A Generic Framework for the Development of Geospatial Processing Pipelines on Clusters

Rémi Cresson, Gabriel Hautreux

A generic framework for the development of geospatial processing pipelines on clusters

Rémi Cresson and Gabriel Hautreux*

The amount of remote sensing data available to applications is constantly growing due to the rise of very-high-resolution sensors and short repeat cycle satellites. Consequently, tackling computational complexity in Earth Observation information extraction is rising as a major challenge. Resorting to High Performance Computing (HPC) is becoming a common practice, since it provides environments and programming facilities able to speed-up processes. In particular, clusters are flexible, cost-effective systems able to perform data-intensive tasks ideally fulfilling any computational requirement. However, their use typically implies a significant coding effort to build proper implementations of specific processing pipelines. This paper presents a generic framework for the development of RS images processing applications targeting cluster computing. It is based on common open sources libraries, and leverages the parallelization of a wide variety of image processing pipelines in a transparent way. Performances on typical RS tasks implemented using the proposed framework demonstrate a great potential for the effective and timely processing of large amount of data.

I. INTRODUCTION

There is an increasing volume of remote sensing (RS) images for earth observation in recent years. Satellite and airborne RS data become widespread and sensors are currently sustaining major technical revolution. In one hand, the use of very high resolution RS data is booming, and both spectral and spatial resolutions are each generation sharper [1]. In the other hand, there is a growing need of sensors with close temporal acquisition in Earth Sciences. With the arrival of satellites such as Sentinel constellation (10m spatial resolution, 5 days revisit cycle), we entered a new era of earth observation. High orbital revisit frequency allow to monitor in near real time the earth's surface, which strongly benefits to many sectors, e.g. agricultural [2]. Currently, Earth Observation is producing permanently a stream of data. Therefore, extracting information from existing RS data became a major computational challenge [3].

For this purpose, high performance computing (HPC) techniques provide solutions to speed-up computations allowing the processing of large data volumes in reasonable time [4]. Basically, it relies on parallelization to increase computational power [5], and involves various computing environments and programming facilities. Several approaches of HPC have been used with RS data, like multiprocessors, computers networks (e.g. clusters, grids, clouds), and hardware such as graphics processing units (GPU). While multiprocessors and GPUs work usually in a shared memory context, i.e. on the same data stored in memory, clusters and clouds are distributed systems, aiming to handle distinct parts of the data bulk. Clusters are homogeneous systems (i.e. similar hardware and software) composed of tightly coupled machines (e.g. Thunderhead Beowulf cluster at NASA's Goddard Space Flight Center), whereas grids and clouds are heterogeneous and loosely coupled systems, generally on a larger scale [6]. It can also be noted that grids and clouds can be composed of several clusters, and most clusters exploit multiprocessors and GPUs as co-processors. Hence, HPC techniques are handled in a complementary pattern. Although each one may suit to a particular application, clusters are considered by the Earth and space sciences community as a cost-effective system able to satisfy specific computational requirements [7]. This is why in this paper, we choose to focus on cluster based systems only. In the following we call "cluster" a group of machines with the same hardware, namely "nodes". Nodes are connected by a fast local area network, sharing a parallel file storage system.

In this context, one crucial point is the software implementation. Often the literature presents algorithms adapted to HPC architectures for one very specific task. Therefore, coding is not generic and each new algorithm implementation require expertise, often at considerable cost. We can distinguish common parallel programing paradigms for various HPC architectures. OpenMP and Pthreads are commonly used to carry out parallel computations with multiple processors in a shared memory environment. For writing programs on GPUs, CUDA and OpenCL frameworks are frequently used. In cluster based systems, the Message Passing Interface (MPI) programming model is commonly used to manage communications between nodes. Besides, hybrid parallel algorithms are regularly employed to achieve the best performances, e.g. MPI + CUDA on a GPU cluster. Hence, developing RS processing pipelines on HPC architectures require some advanced knowledge of both hardware and programming paradigms, holding back its democratization especially for research and academics. Currently, existing popular libraries have programming interfaces embedding multiprocessing in a shared memory environment [8] and GPUs [9]. There is only a few studies presenting frameworks for RS image processing on clusters, mainly relying on MPI [10] and hybrid approaches [11]. However, up to our knowledge, there

* Corresponding author: remi.cresson@irstea.fr

is no available cluster-oriented paradigm benefiting from multiprocessing in a shared memory environment and GPUs. In this paper, we present a generic framework for the development of geospatial processing pipelines on clusters, which is (i) open-source and portable (cross-platform), (ii) developer friendly, with strong abstraction of low level cluster related mechanisms, (iii) based on an existing rich image processing library, Orfeo Toolbox (OTB, [12]) and relying on the MPI standard. Our approach benefits from multiprocessing in a shared memory environment as well as GPU support, while ensuring the distribution of the entire data across clusters. We first give in Section II a detailed description of our cluster-oriented RS image processing parallelization framework. Our approach is then successfully tested with a set of popular pipelines on some Spot 6 satellite images (Section III). We finally discuss the main advantages and limits of our approach in Section IV.

II. METHOD

A. Overview

We consider the library for RS image processing, OTB, built on top of an application development framework widely used in medical image processing (ITK, the Insight Toolkit [8]), and a cluster composed of several nodes with a parallel file system. Our goal is to bring a development framework to exploit the cluster processing capabilities. The parallelization of already implemented pipelines should be enabled with the minimum effort, and the opportunity to build clusters compliant pipelines must be granted to non-expert developers (namely users) without fully understanding the low level MPI implementation. The existing support for multiprocessing in a shared memory environment and GPU brought by ITK and OTB must also be preserved. In the following sections, we provide description of the existing image processing framework of the used libraries (Section II B). Then we introduce the concepts of parallel process objects in Section II C 1 and parallelized pipeline in Section II C 2. Finally, we detail our solution to overcome the problem of geospatial raster data output in Section II D.

B. Description of a pipeline

This section describes the terminology and the execution steps of a RS processing pipeline, from the OTB perspective. A pipeline is a directed graph of process objects, that can be:

- Sources: initiating the pipeline. Generating input data objects (e.g. image file reader),
- Filters: processing the data objects,
- Mappers: terminating the pipeline. Writing data on disk (e.g. image file writer), or interfacing with some other system (e.g. display).

Sources and filters can generate one or multiple data objects (e.g. image, mesh, vector, matrix, number). For the user, building a pipeline consists in connecting process objects together. The execution of a pipeline starts by triggering process objects, usually mappers. More details about these objects and architecture are provided in [13].

The architecture of the libraries hides the complexity of internal mechanisms for pipeline execution, which involve several important steps. First of all, process objects that need to be triggered are determined, to avoid redundant execution.

Then, the execution of the pipeline is started by a mapper trigger. When a filter or a mapper is triggered, a signal is sent upstream to request information about mandatory input data (i.e. information about output data of upstream process object(s)). In this way, it is propagated back through the pipeline, from mappers to sources via filters. Once this request reach sources, information are generated from metadatas. This information can be image size and pixel spacing, that are propagated downstream to mapper. It may be noted that filters can potentially modify these information, according to the process they implement (e.g. resampling might change output image size). Finally, they reach the mapper, initiating the data processing.

Information regarding the size of the image that has to be produced, is then used by the mapper to choose a splitting strategy. Typically, the splitting scheme is based on the system memory specification. Other strategies can be chosen, e.g. striped or tiled regions with fixed dimensions. Once the splitting strategy is determined, the mapper proceed, typically write on disk. The mapper process the image sequentially, region by region, requesting its input filter. The data request and generation is handled through the pipeline in the same way as for the information: once the request reaches the sources, initiating the pipeline, the requested region is produced, then processed through filters, to finally end in the mapper. The pipeline execution continues with the next image region.

C. Toward parallel process objects

The idea is to go through a cluster-oriented parallel approach, while preserving the existing development framework of the libraries. This includes the coding of process object (sources, filters, mappers), and pipelines creation on a higher abstraction level. In the following, we detail two concepts. We first introduce parallel process objects, which are basically process objects implementing a MPI based abstraction layer. Second, we present the parallelized pipeline, which designates the set of pipelines running across the cluster, each one being a different MPI process.

1. Parallel process objects

Regarding the parallel approach, two kind of process objects generating images must be distinguished. The first ones can produce output images in a region independent fashion, meaning that identical pixels are generated whatever the output requested images region. Hence, an entire output image can be gathered from multiple generation of different regions, making these process objects straightforwardly suitable for parallel approach. The second kind of process objects generates output images that are dependent of the requested region and, thus, require specific implementation. However, the ITK and OTB libraries propose a convenient development framework to greatly reduce the developer task in implementing such algorithms. This is particularly useful to design and implement filters which, for instance, process data using multi-process on shared memory (namely *Multithreaded* filters) and filters that persist data through multiple update (namely *Persistent* filters). The advantage of *Multithreaded* filters is that the developer does not need to be an expert in low level threads management, neither knowing the end user's hardware specifications. Typically, he just has to implement some specific methods. These methods are generally data generation on thread region (*ThreadedGenerateData*), and shared resources handling before and/or after the multi-threaded part (e.g. *Before/AfterThreadedGenerateData*). Similarly, *Persistent* filters implement methods to handle variables needing to be updated during the process, e.g. pixels statistics (*Synthesis*, *Reset*). In their parallelized version, such filters must induce communications between MPI processes to aggregate variables over the cluster. This is achieved using the MPI with *many-to-one*, *one-to-many* or *many-to-many* communication patterns in the previously mentioned methods.

2. Parallelized pipeline

As described in the previous part, libraries offer an abstraction layer for the low level multiprocessing paradigm. In addition, a wide range of existing filters use GPU. To take benefit from these advantageous supports, working only in a shared memory context, we focus on the MPI processes grain level to parallelize pipelines: we distribute one pipeline per MPI process, ensuring the shared memory context, then pipelines are executed simultaneously to work in a collaborative fashion. A parallelized pipeline can be composed of parallelized or native process objects, depending if the implemented algorithm produces the same unique result whatever the requested region (as explained in Section II C 1). Given the typology of our parallelized pipeline, it must be terminated with a parallel mapper that ensures the load balancing of the cluster. This is achieved by computing a splitting scheme as described in Section II B, and determining the way of distributing data across the cluster.

Table 1. Characteristics of the dataset

Id	Product type	Image size ($col \times row \times band$)	Values encoding	File size
XS	Multispectral	$10699 \times 11899 \times 4$	16 bits	1.0 Gb
PAN	Panchromatic	$42599 \times 47299 \times 1$	16 bits	4.0 Gb

D. Raster writing on parallel file systems

Most of RS processing applications produce images in raster format. Unfortunately, this kind of data have large size, leading to an I/O (Input/Output) bottleneck [14]. For this purpose, we develop a mapper able to write GeoTiff file on parallel file systems. Thanks to our implementation using MPI-IO (MPI subroutines for file access [15]), multiple MPI processes can write their piece of data simultaneously in the same unique file. We chose to write files in the GeoTiff format, in a row-wise, interleaved pixel fashion, which is faster than tile-wise [16]. Our writer implements various strategies for cluster-oriented splitting scheme described in Section II B. Either size or number of splits can be manually set, or automatically computed using the system specifications (memory and number of MPI processes). Our writer has a static load balancing, meaning that each process has a fixed processing schedule.

III. EXPERIMENTS

The method presented in Section II is implemented in C++ using OTB, ITK and MPI. We parallelized large number of already implemented pipelines in OTB [17]. In this section, we first analyze performances in reading and writing images (Section III A), then we present the results related to the speedup of a set of pipelines frequently used in RS image processing (Section III B). To conduct this series of experiments, we used a dataset of very-high-resolution Spot 6 images acquired within the GEOSUD project[18] presented in Table 1. All experiments are conducted on a cluster composed of 16 DELL R630 Intel Xeon E5-2690 nodes, each one made of 2 sockets of 12 cpus at 2,6Ghz and 64 Gb RAM. Machines are linked with a very fast network (Infiniband 40 Gb/s). All files were stored on a RAID disk array shared to the cluster with a General Parallel File System. We binded MPI process with socket rather than node, to populate each socket exclusively by threads from one unique MPI process. This ensures optimal caching, enabling efficiency of the ITK and OTB multi processing framework which rely on shared memory. Therefore, each node hosts two MPI processes.

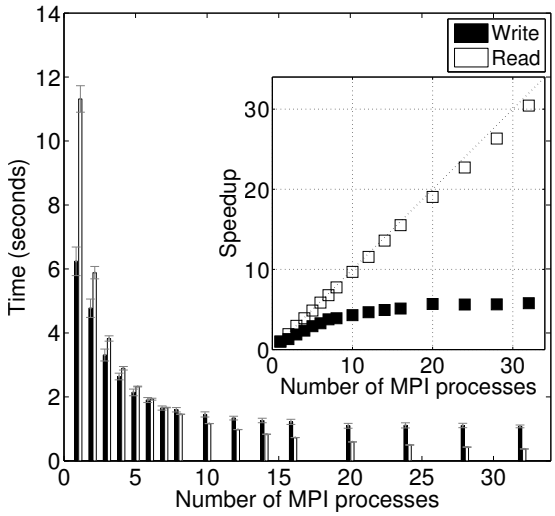

Figure 1. Time consumed by reading (white bars) and writing (black bars) the Spot 6 GeoTiff image stored in the cluster file system, with error bars. An inset presents the speedup of reading (white boxes) and writing (black boxes).

A. I/O performances

We used a simple parallel pipeline, composed of a source and our parallel writer, for measuring I/O related durations.

Figure 1 shows measures dedicated to reading and writing operations from the *XS* GeoTiff image file.

Time spent in reading decreases linearly with the number of MPI processes. This operation is quite scalable, with a speedup close to the number of processes. Yet, the speedup for writing is not linear and reaches a maximum of 6.1 for 32 processes. This poor scalability is related to the striping for writing data, which needs to be fine tuned to fit better the file system use. Besides, writing speedup is expected to be lower than reading because more processes synchronization is required.

B. Parallelized pipelines

We derived the following OTB pipelines P_i in their parallelized flavor, replacing each original image file writer by our parallel one:

(P_1) **Orthorectification:**

(P_2) **Textures Extraction, using Haralick indicators:**

(P_3) **Pansharpening, from *PAN* and *XS* images:**

(P_4) **Image classification, using a random forest based rule:**

(P_5) **Meanshift filtering:**

Table 2. Run times and speedup of pipelines for N MPI processes

N	1	2	4	8
$P1$	1911s ±17s	970s (×**2.0**) ±9.3s (0.02)	481s (×**4.0**) ±4.0s (0.03)	241s (×**7.9**) ±1.6s (0.05)
$P2$	498s ±13s	260s (×**1.9**) ±8.6s (0.06)	125s (×**4.0**) ±0.48s (0.02)	64s (×**7.8**) ±2.0s (0.24)
$P3$	2613s ±18s	1339s (×**2.0**) ±13s (0.02)	674s (×**3.9**) ±2.3s (0.01)	347s (×**7.5**) ±3.0s (0.06)
$P4$	495s ±2.2s	471s (×**1.1**) ±1.7s (0.00)	127s (×**3.9**) ±0.36s (0.01)	63s (×**7.9**) ±0.16s (0.02)
$P5$	1972s ±137s	851s (×**2.4**) ±113s (0.26)	522s (×**3.9**) ±85s (0.61)	284s (×**7.1**) ±41s (0.98)
$P6$	1013s ±2.5s	526s (×**1.9**) ±4.6s (0.02)	262s (×**3.9**) ±1.4s (0.02)	127s (×**8.0**) ±0.21s (0.01)
$P7$	2192s ±16s	1106s (×**2.0**) ±11s (0.02)	561s (×**3.9**) ±1.2s (0.01)	285s (×**7.7**) ±0.82s (0.02)
IO	17s ±0.32s	11s (×**1.6**) ±0.16s (0.02)	5.5s (×**3.2**) ±0.07s (0.04)	3.0s (×**5.7**) ±0.08s (0.15)

N	12	16	32
$P1$	163s (×**12**) ±1.2s (0.08)	125s (×**15**) ±0.36s (0.04)	63s (×**31**) ±0.04s (0.02)
$P2$	42s (×**12**) ±1.00s (0.27)	31s (×**16**) ±0.81s (0.41)	15s (×**32**) ±0.42s (0.85)
$P3$	241s (×**11**) ±2.0s (0.09)	188s (×**14**) ±0.56s (0.04)	115s (×**23**) ±5.2s (0.96)
$P4$	44s (×**11**) ±0.44s (0.12)	34s (×**15**) ±0.31s (0.13)	17s (×**29**) ±0.12s (0.21)
$P5$	186s (×**11**) ±19s (1.1)	152s (×**13**) ±15s (1.4)	69s (×**29**) ±3.6s (1.5)
$P6$	94s (×**11**) ±0.99s (0.11)	64s (×**16**) ±0.15s (0.04)	33s (×**31**) ±0.08s (0.07)
$P7$	196s (×**11**) ±0.70s (0.04)	151s (×**14**) ±0.73s (0.07)	84s (×**26**) ±0.57s (0.18)
IO	2.3s (×**7.6**) ±0.06s (0.21)	1.9s (×**9.0**) ±0.06s (0.29)	1.5s (×**12**) ±0.07s (0.51)

(P_6) **Conversion from Jpeg2000 format to GeoTiff:**

(P_7) **Resampling *XS* image over *PAN* image:**

(***I/O***) **Reading and writing the image in another file:**

To analyze the computational performance of our approach, we decompose run time of each P_i for several number of MPI processes, and calculate the corresponding speedup.

Table 2 presents measured run times in seconds and Figure 2 shows the speedup.

The speedup is more or less equal to the number of MPI processes, characterizing a good scalability, except for the single I/O operation. Besides, Section III A shows that writing operation limits the scalability, explaining why the speedup is slightly below the number of MPI processes for each P_i.

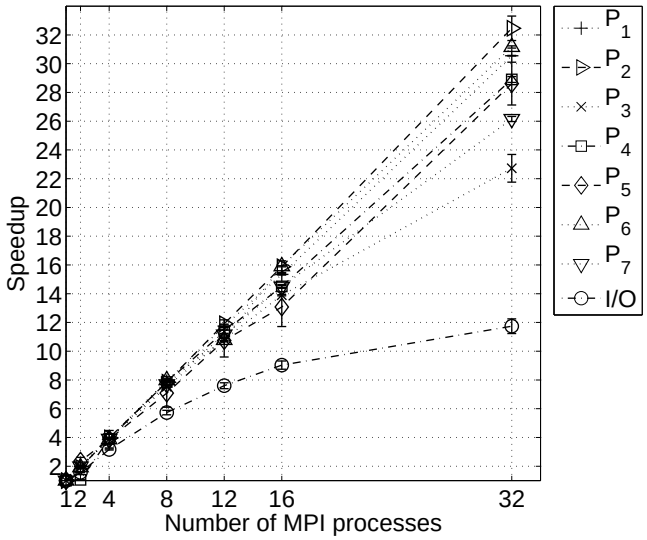

Figure 2. Measured speedup of P_i for increasing number of MPI processes.

IV. DISCUSSION

A. A novel approach to create cluster oriented pipelines

Our first goal was to provide a framework to develop cluster oriented applications for RS images processing. The solution we propose takes roots in the extensively used open-source libraries ITK and OTB. The original development framework was enriched to embed the MPI, for inter-nodes communication and parallel file access. We introduced parallel process objects, and parallel pipelines, enabling the execution of one pipeline on multiple nodes of a cluster.

B. Scalability

Our research also sought to result in operational and scalable processes. The approach was successfully applied to a set of common RS pipelines, on Spot 6 imagery at very-high-resolution. Run times were measured for various number of MPI processes, and demonstrate a good scalability of tested parallel pipelines. In the near future, it will be used to extract a wide range of information for earth observation, on a daily basis from satellite imagery acquired through the GEOSUD platform and the THEIA Land Data Center[19].

C. Limits and further research

It should be noted that the presented method also has limits, which may restrict its use in some cases. To start with, we used a small cluster which has only a dozen of nodes to lead our tests. Hence, we could not fully investigate the limits of our approach in term of scalability on many nodes. Besides, a key drawback of the use of clusters is the I/O bottleneck, which is common in HPC. In our experiments, we show that the scalability of one pipeline is closely dependent on the balance between processing time and file access time. While pipelines are generally scalable, I/O operations might require fine tuning according to the file system specifications. Nevertheless, a cluster should not be used to speed up a process spending a lot more time in writing data than processing. Another possible improvement would be to use a dynamic load balancing, which might tackle the problem of algorithms running in a non-constant time on different image regions. Finally, our approach is only suitable for filters composing a pipeline that could be parallelized using communications between nodes. Depending on the implemented algorithm, some filters might not fulfill this condition, or could lead to a poor speedup. Regarding this last issue, we recommend to split this kind of pipeline in multiple homogeneous parts with uniform scalability and to run them sequentially. This puts forward the question related to the orchestration of multiple connected pipelines execution, which should be addressed in the future.

V. CONCLUSION

This work was carried out with a view toward processing very-high-resolution and high-acquisition-rate satellite images on clusters. We propose a cluster oriented framework for the development of remote sensing images processing applications, using the Orfeo Toolbox and the Message Passing Interface. We parallelized successfully a number of existing pipelines, and demonstrated the good scalability of the processes. Parallel pipelines will be executed every day to extract Earth observation data from very-high-resolution and multi-temporal images acquired through the GEOSUD platform and the THEIA Land Data Center, at CNES (the french Space Agency) and Irstea (the french Research Institute of Science and Technology for Environment and Agriculture). Further research could focus on improving the load balancing strategies, as well as the orchestration of multiple connected pipelines execution. The source code corresponding to the pipelines presented in this paper is available for download at [20], and the exposed framework will be integrated in the forthcoming releases of the Orfeo Toolbox[21].

ACKNOWLEDGMENTS

This work was supported by public funds received through GEOSUD, a project (ANR-10-EQPX-20) of the *Investissements d'Avenir* program managed by the French National Research Agency. The authors would like to thank the reviewers for their valuable

comments and suggestions which have greatly contributed in improving the manuscript. The authors would also like to thank Maxime Lenormand, Raffaele Gaetano and Julien Michel for their great help, and the OTB community for the precious support. They also thank the CINES supercomputing center and the HPC@LR competence center for providing the HPC support.

[1] J. A. Benediktsson, J. Chanussot, and W. M. Moon. Advances in very-high-resolution remote sensing. *Proceedings of the IEEE*, 101, 2013.

[2] D. B. Lobell and G. P. Asner. Cropland distribution from temporal unmixing of modis data. *Remote Sensing of Environment*, 93:412–422, 2004.

[3] J. Dorband, J. Palencia, and U. Ranawake. Commodity computing clusters at goddard space flight center. *J. Space Commun.*, 3:1, 2003.

[4] Antonio J Plaza and Chein-I Chang. *High performance computing in remote sensing*. CRC Press, 2007.

[5] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, et al. The landscape of parallel computing research: A view from berkeley. Technical report, Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, 2006.

[6] Armando Fox, Rean Griffith, Anthony Joseph, Randy Katz, Andrew Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, and Ion Stoica. Above the clouds: A berkeley view of cloud computing. *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, 28(13):2009, 2009.

[7] C. A. Lee, S. D. Gasster, A. Plaza, C-I. Chang, and B. Huang. Recent developments in high performace computing for remote sensing: A review. *IEEE JSTARS*, 4:508–527, 2011.

[8] T. S. Yoo, M. J. Ackerman, W. E. Laorensen, W. Schroeder, V. Chalana, S. Aylward, D. Metaxas, and R. Whitaker. Engineering and algorithm design for an image processing API: A technical report on ITK - The Insight Toolkit. In IOS Press Amsterdam J. Westwood, ed., editor, *Proceedings of Medecine Meets Virtual Reality*, 2002.

[9] E. Christophe, J. Michel, and J. Inglada. Remote sensing processing: From multicore to GPU. *IEEE JSTARS*, 4:643–652, 2011.

[10] Qingfeng Guan, Wen Zeng, Junfang Gong, and Shuo Yun. pRPL 2.0: improving the parallel raster processing library. *Transactions in GIS*, 18(S1):25–52, 2014.

[11] Cheng-Zhi Qin, Li-Jun Zhan, A-Xing Zhu, and Cheng-Hu Zhou. A strategy for raster-based geocomputation under different parallel computing platforms. *International Journal of Geographical Information Science*, 28(11):2127–2144, 2014.

[12] J. Inglada and E. Christophe. The Orfeo Toolbox remote sensing image processing software. In *IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, 2009.

[13] Luis Ibez and Brad King. ITK. Accessed: 2016-05-24.

[14] Lizhe Wang, Yan Ma, Albert Y Zomaya, Dan Chen, et al. A parallel file system with application-aware data layout policies for massive remote sensing image processing in digital earth. *Parallel and Distributed Systems, IEEE Transactions on*, 26(6):1497–1508, 2015.

[15] Rajeev Thakur, William Gropp, and Ewing Lusk. On implementing mpi-io portably and with high performance. In *Proceedings of the sixth workshop on I/O in parallel and distributed systems*, pages 23–32. ACM, 1999.

[16] Cheng-Zhi Qin, Li-Jun Zhan, A Zhu, et al. How to apply the geospatial data abstraction library (GDAL) properly to parallel geospatial raster I/O? *Transactions in GIS*, 18(6):950–957, 2014.

[17] Emmanuel Christophe and Jordi Inglada. Open source remote sensing: Increasing the usability of cutting-edge algorithms. *IEEE Geoscience and Remote Sensing Newsletter*, 35(5):9–15, 2009.

[18] www.equipex-geosud.fr.

[19] http://www.theia-land.fr/en.

[20] Cresson. otbclimpi branch at https://github.com/remicres/otb/tree/otbclimpi. Accessed: 2016-05-11.

[21] http://www.orfeo-toolbox.org.