 113-0033 Japan
{first.last}@exelisvis.com

***Abstract*—Modern SAR and Interferometric SAR image processing make intensive usage of computer hardware resources to cope with the computational power needed to process complex images. An increasing interest in this field is being given to new approaches based on General-Purpose computing on Graphics Processing Units (GPGPU). In this paper we evaluate the performance of three common SAR algorithms: focusing, geocoding, and persistent scatterers. Each algorithm is implemented using different CPU- and GPU-based approaches that are used to evaluate the performance achievable through each technology. According to our measurements and depending on the kind of algorithm, speedups of about one order of magnitude can be obtained with relative low effort.**

I. Introduction

Recent developments in the algorithmic field, an increasing penetration of SAR imagery and related technologies into new application domains, and the imminent availability of new sensors and platforms such as PALSAR-2 and Sentinel-1, delivering regular world-wide coverage, are shifting the exploitation paradigm of SAR-data from mapping to monitoring. While optical sensors typically extend the range of information that can be extracted through Earth Observation (EO) imagery by increasing the number of electromagnetic-spectrum frequencies that can be acquired at once, SAR sensors are most effective when acquisitions are repeated over long time periods [4], [3], [1]. Data gathered in this way allows better investigating of complex phenomena (that are indistinguishable with single-shot acquisitions) and provides additional information coming from repeated measurements that supports the description of their temporal evolution. Moreover, the wide availability of SAR-data opens up to monitoring of very large areas, from continental to even global scale.

While all this significantly increases the number of domains that can potentially take advantage of SAR-data, the complexity of the algorithms and the resources needed for processing very large amounts of information introduce a series of issues related to the efficient solving of computationally-intensive problems [2], [6]. The adoption of solutions based on General-Purpose computing on Graphics Processing Units (GPGPU) to address the aforementioned issues is very attractive and looks like the perfect match since these hardware infrastructures provide:

- Massively-parallel processing capabilities (up to several thousands of cores) available on single workstations (Personal Supercomputers);
- Hardware deployments built on top of widely available, standard products such as various families of GPUs (ranging from laptops to multiple-GPU servers);
- Programmability through standard languages (like C/C++) and porting of existing code with relative low effort;
- High code portability across different hardware platforms.

In this paper we consider different frameworks that enable the hardware-efficient implementation on multicore CPU- and GPU-based platforms of representative SAR- and InSAR-specific algorithms, such as focusing, geocoding, and Persistent Scatterers (PS) analysis. For each algorithm, we evaluate the improved computational efficiency provided by GPU-based approaches versus a multicore-optimized, CPU-based implementation as baseline.

II. Testing Environment and Methodology

Experiments have been run on a desktop PC equipped with an Intel 2.8 GHz i7-930 quadcore CPU with 12 GB of RAM and a Nvidia Tesla C2050 running at 1.15 GHz with 448 CUDA cores and 3 GB of GDDR5 ECC RAM. During experiments, GPU ECC memory has been constantly used.

Performance measurements have been conducted on top of the robust tools and know-how maturated in our related work [15], [14]. Energy saving, dynamic CPU frequency scaling, and Turbo mode have been disabled for improved experimental repeatability. All the measurements reported in this paper are computed as the average of at least five independent runs. We observed less than 5% of overall variance between independent runs.

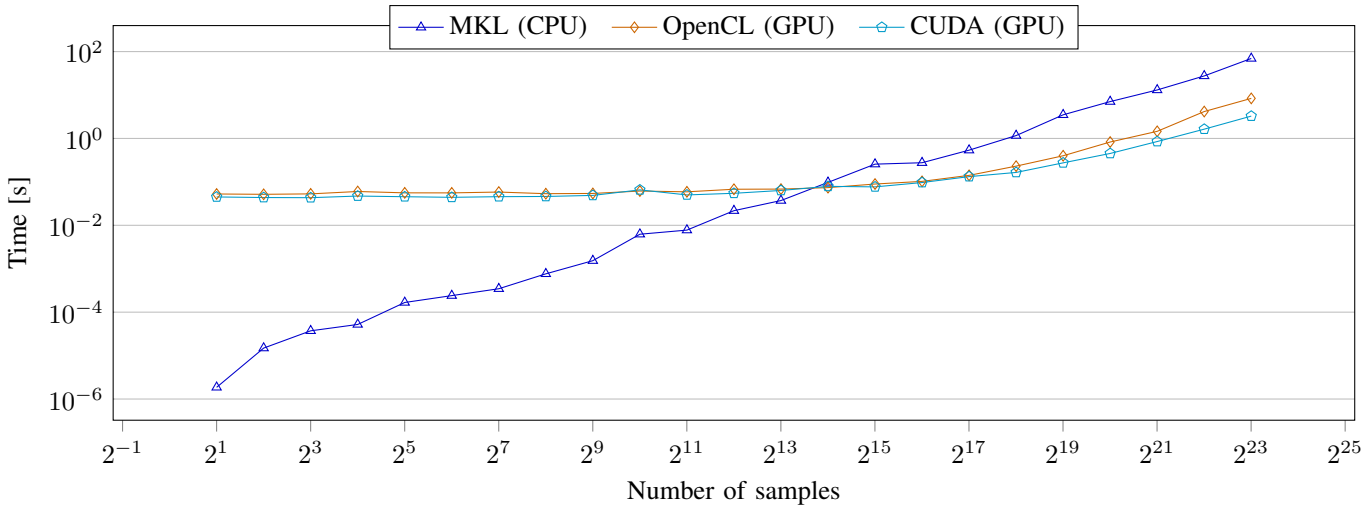

Fig. 1. FFT scalability test on CPU and GPU using an increasing number of samples (N) and three different implementations. Logarithmic scale is used on both axes.

A. Target Frameworks

Several frameworks have emerged over the last years as the most widely used to leverage the computational power provided by modern multicores and GPUs. Among the available solutions, in this paper we focus on these frameworks:

- **Thread Building Blocks (TBB).** Intel developed TBB[1] as a series of C++ template classes designed for taking advantage of multicore processors. TBB reduces software development complexity introduced by low-level multi-threading structures such as threads, locks, etc. With TBB, developers create "tasks" that are automatically parallelized by the runtime. Recently, TBB has been extended to take advantage from the Intel Xeon Phi GPU-like coprocessor [11].
- **Math Kernel Libray (MKL).** Intel's MKL is a highly optimized library implementing a series of common scientific and engineering functions, such as Fourrier transforms, sparse solvers, vector math, etc. It supports Intel and compatible processors.
- **Open Computing Language (OpenCL).** OpenCL is a framework for writing applications that can be executed across a series of heterogeneous computational devices including CPUs, GPUs, DSPs, FPGAs, ARM processors, etc. OpenCL is an open standard maintained and supported by the no-profit consortium Khronos Group[2].
- **Compute Unified Device Architecture (CUDA).** Nvidia developed CUDA as a platform integrating a parallel computing runtime and a programming model running (solely) on their GPUs. CUDA is available as a set of APIs, compilers and tools[3] that eases the creation of software using GPUs to speedup its processing.

[1] http://threadingbuildingblocks.org/
[2] http://www.khronos.org/opencl/
[3] https://developer.nvidia.com/cuda-toolkit

III. Evaluation

In this section we evaluate the CPU and GPU performance of three common SAR algorithms implemented using the frameworks previously described.

A. Focusing

Most of the focusing techniques currently used in SAR imagery intensively use the Fast Fourier Transform (FFT) for doing signal processing. For this reason, the computational efficiency of the FFT implementation is a critical factor determining the overall performance of the focusing procedure, where a faster FFT directly translates into shorter processing times.

Our first experiment is about evaluating three different FFT implementations running on CPU and GPU. On CPU, we use Intel's highly optimized MKL FFT, while on GPU we test an OpenCL implementation (released by Apple) and a CUDA version (cuFFT, released as component of the Nvidia CUDA SDK). The experiment consists in computing a series of FFTs using an increasing number of samples (N). Results are reported in Fig. 1.

According to our measurements, the two GPU-based implementations overcome the CPU version of the FFT. Independently of the programming framework being used, the massive amount of parallel hardware resources featured by the GPU provides a clear speedup over the CPU implementation starting already when a few thousands of samples are used. For instance, with $N = 2^{23}$, the CUDA implementation is 21 times and OpenCL 8 times faster than the MKL version. MKL on CPU is slightly faster than GPU implementations only when small numbers of samples are used (i.e., for $N < 2^{14}$). The reason is that GPUs run into overhead due to preparing kernels for execution and to data-communication costs between host- and device-memory. For small values of N, the overhead voids the speedup obtained through the improved parallelism provided by the GPU.

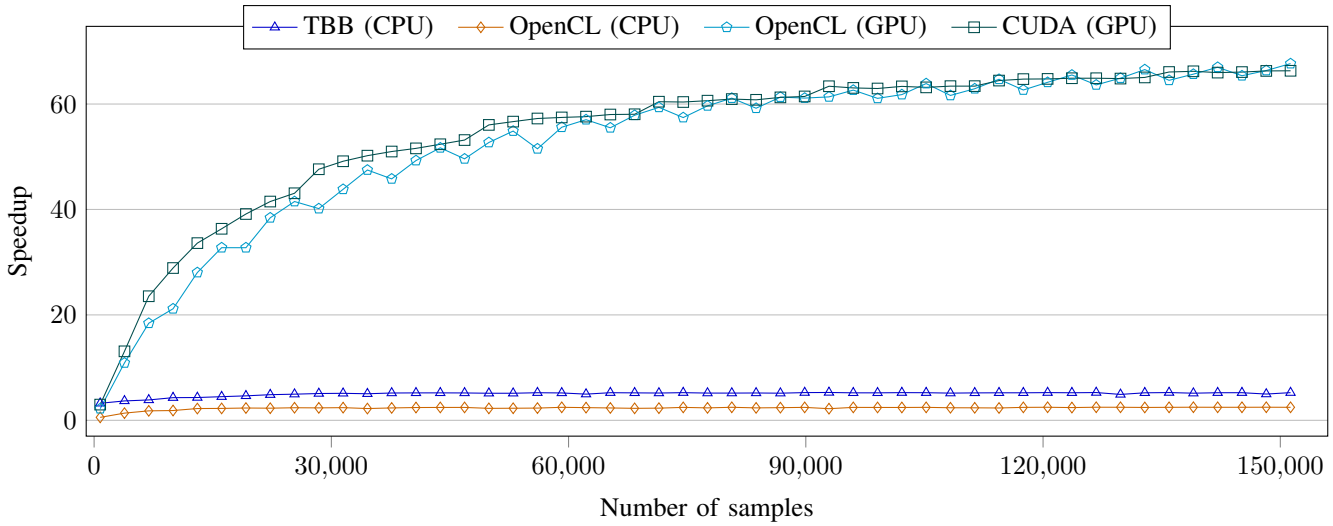

Fig. 2. Geocoding scalability test using a growing number of samples and five different implementations (both on CPU and GPU).

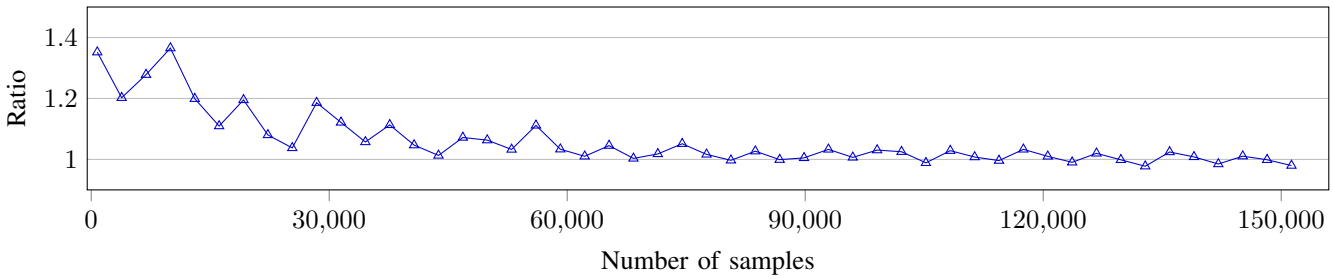

Fig. 3. OpenCL/CUDA execution time ratio.

B. Geocoding

In this experiment, we evaluate three different geocoding implementations. On CPU, we used TBB to write a highly optimized routine taking advantage of the available cores. The second implementation is written in OpenCL and we run it using a CPU-only OpenCL context (i.e., using the same resources used by the first TBB implementation) and a GPU-accelerated OpenCL context. The third implementation is written in CUDA. Results in Fig. 2 show the speedup obtained by the four different configurations with regard to a baseline obtained running a single-core, sequential implementation.

Our measurements show that GPU-based approaches significantly improve performance as soon as large data-chunks are simultaneously processed on the accelerated device, being up to 70 times faster than the baseline and up to 15 times faster than the multicore-optimized TBB implementation. It is interesting to mention that the CPU-only OpenCL implementation, which runs on exactly the same code used for the OpenCL GPU configuration, still provides a reasonable speedup when compared to the baseline and to TBB. Although CPU-only OpenCL is not as optimized as TBB, thanks to OpenCL's code portability, this performance comes "for free" when the CPU is used as a OpenCL device.

Fig. 3 additionally shows that the two GPU-based implementations have a very similar asymptotic performance. From this perspective, both OpenCL and CUDA implementations provide a comparable performance, with OpenCL supporting cross-device code portability as supplementary feature.

C. Persistent Scatterers

Two PS implementations are evaluated using three different configurations. We use TBB on CPU and OpenCL on GPU. As we already did in the previous experiment, we take advantage of OpenCL code portability to run the same OpenCL kernel using an additional, CPU-only configuration.

PS is an algorithm for studying how stable natural reflectors are evolving over time, based on long temporal series of interferometric SAR images. For this reason, we used three different image sizes in our performance evaluation: 812 columns by 202 rows, 6664 by 1223, and 11976 by 3202. Results are depicted in Fig. 4.

According to our measurements, the GPU implementation is between 3 and 10 times faster than the TBB optimized CPU implementation. It is interesting to remark that both the TBB and OpenCL CPU-only implementations perform in a similar way, while in the previous experiment TBB was faster. This means that, in some cases, CPU-only OpenCL can compete in performance with highly-optimized frameworks whilst keeping cross-device portability.

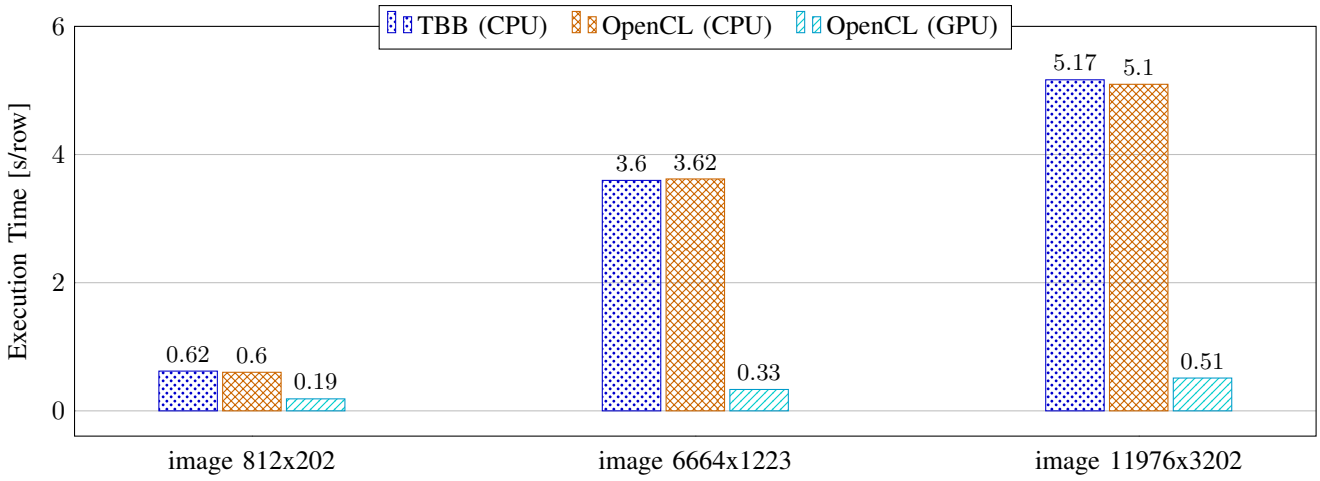

Fig. 4. Persistent scatterers performance comparison experiment using different image sizes and three different configurations.

When smaller images (e.g., 812x202) are used on the GPU, the overhead introduced by OpenCL to copy memory buffers and invoke kernels partially mitigates speedup gains, while larger images maximize the efficiency of the GPU pipeline.

IV. Conclusions

New SAR imagery processing approaches based on GPGPU are gaining increasing interest in the community [13], [8], [12], [9], [10], [5]. While the hype about GPGPU and potential speedup has been somehow mitigated [7], GPUs still provide a significant amount of parallel computational power at reasonable conditions, both in terms of code-migration complexity and infrastructure costs. According to our experiments using common SAR and InSAR imagery algorithms, a significant speedup is achievable by including GPUs in the processing pipelines. Depending on workload sizes and algorithms, GPU-based implementations have been able to increase performance by up to 21 times with regard to the multicore-optimized baseline used in our experiments. On the other hand, small workloads executed on GPU can incur performance degradation due to the overhead required for data-migration and kernel-execution setup.

Acknowledgment

This work is partially funded by the European Space Agency (ESA) with the SARSCAPE Image Processor Accelerator project.

References

[1] P. Berardino, G. Fornaro, R. Lanari, and E. Sansosti. A new algorithm for surface deformation monitoring based on small baseline differential SAR interferograms. *IEEE Transactions on Geoscience and Remote Sensing*, 40(11):2375–2383, 2002.

[2] W. Fan and A. Bifet. Mining big data: current status, and forecast to the future. *SIGKDD Explor. Newsl.*, 14(2):1–5, 2013.

[3] A. Ferretti, C. Prati, and F. Rocca. Nonlinear subsidence rate estimation using permanent scatterers in differential SAR interferometry. *IEEE Transactions on Geoscience and Remote Sensing*, 38(5):2202–2212, 2000.

[4] F. Holecz, M. Barbieri, A. Cantone, P. Pasquali, and S. Monaco. Synergetic use of multi-temporal ALOS PALSAR and ENVISAT ASAR data for topographic/land cover mapping and monitoring at national scale in Africa. In *Proc. of the International Geoscience and Remote Sensing Symposium (IGARSS)*, volume 2, pages 5–8, 2009.

[5] X. Jin and S.-B. Ko. GPU-based parallel implementation of SAR imaging. In *Proc. of the International Symposium on Electronic System Design (ISED)*, pages 125–129, 2012.

[6] S. Kaisler, F. Armour, J. A. Espinosa, and W. Money. Big data: Issues and challenges moving forward. In *Proc. of the 46th Hawaii International Conference on System Sciences (HICSS)*, pages 995–1004, 2013.

[7] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey. Debunking the 100x GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU. *SIGARCH Comput. Archit. News*, 38(3):451–460, 2010.

[8] P. Lindstrom and J. D. Cohen. On-the-fly decompression and rendering of multiresolution terrain. In *Proc. of the SIGGRAPH symposium on Interactive 3D Graphics and Games (I3D)*, pages 65–73, New York, NY, USA, 2010. ACM.

[9] B. Liu, K. Wang, X. Liu, and W. Yu. An efficient SAR processor based on GPU via CUDA. In *Proc. of the 2nd Intenational Congress on Image and Signal Processing (CISP)*, pages 1–5, 2009.

[10] Y. Lu, K. Wang, X. Liu, and W. Yu. A GPU based real-time SAR simulation for complex scenes. In *Proc. of the international Radar conference - Surveillance for a Safer World (RADAR)*, pages 1–4, 2009.

[11] G. Misra, N. Kurkure, A. Das, M. Valmiki, S. Das, and A. Gupta. Evaluation of Rodinia codes on Intel Xeon Phi. In *Proc. of the 4th International Conference on Intelligent Systems Modelling Simulation (ISMS)*, pages 415–419, 2013.

[12] S. Nilakantan, S. Annangi, N. Gulati, K. Sangaiah, and M. Hempstead. Evaluation of an accelerator architecture for speckle reducing anisotropic diffusion. In *Proc. of the 14th international conference on Compilers, architectures and synthesis for embedded systems (CASES)*, pages 185–194, New York, NY, USA, 2011. ACM.

[13] J. Park, P. T. P. Tang, M. Smelyanskiy, D. Kim, and T. Benson. Efficient backprojection-based synthetic aperture radar computation with many-core processors. In *Proc. of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, pages 28:1–28:11, 2012.

[14] A. Peternier, D. Ansaloni, D. Bonetta, C. Pautasso, and W. Binder. Hardware-aware thread scheduling: the case of asymmetric multicore processors. In *Proc. of the 18th International Conference on Parallel and Distributed Systems (ICPADS)*, 2012.

[15] A. Peternier, D. Bonetta, W. Binder, and C. Pautasso. Overseer — Low-level hardware monitoring and management for Java. In *Proc. of the 9th international conference on the Principles and Practice of Programming in Java (PPPJ)*, 2011.