National Technical University of Athens
Msc in Data Science and Machine Learning

Evaluating Strategies for the Exploitation of Multi-node HPC Clusters in Computation Intensive Remote Sensing Tasks

Master Thesis

Konstantinos Kyriakopoulos

Supervisor: Konstantinos Karantzalos
Associate Prof., NTUA

Athens, November 2021

Εθνικό Μετσόβιο Πολυτεχνείο
ΔΠΜΣ Επιστήμη Δεδομένων και Μηχανική Μάθηση

Αξιολόγηση Στρατηγικών για την Εκμετάλλευση Υπερυπολογιστών Πολλαπλών Κόμβων σε Υπολογιστικά Απαιτητικές Εφαρμογές Τηλεπισκόπησης

Μεταπτυχιακή Εργασία
Κωνσταντίνος Κυριακόπουλος

Επιβλέπων: Κωνσταντίνος Καράντζαλος
Αναπλ. Καθηγητής ΕΜΠ

Αθήνα, Νοέμβριος 2021

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 1η Νοεμβρίου 2021.

....	....	....
Κωνσταντίνος Καράντζαλος	Μαρία Βακαλοπούλου	Ιωάννης Παπουτσής
Αναπλ. Καθηγητής ΕΜΠ	Επ. Καθηγήτρια CentraleSupélec Univ. of Paris-Saclay Γαλλία	Εντεταλμένος Ερευνητής Εθνικό Αστεροσκοπείο Αθηνών

Abstract

Over the recent years, the fast growth and evolution in the fields of Big Data, Geoinformatics and Machine Learning, is posing great challenges in terms of robustness, scalability, complexity and computational speed. More specifically, the big size of the datasets, especially in the case of geospatial data, along with the complex calculations that need to be performed when applying ML models, require resources which are not available in common infrastructure. However, this problem can be tackled by massive parallel processing in big hyper computer clusters.

This master thesis focuses on presenting and exploring the capabilities of the ARIS (Advanced Research Information System) cluster while performing complex computations and applying machine learning techniques in parallel scale. For the purpose of this experiment, data from the Sentinel 2 program is used. The dataset consists of 53 tiles which are satellite images that represent the geographical space of Greece.

As part of the experiment, the tile is broken down into smaller parts and each part is processed while applying ML methods for specific versions of the tile(10m and 20m). Once, the process of each part is over, the images that derived from each part are merged into the final image which is the final processed result. During the initial implementation, the process of each part is performed serially, resulting in a lengthy task.

In our approach, each part is being processed in a different node of the cluster as a separate process, resulting in a significant decrease in the overall time that is needed for the experiment. Apart from the gain in time that derives from the parallel processing, we examine the limitations that parallel processing is facing in this experiment along with how the subprocesses are affected by the terrain in the images.

Overally, this thesis provides a very good insight into parallel processing of large geospatial data while applying machine learning methods and presents the challenges, the limitations that exist in the relevant data and infrastructure and how they interact with each other.

Περίληψη

Τα τελευταία χρόνια, η μεγάλη ανάπτυξη και εξέλιξη στους τομείς των Μεγάλων Δεδομένων, της Γεω-Πληροφορικής και της Μηχανικής Μάθησης, προβάλλουν μεγάλες προκλήσεις υπό όρους ανθετικότητας, επεκτασιμότητας, πολυπλοκότητας και υπολογιστικής ταχύτητας. Πιο συγκεκριμένα, το μεγάλο μέγεθος των συνόλων δεδομένων, ειδικά στην περίπτωση των γεωχωρικών δεδομένων, μαζί με τους σύνθετους υπολογισμούς που πρέπει να γίνουν κατά την εφαρμογή μοντέλων ML, απαιτούν πόρους που δεν είναι διαθέσιμοι σε μια κοινή υποδομή. Ωστόσο, αυτό το πρόβλημα μπορεί να αντιμετωπιστεί με μαζική παράλληλη επεξεργασία σε μεγάλες κυψέλες υπερυπολογιστών.

Αυτή η μεταπτυχιακή διατριβή εστιάζει στην παρουσίαση και τη διερεύνηση των δυνατοτήτων του συστήματος ARIS (Advanced Research Information System) ενώ εκτελεί σύνθετους υπολογισμούς και εφαρμόζει τεχνικές μηχανικής μάθησης σε παράλληλη κλίμακα. Για τους σκοπούς αυτού του πειράματος, χρησιμοποιούνται δεδομένα από το πρόγραμμα Sentinel 2. Το σύνολο δεδομένων αποτελείται από 53 πλακίδια τα οποία είναι δορυφορικές εικόνες που αντιπροσωπεύουν τον γεωγραφικό χώρο της Ελλάδας.

Ως μέρος του πειράματος, το πλακίδιο αναλύεται σε μικρότερα μέρη και κάθε μέρος υποβάλλεται σε επεξεργασία ενώ εφαρμόζονται μέθοδοι ML για συγκεκριμένες εκδόσεις του πλακιδίου (10μ και 20μ). Μόλις ολοκληρωθεί η διαδικασία κάθε μέρους, οι εικόνες που προέρχονται από κάθε μέρος συγχωνεύονται στην τελική εικόνα που είναι το τελικό επεξεργασμένο αποτέλεσμα. Κατά την αρχική υλοποίηση, η διαδικασία κάθε τμήματος εκτελείται σειριακά, με αποτέλεσμα μια ιδιαιτερα χρονοβόρα εργασία.

Στην προσέγγισή μας, κάθε τμήμα υποβάλλεται σε επεξεργασία σε διαφορετικό κόμβο του συστήματος ως ξεχωριστή διαδικασία, με αποτέλεσμα τη σημαντική μείωση του συνολικού χρόνου που απαιτείται για το πείραμα. Εκτός από το κέρδος χρόνου που προκύπτει από την παράλληλη επεξεργασία, εξετάζουμε τους περιορισμούς που αντιμετωπίζει η παράλληλη επεξεργασία σε αυτό το πείραμα μαζί με το πώς επηρεάζονται οι υποδιεργασίες από το έδαφος στις εικόνες.

Συνολικά, αυτή η διατριβή παρέχει μια πολύ καλή ανάλυση σχετικά με την παράλληλη επεξεργασία μεγάλων γεωχωρικών δεδομένων κατά την εφαρμογή μεθόδων μηχανικής μάθησης καθώς επίσης παρουσιάζει τις προκλήσεις, τους περιορισμούς που υπάρχουν στα δεδομένα και στις υποδομές και πως αυτοί αλληλεπιδρούν μεταξύ τους.

Acknowledgments

First and foremost, i would like to express my gratitude towards Professor Konstantinos Karantzalos who supervised this thesis. His unconditional support during my studies helped me carry out this task and this work wouldn't have been completed without his contribution.

Furthermore, I would like to thank Zacharias Kandylakis and Christina Karakizi who supported me throughout the whole process of writing this thesis. It was their work about applying machine learning techniques in geospatial data which was used as the basis of our experiments in order to dive deeply into the world of parallel processing.

Of course, many thanks to Dr. Maria Vakalopoulou and Dr. Ioannis Papoutsis for the time they spent to review this thesis.

Last but not least, i need to express my thanks towards my family which has always been by my side during my studies. Their love and support played a vital role in completing my postgraduate studies.

You measure the size of the accomplishment
by the obstacles you have to overcome to reach your goals.
Booker T. Washington

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The rapid growth of Big Data over the past years combined with the evolution of Machine Learning have altered radically the field of computational complexity and data processing. Whereas, in the past a single node was sufficient enough to perform experiments, nowadays the size of the datasets grows exponentially while the new machine learning techniques and libraries require a significant amount of processing power.

Parallel processing has emerged as a revolutionary technique of processing big chunks of data while applying the state of the art ML models and methods. Experimental implementations which normaly require multiple hours or even days of processing in a single node, are tested within minutes in big clusters as the job is broken down into multiple parallel tasks.

Of course, even in parallel processing, there are specific limitations especially in terms of memory and resource handling. Particularly, in the case of Geospatial data it is very important to discover and assess the limitations and the bottlenecks that specific factors like the nature of the terrain create. Such an approach helps towards tuning the testing environment and utilizing the maximum of the cluster's capabilities.

1.1 Thesis Contribution

The objective of this work is to migrate an experimental implementation which applies machine learning methods on Geospatial data from running in a single node as one process into a hypercomputer cluster where each task will run as a parallel process. Moreover, this thesis not only exploits the advantages of parallel multiprocessing in a hypercomputer environment while tackling several technical challenges but also evaluates the correlation between specific features of Geospatial data and the behaviour of multiprocessing in terms of resource allocation and time. The following chapters include all the necessary,theoretical and practical,information about the functionality of the hypercomputer cluster, the background of the ML techniques which are used on the Geospatial data and the assessment of the experiments in terms of parallel processing.

1.2 Thesis Outline

The thesis is organized in the following chapters:

Chapter 2 describes the concept behind the machine learning methods and the nature of the geospatial data that are used in the experiments.

Chapter 3 presents the architecture, the capabilities, the components and the functionality of the Advanced Research Information System (ARIS) .

Chapter 4 contains all the details, results and the observations of the experimental implementation.

Chapter 5 concludes the thesis.

Chapter 2

Theoretical and Experimental Background

2.1 GIS and Remote Sensing

2.1.1 GIS

The acronym GIS stands for Geographic Information Systems. A geographic information system (GIS) is a system which aims to create, manage, analyze, and map all kinds of data. Its functionality is to map location data with different types of explanatory metadata. GIS systems find appliance in many industries and provide an excellent foundation for performing analysis and mapping. Moreover, their functionalities allow users to detect patterns, correlations and trends, observe and monitor changes, apply forecasting, identify problems and manage events. The positives that derive from their usage are many:

- Greater efficiency which results to cost saving. For example GIS can be used in order to optimize routes and this results in reduced costs.
- Improved communication. One of the major advantage of these systems, is that they are able to map the data, both the raw and the metadata into a new language. The users who use GIS are able to understand the data and the results which derive from them and can communicate their findings to other users who use the same systems efficiently and faster.
- Improved decision making. Users and especially professionals have understood how their decision making is enhanced by the numerous and valuable information that the geographic information system (GIS) have to offer.

- Enhanced and robust storage. These systems allow organizations, companies and even individuals to store any kind of geographical data in a robust, scalable and efficient way. Moreover, they allow and even promote the usage of the frameworks between them thus promoting information exchange. It is important to note that very often clever tools and frameworks sit on top of the GIS systems giving enhanced capabilities to their users.

Of course the benefits of the Geographic Information Systems are not limited to the above but we covered the most important of them which have brought GIS being on the edge of technology, machine learning and big data.

2.1.2 Remote sensing

Remote sensing is a type of geospatial technology that samples emitted and reflected electromagnetic (EM) radiation from the Earth's terrestrial, atmospheric, and aquatic ecosystems in order to detect and monitor the physical characteristics of an area without making physical contact. It is one of the methods commonly used for collecting physical data to be integrated into GIS. Remote sensors collect data from objects on the earth without any direct contact. Remote sensing technology has become much more prevalent, accurate and accessible in recent years, and a big range of applications is covered by it. This technique of data collection commonly includes aircraft-based and satellite-based sensor technologies. These technologies are categorized as either passive or active sensors.

On the one hand, passive sensors respond to external stimuli by collecting the radiation that is emitted or reflected by an object or the surrounding area. Reflected sunlight is the most usual source of radiation which is measured by passive remote sensors. Radiometers, Charge-coupled devices, film photography and infrared are popular examples of passive remote sensors.

On the other hand, active sensors use internal stimuli to collect data, emitting energy in order to scan objects and areas whereupon a sensor measures the energy reflected from the target. Typical active remote sensing tools are RADAR and LiDAR. Time delay between emission and return in order to establish the location, direction, and speed of an object are calculated by these sensors. Once the data is gathered, it is afterwards processed and analyzed with hardware and software which are specifically used for remote sensing purposes and are available in both proprietary and open source applications. The sources for the above information regarding remote sensing can be found in references [1], [2] and [3].

2.1.3 Experimental Background

The long term goal of the implementation on which this work is based is to create a common system for image classification regarding satellite images of earth. For the purpose of our experiments we used data from the Sentinel 2 Program and more specifically satellite images from the geographic are of Greece. Further analysis on the dataset will be presented in the next chapter about the actual implementation and the methodology.

One of the big challenges towards the goal of creating the common system is the fact that, due to the orbit of the satellite, there is no consistency between geographic locations and the dates for which we ll have data. There can be data for a specific geographic location for a specific date while there wont be for another location on the same date. In our work, we are aiming to have data for all the geographic regions of Greece on the same dates over a period of 1 year. In that direction, the relevant data can be created through interpolation. We are collecting the data about a geographic location before and after the specific date where we are missing them and through interpolation the data is created on that date.

It is important to note that the images, since they come from a satellite may have clouds. In that case, it is important that atmospheric cleaning takes place. During that process, the necessary corrections are conducted in order to eradicate the alteration of the radiation between the sender and the receiver which occurs due to the atmosphere. The effect of the atmosphere on the sun radiation which reflects on a surface is that one part of the radiation is scattered, another part is absorbed by the surface and the rest finally reaches the sensor of the satellite.

However, not only the direct sun radiation but also the sparse sun radiation which derives from the scattering of the direct sun radiation in the atmosphere before that reaches the ground, reflect on the surface of the ground. The sparse radiation reaches the satellite sensor either directly by the atmosphere or by being reflected on the surface of the ground or on clouds as a routing radiation. The radiation which reaches the sensor is the sum of all the aforementioned components.

In order to perform the atmospheric correction of these images, the MAJA framework is used.Practically what the framework is doing is that it performs the cloud cleaning to the highest possible degree and also creates cloudmasks for the processed image: the cloudmask defines which pixel is "cloudy" by flagging it so that when the interpolation is performed, the value of the "cloudy" pixel wont be picked but instead the neighboring pixels which are not "cloudy" will be used. In our case, the atmospheric cleaning has been performed at the preprocessing stage, outside of our implementation. We will use the "corrected images" and the relevant cloudmasks.

Chapter 3

Methodology and Implementation

3.1 Overview

In our implementation, we are processing satelite images of a specific geographic region for a specific timeline in order to generate images for the dates within the timeline where we dont have available satelite data. As part of our setup, we have performed a pre-processing and the atmosphere of the satelite images has been corrected by removing the clouds.

Initially, we define the time range for which we will provide input data meaning the satelite images. We also define the time window for which we would like to have output data as well as the relevant interval based on which,within the actual dates where we have data, we will get synthetic dates. Obviously, the time window of the output data has to be within the time range of the input data. Also, we define the masks, both the sea and the cloud mask, for the relevant tiles(satelite images) which will be used in the machine learning part. Note that the sea masks are applied only into the tiles which have sea in them.

Once the pipeline has initialized, the necessary time space to be used by the interpolation functions will be created and the synthetic dates will be calculated. After this part, the function which performs the interpolation process is called. In this part, we initially define the number of available cores which will be used in multiprocessing. Note that here we also define the number of parts into which the image will be split. It is important to mention that the number of parts has to be a divident of 5490: 1, 2, 3, 5, 6, 9, 10, 15, 18, 30, 45 etc. and this is related to the dimensions and the pixels of the image. As we will see in the experimental assessment section, the number of parts will be a factor which will determine the number or processes that will run in parallel.

Moving forward, in case the pipeline has water, the pipeline loads the seawater masks and the cloud masks which we defined earlier. Once the masks are loaded, the synthetic dates and the relevant data are calculated: in case the part of the image contain only sea no multiprocessing takes place as the process is not computationally expensive whereas if the part of the image contains land or both land and sea, multiprocessing is performed to accelerate the process.

Of course, in case of multiprocessing factors like the optimal chunksize are calculate in order to ensure an efficient multiprocessing procedure. Note that the process from loading the masks to performing the multiprocessing part, is performed twice: once for the R1 band which are the 10m of the image and once for the R2 band which are the 20m of the image.

Every time the relevant image is created for the synthetic dates in each band for every part these are saved and once the aforementioned process is finished for all the dates and the bands, the images are merged accordingly to create the final images for the synthetic dates.

In this implementation the most important python libraries which are used are multiprocessing, OpenCV, numpy and GDAL:

- The multiprocessing library as the name indicates is used to perform the actual parallel processing by utilizing the available cores in the nodes.
- The OpenCV is used mostly for some resizing functionalities.
- The numpy library is one of the fundamental libraries which is used for scientific computing with Python and is the basis of the code(arrays etc.).
- GDAL is a translator library for raster and vector geospatial data formats that is released under an X/MIT style Open Source License by the Open Source Geospatial Foundation. As a library, it presents a single raster abstract data model and single vector abstract data model to the calling application for all supported formats. It also comes with a variety of useful command line utilities for data translation and processing. The source of the information about GDAL can be found in reference [4]

3.2 The Dataset

For the purpose of our experiments we used data from the Sentinel 2 program. The Sentinel 2 program, launched as part of the European Commission's Copernicus program on June 23, 2015 and was designed specifically to deliver a wealth of data and imagery. The satellite is equipped with an opto-electronic multispectral sensor for surveying with a sentinel-2 resolution of 10 to 60 m in the visible, near infrared (VNIR), and short-wave infrared (SWIR) spectral zones, including 13 spectral channels, which ensures the capture of differences in vegetation state, including temporal changes, and also minimizes impact on the quality of atmospheric photography.

The orbit is an average height of 785 km and the presence of two satellites in the mission allow repeated surveys every 5 days at the equator and every 2-3 days at middle latitudes. The Sentinel-2 data provides GMES (Global Monitoring for Environment and Security) program, jointly implemented by the EC (European Commission) and ESA (European Space Agency) services related, for example, to land management, agricultural production and forestry, and monitoring of natural disasters and humanitarian operations. The source of the information in this section regarding the Sentinel 2 program can be found in reference [5]

We tested our implementation with tiles that cover the geographical area of Greece: 53 tiles of size xxx both for the 10 meter and the 20 meter spatial resolution. The size of each tile varies from 200 GBs to 350 GBs and is dependent to the ratio between sea and land that each tile has. Overally, the 53 tiles have a size of 17 TBs. Also, it important to take into consideration the relevant masks which will be used for each tile that has sea. There are 49 masks, each mask has a size of 117 MBs and overally the masks have a size of 5.5 GBs. As it can be seen in the picture below, the aforementioned tiles have 34 or 35 prefix for the specific are and there is an alphabetic correlation between neighboring tiles.

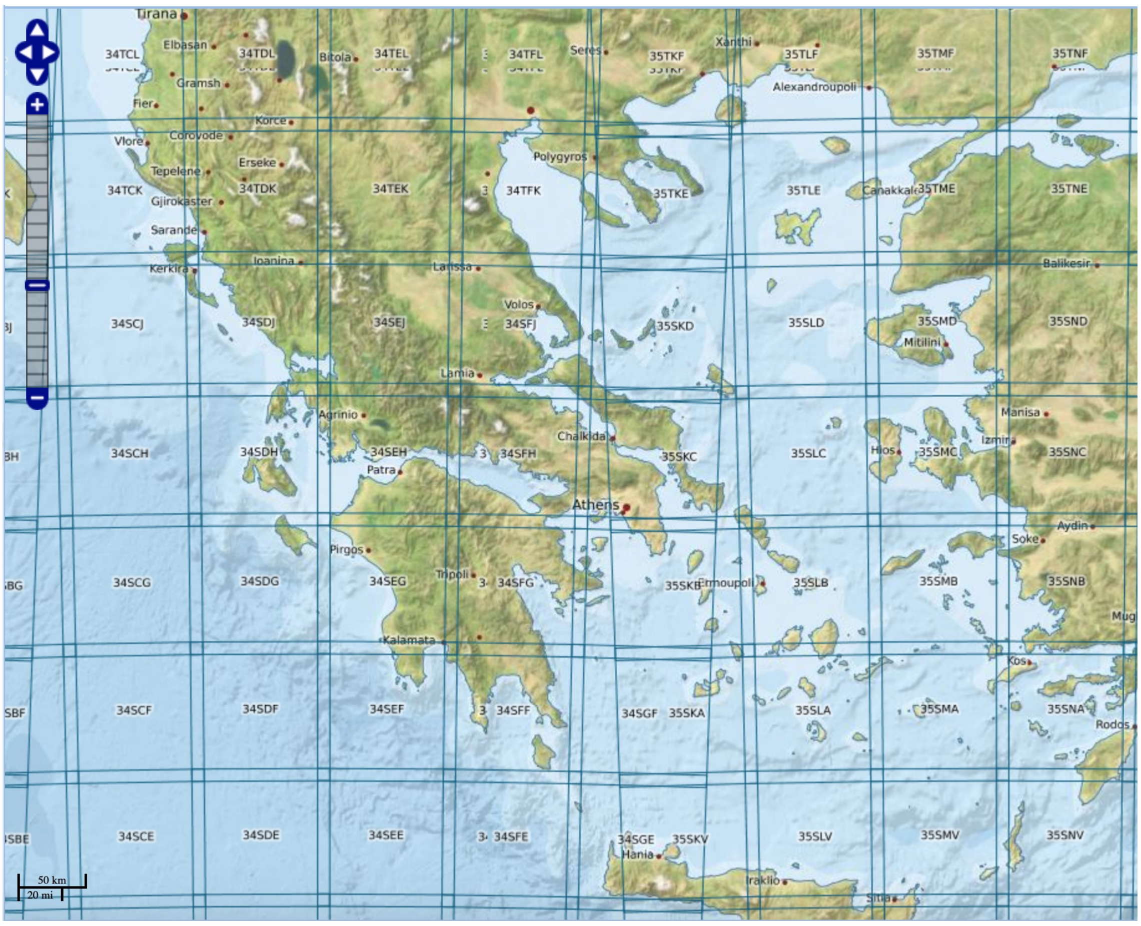

Figure 3.1: Greece - Data from Sentinel 2

3.3 ARIS

3.3.1 Overview of HPC

"Supercomputers" are computer systems used in scientific applications which demand the processing of large volumes of data and the processing of big and extremely complex mathematical operations. Such problems create big obstacles in terms of memory and storage use and consequently demand huge amounts of time to be resolved or cannot be solved at all through normal systems. Hypercomputers overcome these limitations by the utilization of specialized state-of-the-art hardware while exploiting the computing power of multiple units.

A hypercomputer is a powerful computation and research tool. Nowadays, the most complex problems and scientific challenges that humanity is facing like research on climate change, the discovery of new drugs, the origin of the universe and others are being solved by hypercomputers. For instance, the ARIS supercomputer system has been developed and operated in Greece by EDYTE in a very wide range of applications such as:

- The biochemistry for studying biological processes and possible modes of intervention (e.g., drug discovery)
- The chemistry for the study of properties atoms, compounds, e.g. design of new materials.
- The physical simulating phenomena at different levels from the sub-atomic particles to the stars and the universe, helping e.g. astrophysicists interpret observations about our universe.
- Climatology for the study of climate change in the region of Greece and the factors that affect it.
- Meteorology to improve the forecasting models used in Greece.
- Engineering for fluid flow simulation

3.3.2 Description of ARIS

ARIS (Advanced Research Information System) is the most powerful computing system in Greece for scientific applications. It was put into operation in July 2015 by GRNET offering a powerful research tool to the Greek scientific community.

The system at the beginning of its operation was included in the list of the 500 most powerful computers in the world (top500.org) and put Greece on the world map of high performance systems. The ARIS computer system today has a maximum theoretical computing power of 535 TFlop / s (ie it can perform 535 trillion mathematical operations per second) and offers multiple data processing capabilities. [6]

3.3.3 System Architecture

ARIS combines 5 different architectures divided into respective "node islands"
In detail, the infrastructure consists of [6]:

- **An islet which has 426 thin nodes.**

  Each node has two processors and each processor contains 10 processing cores, thus offering a total of 8,520 cores (CPU cores). These nodes are suitable for high-parallel applications that can break their data into many small pieces before processing them.

- **An island of large memory nodes (fat nodes) consisting of 44 nodes.**

  Each node offers 4 processors, 40 cores and 512 GB of central memory per node. These nodes are suitable for applications that need very large central memory and not so much for high scaling.

- **An island of GPU accelerator nodes consisting of 44 nodes.**

  Each node contains 2 processors with 10 cores per processor, 64 GB of memory and 2 NVidia K40 GPU graphics cards. These nodes are suitable for applications that implement computing operations that can utilize graphics cards as co-processors to speed up computing.

- **An island of Xeon Phi accelerator nodes (phi nodes) consisting of 18 nodes.**

  Each node contains 2 processors with 10 cores, 64 GB of memory and 2 co-processors Intel Xeon Phi 7120P. It is suitable for parallel applications that utilize Intel Xeon Phi co-processor technology.

- **A machine learning node island (ml node) consisting of 1 server**

  It contains 2 Intel E5-2698v4 processors, 512 GB of central memory and 8 NVIDIA V100 GPU cards.

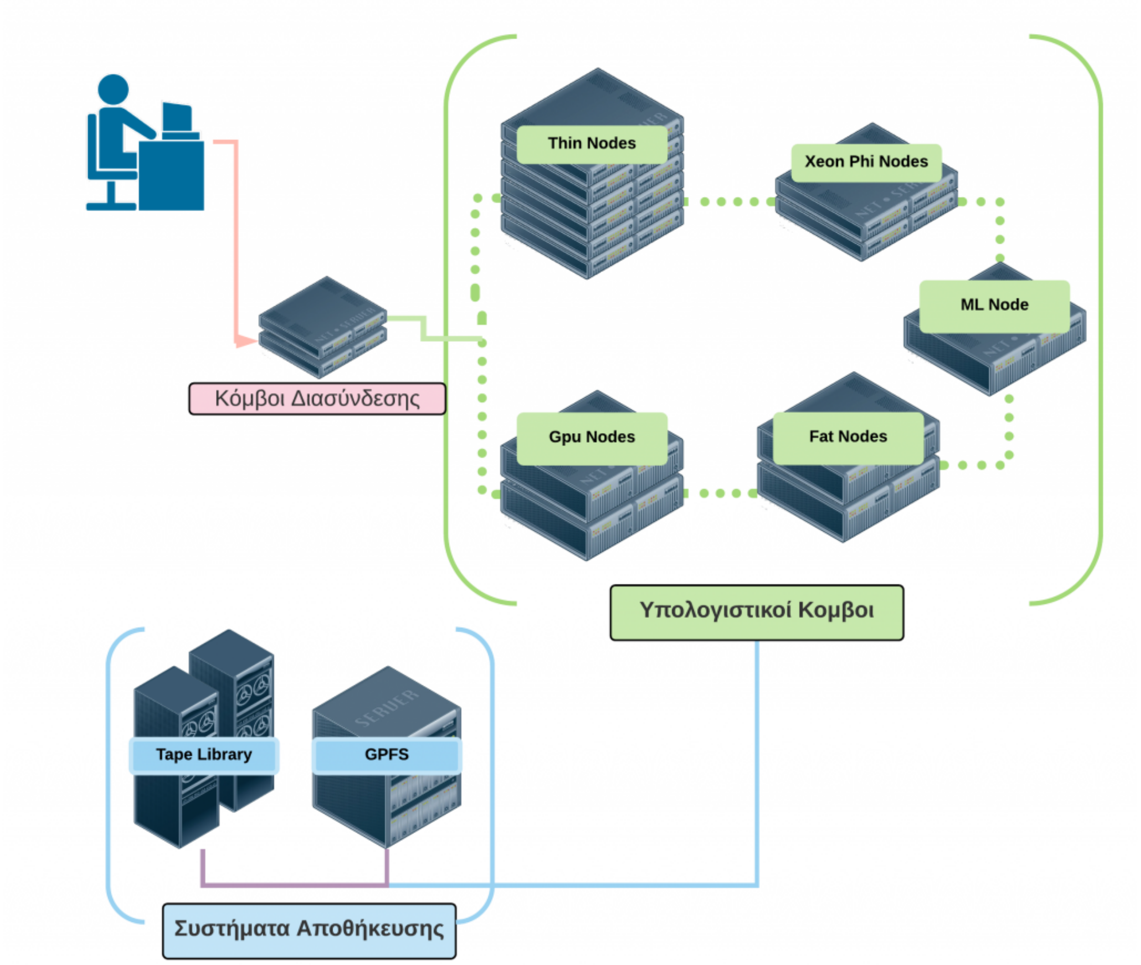

Figure 3.2: ARIS Architecture

3.3.4 Storage System

Programs running on supercomputers potentially generate a huge amount of data, which can be very difficult for standard filesystems and storage infrastructures to manage. Conventional data file systems may have a cap on file size, number of files, or total storage space. File systems used in supercomputers have the ability to expand, quickly transfer large volumes of data, and be accessible simultaneously from all node islands. ARIS for its file system implements IBM General Parallel File System (GPFS) technology by offering 2 PetaBytes of storage space to its users.

In addition to the data used directly, researchers often retain historical data for comparison or as a starting point for future work. Older data is stored in archival storage systems. An example of archiving is the magnetic tape storage system, which can store several petabytes (millions of gigabytes) of data. The ARIS infrastructure has such an IBM TS3500 movie library with a maximum storage capacity of more than 2 PetaByte. The library is used by researchers to archive data over long periods of time (several years) [6].

3.3.5 Data storage

The results of the work are stored in the common GPFS file system which is accessible from all ARIS island nodes and the interface node. A simple approach is to think of GPFS as a very large hard drive that stores all the data and applications in ARIS.

In particular, the ARIS file system offers two storage spaces depending on the data stored and their use:

- **$HOME where emphasis is placed on reliability versus performance.**

  Suitable for storing source code files, compiling applications and temporarily storing data files.

- **$WORKDIR with emphasis on performance (speed).**

  Suitable for storing large files, performing tasks and exporting results.

Users are restricted by file system usage limits. The limits are set according to the project requirements for storage resources.

After the completion of each project, the files corresponding to it are retained in the system for a period of about six months. Each user must ensure that their files are transferred back to their personal computer. The files are transferred to the computer according to the scp, sftp commands. Obviously, if at the request of the project the ARIS library has been requested to be used for data archiving, the appropriate procedures are performed for their preservation for the required period of time [6].

3.4 Profiles of applications running on ARIS

Suitable applications for the ARIS system are those that can be implemented by adopting a parallel processing model. A parallel application during its execution is divided into hundreds or even thousands of individual processes, which are performed simultaneously and collaboratively solve a common problem. To solve the problem, these processes must have access to the same data and communicate with each other by exchanging results. Depending on how this communication is achieved we can differentiate the way these applications are designed and developed and the programming tools to be used.

There are two common models of parallel application development: the shared memory model and the distributed memory model. In a shared memory system, the main memory can be accessed by all processes, while in a distributed memory system, the memory is not accessible between different processes and communication between processes is done by exchanging messages. In this type of applications, the network that connects the computing nodes through which the exchange of messages between processes takes place is very important.

The development of applications in shared memory systems is done with the help of the OpenMP standard, while in distributed memory systems the MPI (Message Passing Interface) standard is applied. In many cases the applications are developed in such a way as to combine both of the above communication models.

In recent years, the use of the Graphical Processing Unit (GPU) for the calculation of numerically demanding algorithms has been observed. The GPU's primary goal is to process two-dimensional and three-dimensional graphics that appear on the screen, which requires the simultaneous execution of commands in a large volume of data. This is achieved through the parallel processing, on which the construction of the GPU was based. ARIS has NVIDIA GPU processing units. To perform general operations on graphics cards, NVIDIA provides the API: CUDA (Compute Unified Device Architecture) programming interface.

The programming languages used to develop parallel applications are in most cases the usual general-purpose languages, such as C / C ++ or Fortran, which are extended with specialized libraries to support MPI, OpenMP, CUDA parallelism. Developers need to design their application in such a way that they run in the form of parallel collaborative processes using the capabilities offered by the MPI and OpenMP libraries [6].

3.5 Operating system

ARIS uses the Linux operating system in RedHat 6.9 and Centos 6.9 distributions. The interaction with the system is done through a command line and not from a graphical environment (as is typically done, for example, in an MS Windows system). The Linux command line is a powerful tool, with significantly more features than the graphical user interface. To use the system, it is necessary to know the basic commands that will be used during the connection, the transfer of files, the compilation of applications and the execution of tasks in the system.

3.5.1 Connection to the system

The only way to connect to the system is using the ssh (secure shell) protocol. Ssh allows a user to connect to the Internet from their own computer and use a remote computer from the command line, just as it would on its own local system. The connection to the system is not made with a password (password) but public key technology (public key cryptography) is used. Each user has a unique key that ssh uses along with the username, instead of a login password.

Access to ARIS login nodes from the user's computer. Interface nodes are the only ARIS nodes that have an Internet connection. From there, the work is sent for execution on one of the islands of computer nodes [6].

3.6 File Transfer

Any file to be used in ARIS, whether it is the application itself or the application data files, must be transferred from the user's computer to the interface nodes using the corresponding scp or sftp file transfer protocols [6].

3.7 Application development

ARIS has all the necessary tools for application development such as compilers, debugging software and profiling software that help optimize applications for the system.

Three different compiler suites are provided for compiling source code in C / C ++ and Fortran languages:

GNU (gcc, g++, gfortran) Intel® Parallel Studio XE Cluster Edition for Linux PGI Cluster Development Kit for Linux The corresponding OpenMPI and IntelMPI software versions are available for MPI parallel libraries. The NVIDIA compiler version for the CUDA nvcc developer model is available for code compression to run on graphics cards [6].

3.8 Pre-installed applications

ARIS offers a large number of pre-installed popular scientific applications and libraries, which are optimized and configured to make efficient use of the system. The best approach suggested to ARIS users is to use the pre-installed applications. Additional applications can be installed with the help of the support team. Finally, for new applications being developed, the appropriate parameters can be compiled by the users themselves using the compilation tools mentioned above [6].

3.9 Running applications

ARIS, like all major systems of its kind, follows the logic of batch job execution. To run an application it must first be described as a task and sent to a scheduler that runs it to run on the computing nodes. ARIS uses the SLURM program to distribute tasks on the supercomputer. SLURM has three basic functions. First, it has exclusive or non-exclusive access to the resources (computing nodes) to the users for a certain period of time, so that they can perform work. Second, it provides a framework for starting, executing, and monitoring tasks. End, ensures that computer nodes are shared across multiple users while avoiding competition issues. This is achieved with a task queue for each node island, in which fair and equal priority is applied to each user depending on the resources required by the application.

The user can route tasks from the interface node to the system via an appropriate SLURM (sbatch) command. In order for any task to be accepted and routed, it must specify some minimum specifications: the preferred island to execute, the number of nodes required, the number of cores per node, the maximum execution time and the program command to be executed . These specifications must be set in a text file called a batch script. For the creation of the specification file for ARIS there is the tool http://doc.aris.grnet.gr/scripttemplate/ which easily exports it.

A task when it is launched may not be executed immediately if the necessary resources are not available. In this case, it enters the appropriate priority queue and waits until the resources needed (eg the number of nodes required to execute it) are released, at which point SLURM will automatically start. The waiting time depends on the requirements of the job and ranges from a few minutes to several hours.

As can be seen from the above, ARIS is suitable for applications that are parallel, run autonomously within a task, with a specific execution time and do not communicate with the user or other applications. It is important to emphasize that it is not possible for the application to communicate via the Internet with external users in the system or to provide it in the form of a web service (web service) which runs in persistent services. The source about ARIS and the information in this chapter can be found in reference [6].

Chapter 4

Experimental Assessment

4.1 Utilization of a single node

In our initial testing, we utilized one of the THIN nodes from Aris: a single node with 64 GBs of memory and a CPU with 20 cores@2.8GHz. Of course, we wont be able to utilise the whole 64 GBs of the memory but instead werequested 56GBs which was still sufficient. We used the original code implementation where the image is split in 18 parts. Each part is processed serially, one after the other and once each processing is completed, the results are temporarily saved as images. Once the whole processing is over, all the images are merged to produce the final result. For this part of testing, the 34SCJ from the Sentinel 2 dataset was used. The results are shown in Table 4.1.

Number of Used Cores	Execution Time(mins)
5	236.19
10	171.88
20	167.82

Table 4.1: Execution in 1 node for different number of cores

We performed the tests for a different number of used cores to observe the impact of multiprocessing at the individual node level. At this point we tested only for 18 parts and didnt test the implementation for a different number of parts as because of the nature of the processing which is performed serially, the behaviour would be the same. In the above table we observe a 28% decrease in the execution time when the number of used cores is increased from 5 to 10 cores. However, increasing the number of used cores beyond the number of 10 used cores doesn't reduce the execution time significantly. Obviously, since each part is processed serially, this is a time consuming process. Later we expanded multiprocessing by utilizing a big number of nodes so that we observe the effects of multiprocessing in the parts level.

4.2 Re-Designing the initial pipeline

As mentioned before, our goal was to parallelize the iteration over the different parts and take advantage of the hardware to the maximum extent. We created a single process for each part: each process calls the virtual dates function for a each individual part. Accordingly the virtual dates function spawns subprocesses in order to perform the calculations on the pixel level. Also, taking into account the hardware capabilities of the THIN nodes in Aris we utilized up to 20 cores on each node - a maximum number of 18 cores was utilised by multiprocessing. We performed testing for 9, 15, 18, 30, 45 parts and also tested parallel processing of parts at the same node. Unfortunately, due to limitations with the memory on each node, we couldnt go to a smaller number of parts as this would require more memory which wasnt available on the THIN nodes of ARIS.

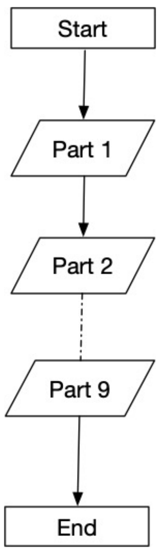

Figure 4.1: Serial Execution in 1 node

As it is seen in in Figure 4.2, in the initial implementation where 1 node is utilized, every part is executed one after the other. However, in the next figure, Figure 4.3 we have parallelized the tasks and each part was processed in parallel with the other parts.

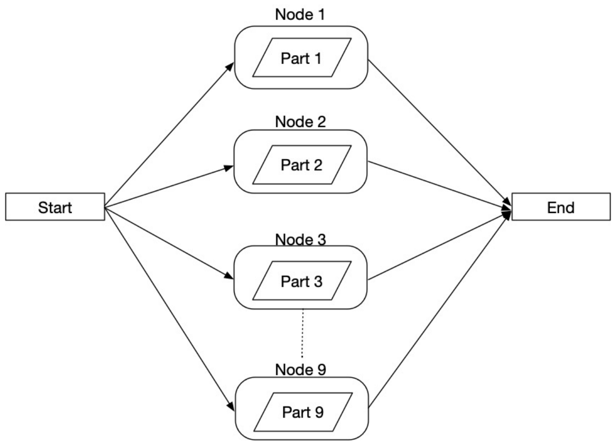

Figure 4.2: Parallel Execution in multiple nodes

It is important to note that the capabilities of the nodes and their technical characteristics allowed us to process more than one part in each node. That was dependent on the size of the memory and the size of each part: each node was able to process as many parts as they could fit in its memory.

4.3 Parallel Processing in multiple nodes

In order to make a comparison between serial execution and parallel processing, we used the 34SCJ tile for parallel processing as in the serial processing of section 4.2. We selected to process 18 parts which corresponds to 18 processes. Initially, we selected to run each process in each individual node so we utilized 18 nodes as it is shown in Figure 4.4:

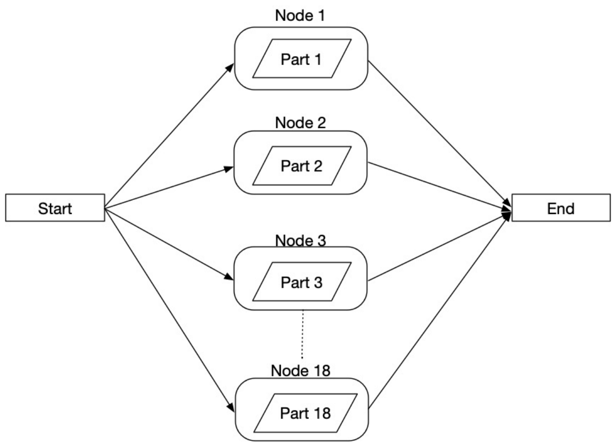

Figure 4.3: Parallel Execution for 34SCJ for 18 nodes

Below, in table 4.2, are the results(execution time in minutes) for the execution of each process:

Process 0	Process 1	Process 2	Process 3	Process 4	Process 5	Process 6	Process 7	Process 8
6.45	6.42	7.53	7.67	6.04	6.67	5.58	5.62	5.58
Process 9	**Process 10**	**Process 11**	**Process 12**	**Process 13**	**Process 14**	**Process 15**	**Process 16**	**Process 17**
5.57	5.61	5.6	5.56	5.55	5.58	5.55	5.44	5.55

Table 4.2: Execution of 18 parts in 18 nodes for the SCJ tile

We observe that the process with the maximum execution time is process 3 which lasted 7.67 minutes. Practically, this is the required amount of time to process tile 34SCJ through parallel processing. If we compare this value with the relevant time from Table 4.1 for the serial execution in 1 node while using 20 cores, we observe that the execution time dropped from 167.82 mins to 7.67 mins. This is a significant decrease in the execution time which pinpoints the tremendous effect of parallel processing on execution time.

As noted before, the technical characteristics of the nodes in terms of memory able to process more than one part in each node. Since the tiles have a size of approx 200-300 GBs, then in case of 18 parts, each part will have a size that ranges from 11.1 to 16.7 GBs. Therefore, we were able to process 2 parts with an overall size of maximum 33.4 GBs in one node since the node has 56GBs of memory. However, reducing the number of parts meant that the size of each part increased and that put a limit in processing more than 1 part in each node for a smaller number of parts. In our implementation, we used 18 parts and 9 nodes: 2 processes/parts per node, as it is shown in Figure:

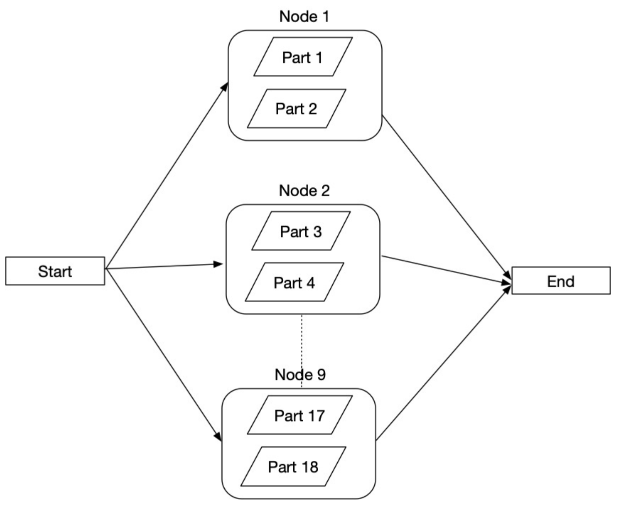

Figure 4.4: Serial Execution in 1 node

Below are the results(execution time in minutes) of the above implementation where we processed 18 parts by utilizing 9 nodes:

Process 0	Process 1	Process 2	Process 3	Process 4	Process 5	Process 6	Process 7	Process 8
7.55	7.63	8.87	9.27	7.44	7.87	7.28	7.43	7.31
Process 9	**Process 10**	**Process 11**	**Process 12**	**Process 13**	**Process 14**	**Process 15**	**Process 16**	**Process 17**
7.44	7.32	7.34	7.26	7.45	7.68	7.29	7.18	7.44

Table 4.3: Execution of 18 parts in 9 nodes for the SCJ tile

We observe that in this case, the maximum execution time was in process 4 which lasted 9.27 minutes and that is then the execution time for this implementation. If we compare the results from Table 4.2 with the results from Table 4.3, we observe that in the latter case, processing the same parts in 9 nodes instead of 18 nodes, lasted longer. That is explained by the fact that, since 2 processes run in 1 node, they had to both share the 18 cores, out of the overall 20 cores on each node, for the multiprocessing part. Therefore, each process had a smaller number of cores to utilize compared to the first implementation where each process run autonomously on each node.

4.4 Multiple Parts - Tile Comparison

In order to broaden our analysis, we proceeded with comparing the execution time and behaviour for a different number of parts-processes. as a baseline in this implementation, we decided that each process will run in a its own node. We performed tests for 9, 15, 18, 30 and 45 parts. Moreover, we decided not to limit our analysis in 1 tile, but rather expand it to several tiles from the dataset which covers the greek geographical area. It was important for us to detect patterns, behaviours and understand how the type of the terrain, sea or land, affects the overall execution and the multiprocessing part. Table 4.4 contains the results for the different number of parts(processes) for each of the 8 tiles that we decided to work with.

Tile	Time for 9 Parts	Time for 15 Parts	Time for 18 Parts	Time for 30 Parts	Time for 45 Parts
34SCJ	9.72	7.45	7.53	6.24	6.16
34SEG	16.88	10.31	8.63	7.31	7.22
34SEJ	16.88	10.31	8.63	7.34	7.22
34TEK	24.62	15.29	12.21	10.93	9.48
34TEL	22.33	13.06	11.43	10.23	9.72
34TDL	21.95	13.47	12.02	10.48	9.47
34TGM	21.27	13.08	11.46	10.14	9.58
34TMG	20.85	12.45	10.85	9.84	9.17

Table 4.4: Execution of multiple parts for different tiles

The results from Table 4.4 are presented in the below Figure:

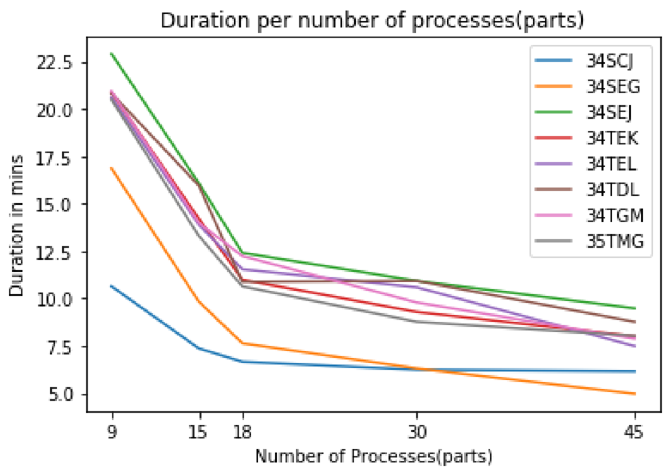

Figure 4.5: Graph comparing duration per number of processes for each tile

We observe that, as expected, the execution time is reduced as the number of processes increases. More specifically, during the transition from 9 to 18 processes, the duration drops approx. 40% to 50%. However, during the transition from 18 to 30 and 45 processes respectively, the decrease is smaller (20%). That implies that the number of 18 processes is the threshold after which multiprocessing doesn't offer a significant gain. In other words, the overhead and the cost in time of the other processes in the pipeline are much bigger than the gain which the multiprocessing part has. This is something that we explored further in later sections.

4.5 Effect of the terrain on multiprocessing

As a next step in our work, we decided to investigate how the terrain affects multiprocessing. More specifically, we wanted to understand how the different percentage of sea and land in each tile, relates to the execution time.

Tile	Percentage of Land	Percentage of Sea	Time for 9 Parts	Time for 15 Parts	Time for 18 Parts
34SCJ	4%	96%	9.72	7.45	7.53
34SEG	61%	39%	16.88	10.31	8.63
34SEJ	99%	1%	24.62	15.29	12.21
34TEK	100%	0%	22.02	13.18	11.23
34TEL	100%	0%	22.33	13.06	11.43
34TDL	100%	0%	21.95	13.47	12.02
34TGM	100%	0%	21.27	13.08	11.46
35TMG	100%	0%	20.85	12.45	10.85
34SDJ	57%	43%	20.89	12.92	11.83
34SEF	9%	91%	13.47	10.18	9.81
34SEH	83%	17%	23.36	14.09	11.77
34SFF	33%	67%	22.33	13.06	11.43
34SFG	78%	22%	22.19	13.82	12.69
34SFH	77%	23%	23.81	13.42	13.43
34SFJ	76%	24%	22.14	14.58	11.78
34SFJ	62%	38%	16.86	11.84	7.6

Table 4.5: Execution of multiple parts for different tiles and analysis on the terrain

Firstly we calculated the percentage of sea and land in each tile. We did this for the tiles from our previous testing which was demonstrated in Table 4.4 and also included new tiles: 34SDJ,34SEF,34SEH,34SFF,34SFG,34SFH,34SFJ,34SEG. We did that in order to have a big variety of tiles with different percentages of sea and land. As in the previous testing for the other tiles, we performed tests for 9, 15, 18, 30 and 45 parts for these tiles as well. In Table 4.5, we have gathered the results for all the tiles: tile name, percentage of sea and land, execution duration in minutes for each process.

We used the above data to create graphs which demonstrate the correlation between the terrain and the execution duration for different number of processes.

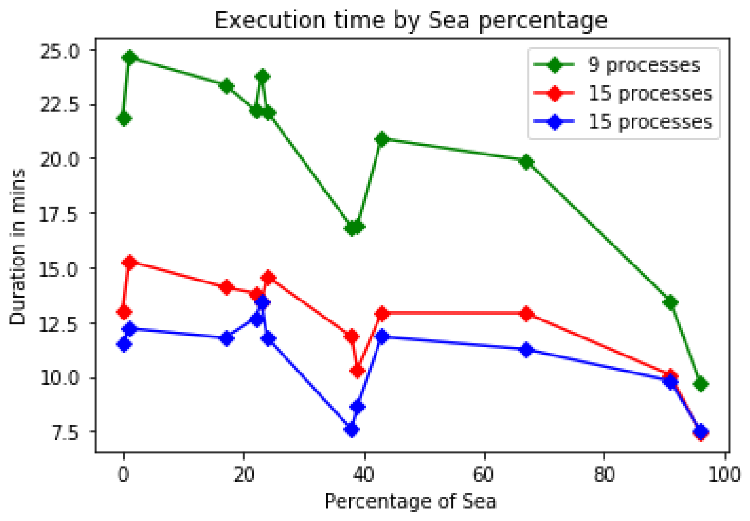

Figure 4.6: Distribution of Sea and Land for each tile – Execution time

We observe that, overally, as the percentage of Sea in the graph increases, the execution time is decreased, especially for the smaller number of processes. This is in accordance with the increase of the relevant processes. However, there are a few exceptions, specifically in the area of 40%, where although the percentage of sea increases, the execution time increases as well. This is also more evident in the bigger number of processes and implies that the execution time is dependent on the distribution of sea in the parts(processes) and there is no guarantee that the percentage of sea is equally distributed between the processes. In other words, the execution time is not only affected by the distribution of land and sea in the tile but also by how the this distribution differs within the subprocesses.

Moreover, we observe that, in some cases, as the percentage of sea increases, the execution time also increases which is not expected. This is evident in the execution time of the 34SEG and 34SDJ tiles which have 38% and 43% of sea respectively.

In that direction it is worth investigating the execution time of each process and the relevant distribution for each of these tiles as well as with the tiles that have 0 and almost 100% sea(tiles 34TEK and 34SEF).

Below, in Figure 4.8 we demonstrate the execution time by process for each of the aforementioned tiles: 34TEK, 34SEF, 34SEG, 34SDJ. We chose to work with 9 parts(processes).

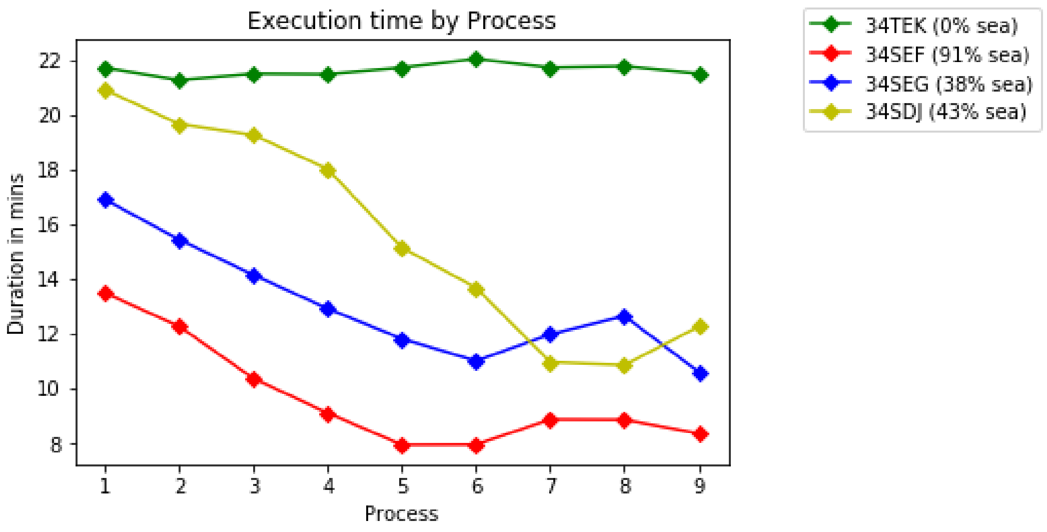

Figure 4.7: Execution time per process for specific tiles

We observe that the execution time is equally distributed for each process at the 34TEK tile which consists of 0% sea and deviates significantly at the 34SDJ tile which consists of 43% sea.

If we analyze further the execution time for the 34SDJ tile we observe that the 1st process lasted 20.89 minutes while the 9th process lasted 12.24 minutes. After reviewing the logs, it appears that the multiprocessing stage for the 10m band in the 1st case lasted 11 mins while the same stage in the 2nd case lasted 4.36 mins. The relevant chink size was 160570 for the 1st process and 65641 for the latter. That explains the fference for the execution time during multiprocessing. The iterable is split into bigger pieces, and each piece is submitted as a separate task which results into a bigger execution time

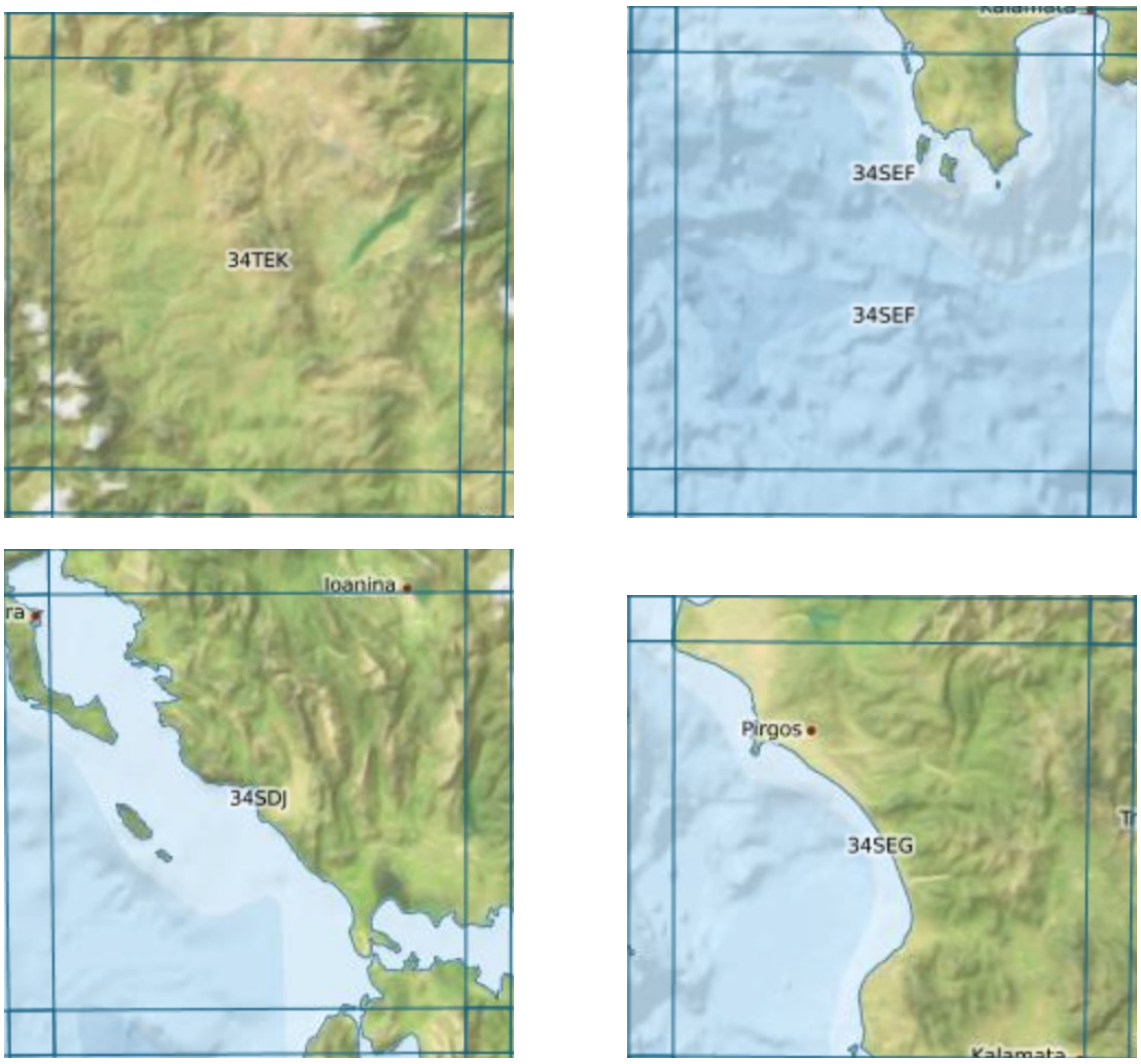

Figure 4.8: Tiles 34TEK, 34SEF, 34SDJ, 34SEG

In the above figrure 4.9 we can see the actual image of each tile. We can see that the 34TEK consists of only land. That explains the consistency in the execution time of each process: every process will process only land so practically the execution time remains almost the same. The other 3 tiles, though, contain both sea and land. It is important to observe how the sea and land is distributed between the different parts that the image is cut into in order to be processed.

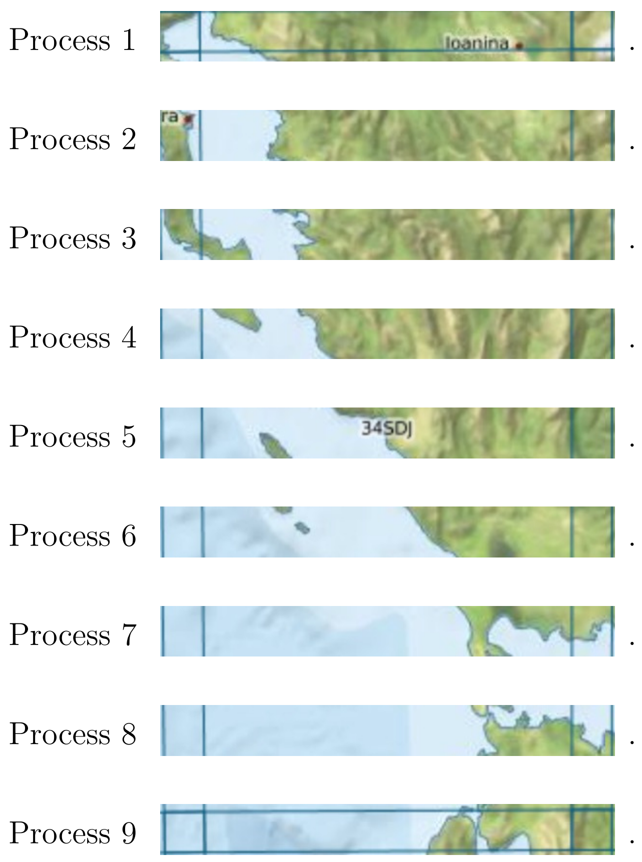

Figure 4.9: Splitting the tile 34SDJ in 9 parts

In Figure 4.10 we observe that the process 1 contains the most land terrain compared to the rest of the processes. Also, it is evident that as we move from the top to the bottom of the tile the percentage of sea in each part/process increases compared to the percentage of land. These observations justify completely the results in Figure 4.8 where the execution time decreases significantly moving from process 1 up to process 9. Accordingly, if we split the the tiles 34SEF and 34SEG into 9 parts as above, we will observe the relevant behaviour in the distribution between land and see in the processes which justifies the results of Figure 4.8.

In order to broaden our analysis, we worked towards the direction of identifying the root cause at the technical level which causes the difference of th execution time between processes. More specifically, in the next section we investigated how the chunksize affects multiprocessing and how the terrain affects the chunksize.

4.6 The effect of chunksize on multiprocessing

The chunksize is a parameter in multiprocessing which defines the size of the pieces into which the iterable will be split. Each of these pieces is submitted as a different task. In multiprocessing, each task is submitted to a core, so practically the chunksize determines how big the task which is submitted to a processor will be. The default value of the chunksize is 1. This means that each worker-core received a new tasks only after it has processed the last received one. However, if the chunksize is set to a value higher than 1, then the worker receives a batch of tasks a the same time and will receive the next batch if it exists, only if it has processed the previous one.

The big challenge here is to determine a good value of the chunksize. Distributing items one-by-one with chunksize=1 increases flexibility of scheduling while it decreases overall throughput, because drip feeding requires more inter-process communication (IPC). A task (as unit of work) consists of chunksize "minitasks". The term "minitasks" is used to differentiate the tasks in each chunksize from the actual tasks as defined in multiprocessing. In case we would be unable to predict how long a minitask would need to be completed e.g. in an optimization problem, where the processing time greatly varies across minitasks.

In this implementation, the minitasks based on our tests, take approximately the same time to finish. So we set the chunksize equal to the result of the division of the iterable and the number of the processes. In case there is a remainder in the division, 1 is added to the chunksize. This means that if a remainder exists, one extra task will be created. Potentially that could severely impact the overall computation time. However, we didnt observe such an issue in our experiments.

As part of our analysis, we extracted the relevant chunksizes in each part(process) of every tile. Since the multiprocessing takes place for every single band, we covered both bands: 10m and 20m. Initially, we related the chunksize to the number of processes as well as to the different bands. From that point, we worked on correlating the execution time with the chunksize and the nature of the terrain. This allowed us to determine how the percentage of land and sea in the tiles affects the chunksize and consequently the execution time of the multiprocessing. We created the relevant diagrams to demonstrate the observations and extract conclusions based on them.

4.6.1 Execution time per chunksize

First of all, it is important to note that multiprocessing takes place only in parts of the tiles where there is land or a mixture of land and sea. In parts of the tiles where there is only sea, multiprocessing is not used as the computational complexity is low. For our tests, we selected a few datasets: 34SCJ , 34SEG, 34TEK and 34SDJ. We performed the tests separately for 10m and 20m bands. That allowed us to get a broad variety of different chunksizes and the relevant execution times. In order to perform the tests, since we used 9 parts(processes) per band, we utilised 18 nodes from Aris. In the following tables we have summoned the relevant results for the 10m band.

Tile	Process 1	Process 2	Process 3	Process 4	Process 5	Process 6	Process 7	Process 8	Process 9
34SCJ	2.38	1.73	1.32	0.71					
34SEG	9.74	8.42	7.6	5.98	5.05	4.69	5.22	5.6	4.5
34TEK	11.66	11.62	11.59	11.54	11.68	11.70	11.71	11.53	11.68
34SDJ	11.07	10.02	10.03	8.62	6.71	5.37	3.12	2.62	4.36

Table 4.6: Execution time per process for the 10m band in selected tiles

Tile	Process 1	Process 2	Process 3	Process 4	Process 5	Process 6	Process 7	Process 8	Process 9
34SCJ	31817	19486	11883	219					
34SEG	161306	142598	130762	103085	87152	79760	90708	96309	76668
34TEK	176258	176258	176258	176258	176258	176258	176258	176258	176258
34SDJ	160570	147251	146713	127312	101071	81626	45268	36484	65641

Table 4.7: Chunksize per process for the 10m band in selected tiles

Although, we had the option to perform the tests either by using 15 or 18 parts per band, there would be no value as the chunksizes and the execution times from the testing in 9 parts cover the necessary range.

Below are the relevant results for the 20m band in the selected tiles:

Tile	Process 1	Process 2	Process 3	Process 4	Process 5	Process 6	Process 7	Process 8	Process 9
34SCJ	0.59	0.42	0.32	0.16					
34SEG	1.93	1.73	1.6	1.27	1.12	1.03	1.15	1.21	0.97
34TEK	2.27	2.28	2.26	2.25	2.26	2.26	2.25	2.26	2.26
34SDJ	2.29	2.09	2.07	1.8	1.44	1.18	0.74	0.63	1.00

Table 4.8: Execution time per process for the 20m band in selected tiles

Tile	Process 1	Process 2	Process 3	Process 4	Process 5	Process 6	Process 7	Process 8	Process 9
34SCJ	7956	4874	2971	56					
34SEG	40324	35651	32691	25772	21788	19938	22675	24075	19170
34TEK	44065	44065	44065	44065	44065	44065	44065	44065	44065
34SDJ	40145	36813	36677	31830	25268	20408	11321	9118	16407

Table 4.9: Chunksize per process for the 20m band in selected tiles

After combining the results from the tables 4.6, 4.7, 4.8 and 4.9 we created a diagram which demonstrates the correlation between the chunksize and the execution time for the specific bands.

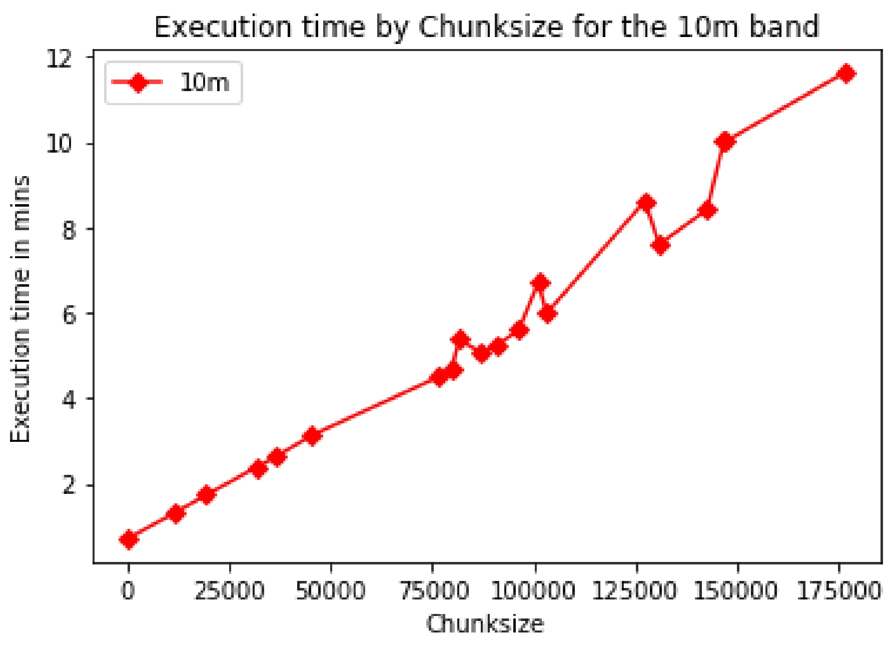

Figure 4.10: Execution time by Chunksize for the 10m band

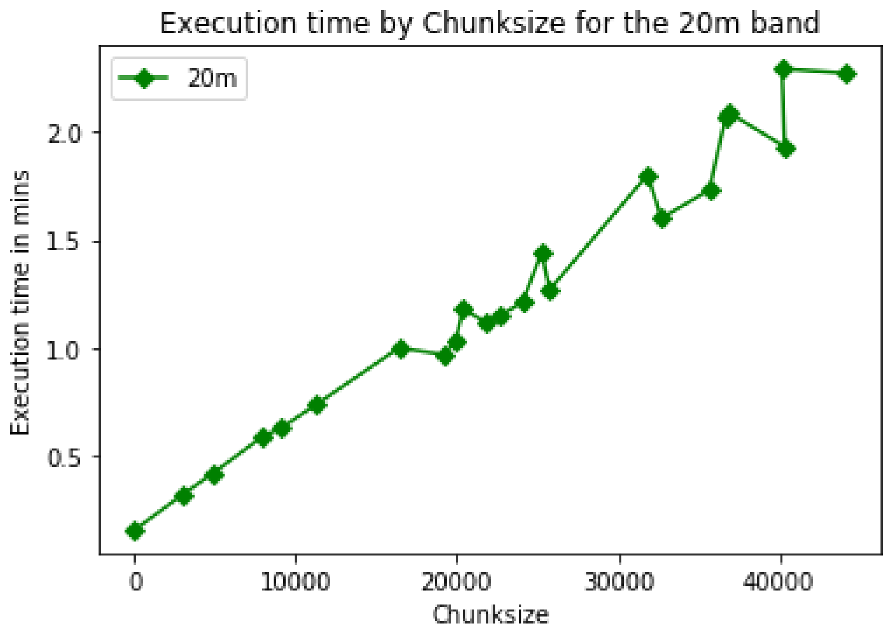

Figure 4.11: Execution time by Chunksize for the 20m band

In both cases, we observe that as the chunksize inceases so does the execution time. There a few exeptions between neighboring observations regarding the chunksize but the difference in the execution time is negligible. In any case, it can be justified in terms of computational complexity and hardware characteristics at the time of the execution.

Another observation that we make is that the chunksize is considerably bigger in the processes of the 10m band than the chunksize in the processes of the 20m band. That is justified by the fact that number of pixels that are processed is smaller: the iterable is smaller and, based on the algorithm that we use to compute it, the chunksize is smaller.

Last but not least, similarly to the previous observation about the difference in the bands regarding the chunksize, we can correlate the chunksie to the percentage of land and sea in the tiles. Taking into consideration the percentage of land sea in the relevant tiles from Table 4.5, combined with the results in Tables 4.6-4.9, we observe that as the percentage of land increases so does the chunksize and the execution time. Same way, as the percentage of sea in the tile increases, the chunksize is reduced as well as the execution time. However, note that this is the general observation as the distribution of land and sea within the processes is also very important: a tile may have a relatively small percentage of land but the vast majority of it is collected by one process. That process will have a considerably bigger chunksize in multiprocessing compared to the others, and that can lead to a significantly increased execution time.

Chapter 5

Conclusion

In this work, the main idea was to exploit and define strategies to perform remote sensing tasks which are computationally intensive while utilizing the cluster of a HPC. Our first goal, of fully taking advantage the capabilities of the cluster and the nodes in terms of parallel computing was achieved: we managed to drop dramatically the execution time as the execution which used to take around 3 hours for each tile was performed in a few minutes. Based on the limitations in terms of memory in the THIN nodes of Aris, we run 2 processes per node in parallel, fully utilizing the resources.

Moreover, we identified the threshold where further parallelization doesnt offer a significant gain: based on the technical specifications of our implementation(size of data, memory availability on the nodes etc.) that occurred beyond the 18 processes. Before reaching that threshold we analyzed the behaviour of the processes while increasing the number of them.

It is important to note that we gained very useful insights about the effect of the terrain on multiprocessing for our implementation: as a general rule, as the percentage of sea in the tiles increases, the execution time is reduced. Vice versa, the more land a tile has, the more time it will take to be processed. In order to undesrstand fully this behavior, we broke down our analysis to the process level.

We came to the conclusion that the distribution of land and sea within the processes plays a significant role in the execution time. Since the overall execution time of a tile is determined by the last process to finish, the percentage of land or sea in that particular process will affect the overall distribution time. A tile with a relatively small percentage of land may still a considerable amount of time, compared to other tiles, simply because a single process may possess the biggest part of that land.

Last but not least, we determined the correlation between the chunksize which is the most important factor of multiprocessing, the terrain in the tile and the execution time. The chunksize increases as the process which corresponds to a part of the tile, contains more land and accordingly so does the execution time. On the other hand, more sea in the tile means a smaller chunksize resulting in a faster execution.

Overally, through this work, we determined some very useful strategies not only to fully take advantage of the capabilities of a hypercomputer cluster like Aris and optimize our implementation in terms of parallel processing but also make useful observations about how this process is affected by certain characteristics of remote sensing tasks.

5.1 Future Work

The vast capabilities of the ARIS HPC definitely provide room for expansion of the current work. First of all, it would be really interesting to utilize the FAT nodes in ARIS. The enhanced capabilities of these nodes in terms of memory, cpu etc. would be really useful tools in the quest to optimize the parallel processing of our implementation. Furthermore, the core multiprocessing process could become more efficient by exploiting the CUDA library that ARIS offers: by having the option to run the multiprocessing part in graphic cards, the execution time of that step would be significantly decreased. It is also worth noting that optimization could also take at the code level to make the whole process more efficient. Last but not least, one topic for future work would be to apply the aforementioned techniques and implementation in a different dataset: different pictures with other terrain characteristics could potentially pose new challenges in terms of complexity and optimization of the parallelization process.

Bibliography

[1] https://www.omnisci.com

[2] https://https://www.esri.com

[3] https://https://gisandscience.com

[4] https://gdal.org/about.html

[5] https://eos.com/find-satellite/sentinel-2/

[6] https://hpc.grnet.gr

[7] Thomas Sterling, Maciej Brodowicz, Matthew Anderson *High Performance Computing Modern Systems and Practices* 2017.

[8] Bertil Schmidt, Jorge Gonzalez-Dominguez, Christian Hundt, Moritz Schlarb *Parallel Programming Concepts and Practice* 2017.

[9] Jim Jeffers, James Reinders *High Performance Parallelism Pearls Volume Two Multicore and Many-core Programming Approaches* 2015.

[10] Gerhard Joubert, Wolfgang Nagel, Frans Peters, Wolfgang Walter *Parallel Computing: Software Technology, Algorithms, Architectures & Applications* 2004.

[11] Senapathi Venkatramanan, Prasanna Mohan Viswanathan, Sang Yong Chung *GIS and Geostatistical Techniques for Groundwater Science* 2019.

[12] Seonmyeong Bak, Colleen Bertoni, Swen Boehm, Reuben Budiardja, Barbara M.Chapman, Johannes Doerfert, Markus Eisenbach, Hal Finkel, Oscar Hernandez, Joseph Huber, Shintaro Iwasaki, Vivek Kaleb, Paul R.C. Kent, JaeHyuk Kwack, Meifeng Lin, Piotr Luszczek, Ye Luo, Buu Pham, Swaroop Pophale, Kiran Ravikumar, Vivek Sarkar, Thomas Scogland, Shilei Tian, P.K.Yeung *OpenMP application experiences: Porting to accelerated nodes* 2021.

[13] Mehmet Deveci, Christian Trott, Sivasankaran, Rajamanickam *Multithreaded sparse matrix-matrix multiplication for many-core and GPU architectures* 2019.

[14] Vasco Amaral, Beatriz Norberto, Miguel Goulão, Marco Aldinucci, Siegfried Benkner, Andrea Bracciali, Paulo Carreira, Edgars Celms, Luís Correia, Clemens Grelck, Helen Karatza, Christoph Kessler, Peter Kilpatrick, Hugo Martiniano, Ilias Mavridis, Sabri Pllana, Ana Respício, José Simão, Luís Veiga, Ari Visa *Programming languages for data-Intensive HPC applications: A systematic mapping study* 2019.

[15] I. Masliah, A. Abdelfattah, A. Haidar, S. Tomov, M. Baboulin, J. Falcou, J. Dongarra *Algorithms and optimization techniques for high-performance matrix-matrix multiplications of very small matrices* 2018.

[16] Ivy Bo Peng, Roberto Gioiosa, Gokcen Kestor, Jeffrey S.Vetter, Pietro Cicotti, Erwin Laure, Stefano Markidis *Characterizing the performance benefit of hybrid memory system for HPC applications* 2018.

[17] Kurt B.Ferreira, Scott Levy *Evaluating MPI resource usage summary statistics* 2021.

[18] Craig A. Lee, Samuel D. Gasster, Antonio Plaza, Chein-I Chang, Bormin Huang *Recent Developments in High Performance Computing for Remote Sensing: A Review* 2011.

[19] Ujwala M. Bhangale, Kuldeep R. Kurte, Surya S. Durbha, Roger L. King, Nicolas H. Younan *Big data processing using hpc for remote sensing disaster data* 2016.

[20] Mingmin Chi, Antonio Plaza, Jón Atli Benediktsson, Zhongyi Sun, Jinsheng Shen, Yangyong Zhu *Big Data for Remote Sensing: Challenges and Opportunities* 2016.

[21] Yan Ma, Haiping Wub, Lizh Wang, Bormin Huang, Rajiv Ranjan, Albert Zomaya, Wei Jie *Remote sensing big data computing: Challenges and opportunities* 2016.

[22] S.N.V.Kalluri, J.Jaja, D.A.Bader, Z.Zhang , J.R.G.Townshend, H.Fallah-Adl *High performance computing algorithms for land cover dynamics using remote sensing data* 2010.

[23] Antonio J. Plaza, Chein-I Chang*High performance computing in Remote Sensing* 2008.

[24] Emmanuel Christophe, Julien Michel, Jordi Inglada*Remote Sensing Processing: From Multicore to GPU* 2011

[25] Emmanuel Christophe, Julien Michel, Jordi Inglada*A Hadoop-based distributed framework for efficient managing and processing big remote sensing images.* 2015

[26] Jinwei Dong, Graciela Metternicht, Patrick Hostert, Rasmus Fensholt, Rinku Roy Chowdhury*Remote sensing and geospatial technologies in support of a normative land system science: status and prospects* 2019

[27] Jianting A Zhang*Towards personal high-performance geospatial computing (HPC-G): perspectives and a case study* 2010

[28] Natalija Stojanovic, Dragan Stojanovic*High–performance computing in GIS: techniques and applications* 2013

[29] Kenneth A Hawick, P.D Coddington, H.A James*Distributed frameworks and parallel algorithms for processing large-scale geographic data* 2003

[30] M.Drusch, U.Del Bello, S.Carlier, O.Colin, V.Fernandez, F.Gascon, B.Hoersch, C.Isola, P.Laberinti, P.Martimort, A.Meygret, F.Spoto, O.Sy, F.Marchese, P.Bargellini*Sentinel-2: ESA's Optical High-Resolution Mission for GMES Operational Services* 2012

[31] Jie Yang, Anmol Paudel, Satish Puri*Spatial Data Decomposition and Load Balancing on HPC Platforms* 2019

[32] Karantzalos K., Bliziotis D., Karmas A.*A Scalable Geospatial Service for Near Real-Time, High-Resolution Land Cover Mapping* 2015

[33] Karmas A., Tzotsos A., Karantzalos K.*Big Geospatial Data for Environmental and Agricultural Applications, Yu and Guo (eds.), Big Data Concepts, Theories and Applications* 2016

[34] Karakizi C., Karantzalos K., Vakalopoulou M., Antoniou G.*Detailed Land Cover Mapping from Multitemporal Landsat-8 Data of Different Cloud Cover, Remote Sensing* 2018