Multi-mode Real-Time SAR On-Board Processing

Conference Paper · June 2016

CITATIONS
5

READS
460

4 authors, including:

Octavio Ponce

40 PUBLICATIONS **344** CITATIONS

SEE PROFILE

Stefan Valentin Baumgartner

German Aerospace Center (DLR)

122 PUBLICATIONS **972** CITATIONS

SEE PROFILE

Rolf Scheiber

German Aerospace Center (DLR)

283 PUBLICATIONS **3,879** CITATIONS

SEE PROFILE

Multi-mode Real-Time SAR On-Board Processing

Russel Que, Octavio Ponce, Stefan V. Baumgartner, Rolf Scheiber
German Aerospace Center (DLR), Microwaves and Radar Institute
Oberpfaffenhofen, Germany
E-mail: russel.que@dlr.de

Abstract

A novel distributed computer framework for onboard processing is presented. It enables real-time high-resolution synthetic aperture radar (SAR) processing and multi-channel ground moving target indication (GMTI). To handle the different applications running on the onboard processor, a new scripting language was developed. The framework is capable of running algorithm implementations for multicore processing or graphical processing unit (GPU), e.g., fast-factorized back-projection (FFBP) and direct back-projection (DBP) algorithms . The software architecture of the framework and the operational modes as well as some applications are discussed in detail. Furthermore, first experimental results obtained with the framework are presented.

1 Introduction

In the past two decades, synthetic aperture radar (SAR) has been widely used for Earth observation. Due to its weather and day-light independence, SAR is suitable to monitor the Earth's dynamics. One of the limitations of SAR, compared to other remote sensing sensors, is the high-computational burden required to obtain a well-focused image. In the past few years, several attempts have been done to achieve real-time SAR imaging [1, 2, 3, 4]. In [1, 2], the proposed solutions were based on frequency-domain algorithms because the computational burden mainly relies on FFTs. The operational mode was stripmap SAR, and digital signal processors (DSP) and graphic processing units (GPU) were used as hardware. In [3, 4], time-domain approaches were used for real-time processing in the spotlight SAR imaging mode. The fast factorized back-projection (FFBP) was the chosen algorithm, since it has a computational load similar to the frequency domain approaches. Most of these realisations were only supporting single channel SAR image formation. However, polarimetric modes as well as ground moving target indication (GMTI) aks for parallel processing of several channels.

In this paper, a real-time multinode on-board processor framework is proposed. As first functional application case, a tailored solution of the FFBP is implemented to support linear and non-linear modes, e.g., stripmap and circular SAR [5], as well as a knowledge-based GMTI algorithm [6]. The challenge in a real time scenario on top of a fast algorithm is not just the computational environment in which the algorithm will run, but also the control of the processor, the integration of the input real-time data and the management of the resulting images and meta data, which are the primary focus of this paper.

2 Bifrost Real-time Multiprocessor Framework

A suitable framework must offer fast prototyping, just like in most high-level languages (IDL, Matlab, Python). However, in the case of radar applications requiring real time output products like GMTI, none of the high-level languages has a unified approach to multi-node computing or high performance computing (HPC) that fits a fully connected processing topology. One could perhaps use Message Passing Interface (MPI), however, many researchers will be working on the framework. Hence, it is prudent that the framework provides an easy programming interface and be as well structured as any of those languages where each application can have its own function module 'toolbox'. Then all one has to do is divide one's application into different functional parts, then write each algorithmic part in C++ or others, and then lastly, write the workflow script that describes the interconnection of all those parts. The first attempt to implement such a framework for SAR processing has been done in [7]. However, developed for DLR's next generation airborne SAR systems, the new framework, named 'Bifrost' after the rainbow bridge between worlds in norse mythology, is completely new while retaining some features from its predecessor. The focus this time is on a unified variable system for data streaming and communication, a more powerful scripting to enable ease in test development and an operator-friendly interface during realtime operation.
The framework is best understood by simply listing the features it supports. The technical aspects of these software implementations are not covered here but nonetheless enumerated to give an overview on the visible interface and the underlying capabilities of the framework:

1. The radar operator uses a GUI-based user-friendly

interface for processing radar raw data into SAR images or info-based products (GMTI).

2. A viewing widget is implemented to display outputs of the processor and tools like K Desktop Environment (KDE) Marble [10] (similar to Google Earth) are embedded in it.
3. An Integrated Development Environment (IDE) for testing the function modules, write the workflow scripts, view workspace variables, check debug variables from a function in another node, plot/display variable contents (see **Figure 1**).
4. Interface to a separate communication protocol of the radar hardware control system.
5. Universal interfaces to all I/O hardware components like ADCs and navigation devices, be they in the form of TCP, FPGA, UDP, etc.
6. The universal interface also includes a file interface for offline access of raw data, necessary for offline processing and algorithm development.
7. Algorithms/functions to be implemented are structured into toolboxes by its functional group.
8. Algorithms/functions can be executed in any node and are able to connect to a module in another node.
9. Any stream input's fan-in of the module is limited to 1 but the output fan-out can be broadcasted to any number of recipients.
10. A network wide unified variable system was created such that developers can easily access data streams of varying types without writing a specific parser for each type. The variables support all basic numeric types including complex numbers, arrays and structures of all types.

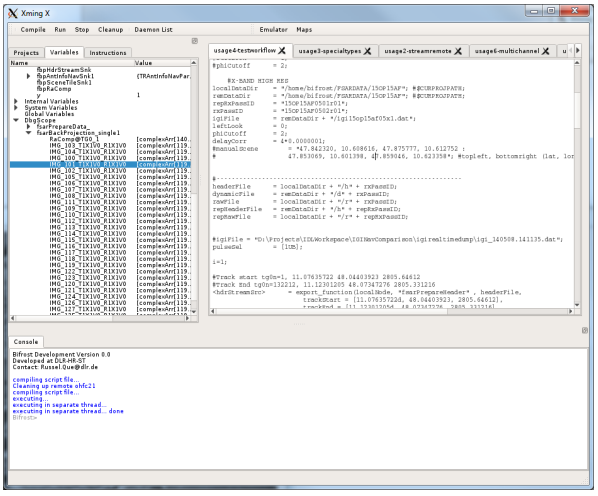


Figure 1: IDE: The left pane contains the project tree, variables created by the script and debug variables sent by functions from the nodes. To the right are the workflow scripts editor. The bottom pane is the console for receiving warnings and errors. The top buttons are used for running, stopping and launching the widget for viewing products.

11. A workflow script was designed using the variables mentioned previously and fitted with standard constructs for assignment, arithmetic operations, structures, arrays and others. Its special feature is a syntax construct to spawn functions asynchronously in any part of the network.
12. It is equipped with multi-threading safeguards to APIs like FFTW, Graphics Processing Unit (GPU), etc.

2.1 Software Architecture

In order to control the flow of data, from a function in one node to a function in another node, a scripting language is needed. The idea is to have a workflow script using a functional programming approach common to almost all high-level languages, as shown in **Figure 2**. To manage the execution of the script, two applications/processes are implemented, namely, the "Master App" and the "Daemon". The master app takes care of the compilation of the script and its execution. The execution of the script includes basic computations, like assignment, arithmetic, concatenation, standard function calls, etc. But most importantly it has a dispatch routine to export a toolbox function into a node by communicating with the remote handler's daemon. Thereafter, the intercommunication of these toolbox functions are left to be handled between daemons. The master app waits for the TCP streams of the function to be established, then proceeds to execute the next line in the script.

At the heart of both processes is the session handler which acts both as the script interpreter and function dispatcher. If the code calls for a basic computation, it executes it right away and then returns. However, if it receives a request to dispatch a function, it executes it by calling a thread. The function then runs and lives in the daemon indefinitely until aborted or until it has completed its execution. While the dispatched function is running it can accept data streams and send out data streams as well, thereby completing the chain processor as dictated by the workflow script.

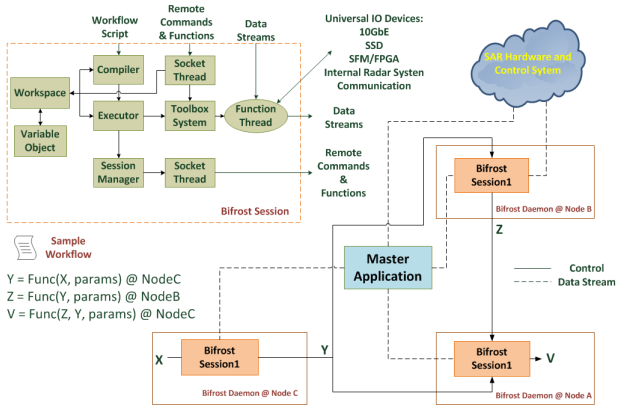


Figure 2: Bifrost architecture: The session module serves as the execution and communication engine for all function modules located in different nodes.

2.2 Operational Modes

The framework described in the previous section provides the platform where one can implement any application distributed over a network. In this case, the application corresponds to one of the operational modes of the radar system. In order to best utilize all resources in the network, the application is divided into functional parts (e.g. navigation, range compression, focusing algorithm, post processor e.g., polarimetry, GMTI). It is typical for applications like GMTI or polarimetric SAR imaging that the multiple channels required be processed in parallel by spawning the focusing algorithm at different nodes. Afterwards, the results are combined by the post processor corresponding to the application. Pertaining to the navigation module, it may not be as computationally intensive as the rest, nonetheless, it plays an important role in conditioning the realtime navigation data stream which may contain unwanted clicks and jumps. It is also responsible in converting the navigation data into other forms such as rotation matrices.

2.2.1 Fully polarimetric SAR imaging

The radar data processing is implemented using time-domain fast factorized-backprojection (FFBP) [5, 3]. The FFBP has been tailored to be used for linear and non-linear synthetic apertures, e.g., stripmap and circular SAR, and it is able to focus at a custom resolution. Additionally, the direct back-projection (DBP) has been implemented for reference and comparison purposes. The execution of both can be done either in an n-core CPU or in a GPU.

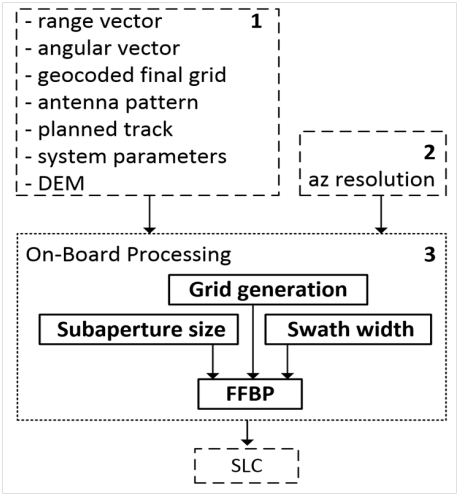


Figure 3: Processing chain of the real-time processing with the fast factorized back-projection (FFBP). 1 and 2 are input parameters to the on-board processing (indicated with the number 3). The output is the single-look complex (SLC) image.

An overview of the processing chain is given in **Figure 3**. In order to reduce the computational burden, there are parameters that can be computed during the mission planning, such as range and angular vectors of the polar grids, the geocoded final grid, the antenna pattern, planned track and the system parameters. In order to achieve real-time imaging, there exist a compromise between the desired geometric resolution and the swath width. The main differences between stripmap and circular SAR during the on-board processing are the subaperture size and the geocoded final grid. In stripmap SAR there should be an overlap of 50% to avoid discontinuities, whereas in the CSAR mode subaperture overlap is not necessary. Regarding the geocoded final grid, in stripmap SAR it will be filled as the platform moves in the along-track direction, whereas in the CSAR mode all pixels will be illuminated with the first SLC, thus being able to potentially provide constant monitoring.

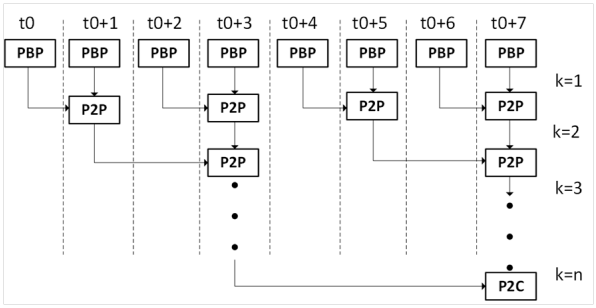


Figure 4: Fast factorized back-projection (FFBP) dependencies over time for the real-time processor. Depending on the desired resolution, the processing chain can be stopped at a given time $t_0 + i$ and stage k. PBP indicates the polar back-projection, while P2P and P2C refers to polar-to-polar interpolations and polar-to-cartesian interpolation.

As known from the literature, one of the main advantages of focusing with time-domain approaches is the accuracy it provides, since no reference track is needed. Specifically, for the FFBP the Bifrost can start processing for the very beginning of the acquisition with just few pulses. In order to fit the FFBP for real-time imaging, the sub-aperture dependencies should be understood (see **Figure 4**). With this information the on-board processor is able to decide at what instant in time $t_0 + i$ and at what stage k the FFBP should interpolate to the geocoded grid.

2.2.2 Ground moving target indication (GMTI)

For performing GMTI the *a priori* knowledge-based algorithm discussed in detail in [8] is implemented. This algorithm takes into account the known road network obtained from the freely available OpenStreetMap database [9]. The road axes of interest are directly mapped into the range-compressed and clutter-suppressed data array. Only small areas around the road axes need to be processed as depicted in **Figure 5**. This is one of the major reasons why the algorithm achieves real-time capability with manageable processing hardware effort.

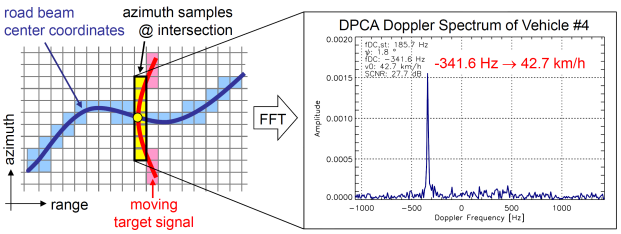


Figure 5: Principle of the *a priori* knowledge-based GMTI algorithm.

In the near future, a conventional post-Doppler (PD) space-time adaptive processing (STAP) algorithm as well as a novel *a priori* knowledge-based PD STAP algorithm will be embedded into the Bifrost framework. First results obtained from these algorithms are presented in [6].

3 Experimental results

The framework and the algorithm prototypes were tested in off-line mode using available F-SAR data. To process a single channel X-band data with 384 MHz bandwidth, the workflow is subdivided into 4 function modules, one of which is the actual focusing algorithm using backprojection while the rest of the modules are navigation, range compression, and radar settings parser. The illuminated scene on the ground is automatically selected and divided into tiles by the navigation module. This is done by selecting the region sandwiched by the near range and far range track projections on the ground. **Figure 6** shows the uppermost yellow line corresponding to the near-range track, while the blue line traces the track of the antenna main beam which looks squigglier than the near-range track due to the platform's attitude being factored in. The scene is depicted to be in the middle of processing and thereby shows the tiles popping into view. The tiles are geocoded to the mapping widget Marble, thus providing the operator a quick visual evaluation of the radar performance.

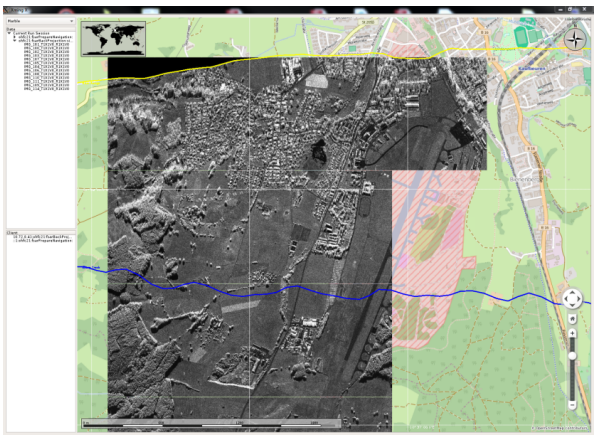


Figure 6: Embedding Marble to the output display widget. Image is processed with 4 looks DBP-GPU.

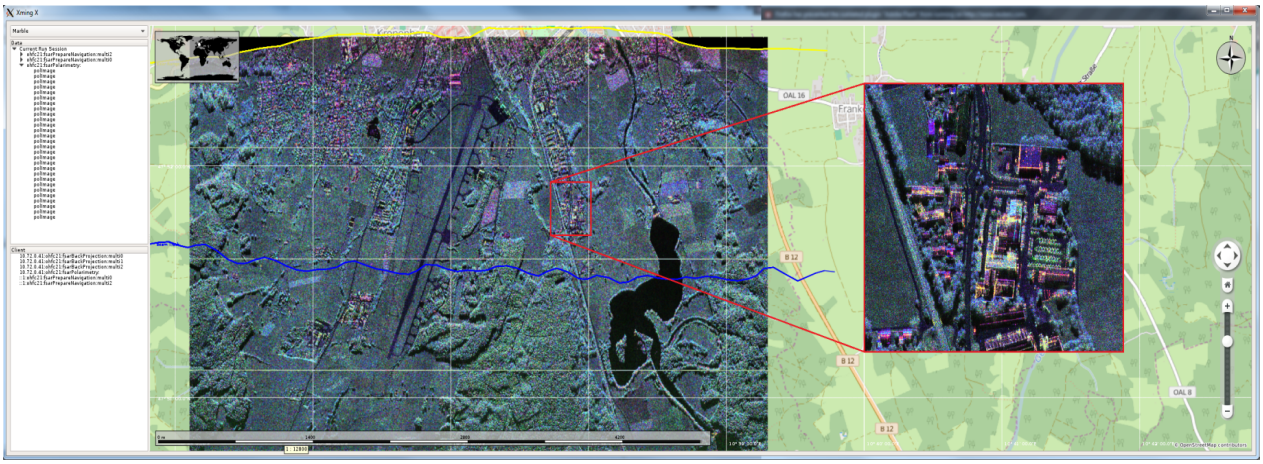
A test tile of size 1550 m azimuth x 930 m range of 50 cm resolution was processed with algorithms implemented as DBP-GPU and FFBP-CPU combinations. Both being configured with single-look, 0.5 m resolution and no presumming, the DBP-GPU implementation with 7° integration took 136 seconds while the preliminary FFBP C++ implementation took 480 seconds and using 4 cores at most. The FFBP implementation is currently undergoing optimization tests so as to make it as fast as possible and is expected to perform even better in future versions. Real-time operation of FFBP require either a trade-off between resolution and swath width or by offloading the execution to a GPU.

Once the images are generated for each channel, those can then be fed into another function module called 'polarimetry'. In this module, phase calibration is performed together with a Pauli polarimetric decomposition to produce the output as shown in **Figure 7**. As for the radiometric calibration, it is applied beforehand inside the focusing algorithm module by using the Tx and Rx antenna diagrams and navigation data.

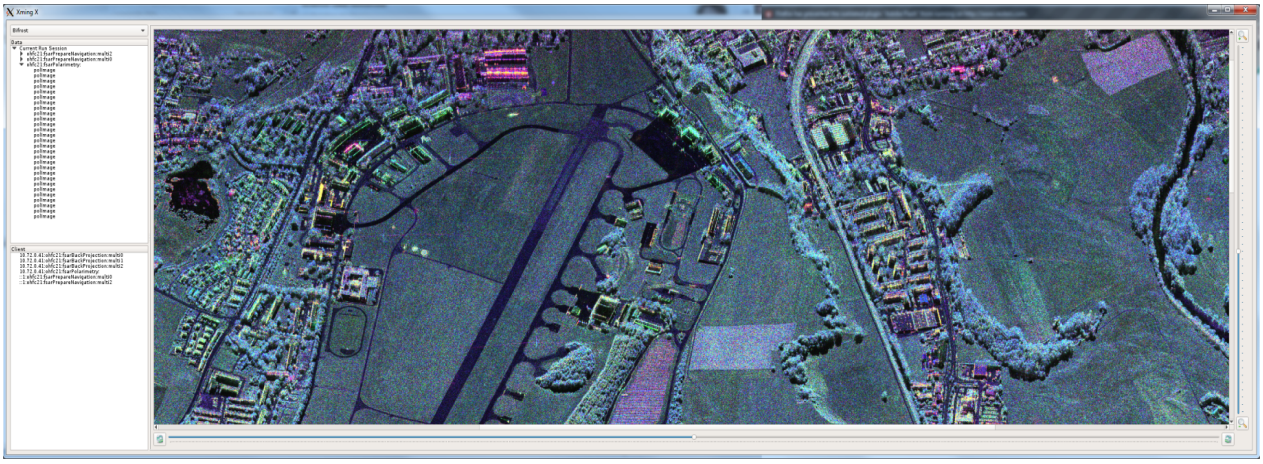
The whole scene in **Figure 7a** was processed in about 10 minutes (parallel execution of 3 polarimetric channels), with settings of 2° aperture in azimuth, single-look, 1 m x 1 m output grid, no presumming and using DBP-GPU. In order to investigate real-time operation, with a bit of trade-off in image quality, the azimuth aperture was decreased to 0.8° and an azimuth presumming of 4 was used. This took 3.5 minutes of processing. The tests for this Polarimetry application were done in a workstation with Intel Core i7-4790K (for range compression, presumming and calibration) and a single NVIDIA GeForce GTX 960 Card (for the DBP algorithm). Included already in this processing time is the latency incurred in performing a block FFT-based presumming. The workstations used here are first order test cases and do not reflect the future on-board configuration. The final operational hardware is still in development and hopefully surpasses the current test configurations.

4 Conclusions

The initial focus of the development has been on the conceptualization, design and C++ implementation of the real time processing framework. Test algorithms such as knowledge-based GMTI, single-channel quick-look and multilooking, and polarimetry were executed in the framework. The framework is scalable to the amount of computing hardware and therefore flexible enough to meet the real time requirements of an application. This has the advantage of being able to estimate the hardware requirements of an operational SAR processing system given a specific target application. Moreover, by using a script based launcher of network interconnected distributed processes, the time needed to upgrade a development system setup to an operational one can be greatly minimized. Another important feature of the framework is that big data inputs can be directly sourced streaming from either a device (e.g., ADC), file, UDP or TCP, and that the output results can be directly streamed to an operator interface or file storage systems for further evalua-



(a) Marble with polarimetric image overlay and a zoomed area. Yellow and blue lines correspond to near-range and beam tracks respectively.



(b) Bifrost image viewer. Instead of Marble, this view can be switched into from the topleft drop-down box menu.

Figure 7: Three channel polarimetric images with 384 MHz bandwidth, single-look, no presumming, 1m x 1m grid and 2° azimuthal aperture. Processed with direct backrojection on GPU. The test site is in Kaufbeuren, Germany.

tions.
It has been made evident in the test applications like polarimetry how the GPUs, together with multiprocessors, could hold the key in accelerating algorithms to truly perform in real-time scenarios. Future work is needed on the optimization of the algorithms and trade-off balancing, depending on the target end-to-end processing time required by the application. It is also assuredly expected that the algorithms will be adapted in the near future to stretch into the limits of computing via multiple GPUs on a node.

References

[1] Cantalloube, H.M.J.: *Real-time Airborne SAR Imaging. Motion compensation and Autofocus issues*, EUSAR Conference 2012, Munich, Germany, 2012.

[2] Simon-Klar, C., Friebe, L., Kloos, H., Lieske, H., Hinrichs, W., Pirsch, P.: *A Multi DSP Board for Real Time SAR Processing using the HiPAR-DSP 16*, IEEE,, 2002.

[3] Lidberg, C., Olin, J.: *Optimization of Fast Factorized Backprojection execution performance*, Chalmers University of Technology, Sweden, 2012.

[4] Hast, A., Johansson, L.: *Fast Factorized Back-Projection in an FPGA*, Halmstad University, Sweden, 2006.

[5] Ponce, O., Prats, P., Pinheiro, M., Rodriguez-Cassola, M., Scheiber, R., Reigber, A., Moreira, A.: *Fully-Polarimetric High-Resolution 3-D Imaging with Circular SAR at L-Band*, IEEE Trans. Geosci. Remote Sensing, Vol. 52, 2014.

[6] da Silva, A.B.C., Baumgartner, S.V.: *A Priori Knowledge-Based STAP for Traffic Monitoring Applications: First Results*, 11th European Conference on Synthetic Aperture Radar (EUSAR)

[7] Andres, C.; Keil, T.; Herrmann, R.; Scheiber, R.: *A multiprocessing framework for SAR image processing*, Geoscience and Remote Sensing Symposium, IGARSS 2007.

[8] Baumgartner, S.V., Krieger, G.: *Fast GMTI Algorithm For Traffic Monitoring Based On A Priori*

Knowledge, IEEE Trans. Geosci. Remote Sensing, Vol. 50, No. 11, November 2012.

[9] Haklay, M., Weber P.: *OpenStreetMap: User-Generated Street Maps*, Pervasive Computing, vol. 7, no. 4, pp. 12 - 18, Oct. 2008.

[10] https://marble.kde.org/

[11] https://developer.nvidia.com/cuda-gpus