Χαροκόπειο Πανεπιστήμιο
Τμήμα Πληροφορικής και Τηλεματικής

Παραλληλοποίηση αλγορίθμων χαρτογράφησης καμένων εκτάσεων σε δορυφορικά δεδομένα

Πτυχιακή Εργασία
του
Νικόλαου Κανακάρη

Επιβλέπων: Δημήτριος Μιχαήλ

Αθήνα, Απρίλιος 2016

Επιβλέπων: Δημήτριος Μιχαήλ
Επίκουρος καθηγητής
Ph.D. Max-Planck-Institut für Informatik

Μέλος επιτροπής: Αλέξανδρος Δημόπουλος
Εξωτερικός διδάσκων
Ph.D. Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών Ε.Μ.Π.

Μέλος επιτροπής: Κωνσταντίνος Τσερπές
Λέκτορας
Ph.D. Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών Ε.Μ.Π.

To my family and Haroula.
Thank you for all the support and love.

In theory, there is no difference between theory and practice. But, in practice, there is.

- Jan L. A. van de Snepscheut

Περίληψη

Στις μέρες μας, η χλωρίδα του πλανήτη μειώνεται με ραγδαίο ρυθμό. Αντίθετα, οι πυρκαγιές που δημιουργούνται σε δασικές εκτάσεις όλο και αυξάνονται. Έτσι, έχουν δημιουργηθεί αρκετά προγράμματα και αποστολές τηλεπισκόπησης με σκοπό τη συλλογή δεδομένων προκειμένου να χαρτογραφηθούν καμένες εκτάσεις. Το πρόγραμμα Landsat της NASA (National Aeronautics and Space Administration) ασχολείται με την παροχή δεδομένων τα οποία μπορούν να χρησιμοποιηθούν για λόγους χαρτογράφησης. Η ESA (European Space Agency) αντίστοιχα παρέχει δορυφορικά δεδομένα χάρη στο πρόγραμμα Copernicus, με το οποίο σχεδιάζεται μία σειρά δορυφορικών αποστολών με όνομα Sentinels.

Το Εθνικό Αστεροσκοπείο Αθηνών έχει αναπτύξει διάφορα συστήματα που υλοποιούν αλγόριθμους και φίλτρα ψηφιακής επεξεργασίας εικόνας με στόχο τη χαρτογράφηση των καμένων εκτάσεων της ελληνικής επικράτειας. Τα δεδομένα που χρησιμοποιούνται είναι κυρίως από τους δορυφόρους του προγράμματος Landsat. Οι δορυφορικές εικόνες τύπου Landsat είναι υψηλής ανάλυσης και μεγάλου μεγέθους. Συνεπώς, η ανάγκη για επεξεργασία πολλών και μεγάλων εικόνων καθιστά τη σειριακή υλοποίηση του συστήματος μη ικανοποιητική από άποψη χρόνου.

Η παρούσα πτυχιακή εργασία ασχολείται με την παραλληλοποίηση των αλγορίθμων και των φίλτρων του συστήματος χαρτογράφησης καμένων εκτάσεων που είχε υλοποιηθεί από το Εθνικό Αστεροσκοπείο Αθηνών. Για τις ανάγκες της πτυχιακής χρησιμοποιείται η γλώσσα προγραμματισμού Python και η διεπαφή περάσματος μηνυμάτων MPI (Message Passing Interface). Η παραλληλοποίηση καταφέρνει και μειώνει το μέσο συνολικό χρόνο εκτέλεσης του συστήματος από 14.7 λεπτά σε λιγότερο από 1 λεπτό, χρησιμοποιώντας έως και 33 τετραπύρηνους υπολογιστές του εργαστηρίου του τμήματος Πληροφορικής και Τηλεματικής.

Λέξεις κλειδιά

burn scar mapping, τηλεπισκόπηση, ψηφιακή επεξεργασία εικόνας, χωρικό φίλτρο, λευκαύγεια (Albedo), δείκτης βλάστησης NDVI, NBR, παράλληλος προγραμματισμός, προγραμματισμός περάσματος μηνυμάτων, MPI, επιτάχυνση παράλληλου συστήματος, απόδοση παράλληλου αλγόριθμου, Breadth-First Search (BFS), union-find

Abstract

Nowadays, the plant life all over the world is decreasing rapidly. On the contrary, the breakouts of fires, in forests, are increasing. As a result, many programs and missions have been created, as far as remote sensing is concerned. Their goal is to collect data, in order to map out burnt areas. NASA's Landsat Program provides information that can be used in terms of 'burn-scar' mapping. ESA (European Space Agency) also offers its services via the program 'Copernicus' which is responsible for satellite missions called 'Sentinels'.

The National Observatory of Athens has played a significant role in mapping out burnt areas throughout Greek territory, by developing different systems, which apply algorithms and filters on digital image processing. The data used is mostly from the Landsat Program, due to the fact its' satellite images are high quality and large in size. Therefore, the need for processing a large number of images that are also big in size, makes sequential implementation of the system not sufficient enough, as far as time is concerned.

This thesis has worked on the algorithm and filter parallelization of the 'burn-scar' mapping system that was implemented by the National Observatory of Athens. For the actualization of the thesis, the programming language Python and Message Passing Interface (MPI) are used. The parallelization achieves a decrease of the total execution time from 14,7 minutes to less than 1 minute, using up to 33 quad-core computers at the lab of the Department of Informatics and Telematics.

Keywords

burn scar mapping, remote sensing, digital image processing, spatial filter, Albedo, Normalized Difference Vegetation Index (NDVI), Normalized Burn Ratio (NBR), parallel programming, distributed memory, Message Passing Interface (MPI), speedup, efficiency, Breadth-First Search (BFS), union-find, disjoint sets

Ευχαριστίες

Αρχικά, χρωστάω ένα μεγάλο ευχαριστώ στον κ. Δημήτριο Μιχαήλ, ο οποίος με εμπιστεύθηκε για την εκπόνηση αυτής της πτυχιακής εργασίας. Ακόμα, τον ευχαριστώ για τις γνώσεις, τις συμβουλές και τις άπειρες ώρες συζητήσεων μαζί του, για αρκετά θέματα, σχετικά με την επιστήμη των υπολογιστών. Τέλος, θεωρώ πως ήταν εκείνος που βοήθησε στο να αγαπήσω περισσότερο τον προγραμματισμό, μέσα από τη διδασκαλία του.

Έπειτα, θα ήθελα να ευχαριστήσω τον κ. Αλέξανδρο Δημόπουλο, τον κ. Κωνσταντίνο Τσερπέ, καθώς και τον κ. Ηρακλή Βαρλάμη, για τις γνώσεις που μου έδωσαν, καθ' όλη τη διάρκεια της φοίτησής μου στο τμήμα.

Θα ήθελα να ευχαριστήσω όλους τους φίλους μου από τη σχολή, οι οποίοι μου πρόσφεραν μία όμορφη παρέα. Επίσης, θα ήθελα να ευχαριστήσω τη φίλη μου Κατερίνα Ρουκουνάκη[1] που μου δίδαξε να μην θεοποιώ κανέναν, και απλά να προσπαθώ να μάθω αυτά που ξέρει.

Οφείλω ένα από τα μεγαλύτερα ευχαριστώ στην οικογένειά μου, που μου δείχνει την αγάπη της καθημερινά, και παρά τον περίεργο χαρακτήρα μου, με στηρίζει σε όλες τις επιλογές μου. Τέλος, ένα μεγάλο ευχαριστώ πηγαίνει στη Χαρούλα Μπαλιάκα, η οποία με βοηθάει και ακούει τους προβληματισμούς μου κάθε μέρα, αλλά και επειδή ήταν εκείνη που προσπάθησε να διορθώσει αρκετά εκφραστικά λάθη που είχε αυτό το κείμενο, παρά το γεγονός ότι αρνήθηκα να διορθώσω μερικά από αυτά.

[1] std::cout « "Java is nothing but a piece of shit!" « std::endl;

Προς τους αναγνώστες

Για την κατανόηση της πτυχιακής εργασίας, θεωρείται πως ο αναγνώστης είναι τουλάχιστον δευτεροετής φοιτητής σε τμήμα Πληροφορικής, Επιστήμης Υπολογιστών ή Ηλεκτρολόγων Μηχανικών, που προορίζεται για την κατεύθυνση λογισμικού (Λ). Επομένως, ο αναγνώστης γνωρίζει τι σημαίνει αλγόριθμος, γράφημα (graph), πίνακας, καθώς και ορισμένους αλγόριθμους, όπως ο BFS ή τη δομή ξένων συνόλων Union-Find. Επίσης, χρήσιμες είναι οι γνώσεις σε τεχνικές ψηφιακής επεξεργασίας εικόνας και προγραμματισμού.

Για την υλοποίηση της εφαρμογής χρησιμοποιήθηκε η γλώσσα προγραμματισμού Python 2.7. Μερικές από τις βιβλιοθήκες (modules) που χρησιμοποιήθηκαν ήταν οι mpi4py[2], gdal[3], NumPy[4], collections, logging. Για το profiling της εφαρμογής χρησιμοποιήθηκε η βιβλιοθήκη cProfile[5]. Ο κώδικας αποτελεί ιδιοκτησία του Εθνικού Αστεροσκοπείου Αθηνών, αλλά μπορεί να παρουσιαστεί κατόπιν συνεννόησης (michail@hua.gr, it21113@hua.gr).

Για τη συγγραφή του κειμένου, χρησιμοποιήθηκε το σύστημα προετοιμασίας εγγράφων (document preparation system) LaTeX. Αρκετά από τα σχήματα και διαγράμματα ροής σχεδιάστηκαν με την εφαρμογή Dia[6] καθώς και με τη χρήση του πακέτου TikZ[7]. Οι γραφικές παραστάσεις αρχικά έγιναν με gnuplot[8], αλλά αργότερα με τη βιβλιοθήκη matplotlib[9]. Για τη δημιουργία γραφημάτων χρησιμοποιήθηκε η γλώσσα DOT[10]. Για την παρουσίαση δορυφορικών εικόνων χρησιμοποιήθηκε η εφαρμογή QGIS[11]. Τέλος, όπως και για τον κώδικα, έτσι και για το κείμενο, ως επεξεργαστής κειμένου χρησιμοποιήθηκε ο Vim[12] στις διανομές ubuntu και debian του Linux, χρησιμοποιώντας ως window manager το xmonad[13].

[2] https://pythonhosted.org/mpi4py
[3] http://www.gdal.org
[4] http://www.numpy.org
[5] https://docs.python.org/2/library/profile.html
[6] http://dia-installer.de
[7] http://www.texample.net/tikz
[8] http://www.gnuplot.info
[9] http://matplotlib.org
[10] https://en.wikipedia.org/wiki/DOT_(graph_description_language)
[11] http://www.qgis.org/en/site
[12] http://www.vim.org
[13] http://xmonad.org

Περιεχόμενα

1 Εισαγωγή **1**

2 Θεωρητικό υπόβαθρο **3**

2.1 Τηλεπισκόπηση 3

2.2 Πρόγραμμα Landsat 4

2.2.1 Δορυφόρος Landsat 7 5

2.3 Δορυφορικά δεδομένα 6

2.4 Δείκτες 7

2.4.1 Λευκαύγεια 7

2.4.2 Δείκτης βλάστησης NDVI 8

2.4.3 Δείκτης NBR 8

2.5 Χαρτογράφηση καμένων εκτάσεων 9

2.6 Παράλληλος προγραμματισμός 9

2.6.1 Απόδοση παράλληλων αλγορίθμων 10

2.6.2 Νόμος του Amdahl 11

2.6.3 OpenMP 12

2.6.4 MPI 13

2.6.5 Σύγκριση OpenMP και MPI 15

2.6.6 Υβριδικά συστήματα 15

3 Διαδικασία χαρτογράφησης καμένων εκτάσεων Εθνικού Αστεροσκοπείου Αθηνών 16

3.1 Προεπεξεργασία και μετα-επεξεργασία δεδομένων 17

3.2 Κύρια επεξεργασία δεδομένων 17

3.2.1 Περιγραφή υλοποίησης 18

3.2.2 Παράδειγμα εκτέλεσης 24

4 Παραλληλοποίηση διαδικασίας χαρτογράφησης καμένων εκτάσεων του Εθνικού Αστεροσκοπείου Αθηνών 27

4.1 Περιγραφή παράλληλης υλοποίησης 27

4.2 Παράλληλη αλυσίδα φίλτρων 33

5 Αποτελέσματα 39

5.1 Επιτάχυνση εφαρμογής 40

5.2 Χρόνος εκτέλεσης 45

5.3 Βελτιστοποίηση φίλτρου ομαδοποίησης καμένων pixel 51

5.4 Βελτιστοποίηση φίλτρου ένωσης ομάδων καμένων pixel 54

6 Σύνοψη 57

Βιβλιογραφία 59

Κατάλογος σχημάτων

2.1 Επιτάχυνση σε σχέση με τον αριθμό των επεξεργαστών, σύμφωνα με το νόμο του Amdahl 11

2.2 Σύστημα κοινής μνήμης 12

2.3 Χρήση MPI στη γλώσσα C++ 14

3.1 Περιγραφή σταδίων της εφαρμογής 18

3.2 Εφαρμογή χωρικού φίλτρου $3x3$ στο κελί με τιμή $a_{3,3}$ 21

3.3 Αναζήτηση σε γειτονικά pixel του σημειού με τιμή $a_{3,3}$ με σκοπό τη δημιουργία ομάδων καμένων περιοχών 22

3.4 Συγχώνευση ομάδων καμένων περιοχών που βρίσκονται σε κοντινή απόσταση . 23

3.5 Δεδομένα που συνδυάζονται για την παρουσίαση του αποτελέσματος 26

3.6 Παρουσίαση αποτελέσματος χρησιμοποιώντας ως επίπεδα την μπάντα 3 και την έξοδο της εφαρμογής 26

4.1 Διαμοιρασμός πίνακα $12xn$ σε σύστημα με 3 διεργασίες 28

4.2 Ροή δεδομένων μεταξύ των διεργασιών κατά τη διάρκεια εκτέλεσης ενός φίλτρου . 29

4.3 Αφηρημένη αρχιτεκτονική παράλληλης εφαρμογής 30

4.4 Διάγραμμα ροής φίλτρου ομαδοποίησης καμένων pixel 35

4.5 Ένωση (union) ομάδων με χρήση της κλάσης QuickFindUnionFind 37

4.6 Ιδανική ένωση (union) ομάδων με χρήση της κλάσης QuickFindUnionFind 37

4.7 Διάγραμμα ροής φίλτρου ένωσης ομάδων καμένων pixel 38

5.1 Παρουσίαση της συνολικής επιτάχυνσης της εφαρμογής χρησιμοποιώντας δεδομένα από τα έτη 2004-2005 41

5.2 Παρουσίαση της επιτάχυνσης των φίλτρων χρησιμοποιώντας δεδομένα από τις χρονολογίες 2004-2005 42

5.3 Παρουσίαση της συνολικής επιτάχυνσης της εφαρμογής χρησιμοποιώντας δεδομένα από τα έτη 2011-2012 43

5.4 Παρουσίαση της επιτάχυνσης των φίλτρων χρησιμοποιώντας δεδομένα από τις χρονολογίες 2011-2012 44

5.5 Παρουσίαση του συνολικού χρόνου εκτέλεσης της εφαρμογής χρησιμοποιώντας δεδομένα από τις χρονολογίες 2004-2005 46

5.6 Παρουσίαση του χρόνου εκτέλεσης των φίλτρων χρησιμοποιώντας δεδομένα από τις χρονολογίες 2004-2005 47

5.7 Παρουσίαση του συνολικού χρόνου εκτέλεσης της εφαρμογής χρησιμοποιώντας δεδομένα από τις χρονολογίες 2011-2012 48

5.8 Παρουσίαση του χρόνου εκτέλεσης των φίλτρων χρησιμοποιώντας δεδομένα από τις χρονολογίες 2011-2012 49

5.9 Ποσοστό του συνολικού χρόνου που καταναλώνει κάθε εργασία 51

5.10 Σύνολα που δίνονται ως είσοδος στη κλασική union-find δομή καθώς και στη zigzag Find union-find δομή 55

5.11 Έξοδος κλασικής union-find δομής 55

5.12 Έξοδος zigzag Find union-find δομής 56

Κατάλογος πινάκων

2.1 Συστήματα σχετικά με την Τηλεπισκόπηση 4

2.2 Χρονικό διάστημα λειτουργίας των Landsat δορυφόρων. Πηγή: [11] 5

2.3 Χωρική ανάλυση ενισχυμένου θεματικού χαρτογράφου του δορυφόρου Landsat 7. Πηγή: [12] 6

2.4 Χρησιμότητα φασματικών μπάντων. Πηγή: [13] 7

2.5 MPI βιβλιοθήκες 14

3.1 Είσοδος παραδείγματος για τα έτη 2004-2005 24

3.2 Έξοδος ανά φίλτρο 25

4.1 Python MPI βιβλιοθήκες 32

4.2 Οι 12 σημαντικότερες μέθοδοι ανά Python βιβλιοθήκη. Πηγή:[Lin] 33

5.1 Επιτάχυνση εργασιών σύμφωνα με τη συνάρτηση γραμμικής παλινδρόμησης με 120 διεργασίες με δεδομένα από τα έτη 2004-2005 και 132 διεργασίες με δεδομένα από τα έτη 2011-2012 45

5.2 Σειριακός και παράλληλος χρόνος εκτέλεσης σε δευτερόλεπτα για τα δεδομένα από τα έτη 2004-2005 50

5.3 Σειριακός και παράλληλος χρόνος εκτέλεσης σε δευτερόλεπτα για τα δεδομένα από τα έτη 2011-2012 50

Κεφάλαιο 1

Εισαγωγή

Most people do not listen with the intent to understand; they listen with the intent to reply.

- Stephen R. Covey

Χαρτογράφηση καμένων εκτάσεων ονομάζεται η διαδικασία αναγνώρισης και επισήμανσης καμένων δασικών περιοχών. Τα τελευταία χρόνια η σπουδαιότητα και η χρησιμότητα της χαρτογράφησης καμένων περιοχών έχει αυξηθεί, λόγω της μείωσης της χλωρίδας του πλανήτη και της αύξησης των πυρκαγιών ανά χώρα.

Διάφορα προγράμματα παρέχουν δεδομένα για χαρτογράφηση καμένων περιοχών. Τέτοια προγράμματα είναι το πρόγραμμα Landsat [Nasa] της Nasa, το σύστημα FIRMS [Nasa] και το νεότερο πρόγραμμα Copernicus [Esa] της ESA το οποίο αποσκοπεί στη συλλογή πληροφοριών, για τη Γη, χάρη στη δορυφορική αποστολή με όνομα Sentinels [Esa].

Στην Ελλάδα, το Εθνικό Αστεροσκοπείο Αθηνών έχει αναπτύξει και συμμετέχει σε προγράμματα σχετικά με χαρτογράφηση καμένων εκτάσεων και πρόληψη για αποφυγή πυρκαγιών. Τέτοια προγράμματα είναι το TELEIOS [Noa] όπου και συμμετείχε, το οποίο παρέχει ζωντανή παρακολούθηση πυρκαγιών (real-time fire monitoring) και χαρτογράφηση καμένων εκτάσεων, καθώς και το FireHub [Noa]. Για το σκοπό αυτό, στα προγράμματα TELEIOS και FireHub υλοποιείται μία σειρά διαδικασιών οι οποίες επεξεργάζονται δορυφορικές εικόνες, μεγάλου όγκου, τύπου Landsat και τύπου MODIS [Nasa].

Η ανάγκη για γρήγορη επεξεργασία πολλών δορυφορικών εικόνων είναι το θέμα της παρούσας πτυχιακής εργασίας. Η παρούσα πτυχιακή εργασία ασχολείται με την παραλληλοποίηση των αλγορίθμων και φίλτρων που εφαρμόζονται σε Landsat εικόνες από το Εθνικό Αστεροσκοπείο Αθηνών, με σκοπό τη μείωση του χρόνου της διαδικασίας χαρτογράφησης καμένων εκτάσεων. Στο Κεφάλαιο 3 παρουσιάζεται η σειριακή εφαρμογή από το Εθνικό Αστεροσκοπείο Αθηνών, ενώ στο Κεφάλαιο 4 παρουσιάζεται η παράλληλη υλοποίηση.

Για την παραλληλοποίηση της εφαρμογής χρησιμοποιείται η αρχιτεκτονική περάσματος μηνυμάτων (Message Passing Interface) [8] και η γλώσσα προγραμματισμού Python.

Κεφάλαιο 2

Θεωρητικό υπόβαθρο

If Java had true garbage collection, most programs would delete themselves upon execution.

- Robert Sewell

There are only two kinds of languages: the ones people complain about and the ones nobody uses.

- Bjarne Stroustrup

2.1 Τηλεπισκόπηση

Τηλεπισκόπηση ή τηλεανίχνευση (Remote Sensing) [Schowengerdt] ονομάζεται η διαδικασία απόκτησης πληροφορίας για ένα αντικείμενο από απόσταση, χωρίς να υπάρχει φυσική επαφή με αυτό. Η τηλεπισκόπηση είναι επιστημονικός κλάδος της Γεωγραφίας. Η τηλεπισκόπηση είναι συνδυασμός επιστημών όπως τα Μαθηματικά, η Πληροφορική και η Χημεία. Χρησιμοποιείται για την παρακολούθηση αντικειμένων της γης καθώς και όλων των ουράνιων σωμάτων. Τα αντικείμενα αυτά μπορεί να βρίσκονται είτε στην επιφάνεια, είτε στην ατμόσφαιρα ή στους ωκεανούς του ουράνιου σώματος. Η παρακολούθηση, γίνεται μελετώντας την αλληλεπίδραση των αντικειμένων με την ηλεκτρομαγνητική ακτινοβολία η οποία τις περισσότερες φορές εκπέμπεται από τον ήλιο. Ο τρόπος συλλογής δεδομένων γίνεται μέσω δορυφόρων που αποτελούνται, συνήθως, από ένα σύνολο παθητικών αισθητήρων. Κάθε υλικό-αντικείμενο αντανακλά την προσπίπτουσα ηλεκτρομαγνητική ακτινοβολία με ξεχωριστό τρόπο, σε διαφορετικά μήκη κύματος, με αποτέλεσμα να ανιχνεύεται το είδος της επιφάνειας που απεικονίζεται στα δεδομένα (νερό, δάσος, χιόνι, καμένες εκτάσεις).

Υπάρχουν αρκετά συστήματα που ασχολούνται με την τηλεπισκόπηση. Παραδείγματα τέτοιων συστημάτων φαίνονται στον Πίνακα 2.1.

Όνομα Συστήματος
Landsat 1, 2, 3, 4, 5, 6, 7, 8
AVHRR/3
MODIS
QuickBird
GOES
Ikonos
SPOT
RADARSAT
ASTER

Πίνακας 2.1: Συστήματα σχετικά με την Τηλεπισκόπηση

Τα δεδομένα που λαμβάνονται, αποθηκεύονται σε μορφή δισδιάστατου πίνακα. Η ποιότητα τους χαρακτηρίζεται από τα ακόλουθα χαρακτηριστικά [Πλένιου]:

Χωρική ανάλυση: ο αριθμός των pixel της εικόνας

Φασματική ανάλυση: το πλήθος των φασματικών μπάντων, στις οποίες υπάρχει η δυνατότητα να ληφθούν δεδομένα

Ραδιομετρική ανάλυση: το πλήθος των διαφορετικών εντάσεων ακτινοβολίας που μπορούν να ληφθούν

Μερικά από τα θετικά στοιχεία της τηλεπισκόπησης είναι πως i) μπορούν να συλλεχθούν δεδομένα για δύσβατες ή μακρινές περιοχές όπως η Ανταρκτική, ii) μπορούν να ληφθούν δεδομένα σε διαφορετικές φασματικές περιοχές, iii) το κόστος είναι σχετικά μικρό .

Μερικά από τα αρνητικά στοιχεία είναι i) η προσθήκη θορύβου λόγω καιρικών συνθηκών (σύννεφα), ii) η ανάγκη για βαθμονόμηση της έντασης της ηλεκτρομαγνητικής ακτινοβολίας που λαμβάνεται από τους αισθητήρες των δορυφόρων .

2.2 Πρόγραμμα Landsat

Το πρόγραμμα Landsat [Nasa] της Εθνικής Υπηρεσίας Αεροναυτικής και Διαστήματος (National Aeronautics and Space Administration - NASA) είναι το πιο μακρόχρονο πρόγραμμα συλλογής δορυφορικών δεδομένων-εικόνων που αφορούν τη Γη. Ξεκίνησε το 1963 ως Earth Resources Technology

Satellite. Ο πρώτος δορυφόρος με όνομα Earth Resources Technologoy Satellite εκτοξεύθηκε στις 23 Ιουλίου 1972. Αργότερα έγινε μετανομασία του δορυφόρου αυτού σε Landsat 1. Έκτοτε έχουν εκτοξευθεί άλλοι 7 δορυφόροι για τον ίδιο σκοπό, από τους οποίους ο τελευταίος με όνομα Landsat 8 εκτοξεύθηκε στις 11 Φεβρουαρίου 2013.
Ο Πίνακας 2.2 περιγράφει το χρονικό διάστημα λειτουργίας του κάθε Landsat δορυφόρου.

Όνομα	Εκτόξευση	Τέλος λειτουργίας
Landsat 1	23 Ιουλίου 1973	6 Ιανουαρίου 1978
Landsat 2	22 Ιανουαρίου 1975	25 Φεβρουαρίου 1982
Landsat 3	5 Μαρτίου 1978	31 Μαρτίου 1983
Landsat 4	16 Ιουλίου 1982	14 Δεκεμβρίου 1993
Landsat 5	1 Μαρτίου 1984	5 Ιουνίου 2013
Landsat 6	5 Οκτωβρίου 1993	5 Οκτωβρίου 1993
Landsat 7	15 Απριλίου 1999	Ενεργός έως και σήμερα
Landsat 8	11 Φεβρουαρίου 2013	Ενεργός έως και σήμερα

Πίνακας 2.2: Χρονικό διάστημα λειτουργίας των Landsat δορυφόρων. Πηγή: [11]

Τα δεδομένα που συλλέγονται είναι από διάφορες φασματικές μπάντες (spectral bands) και αποθηκεύονται έτσι ώστε να χρησιμοποιηθούν σε εφαρμογές γεωργικές-αγροτικές, οικολογικές, παρατήρησης, περιφεριακού σχεδιασμού, δασοκομίας, γεωλογίας, χαρτογράφησης ή εκπαιδευτικές. Ο συνδυασμός διάφορων μπάντων ικανοποιεί τις ανάγκες των παραπάνω επιστημονικών εφαρμογών. Αναφορικά, ο δορυφόρος Landsat 8 διαθέτει δεδομένα σε 11 διαφορετικές φασματικές μπάντες.

2.2.1 Δορυφόρος Landsat 7

Ο δορυφόρος Landsat 7 είναι ένας από τους πιο σημαντικούς δορυφόρους του προγράμματος Landsat. Είχε σχεδιασθεί να παρέχει υψηλής ακρίβειας και ποιότητας δεδομένα με τη βοήθεια του ενισχυμένου θεματικού χαρτογράφου (Enchanced Thematic Mapper) [12]. Τα δορυφορικά δεδομένα που χρησιμοποιούνται για την παρούσα πτυχιακή εργασία προέρχονται από τους Landsat 4, Landsat 5 και Landsat 7.

Ο Landsat 7 αποτελείται από ένα σύνολο αισθητήρων, χρήσιμων για τη λήψη δορυφορικών εικόνων στις διάφορες φασματικές μπάντες. Επίσης διαθέτει ειδικό φίλτρο εξάλειψης θορύβων από σύννεφα. Ακόμα έχει τη δυνατότητα λήψης και αποστολής έως και 532 εικόνων ημερησίως.

Ο Landsat 7 χρησιμοποιώντας τον ενισχυμένο θεματικό χαρτογράφο λαμβάνει εικόνες μεταξύ

των μπάντων 1 έως 8. Η χωρική ανάλυση για τις μπάντες από 1 έως 7 είναι 30 μέτρα ενώ για τη μπάντα 8 είναι 15 μέτρα. Ο Πίνακας 2.3 περιγράφει τη χωρική ανάλυση σε κάθε μπάντα για τον ενισχυμένο θεματικό χαρτογράφο.

Μπάντα	Μήκος κύματος	Χωρική ανάλυση
1	$0.45\mu m - 0.52\mu m$	$30m$
2	$0.52\mu m - 0.60\mu m$	$30m$
3	$0.63\mu m - 0.69\mu m$	$30m$
4	$0.77\mu m - 0.90\mu m$	$30m$
5	$1.55\mu m - 1.75\mu m$	$30m$
6	$10.40\mu m - 12.50\mu m$	$30m$
7	$2.09\mu m - 2.35\mu m$	$30m$
8	$0.52\mu m - 0.90\mu m$	$15m$

Πίνακας 2.3: Χωρική ανάλυση ενισχυμένου θεματικού χαρτογράφου του δορυφόρου Landsat 7. Πηγή: [12]

2.3 Δορυφορικά δεδομένα

Τα δορυφορικά δεδομένα που αποκτούνται σε κάθε φασματική μπάντα χρησιμοποιούνται για τη διεξαγωγή πληροφοριών, χρήσιμων για αρκετές επιστήμες. Ο Πίνακας 2.4 περιγράφει τη χρησιμότητα των δεδομένων ανά φασματική μπάντα.

Band	Wavelength	Useful for mapping
1 - blue	$0.45\mu m - 0.52\mu m$	Bathymetric mapping, distinguishing soil from vegetation and deciduous from coniferous vegetation
2 - green	$0.52\mu m - 0.60\mu m$	Emphasizes peak vegetation, which is useful for assessing plant vigor
3 - red	$0.63\mu m - 0.69\mu m$	Discriminates vegetation slopes
4 - Near Infrared	$0.77\mu m - 0.90\mu m$	Emphasizes biomass content and shorelines
5 - Short-wave Infrared	$1.55\mu m - 1.75\mu m$	Discriminates moisture content of soil and vegetation; penetrates thin clouds
6 - Thermal Infrared	$10.40\mu m - 12.50\mu m$	Thermal mapping and estimated soil moisture
7 - Short-wave Infrared	$2.09\mu m - 2.35\mu m$	Hydrothermally altered rocks associated with mineral deposits
8 - Panchromatic	$0.52\mu m - 0.90\mu m$	15 meter resolution, sharper image definition

Πίνακας 2.4: Χρησιμότητα φασματικών μπάντων. Πηγή: [13]

Τα δορυφορικά δεδομένα που χρησιμοποιούνται για την παρούσα πτυχιακή εργασία είναι κυρίως από την περιοχή της Πελοποννήσου και αντιπροσωπεύουν το χρονικό διάστημα από το 2004 έως το 2012.Τα δεδομένα αυτά είναι από τις φασματικές μπάντες 3, 4 και 7 των δορυφόρων Landsat 5 και 7.

2.4 Δείκτες

Τα δορυφορικά δεδομένα που λαμβάνονται από τις διάφορες μπάντες, επεξεργάζονται για να δημιουργηθεί γνώση σχετικά με τις περιοχές οι οποίες απεικονίζονται στα δεδομένα. Για να παραχθούν σωστά αποτελέσματα, χρησιμοποιούνται διάφοροι συντελεστές/δείκτες, οι οποίοι προέρχονται κυρίως από θετικές επιστήμες όπως η Φυσική, η Χημεία και η Γεωλογία. Για την αναγνώριση καμένων περιοχών που ίσως απεικονίζονται σε δορυφορικά δεδομένα χρησιμοποιούνται οι δείκτες που περιγράφονται σε αυτό το κεφάλαιο. Οι δείκτες αυτοί συνδυάζονται με σκοπό να δημιουργήσουν τον αλγόριθμο BSM_NOA [Kontoes, Poilvé, Florsch, *et al.*], ο οποίος στην πραγματικότητα είναι ένα ταξινομητής (classifier). Ο ταξινομητής αυτός βοηθάει στη λήψη της απόφασης για ένα οποιοδήποτε σημείο (pixel), από τα δορυφορικά δεδομένα, για το αν είναι καμένο ή όχι.

2.4.1 Λευκαύγεια

Η λευκαύγεια (Albedo) [15] ή συντελεστής αντανάκλασης είναι ο δείκτης που μετράει την αντανάκλαση ηλεκτρομαγνητικής ακτινοβολίας ενός σώματος. Υπολογίζεται ως ο λόγος της ανακλώμενης ηλεκτρομαγνητικής ακτινοβολίας προς την προσπίπτουσα ηλεκτρομαγνητική ακτινο-

βολία.

Η χρησιμότητα της λευκαύγειας βρίσκεται στο γεγονός ότι διαφορετικά υλικά έχουν και διαφορετική λευκαύγεια. Για παράδειγμα η λευκαύγεια του χιονού είναι περίπου 0.9 ή 90% ενώ η μέση λευκαύγεια της γης κυμαίνεται στο 37-39%. Έτσι, ξέροντας από πριν τη λευκαύγεια κάθε υλικού, εύκολα μπορεί να αναγνωρισθεί το ίδιο το υλικό στα δορυφορικά δεδομένα.

2.4.2 Δείκτης βλάστησης NDVI

Ο δείκτης βλάστησης κανονικοποιημένης διαφοράς (Normalized Difference Vegetation Index ή NDVI) [16] υπολογίζεται από την κόκκινη και εγγύς υπέρυθρη ηλεκτρομαγνητική ακτινοβολία που αντανακλάται από τη βλάστηση. Η υγιής βλάστηση απορροφά περισσότερη από την ορατή ακτινοβολία που τη χτυπά και αντανακλά μεγάλο μέρος από την εγγύς υπέρυθρη ακτινοβολία. Αντίθετα, η μη υγιής ή αραιή βλάστηση αντανακλά περισσότερη ορατή ακτινοβολία και λιγότερη υπέρυθρη ακτινοβολία.

Η παραπάνω περιγραφή του NDVI μπορεί να εκφραστεί και από την ακόλουθη εξίσωση σε συνάρτηση με τις τιμές NIR[1] και VIS[2]:

$$NDVI = \frac{NIR - VIS}{NIR + VIS} \quad \text{, με} \quad -1 \leq NDVI \leq 1 \tag{2.1}$$

Όταν το αποτέλεσμα της (2.1) ισούται με -1 ή τείνει προς το -1 σημαίνει πως δεν υπάρχει καθόλου βλάστηση, ενώ ισχύει το αντίθετο όταν ισούται ή τείνει με 1.

2.4.3 Δείκτης NBR

Ο δείκτης NBR (Normalized Burn Ratio) [17] ορίσθηκε με σκοπό να επισημάνει τις καμένες περιοχές. Η συνάρτηση που υπολογίζει το δείκτη NBR είναι παρόμοια με τη συνάρτηση του δείκτη NDVI, απλά σε αυτήν την περίπτωση χρησιμοποιείται η τιμή MIR[3] αντί της τιμής VIS:

$$NBR = \frac{NIR - MIR}{NIR + MIR} \quad \text{, με} \quad -1 \leq NBR \leq 1 \tag{2.2}$$

[1] τιμή εγγύς υπέρυθρης ακτινοβολίας
[2] τιμή ορατής ακτινοβολίας
[3] τιμή μέσης υπέρυθρης ακτινοβολίας

2.5 Χαρτογράφηση καμένων εκτάσεων

Η χαρτογράφηση καμένων εκτάσεων (Burn Scar Mapping - BSM) έχει ως στόχο την αναγνώριση και σήμανση καμένων περιοχών. Η χαρτογράφηση είναι δύσκολο να γίνει χωρίς τη χρήση δεδομένων που έχουν ληφθεί από τον αέρα. Αρχικά, τα δεδομένα λαμβάνονταν είτε από αεροπλάνο είτε από κάποιον ειδικό δορυφόρο. Πλέον, τα δεδομένα λαμβάνονται κυρίως από ειδικούς δορυφόρους τηλεπισκόπησης, όπως αυτοί του προγράμματος Landsat.

Η χαρτογράφηση καμένων εκτάσεων, πλέον είναι μία αυτόματη διαδικασία όπου γίνεται μέσω ειδικών λογισμικών που έχουν αναπτυχθεί από διάφορους οργανισμούς. Για παράδειγμα, η δημοσίευση [18] παρουσιάζει έναν αλγόριθμο χρήσιμο για χαρτογράφηση καμένων εκτάσεων.

Στην Ελλάδα, το Εθνικό Αστεροσκοπείο Αθηνών έχει αναπτύξει λογισμικό για επεξεργασία δορυφορικών εικόνων τύπου Landsat TM 4, 5, 7 με σκοπό την αυτόματη αναγνώριση καμένων περιοχών. Το λογισμικό αυτό χρησιμοποιείται με επιτυχία σε δορυφικά δεδομένα που συλλέγονται από το 1984 και αφορούν μόνο την Ελλάδα. Η παρούσα πτυχιακή εργασία βασίζεται και είναι μία επέκταση ενός μέρους αυτού του λογισμικού.

2.6 Παράλληλος προγραμματισμός

Σύμφωνα με τον [Pacheco], παράλληλος προγραμματισμός είναι η δημιουργία προγραμμάτων ή η τροποποίηση σειριακών προγραμμάτων για την εκμετάλλευση παράλληλων συστημάτων. Στις περισσότερες περιπτώσεις οι εφαρμογές που προορίζονται για μονοπύρηνα συστήματα δεν μπορούν να χρησιμοποιήσουν όλη την ισχύ των πολυπύρηνων συστημάτων. Για παράδειγμα η ανάγκη να εκτελεσθεί πολλές φορές ταυτόχρονα μία εφαρμογή, σε διαφορετικούς επεξεργαστές, δεν είναι πάντα αυτό που είναι αναγκαίο. Συνήθως, χρειάζεται μία εργασία να κατανεμηθεί στους διαθέσιμους υπολογιστές/επεξεργαστές του συστήματος με κυριότερο σκοπό τη γρηγορότερη εκτέλεση της. Παραδείγματα τέτοιων εφαρμογών μπορούν να βρεθούν στον τομέα της ψηφιακής επεξεργασίας εικόνας. Ο παράλληλος προγραμματισμός απαιτείται σε περιπτώσεις όπου το πρόγραμμα χρειάζεται να επεξεργαστεί ή να αποθηκεύσει περισσότερα δεδομένα από ότι μπορεί να φιλοξενήσει η μνήμη ενός υπολογιστή. Έτσι, για να λυθεί το πρόβλημα, τα δεδομένα κατανέμονται στους υπολογιστές ενός υπολογιστικού συστήματος.

Ο προγραμματισμός ενός παράλληλου συστήματος εξαρτάται από την αρχιτεκτονική του. Αν οι κόμβοι του συστήματος έχουν κοινή μνήμη, τότε μπορεί να χρησιμοποιηθεί η διασύνδεση OpenMP [20]. Αντίθετα, αν οι κόμβοι του συστήματος έχουν ιδιωτική μνήμη, τότε μπορεί να χρησιμοποιηθεί η διασύνδεση MPI (Message Passing Interface).

Γενικά, κάθε είδους τεχνική παράλληλου προγραμματισμού πρέπει να χρησιμοποιείται μόνο σε στιγμές αληθινής ανάγκης. Τέτοιες στιγμές είναι όταν έχουν εξαντληθεί όλα τα ενδεχόμενα βελτιστοποίησης του αντίστοιχου σειριακού προγράμματος ή όταν το σειριακό πρόγραμμα έχει ανάγκη από περισσότερους πόρους, σε σχέση με αυτούς που προσφέρονται από ένα σύστημα ενός

υπολογιστή (μνήμη, υπολογιστική ισχύς). Σε αντίθετη περίπτωση, η τεχνική του παράλληλου προγραμματισμού είναι περιττή και δυσκολεύει το έργο του προγραμματιστή.

2.6.1 Απόδοση παράλληλων αλγορίθμων

Όπως αναφέρθηκε προηγουμένως, κύριος στόχος των παράλληλων προγραμμάτων είναι η μείωση του χρόνου εκτέλεσης μίας λειτουργίας. Επομένως, πρέπει να ορισθούν κάποια κριτήρια με τα οποία γίνεται ο έλεγχος της ποιότητας των παράλληλων προγραμμάτων/αλγορίθμων. Τέτοια κριτήρια είναι η επιτάχυνση, ο χρόνος παράλληλης εκτέλεσης, η αποδοτικότητα και η επεκτασιμότητα [Παπαδάκης and Διαμαντάρας].

Χρόνος παράλληλης εκτέλεσης (t_p): Ο χρόνος παράλληλης εκτέλεσης στην ιδανική περίπτωση υπολογίζεται ως $t_p = \frac{t_s}{p}$, όπου t_s είναι ο σειριακός χρόνος εκτέλεσης του προγράμματος και p το πλήθος των υπολογιστών. Αυτό σημαίνει ότι ο παράλληλος αλγόριθμος δεν προσθέτει επιπλέον χρόνο προκειμένου να λειτουργήσει σωστά. Στην πραγματικότητα, πάντοτε προστίθεται ελάχιστος χρόνος, για παράδειγμα, για επικοινωνία ή συγχρονισμό των υπολογιστών του συστήματος.

Επιτάχυνση (speedup): Θεωρώντας ένα σύστημα με $p > 1$ υπολογιστές, τότε η επιτάχυνσή του υπολογίζεται από τον τύπο $S(p) = \frac{t_s}{t_p}$. Στην ιδανική περίπτωση ισχύει $S(p) = p$. Όμως, στην πραγματικότητα η επιτάχυνση ποτέ δεν αγγίζει τη μέγιστη τιμή της. Σε περιπτώσεις που η επιτάχυνση είναι μεγαλύτερη από p σημαίνει ότι ο σειριακός αλγόριθμος δεν ήταν ο βέλτιστος.

Αποδοτικότητα (efficiency): Για τον υπολογισμό της επιτάχυνσης χρησιμοποιείται μόνο ο χρόνος σειριακής και παράλληλης εκτέλεσης του προγράμματος, αγνοώντας τον αριθμό των επεξεργαστών. Αυτό σημαίνει πως όσο αυξάνονται οι επεξεργαστές, θεωρητικά, αυξάνεται και η επιτάχυνση, με αποδοτικό τρόπο. Στην πραγματικότητα αυτό δεν ισχύει. Έτσι, χρειάζεται να βρεθεί ένα κριτήριο το οποίο περιγράφει την ποιότητα της παραλληλοποίησης. Αυτό το κριτήριο ονομάζεται αποδοτικότητα (E) και υπολογίζεται με τον τύπο (2.3).

$$E = \frac{s}{p} \quad \text{, με } E \leq 1 \text{ αφού } s \leq p \tag{2.3}$$

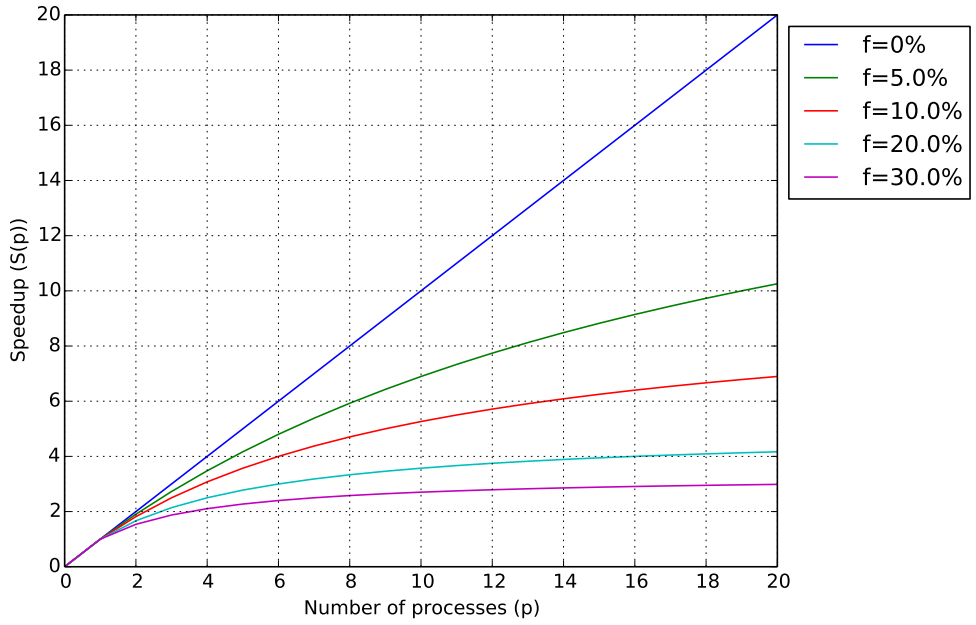
Επεκτασιμότητα (scalability): Η επεκτασιμότητα αναλύει τις δυνατότητες του παράλληλου προγράμματος. Με την επεκτασιμότητα καταγράφεται τι θα συμβεί στην επιτάχυνση αν αυξηθούν οι υπολογιστές ενός παράλληλου συστήματος. Έτσι, ο προγραμματιστής καταλαβαίνει τα όρια του παράλληλου προγράμματος και αν αξίζει να αυξηθεί ο αριθμός των υπολογιστών.

2.6.2 Νόμος του Amdahl

Σύμφωνα με τους [Παπαδάκης and Διαμαντάρας] η θεωρία ότι η επιτάχυνση είναι γραμμική σε σχέση με τον αριθμό των επεξεργαστών απέχει από την πραγματικότητα, όπως αναφέρθηκε προηγουμένως. Τη δεκαετία του 1960 ο Gene Amdahl παρατήρησε ότι ακόμα και αν αγνοηθεί ο χρόνος για επικοινωνία και συγχρονισμό, δεν θα επιτευχθεί ποτέ η μέγιστη θεωρητική επιτάχυνση. Αυτό συμβαίνει διότι σε κάθε πρόγραμμα υπάρχει τουλάχιστον ένα κομμάτι, το οποίο δεν γίνεται να παραλληλοποιηθεί. Αν το ποσοστό του προγράμματος που δεν παραλληλοποιείται είναι f τότε ο χρόνος εκτέλεσης του προγράμματος υπολογίζεται από τον τύπο $t_p = f \cdot t_s + (1-f) \cdot \frac{t_s}{p}$. Επομένως, η επιτάχυνση υπολογίζεται από τον τύπο $S(p) = \frac{t_s}{t_p} = \frac{1}{f + \frac{1-f}{p}}$.

Ο παραπάνω τύπος δείχνει πως η μέγιστη επιτάχυνση ισούται με $\frac{1}{f}$ για $p \to +\infty$. Τελικά, σύμφωνα με το νόμο του Amdahl ισχύει ότι $S(p) \leq \frac{1}{f + \frac{1-f}{p}}$.

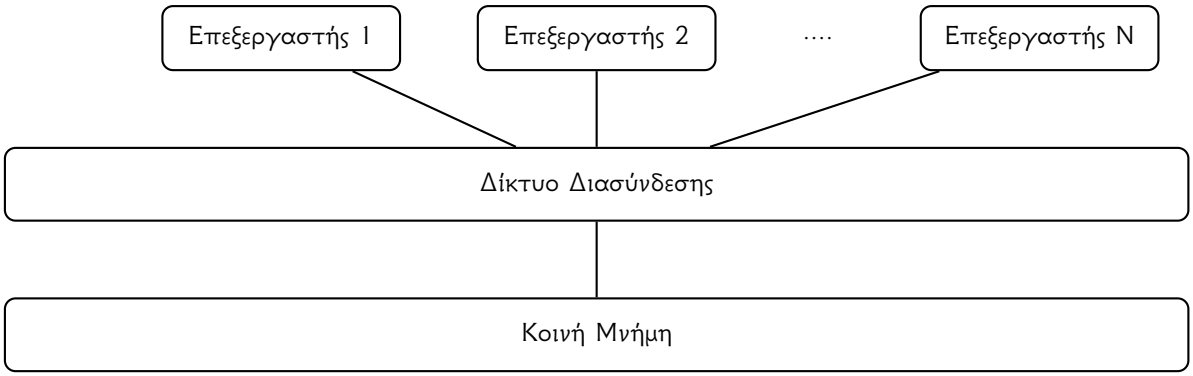
Το Σχήμα 2.1 παρουσιάζει την επιτάχυνση σε σχέση με τον αριθμό των επεξεργαστών, σύμφωνα με το νόμο του Amdahl.



Σχήμα 2.1: Επιτάχυνση σε σχέση με τον αριθμό των επεξεργαστών, σύμφωνα με το νόμο του Amdahl

2.6.3 OpenMP

Η OpenMP είναι διασύνδεση για παράλληλο προγραμματισμό κοινής μνήμης (shared memory). Η OpenMP είναι σχεδιασμένη για συστήματα στα οποία κάθε υπολογιστής έχει πρόσβαση στην κύρια μνήμη του συστήματος. Κάθε υπολογιστής/κόμβος θεωρείται ένα νήμα (thread) του συστήματος. Κατά τον προγραμματισμό με OpenMP ιδιαίτερη προσοχή πρέπει να δοθεί στο ότι όλοι οι υπολογιστές έχουν πρόσβαση στα κοινά δεδομένα και επομένως μπορούν να δημιουργηθούν προβλήματα κατά την εγγραφή ή την προσπέλασή της. Επομένως, πρέπει να υπάρχει συγχρονισμός κατά την εγγραφή δεδομένων στη μνήμη από τα διάφορα νήματα. Στην πραγματικότητα, η διασύνδεση OpenMP χρησιμοποιείται κυρίως για τον προγραμματισμό υπολογιστών που διαθέτουν πολλούς επεξεργαστές. Σε αυτούς τους επεξεργαστές μοιράζονται οι εργασίες που δρομολογούνται από τη διασύνδεση. Το Σχήμα 2.2 παρουσιάζει την αρχιτεκτονική του συστήματος κοινής μνήμης για το οποίο μπορεί να χρησιμοποιηθεί η διασύνδεση OpenMP κατά τον προγραμματισμό του.



Σχήμα 2.2: Σύστημα κοινής μνήμης

Η διασύνδεση OpenMP παρέχεται στις γλώσσες C, C++, `Fortran`. Στις γλώσσες C και C++ η διασύνδεση υλοποιείται μέσω pragmas, τα οποία στην πραγματικότητα είναι οδηγίες προς το μεταγλωττιστή. Το Παράδειγμα 2.1 παρουσιάζει τη χρήση της διασύνδεσης OpenMP στη γλώσσα C++ με 5 threads.

```
1  #include <iostream>
2  #include <omp.h>
3
4  using std::cout;
5  using std::endl;
6
7  int main(int argc, char *argv[])
8  {
9    int num_of_threads;
10   int id;
11
12   omp_set_num_threads(5);
13
```

```
14    #pragma omp parallel private(id) shared(num_of_threads)
15    {
16      id = omp_get_thread_num();
17      num_of_threads = omp_get_num_threads();
18
19      #pragma omp critical
20      {
21        cout << "Thread " << id << " of " << num_of_threads << endl;
22      }
23    }
24  }
```

Παράδειγμα 2.1: Χρήση OpenMP στη γλώσσα C++

2.6.4 MPI

Η MPI (Message Passing Interface) είναι διασύνδεση για παράλληλο προγραμματισμό κατανεμημένης μνήμης (distributed memory). Η MPI είναι σχεδιασμένη για συστήματα στα οποία κάθε υπολογιστής έχει τη δική του προσωπική μνήμη και δεν έχει άμεση πρόσβαση σε μνήμη άλλου υπολογιστή. Σε αντίθεση με την OpenMP, η MPI λειτουργεί με διεργασίες (processes) και όχι με νήματα. Επίσης, κύριος στόχος της MPI είναι ο προγραμματισμός συστοιχιών υπολογιστών (clusters) ή υπολογιστών που βρίσκονται σε ένα κοινό δίκτυο. Παράδειγμα τέτοιων υπολογιστών είναι οι υπολογιστές στα εργαστήρια πανεπιστημίων. Η επικοινωνία μεταξύ των διεργασιών γίνεται μέσω ενός δικτύου (communicator) το οποίο διαχειρίζεται η MPI. Ο χρήστης χρειάζεται να υλοποιήσει την ανταλλαγή δεδομένων, όπου και αν χρειάζεται, μεταξύ των υπολογιστών/διεργασιών με χρήση των συναρτήσεων περάσματος μηνυμάτων που παρέχονται από την MPI.

Κάθε διεργασία που δημιουργείται έχει ένα ξεχωριστό αριθμό (rank) ο οποιός είναι χρήσιμος για το χρήστη. Η MPI ακολουθεί τη φιλοσοφία των συστημάτων Ένα Πρόγραμμα, Πολλά Δεδομένα (Single Program Multiple Data - SPMD) [Wilkinson and Allen]. Σύμφωνα με αυτήν τη φιλοσοφία ο χρήστης χρειάζεται να υλοποιήσει ένα κοινό πρόγραμμα, στο οποίο θα καθορίζει με δομές if-then-else τι πρέπει να εκτελεσθεί από κάθε διεργασία. Η διαχώριση των εργασιών μεταξύ των διεργασιών γίνεται μέσω του ξεχωριστού αριθμού που διαθέτουν.

Τα συστήματα που χρησιμοποιούν MPI, τις περισσότερες φορές λειτουργούν σύμφωνα με την τεχνολογία master - slave/worker. Αυτό σημαίνει πως ο χρήστης ορίζει μία κύρια διεργασία (master), συνήθως η διεργασία με αριθμό 0, η οποία μοιράζει και λαμβάνει δεδομένα από όλες τις υπόλοιπες διεργασίες (slaves/workers). Έτσι, η σωστή κατανομή των δεδομένων στις διάφορες διεργασίες είναι ευθύνη του χρήστη.

Η διασύνδεση MPI παρέχεται σε αρκετές γλώσσες μέσω βιβλιοθηκών. Ο Πίνακας 2.5 περιέχει ορισμένα παραδείγματα βιβλιοθηκών στις αντίστοιχες γλώσσες προγραμματισμού.

	Βιβλιοθήκη	Γλώσσα προγραμματισμού
1	Open MPI	C, C++, Fortran
2	MPI for Python (mpi4py)	Python
3	pypar	Python
4	MYMPI	Python
5	pyMPI	Python
6	Scientific.MPI	Python
7	boostmpi	Python
8	haskell-mpi	Haskell
9	OCamlMPI	OCaml

Πίνακας 2.5: MPI βιβλιοθήκες

Το Παράδειγμα 2.3 παρουσιάζει τη χρήση της διασύνδεσης MPI στη γλώσσα προγραμματισμού C++.

```
#include <iostream>
#include <mpi.h>

using std::cout;
using std::endl;

int main(int argc, char *argv[])
{
  // Initialize MPI.
  MPI::Init(argc, argv);

  // Get rank number.
  int rank = MPI::COMM_WORLD.Get_rank();

  // Get number of processes.
  int size = MPI::COMM_WORLD.Get_size();

  cout << "Process: " << rank << " of: " << size << endl;

  // Finalize MPI.
  MPI::Finalize();
}
```

Σχήμα 2.3: Χρήση MPI στη γλώσσα C++

2.6.5 Σύγκριση OpenMP και MPI

Ο προγραμματισμός με χρήση της διασύνδεσης MPI δεν είναι εύκολος όσο με τη διασύνδεση OpenMP για τους εξής λόγους [23]:

- Υλοποίηση της επικοινωνίας από τον προγραμματιστή
- Επιρρέπεια σε σφάλματα.
- Δυσκολία στην εύρεση λαθών του προγράμματος

Ωστόσο, λόγω του ότι κάθε διεργασία έχει τη δική της μνήμη δε χρειάζεται συνεχής συγχρονισμός, κάτι που είναι αρκετά χρήσιμο για τον προγραμματιστή.

Η διασύνδεση που χρησιμοποιείται για την παρούσα πτυχιακή εργασία είναι η MPI, στη γλώσσα προγραμματισμού Python μέσω της βιβλιοθήκης mpi4py.

2.6.6 Υβριδικά συστήματα

Σε περιπτώσεις που χρειάζεται η μέγιστη απόδοση ενός συστήματος από πολυπύρηνους κόμβους, κάθε κόμβος προγραμματίζεται με χρήση προγραμματισμού κοινής μνήμης και εφαρμόζεται προγραμματισμός κατανεμημένης μνήμης μεταξύ των κόμβων. Αυτή η τεχνική πρέπει να χρησιμοποιείται μόνο σε ακραίες περιπτώσεις αφού η συγγραφή τέτοιου λογισμικού είναι επίπονη διαδικασία.

Κεφάλαιο 3

Διαδικασία χαρτογράφησης καμένων εκτάσεων Εθνικού Αστεροσκοπείου Αθηνών

Namespaces are one honking great idea -let's do more of those!

- Tim Peters

I invented the term 'Object-Oriented', and I can tell you I did not have C++ in mind.

- Alan Kay

Η διαδικασία χαρτογράφησης καμένων εκτάσεων του Εθνικού Αστεροσκοπείου Αθηνών περιγράφεται στη δημοσίευση [24]. Η διαδικασία αυτή, χωρίζεται σε τρία κύρια στάδια:

1. Προεπεξεργασία δορυφορικών δεδομένων
2. Κύρια επεξεργασία δορυφορικών δεδομένων
3. Μετα-επεξεργασία δορυφορικών δεδομένων

Το συνολικό αποτέλεσμα από αυτό το λογισμικό χρησιμοποιείται και μπορεί να βρεθεί στο σύνδεσμο http://ocean.space.noa.gr/diachronic_bsm, όπου ο χρήστης μπορεί να παρατηρήσει τις καμένες εκτάσεις της Ελλάδος από το 1984.

3.1 Προεπεξεργασία και μετα-επεξεργασία δεδομένων

Σύμφωνα με τους [Kontoes, Papoutsis, Herekakis, *et al.*] κατά την προεπεξεργασία δεδομένων i) αναγνωρίζονται και αποθηκεύονται χρήσιμα δεδομένα όπως δορυφορικά δεδομένα, ii) γίνεται γεω-αναφορά (georeferencing) των δορυφορικών εικόνων με χρήση του λογισμικού NASA AROP [25] και iii) παράγονται οι μάσκες νερού και συννέφων.

Στη συνέχεια η μετα-επεξεργασία δεδομένων i) ελέγχει αν τα αποτελέσματα από την προ-επεξεργασία και την κύρια επεξεργασία δεδομένων είναι πλήρως σωστά και ii) συλλέγει όλα τα στρώματα (layers) δεδομένων και εικόνων που χρειάζονται με σκοπό να δημιουργηθεί ένα αποτέλεσμα φιλικό προς τον άνθρωπο. Κατά τη διάρκεια της μετα-επεξεργασίας, το τελικό αποτέλεσμα εμπλουτίζεται με εξωτερική γνώση, όπως δεδομένα από το πρόγραμμα CORINE Land Cover [26], με αποτέλεσμα την αφαίρεση λαθών που μπορεί να προέκυψαν από την κύρια επεξεργασία των δεδομένων.

3.2 Κύρια επεξεργασία δεδομένων

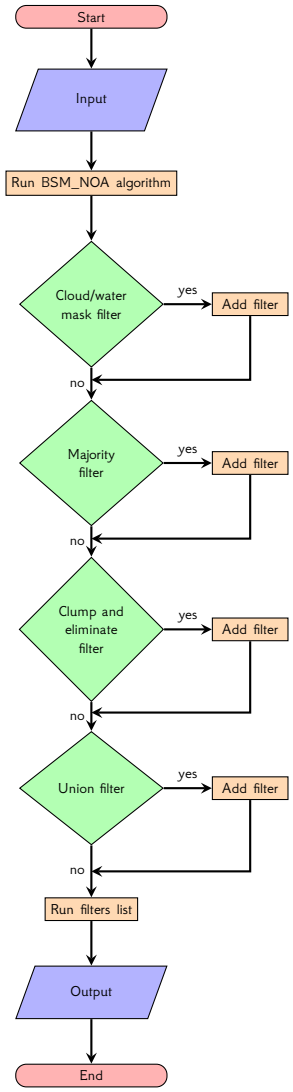
Η κύρια επεξεργασία δεδομένων λαμβάνει ως είσοδο τις Landsat δορυφορικές εικόνες που έχουν υποστεί επεξεργασία από το προηγούμενο στάδιο και εφαρμόζει μία σειρά-αλυσίδα φίλτρων με τελικό σκοπό την παραγωγή μίας εικόνας στην οποία περιγράφονται οι ομάδες των καμένων περιοχών. Η παρούσα πτυχιακή εργασία ασχολείται με την παραλληλοποίηση της κύριας επεξεργασίας δεδομένων.

Η κύρια επεξεργασία δεδομένων υλοποιεί εσωτερικά τον αλγόριθμο ομαδοποίησης BSM_NOA. Η κύρια επεξεργασία δεδομένων λαμβάνει ως είσοδο τις δορυφορικές εικόνες μίας χρονιάς ή από δύο διαφορετικά έτη/ημερομηνίες, διεξάγοντας με ανάλογο τρόπο το αποτέλεσμα σε κάθε περίπτωση. Τα κύρια κριτήρια που χρησιμοποιούνται για να γίνει σωστή ομαδοποίηση του αλγόριθμου BSM_NOA, όταν δέχεται ως είσοδο δεδομένα ενός έτους, είναι ο δείκτης NBR, ο δείκτης ALBEDO και ο δείκτης βλάστησης NDVI. Όταν δέχεται ως είσοδο δεδομένα από δύο έτη/ημερομηνίες, λαμβάνει υπόψη και τη διαφορά μεταξύ των NDVI δεικτών. Το αποτέλεσμα που λαμβάνεται από την παραπάνω διαδικασία επεξεργάζεται περαιτέρω από μία σειρά χωρικών φίλτρων. Αρχικά, εφαρμόζεται φίλτρο εξάλειψης θορύβου από σύννεφα και νερό. Έπειτα, εφαρμόζεται ένα φίλτρο $3x3$ ή $5x5$ παραθύρου για κάθε pixel της εικόνας. Αν ένα pixel έχει οριστεί ως μη καμένο, αλλά η πλειοψηφία των γειτονικών του pixel έχουν οριστεί ως καμένα τότε αλλάζει και η δική του κατάσταση. Το επόμενο φίλτρο δημιουργεί ομάδες καμένων εκτάσεων. Ο αλγόριθμος υποθέτει πως δύο pixel βρίσκονται στην ίδια ομάδα αν η απόστασή τους σε pixel είναι μικρότερη από μία προκαθορισμένη απόσταση. Το τελευταίο φίλτρο ενώνει τις ομάδες σε μεγαλύτερες ομάδες, χρησιμοποιώντας την ίδια λογική, όπως για τη δημιουργία απλών ομάδων καμένων εκτάσεων.

Η παραπάνω διαδικασία είναι πλήρως ρυθμιζόμενη από το χρήστη. Επομένως, ορισμένα φίλτρα και διαδικασίες μπορούν να ενεργοποιηθούν ή να απενεργοποιηθούν.

3.2.1 Περιγραφή υλοποίησης

Το λογισμικό για την κύρια επεξεργασία δεδομένων υλοποιήθηκε στη γλώσσα προγραμματισμού Python 2.7 και εκτελείται σειριακά. Απευθύνεται σε υπολογιστές με Unix λειτουργικό σύστημα και εκτελείται μέσω τερματικού. Για τη συγγραφή του χρησιμοποιήθηκαν βιβλιοθήκες όπως η gdal [27] και η NumPy [28]. Το Σχήμα 3.1 περιγράφει τη διαδικασία του λογισμικού καθώς και την αλυσίδα φίλτρων που εφαρμόζεται.



Σχήμα 3.1: Περιγραφή σταδίων της εφαρμογής

Είσοδος δεδομένων

Αρχικά η εφαρμογή αποθηκεύει τις μπάντες της/ων Landsat εικόνας/ων που λαμβάνει ως είσοδο σε NumPy πίνακες 2 διαστάσεων. Η ανάγνωση της/ων Landsat εικόνας/ων γίνεται με χρήση της βιβλιοθήκης gdal.

Αλγόριθμος BSM_NOA (Classifier)

Από τη στιγμή που δημιουργεί τους πίνακες με την πληροφορία, εκτελεί τον αλγόριθμο BSM_NOA, ο οποίος υλοποιείται από την κλάση LandsatFireClassifier, αν είσοδος είναι μία Landsat εικόνα, δηλαδή μία μόνο ημερομηνία προς επεξεργασία, και από την κλάση LandsatFireClassifer2, αν είσοδος είναι δύο Landsat εικόνες, δηλαδή δύο ημερομηνίες προς επεξεργασία. Σε αυτόν τον αλγόριθμο επεξεργάζονται όλα τα pixel της Landsat εικόνας, ένα προς ένα, και εφαρμόζονται σε αυτά οι συντελεστές ALBEDO και NDVI. Αν το συγκεκριμένο στάδιο αποφασίσει πως ένα pixel είναι καμένο, τότε θέτει την τιμή 1 σε αυτό, αλλιώς θέτει την τιμή 0. Το αποτέλεσμα από αυτό το στάδιο είναι μία δυαδική εικόνα ίδιων διαστάσεων και μεγέθους με την αρχική Landsat εικόνα. Για να παρθεί η απόφαση αν ένα pixel είναι καμένο ή όχι, σε αυτό το στάδιο, επεξεργάζονται οι μπάντες 3, 4 και 7 της/ων Landsat εικόνας/ων. Το Παράδειγμα 3.1 παρουσιάζει τη συνάρτηση που εφαρμόζει τον αλγόριθμο BSM_NOA ανά pixel των μπάντων 3, 4, και 7 θεωρώντας πως η είσοδος είναι μόνο μία Landsat εικόνα.

```
def __landsatFirePredicate(self, b3, b4, b7):
    """ Fire predicate for Landsat.
    """
    if b3 == 0 or b4 == 0 or b7 == 0:
        return 0

    indexNIR = float(b4)
    if indexNIR > 60.0:
        return 0
    indexALBEDO = math.floor((float(b3) + float(b4)) / 2.0)
    if indexALBEDO > 50.0:
        return 0

    indexNBR = 0.0
    sum47 = float(b4) + float(b7)
    diff47 = float(b4) - float(b7)
    if sum47 != 0.0:
        indexNBR = diff47 / sum47
        indexNBR = (indexNBR + 1.0) * 255.0 / 2.0

    if indexNBR > 126.0:
        return 0

    return 1
```

Παράδειγμα 3.1: Η συνάρτηση που κάνει ταξινόμηση (classify) στον αλγόριθμο BSM_NOA ανά pixel

Φίλτρο εξάλειψης θορύβου από νερό και σύννεφα

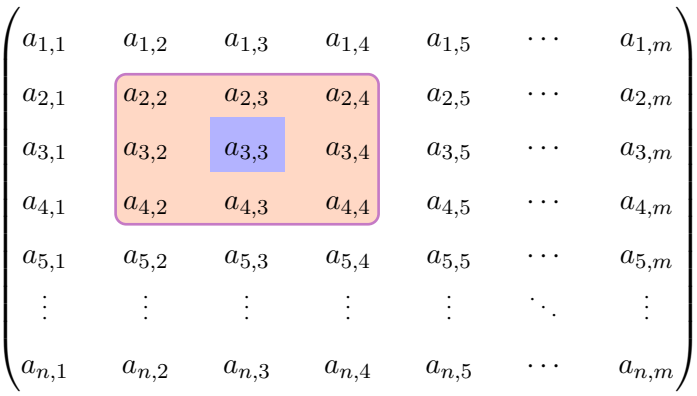
Το πρώτο σε σειρά φίλτρο από την αλυσίδα φίλτρων είναι το φίλτρο που υλοποιείται από την κλάση CloudWaterMaskEraseFireFilter. Σε αυτό το φίλτρο το αποτέλεσμα του προηγούμενου σταδίου συγκρίνεται με μία εικόνα στην οποία περιγράφεται τυχόν ύπαρξη θορύβου από σύννεφα. Και σε αυτό το στάδιο η επεξεργασία της εικόνας γίνεται ανά pixel, συγκρίνοντας το x, y σημείο της κύριας εικόνας με το αντίστοιχο σημείο της εικόνας στην οποία περιγράφονται οι θόρυβοι. Το φίλτρο αυτό θέτει ως μη καμένο ένα pixel το οποίο είχε ορισθεί ως καμένο λόγω θορύβου από σύννεφα. Το Παράδειγμα 3.2 παρουσιάζει τη συνάρτηση που εφαρμόζει το φίλτρο ανά pixel.

```
def run(self, fireslmg):
    maskArray = self.__mask.getBand(1)

    img = fireslmg.getBand(1)
    XSize = maskArray.shape[0]

    for i in range(XSize):
        idx = (maskArray[i] != self.__passValue)
        img[i][idx] = 0

    return fireslmg
```

Παράδειγμα 3.2: Η συνάρτηση που εφαρμόζει το φίλτρο εξάλειψης θορύβου από σύννεφα ή νερό

Φίλτρο πλειοψηφίας γειτονικών pixel

Το δεύτερο σε σειρά φίλτρο από την αλυσίδα φίλτρων είναι το φίλτρο που υλοποιείται από την κλάση MajoritySetFireFilter. Σε αυτό το φίλτρο το αποτέλεσμα του προηγούμενου σταδίου επεξεργάζεται από ένα χωρικό φίλτρο με σκοπό να αναγνωριστούν τυχόν λανθασμένες αποφάσεις για ένα pixel. Ένα μη καμένο pixel θεωρείται ως καμένο αν η πλειοψηφία των γειτονικών καμένων pixel είναι μεγαλύτερη ή ίση από μία τιμή-κατώφλι που δίνεται ως είσοδος στο λογισμικό. Σε κάθε pixel της εικόνας, εφαρμόζεται ένα φίλτρο παραθύρου, όπως παρουσιάζεται στο Σχήμα 3.2, όπου οι εξ ορισμού διαστάσεις του είναι $3x3$.



Σχήμα 3.2: Εφαρμογή χωρικού φίλτρου $3x3$ στο κελί με τιμή $a_{3,3}$

Το Παράδειγμα 3.3 παρουσιάζει τις συναρτήσεις που υλοποιούν αυτό το φίλτρο και το εφαρμόζουν ανά pixel.

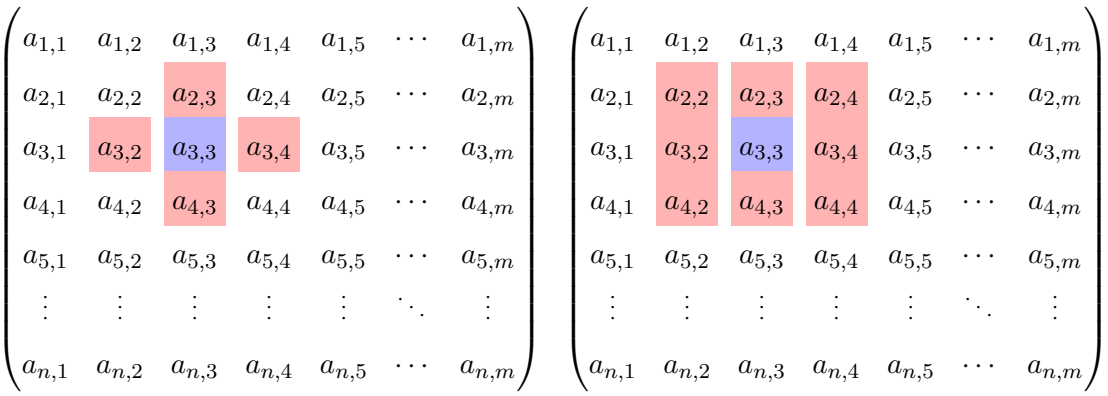
```
def run(self, fireslmg):
    footprint = numpy.ones((self.windowSize, self.windowSize))
    filtered = generic_filter(fireslmg.getBand(1), self.__majority, footprint=
    footprint, mode='constant', cval=0.0)
    result = bsm.image.Image.createFromNumpyArray(filtered)
    return result

def __majority(self, bufferArray):
    middle = self.windowSize * self.windowSize / 2
    if bufferArray[middle] > 0 or bufferArray.sum() >= self.majorityThreshold:
        return 1
    return 0
```

Παράδειγμα 3.3: Οι συναρτήσεις που εφαρμόζουν και υλοποιούν το φίλτρο πλειοψηφίας γειτόνων

Τα παραπάνω φίλτρα, ουσιαστικά ταξινομούν τα pixel σε καμένα ή μη καμένα και αφαιρούν τυχόν θόρυβο που υπάρχει. Τα ακόλουθα φίλτρα, έχουν ως στόχο τη δημιουργία ομάδων καμένων pixel.

Φίλτρο ομαδοποίησης καμένων pixel

Το τρίτο σε σειρά φίλτρο από την αλυσίδα φίλτρων υλοποιείται από την κλάση ClumpAndEliminateFilter. Το φίλτρο αυτό δημιουργεί ομάδες καμένων pixel. Για να είναι δύο καμένα pixel στην ίδια ομάδα πρέπει να υπάρχει τουλάχιστον ένα μονοπάτι καμένων pixel το οποίο τα ενώνει. Η εφαρμογή δίνει ως είσοδο στο φίλτρο, έναν αριθμό-κατώφλι ο οποίος είναι το πλήθος των καμένων pixel που πρέπει να υπάρχουν σε κάθε ομάδα ώστε να μην θεωρηθεί ως θόρυβος, και τον αριθμό των γειτονικών pixel, ενός σημείου που πρέπει να αναλυθούν περαιτέρω όπως φαίνεται στο Σχήμα 3.3. Ο αριθμός αυτός μπορεί να πάρει τις τιμές 4 ή 8.



Αναζήτηση στα 4 γειτονικά pixel

Αναζήτηση στα 8 γειτονικά pixel

Σχήμα 3.3: Αναζήτηση σε γειτονικά pixel του σημειού με τιμή $a_{3,3}$ με σκοπό τη δημιουργία ομάδων καμένων περιοχών

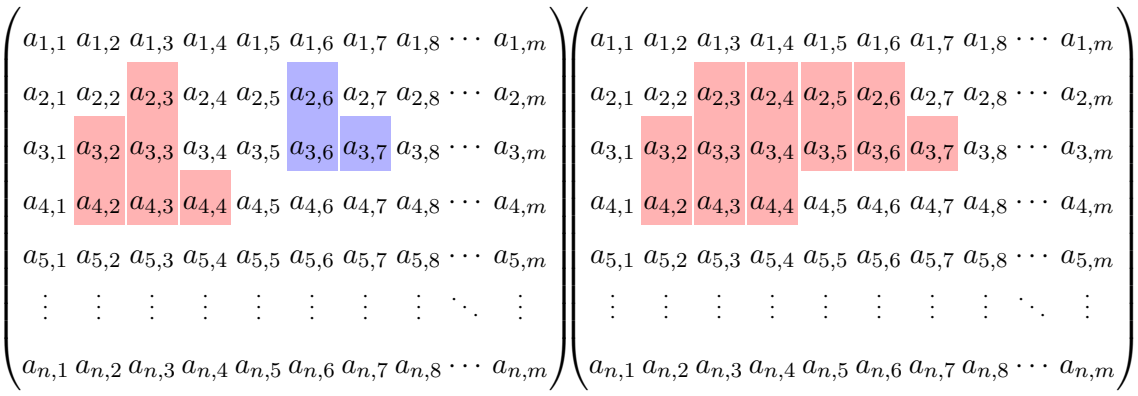
Για κάθε καμένο pixel της επεξεργασμένης από τα προηγούμενα στάδια εικόνας εφαρμόζεται ο αλγόριθμος Breadth-First Search (BFS) [Skiena], με σκοπό την εύρεση μονοπατιών καμένων pixel. Έπειτα από την ανωτέρω επεξεργασία, κάθε καμένο pixel διαθέτει ως τιμή το θετικό αριθμό της ομάδας στην οποία ανήκει. Το φίλτρο αφαιρεί τις ομάδες οι οποίες διαθέτουν λιγότερα καμένα pixel από ότι το κατώφλι. Το Παράδειγμα 3.4 παρουσιάζει τμήμα αυτού του φίλτρου.

```
def run(self, fireslmg):
    img = fireslmg.getBand(1)
    XSize = img.shape[0]
    YSize = img.shape[1]
    self.components = {}
    component = 1

    for i in range(XSize):
        for j in range(YSize):
            if img[i, j] == 1:
                if self.neighbors == 8:
                    (component, size) = self.__bfs8(img, i, j, component)
                else:
                    (component, size) = self.__bfs4(img, i, j, component)
                self.components[component] = size

    self.components = dict(filter(lambda(c, f): f > self.eliminateThreshold ,
    self.components.items()))

    for i in range(XSize):
        for j in range(YSize):
            if img[i, j] == 0:
                continue
            if img[i, j] not in self.components:
                img[i, j] = 0
    return fireslmg
```

Παράδειγμα 3.4: Η κύρια συνάρτηση που εφαρμόζει και υλοποιεί το φίλτρο ομαδοποίησης καμένων pixel

Φίλτρο ένωσης ομάδων καμένων pixel

Το τελευταίο φίλτρο είναι το φίλτρο που υλοποιείται από την κλάση UnionCloseByFireFilter. Το φίλτρο αυτό ενώνει ομάδες καμένων pixel, δημιουργώντας μεγαλύτερες ομάδες. Δύο ομάδες πρέπει να ενωθούν αν η απόσταση μεταξύ δύο pixel τους είναι μικρότερη ή ίση με 3 pixel, είτε διαγώνια είτε σε ευθεία γραμμή. Τυχόν μη καμένα pixel μεταξύ των ομάδων αυτών, ορίζονται ως καμένα. Για τη διαχείριση των ομάδων χρησιμοποιείται η δομή ξένων συνόλων (disjoint-set data structure ή union-find) [Cormen, Leiserson, Rivest, *et al.*] Η υλοποίηση της δομής union-find μπορεί να βρεθεί στο σύνδεσμο http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/215912. Τα pixel των ομάδων που πρέπει να ενωθούν έχουν πλέον την ίδια τιμή. Στο Σχήμα 3.4 η μπλε ομάδα ενώνεται με την κόκκινη ομάδα.



1) Είσοδος φίλτρου 2) Ένωση ομάδων

Σχήμα 3.4: Συγχώνευση ομάδων καμένων περιοχών που βρίσκονται σε κοντινή απόσταση

Έξοδος εφαρμογής

Στο τέλος, η εφαρμογή αποθηκεύει το αποτέλεσμα που προκύπτει από την παραπάνω σειρά φίλτρων σε ένα αρχείο τύπου shapefile. Επίσης η εφαρμογή δημιουργεί και επιπλέον βοηθητικά δεδομένα, χρήσιμα για την επεξεργασία της κύριας εικόνας shapefile. Το αποτέλεσμα από την κύρια επεξεργασία δεδομένων επεξεργάζεται περαιτέρω από τη μετα-επεξεργασία δεδομένων.

3.2.2 Παράδειγμα εκτέλεσης

Οι Πίνακες 3.1 και 3.2 εξηγούν την εκτέλεση της εφαρμογής, μέσω των αλλαγών στα δεδομένα της κατά τη διάρκεια της εκτέλεσής της. Τα δεδομένα είναι κομμάτι από τα έτη 2004-2005. Για λόγους απλότητας η κάθε εικόνα έχει μέγεθος $6x6$. Επίσης, χρησιμοποιείται φίλτρο πλειοψηφίας $3x3$ παραθύρου με κατώφλι τον αριθμό 1, όπως και ο αριθμός-κατώφλι για το φίλτρο εξάλειψης θορύβου (ClumpAndEliminateFilter) ορίζεται ως 1. Το κομμάτι κώδικα στο Παράδειγμα 3.5 παρουσιάζει το πρόγραμμα (script) bash φλοιού που χρησιμοποιείται για να λειτουργήσει η εφαρμογή:

```
CLOUD_WATER_MASK_2=LT71820312005211_mask.tif
BAND3_FNAME=LT71820312005211B03.tif
BAND4_FNAME=LT71820312005211B04.tif
BAND7_FNAME=LT71820312005211B07.tif

BAND3_2_FNAME=LT71820312004257B03.tif
BAND4_2_FNAME=LT71820312004257B04.tif

./chain.py -q -s  --eliminate-filter-threshold=1 --cloud-water-mask2=
    $CLOUD_WATER_MASK_2 --band3-filename=$BAND3_FNAME --band4-filename=
    $BAND4_FNAME --band7-filename=$BAND7_FNAME --band3-second-filename=
    $BAND3_2_FNAME --band4-second-filename=$BAND4_2_FNAME
```

Παράδειγμα 3.5: Πρόγραμμα bash φλοιού που χρησιμοποιείται για τη ρύθμιση της κύριας εφαρμογής

$$\begin{pmatrix} 32 & 0 & 32 & 10 & 20 & 0 \\ 10 & 0 & 32 & 0 & 32 & 0 \\ 10 & 0 & 0 & 0 & 0 & 0 \\ 10 & 20 & 10 & 0 & 0 & 20 \\ 32 & 0 & 0 & 10 & 0 & 0 \\ 20 & 0 & 0 & 10 & 20 & 0 \end{pmatrix}$$
Μπάντα 3 2004

$$\begin{pmatrix} 0 & 0 & 15 & 20 & 20 & 15 \\ 16 & 20 & 20 & 0 & 16 & 0 \\ 0 & 0 & 20 & 0 & 0 & 0 \\ 0 & 15 & 20 & 20 & 20 & 0 \\ 0 & 0 & 0 & 0 & 20 & 16 \\ 20 & 0 & 20 & 15 & 16 & 0 \end{pmatrix}$$
Μπάντα 4 2004

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$
Μάσκα σύννεφου 2005

$$\begin{pmatrix} 0 & 10 & 20 & 0 & 54 & 54 \\ 0 & 0 & 20 & 20 & 0 & 10 \\ 20 & 0 & 0 & 54 & 54 & 10 \\ 0 & 0 & 10 & 54 & 0 & 20 \\ 0 & 10 & 20 & 0 & 0 & 54 \\ 20 & 0 & 10 & 20 & 20 & 54 \end{pmatrix}$$
Μπάντα 3 2005

$$\begin{pmatrix} 60 & 0 & 64 & 20 & 0 & 60 \\ 0 & 30 & 30 & 60 & 0 & 60 \\ 30 & 64 & 30 & 64 & 30 & 30 \\ 0 & 64 & 60 & 0 & 20 & 20 \\ 64 & 0 & 20 & 20 & 64 & 20 \\ 60 & 0 & 64 & 30 & 30 & 30 \end{pmatrix}$$
Μπάντα 4 2005

$$\begin{pmatrix} 0 & 40 & 40 & 0 & 0 & 0 \\ 20 & 40 & 45 & 20 & 45 & 0 \\ 40 & 0 & 45 & 45 & 0 & 0 \\ 0 & 40 & 45 & 20 & 45 & 0 \\ 20 & 45 & 0 & 45 & 20 & 0 \\ 40 & 0 & 0 & 45 & 0 & 0 \end{pmatrix}$$
Μπάντα 7 2005

Πίνακας 3.1: Είσοδος παραδείγματος για τα έτη 2004-2005

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

BSM_NOA

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Cloud water mask

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Majority

$$\begin{pmatrix} 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 & 0 \\ 0 & 5 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Clump and eliminate

$$\begin{pmatrix} 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 5 & 5 & 0 & 0 & 0 \\ 0 & 5 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

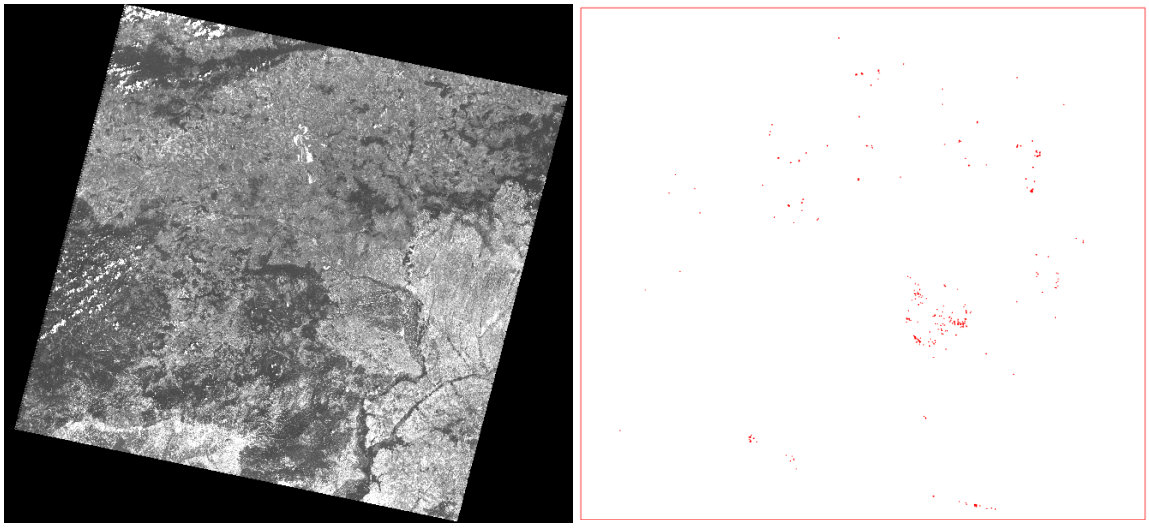
Union close by fire filter

Πίνακας 3.2: Έξοδος ανά φίλτρο

Στην πραγματικότητα, η εκτέλεση της εφαρμογής με χρήση των ρυθμίσεων του προγράμματος φλοιού 3.5 δε δημιουργεί λογικό αποτέλεσμα. Έτσι, για την παρουσίαση της εξόδου σε μορφή εικόνας χρησιμοποιείται το πρόγραμμα φλοιού 3.6. Η Εικόνα 3.5β′ είναι το αποτέλεσμα απο την εκτέλεση του προγράμματος. Όμως, λόγω του ότι είναι δύσκολο να κατανοηθεί η έξοδος χωρίς βοηθητικά δεδομένα χρησιμοποιείται η δορυφορική Εικόνα 3.5α′. Έτσι, επεξεργάζοντας τις Εικόνες 3.5β′ και 3.5α′ με το λογισμικό qgis [31] δημιουργείται το αποτέλεσμα της Εικόνας 3.6.

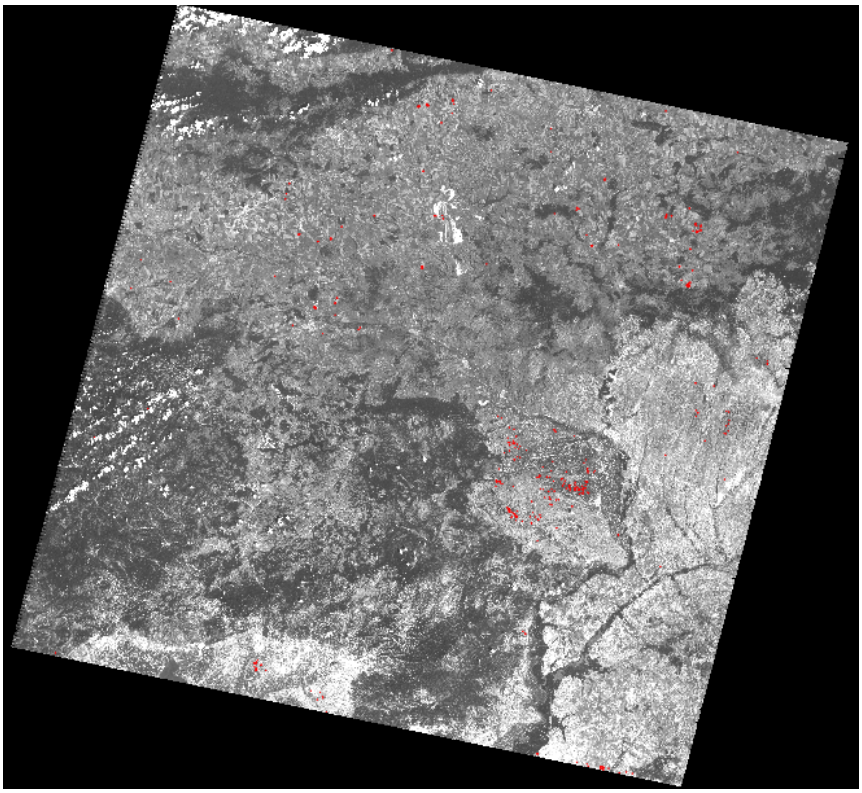
```
./chain.py --verbose -s --cloud-water-mask2=LT71820312005211_mask.tif --band3-
    filename=LT71820312005211B03.tif --band4-filename=LT71820312005211B04.tif
    --band7-filename=LT71820312005211B07.tif --band3-second-filename=
    LT71820312004257B03.tif --band4-second-filename=LT71820312004257B04.tif
```

Παράδειγμα 3.6: Πρόγραμμα bash φλοιού που χρησιμοποιείται για τη δημιουργία των αποτελεσμάτων



(α΄) Μπάντα 3 2005 (β΄) Έξοδος σειριακής εκτέλεσης εφαρμογής

Σχήμα 3.5: Δεδομένα που συνδυάζονται για την παρουσίαση του αποτελέσματος



Σχήμα 3.6: Παρουσίαση αποτελέσματος χρησιμοποιώντας ως επίπεδα την μπάντα 3 και την έξοδο της εφαρμογής

Κεφάλαιο 4

Παραλληλοποίηση διαδικασίας χαρτογράφησης καμένων εκτάσεων του Εθνικού Αστεροσκοπείου Αθηνών

Any fool can write code that a computer can understand. Good programmers write code that humans can understand.

- Martin Fowler

Keep it simple, stupid!

- The KISS principle

Η παρούσα πτυχιακή εργασία ασχολείται με την παραλληλοποίηση της κύριας επεξεργασίας δορυφορικών δεδομένων του αλγόριθμου burn scar mapping (bsm). Στην πραγματικότητα είναι η μετατροπή του σειριακού αλγόριθμου κύριας επεξεργασίας δορυφορικών δεδομένων σε παράλληλο, και όχι η δημιουργία ενός καινούργιου αλγόριθμου. Η παράλληλη υλοποίηση έχει ως στόχο να εκτελείται σε συστοιχία και όχι σε έναν και μόνο υπολογιστή.

4.1 Περιγραφή παράλληλης υλοποίησης

Τα δεδομένα που επεξεργάζεται η σειριακή υλοποίηση, στην πραγματικότητα, είναι Landsat δισδιάστατες εικόνες οι οποίες μπορούν να αναπαρασταθούν σε μορφή δισδιάστατων πινάκων. Σε αυτές τις Landsat εικόνες εφαρμόζονται τα διάφορα φίλτρα του αλγόριθμου. Η ιδέα πίσω από

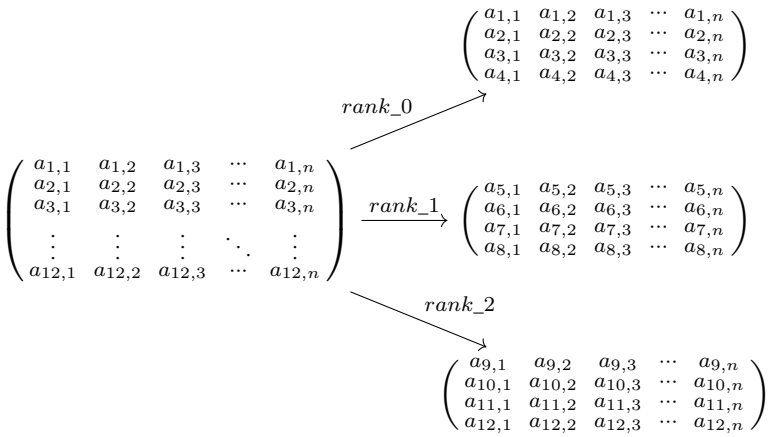
την παραλληλοποίηση είναι πως αρκετά από τα φίλτρα αυτά εφαρμόζονται κατά σημείο (ανά pixel). Έτσι, υπάρχει η δυνατότητα να χωρισθεί η Landsat εικόνα, που λαμβάνει ως είσοδο η εφαρμογή κάθε φορα, και να μοιρασθεί στις διεργασίες με σκοπό να εκτελεσθεί παράλληλα το ίδιο φίλτρο σε διαφορετικά pixel της εικόνας. Με αυτόν τον τρόπο κάθε διεργασία έχει ένα κομμάτι της μεγαλύτερης εικόνας ώστε να εφαρμόσει σε αυτό τα φίλτρα. Έστω ότι υπάρχουν n γραμμές δεδομένων και επιθυμείται να χωρισθούν σε s διεργασίες, κάθε διεργασία i θα πάρει:

$$n_i = n/s + f(x) \text{ όπου } f(x) = \begin{cases} 1 & \text{αν } n \mod s > i \\ 0 & \text{αλλιώς} \end{cases}$$

Έστω ότι η είσοδος βρίσκεται σε έναν δισδιάστατο πίνακα, τότε το διάστημα των γραμμών του πίνακα που θα πάρει κάθε διεργασία i θα είναι τό $[a_i, b_i)$ όπου $a_i = i \cdot n/s + min(i, n \mod s)$ και $b_i = n_i + b_i$.
Ακολουθεί η συνάρτηση calculate_limits η οποία βρίσκει το διάστημα και είναι υλοποιημένη σε γλώσσα Python:

```
def calculate_limits(n,s):
    for i in range(s):
        ai = i*(n/s) + min(i,n%s)
        ni = (n/s) + (lambda x: 1 if n%s > x else 0)(i)
        bi = ai + ni
```

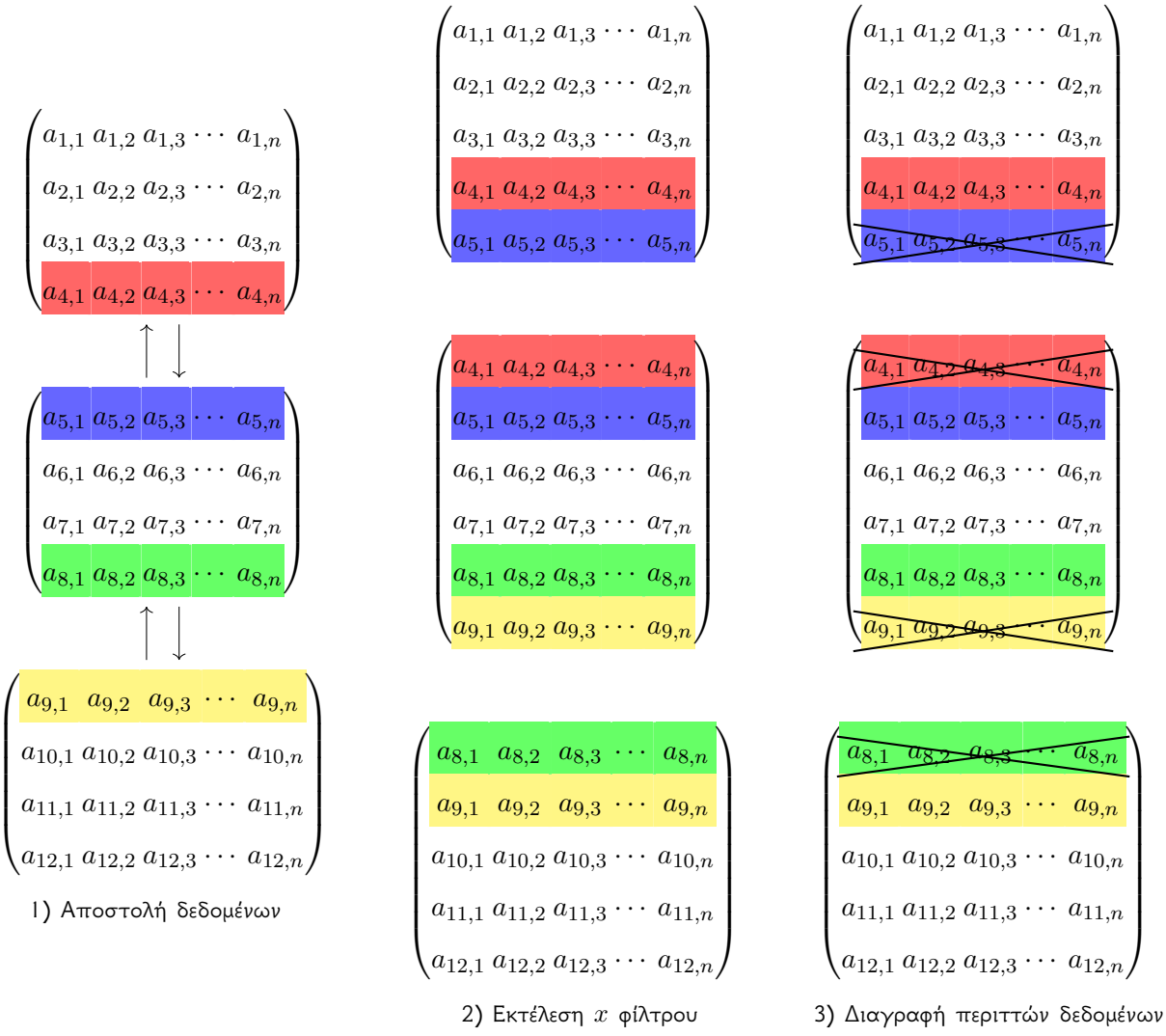
Επομένως, η παραλληλοποίηση γίνεται σε επίπεδο δεδομένων και όχι σε επίπεδο εργασιών προς εκτέλεση, όπως παρουσιάζεται στο Σχήμα 4.1.



Σχήμα 4.1: Διαμοιρασμός πίνακα $12xn$ σε σύστημα με 3 διεργασίες

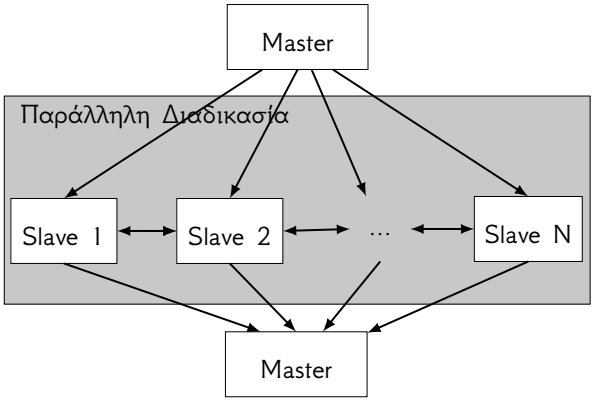
Επικοινωνία μεταξύ των διεργασιών χρειάζεται μόνο στα φίλτρα που χρειάζονται τα γειτονικά pixel ενός pixel, ώστε να παρθεί η απόφαση για την τιμή του. Παράδειγμα τέτοιου φίλτρου είναι

το φίλτρο πλειοψηφίας γειτονικών pixel. Η ροή δεδομένων στα φίλτρα που χρειάζεται επικοινωνία παρουσιάζεται στο Σχήμα 4.2.



Σχήμα 4.2: Ροή δεδομένων μεταξύ των διεργασιών κατά τη διάρκεια εκτέλεσης ενός φίλτρου

Το Σχήμα 4.3 παρουσιάζει την αφηρημένη αρχιτεκτονική της παράλληλης εφαρμογής.



Σχήμα 4.3: Αφηρημένη αρχιτεκτονική παράλληλης εφαρμογής

Τέλος, η παραλληλοποίηση γίνεται μόνο στα φίλτρα που εφαρμόζονται στις Landsat εικόνες. Η είσοδος της εφαρμογής, διαβάζεται μόνο από την κύρια διεργασία η οποία έπειτα μοιράζει σε κάθε διεργασία το αντίστοιχο κομμάτι της εικόνας για επεξεργασία. Αντίστοιχα η έξοδος δημιουργείται από την κύρια διεργασία. Σε αυτά τα στάδια της εφαρμογής οι υπόλοιπες διεργασίες περιμένουν την κύρια διεργασία να ολοκληρώσει την εργασία της.

Γλώσσα προγραμματισμού

Για την παράλληλη υλοποίηση χρησιμοποιείται η γλώσσα προγραμματισμού Python, λόγω του ότι και η σειριακή υλοποίηση είναι υλοποιημένη σε αυτήν. Επίσης, η γλώσσα Python χρησιμοποιείται σχεδόν σε όλες τις σύγχρονες γεωγραφικές και επεξεργασίας εικόνας εφαρμογές, παρέχοντας αρκετές βοηθητικές βιβλιοθήκες (modules) για το σκοπό αυτό. Παράδειγμα τέτοιων βιβλιοθηκών είναι η gdal και η opencv [32]. Τέλος, η γλώσσα Python επικοινωνεί και χρησιμοποιεί εύκολα λογισμικό που είναι υλοποιημένο στις γλώσσες C και C++, όπως και η γλώσσα προγραμματισμού Lua, γεγονός που είναι χρήσιμο για βελτιστοποίηση κρίσιμων σημείων του λογισμικού. Για τους παραπάνω λόγους, η γλώσσα Python θεωρείται ως η πιο κατάλληλη για τη συγγραφή του λογισμικού αυτού.

Κύρια δομή δεδομένων

Η δομή δεδομένων που χρησιμοποιείται περισσότερο για την υλοποίηση της εφαρμογής είναι ο πίνακας τύπου ndarray (ή NumpPy array) που υλοποιείται στη βιβλιοθήκη NumPy. Η NumPy βιβλιοθήκη χρησιμοποιείται και για τη σειριακή εφαρμογή. Η NumPy χρησιμοποιείται κυρίως για την επεξεργασία και αποθήκευση δεδομένων σε επιστημονικές εφαρμογές. Αποτελείται από:

- τη ndarray δομή δεδομένων

- εργαλεία για ενσωμάτωση C/C++ και Fortran κώδικα
- μεθόδους και δυνατότητες για γραμμική άλγεβρα, μετασχηματισμό Fourier και τυχαίους αριθμούς

Η βιβλιοθήκη NumPy μπορεί να χρησιμοποιηθεί και για απλή χρήση, όπου χρειάζεται εξικονόμηση μνήμης και ταχύτητα. Ένας χρήστης ο οποίος έχει χρησιμοποιήσει Matlab ή Octave μπορεί εύκολα να χρησιμοποιήσει τη βιβλιοθήκη αυτή αφού οι μέθοδοι της έχουν παρόμοια ονομασία και παραμέτρους με τις αντίστοιχες σε Matlab ή Octave. Η βιβλιοθήκη NumPy αποτελεί ενσωματωμένο κομμάτι της βιβλιοθήκης SciPy [33], μία βιβλιοθήκη όπου συγκεντρώνει χρήσιμα πακέτα (packages) για επιστημονική χρήση της γλώσσα Python. Τέλος η βιβλιοθήκη NumPy χρησιμοποιείται και από τη βιλβιοθήκη mpi4py [Dalcín, Paz, Storti, *et al.*] κάτι που βοηθάει στη σύνταξη του παράλληλου προγράμματος.

Η δομή δεδομένων ndarray (n-dimensional array), σε αντίθεση με τη δομή δεδομένων list της Python, μπορεί να αποθηκεύσει μόνο ίδιου τύπου δεδομένα, όπως οι πίνακες στις γλώσσες C/C++. Όπως και οι πίνακες σε αυτές τις γλώσσες, ο ndarray πίνακας αποτελείται από συνεχόμενες θέσεις μνήμης. Ένας πίνακας ndarray έχει σταθερό μέγεθος το οποίο καθορίζεται κατά τη δημιουργία του. Τυχόν αλλαγή στο μέγεθος του σημαίνει αντιγραφή του ίδιου πίνακα σε νέες θέσεις μνήμης και διαγραφή του παλιού στιγμιότυπου του πίνακα.

Η δομή δεδομένων ndarray μειονεκτεί σε σύγκριση με τον built-in τύπο list της Python, από άποψη ευκολίας χρήσης, αλλά ισχύει το εντελώς αντίθετο για τις επιστημονικές εφαρμογές. Λόγω των ιδιοτήτων της, η διαχείριση ndarray πινάκων είναι γρηγορότερη, αφού η προσπέλαση μίας θέσης αυτών των πινάκων χρειάζεται σταθερό χρόνο $O(1)$. Επίσης η κατανάλωση μνήμης μειώνεται αφού δεν χρειάζεται αρκετή βοηθητική πληροφορία για τη σωστή λειτουργία της δομής (μειώνεται το overhead). Ακόμα, υπάρχει η δυνατότητα να ορισθεί ο τύπος των δεδομένων, κάτι που μπορεί να μειώσει ακόμα περισσότερο την κατανάλωση μνήμης. Όλα τα παραπάνω καθιστούν τους NumPy πίνακες εξίσου αρκετά γρήγορους από άποψη ταχύτητας εκτέλεσης όπως τους αντίστοιχους της γλώσσας C.

Το Παράδειγμα 4.1 παρουσιάζει τη δημιουργία ενός πίνακα NumPy $10x10$ διαστάσεων και τύπου δεδομένων uint8 που περιέχει αρχικά σε όλες τις θέσεις μνήμης του τον αριθμό μηδέν (0).

```
1 import numpy
2 array = numpy.zeros([10, 10], dtype=numpy.uint8)
```

Παράδειγμα 4.1: Χρήση της βιβλιοθήκης NumPy στη γλώσσα Python

Παράλληλη διασύνδεση

Η διασύνδεση που χρησιμοποιείται είναι η MPI, χρησιμοποιώντας τη βιβλιοθήκη mpi4py. Η βιβλιοθήκη mpi4py θεωρείται ως η πιο κατάλληλη μεταξύ αρκετών βιβλιοθηκών για τους λόγους που περιγράφονται παρακάτω.

Για τη γλώσσα Python έχουν υλοποιηθεί αρκετές βιβλιοθήκες για παράλληλο προγραμματισμό με χρήση MPI. Τέτοιες είναι οι ακόλουθες:

	Modules
1	mpi4py
2	Pypar
3	MYMPI
4	pyMPI
5	boostmpi
6	Scientific.MPI

Πίνακας 4.1: Python MPI βιβλιοθήκες

Η mpi4py θεωρείται η καλύτερη λόγω του ότι υλοποιεί τις περισσότερες λειτουργίες της διασύνδεσης MPI. Επίσης είναι υλοποιημένη στην υβριδική γλώσσα προγραμματισμού Cython με σκοπό τα κρίσιμα μέρη της υλοποίησης να έχουν βέλτιστη απόδοση. Η βιβλιοθήκη αυτή, αν και όχι η πιο εύκολη σε χρήση, είναι η μόνη που εξασφαλίζει τη βέλτιστη απόδοση του συστήματος, από άποψη χρόνου και κατανάλωση μνήμης, σε σχέση με τις υπόλοιπες. Παρέχει στο χρήστη μεθόδους με τις οποίες μπορεί να ανταλλάξει Python αντικείμενα (pickle-based communication) αλλά ταυτόχρονα παρέχει τις αντίστοιχες μεθόδους για ανταλλαγή δεδομένων τύπου numpy πινάκων (array data communication). Η διαφορά μεταξύ των δύο τρόπων είναι πως με numpy πίνακες η μεταφορά δεδομένων είναι γρηγορότερη αλλά ο προγραμματισμός της είναι δυσκολότερος. Για την παρούσα πτυχιακή εργασία χρησιμοποιείται η ανταλλαγή δεδομένων τύπου numpy πινάκων. Ο λόγος είναι η επιθυμία για βέλτιστη απόδοση και το γεγονός ότι τα δεδομένα είναι ήδη αποθηκευμένα σε numpy πίνακες στη σειριακή υλοποίηση της εφαρμογής. Τέλος, μπορεί εύκολα να χρησιμοποιηθεί από ένα χρήστη που έχει χρησιμοποιήσει την αντίστοιχη βιβλιοθήκη MPI για τις γλώσσες C/C++. Ο Πίνακας 4.2 παρουσιάζει τις 12 σημαντικότερες MPI μεθόδους στις διάφορες Python βιβλιοθήκες.

	pypar	mpi4py	myMPI	pyMPI	Scientific.MPI	boostmpi
MPI_Send	✓	✓	✓	✓	✓	✓
MPI_Recv	✓	✓	✓	✓	✓	✓
MPI_Sendrecv		✓		✓		
MPI_Isend		✓		✓	✓	✓
MPI_Irecv		✓		✓	✓	✓
MPI_Bcast	✓	✓	✓	✓	✓	✓
MPI_Reduce	✓	✓	✓	✓	✓	✓
MPI_Allreduce		✓		✓	✓	✓
MPI_Gather	✓	✓	✓	✓		✓
MPI_Allgather		✓		✓		✓
MPI_Scatter	✓	✓	✓	✓		✓
MPI_Alltoall		✓	✓			✓

Πίνακας 4.2: Οι 12 σημαντικότερες μέθοδοι ανά Python βιβλιοθήκη. Πηγή:[Lin]

4.2 Παράλληλη αλυσίδα φίλτρων

Η αλυσίδα φίλτρων παραλληλοποιείται πλήρως χρησιμοποιώντας ελάχιστη επικοινωνία μεταξύ των διεργασίων. Αφού κάθε διεργασία λαμβάνει το κομμάτι της εικόνας που πρέπει να εφαρμόσει τα φίλτρα, ξεκινά η παράλληλη διαδικασία.

Αλγόριθμος BSM_NOA (Classifier)

Η εκτέλεση του αλγόριθμου γίνεται ανά pixel οπότε δεν χρειάζεται επικοινωνία μεταξύ των διεργασίων. Ο λόγος είναι πως κάθε διεργασία εκτελεί τον classifier στο κομμάτι της εικόνας που της αντιστοιχεί.

Φίλτρο εξάλειψης θορύβου από νερό και σύννεφα

Στο πρώτο κατά σειρά φίλτρο η κάθε διεργασία εφαρμόζει το φίλτρο που υλοποιείται από την κλάση CloudWaterMaskEraseFireFilter στο κομμάτι της εικόνας που της δίνεται. Και σε αυτό το στάδιο δεν χρειάζεται επιπλέον επικοινωνία μεταξύ των διεργασιών. Ο λόγος είναι πως και

οι δύο εικόνες (η εικόνα φωτιών και η εικόνα θορύβων από σύννεφα και νερό), είναι χωρισμένες με τον ίδιο τρόπο έτσι ώστε κάθε διεργασία να έχει το σωστό κομμάτι της κάθε εικόνας.

Φίλτρο πλειοψηφίας γειτονικών pixel

Στο δεύτερο κατά σειρά φίλτρο η κάθε διεργασία εφαρμόζει το φίλτρο που υλοποιείται από την κλάση MajorityFireFilter στο κομμάτι της εικόνας που της δίνεται. Σε αυτό το στάδιο, χρειάζεται επικοινωνία μεταξύ των διεργασιών. Η επικοινωνία χρειάζεται λόγω του ότι εφαρμόζεται χωρικό φίλτρο παραθύρου για αυτό το φίλτρο, όπως αναφέρεται στο Σχήμα 3.2. Τα pixel που βρίσκονται στις εξωτερικές γραμμές των δεδομένων που διαθέτει κάθε διεργασία, χρειάζονται πληροφορία από τα δεδομένα των γειτονικών διεργασιών, με σκοπό να υλοποιηθεί σωστά το φίλτρο. Έτσι κάθε διεργασία στέλνει στις γειτονικές διεργασίες της, τις εξωτερικές γραμμές από το δισδιάστατο πίνακα που διαθέτει. Αντίστοιχα λαμβάνει από τις ίδιες τις εξωτερικές τους γραμμές. Το πλήθος των γραμμών είναι ανάλογο με το μέγεθος του παραθύρου. Αφού ολοκληρώνεται η προηγούμενη διαδικασία εκτελείται το φίλτρο και έπειτα οι εξωτερικές γραμμές που είχαν ληφθεί προηγουμένως, διαγράφονται από τα δεδομένα, όπως περιγράφεται στο Σχήμα 4.2.

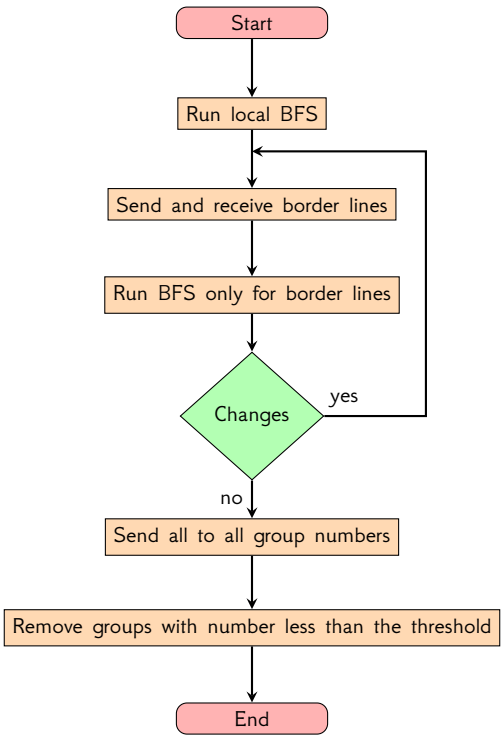
Φίλτρο ομαδοποίησης καμένων pixel

Στο τρίτο κατά σειρά φίλτρο η κάθε διεργασία εφαρμόζει το φίλτρο που υλοποιείται από την κλάση ClumpAndEliminateFilter στο κομμάτι της εικόνας που της δίνεται. Σε αυτό το στάδιο χρειάζεται περισσότερη επικοινωνία μεταξύ των διεργασιών σε σχέση με το προηγούμενο. Ο λόγος είναι διότι, σε αυτό το φίλτρο χρειάζεται να υλοποιηθεί ο αλγόριθμος BFS με παράλληλο τρόπο. Αρχικά κάθε διεργασία εφαρμόζει στα δεδομένα της τον αλγόριθμο BFS, δημιουργώντας τοπικές ομάδες καμένων περιοχών. Ο μέγιστος αριθμός ομάδων που μπορούν να υπάρχουν σε κάθε διεργασία ορίζεται από τη συνάρτηση

$$f(x) = \left\lceil \frac{(\text{number of rows of } x \text{ rank}) \cdot (\text{number of columns of } x \text{ rank})}{2} \right\rceil . \tag{4.1}$$

Κάθε διεργασία έχει ξεχωριστό εύρος αριθμών με το οποίο μπορεί να αριθμήσει τις ομάδες της. Λόγω του ότι η κύρια Landsat εικόνα χωρίζεται σε κομμάτια και επεξεργάζεται από κάθε διεργασία το αντίστοιχο κομμάτι, υπάρχει ενδεχόμενο η ίδια ομάδα να βρίσκεται σε δύο ή περισσότερα διαφορετικά κομμάτια, δηλαδή σε δύο ή περισσότερες διαφορετικές διεργασίες. Για τον λόγο αυτό, οι εξωτερικές γραμμές στέλνονται στις γειτονικές διεργασίες μίας διεργασίας και διαβάζονται από τις γειτονικές αντίστοιχα. Η διαδικασία αποστολής και λήψης των γειτονικών γραμμών γίνεται μέχρις ότου να μην υπάρχουν αλλαγές στις ομαδοποιήσεις των τοπικών ομάδων καμένων περιοχών, σε όλες τις διεργασίες. Στο σημείο αυτό διαγράφονται οι εξωτερικές γραμμές που είχαν ληφθεί από κάθε γειτονική διεργασία, όπως περιγράφεται στο Σχήμα 4.2. Έπειτα αποστέλνονται δεδομένα που περιγράφουν το πλήθος των pixel που διαθέτει κάθε ομάδα σε κάθε διεργασία, από και προς όλες τις διεργασίες. Αφότου στέλνονται τα δεδομένα σε κάθε διεργασία, η κάθε μία ξεχωριστά διαγράφει ομάδες που έχουν δημιουργηθεί από τυχόν θόρυβο, δηλαδή

ομάδες που το πλήθος των pixel τους δεν ξεπερνάει το κατώφλι που δίνεται ως είσοδος. Στό Σχήμα 4.4 παρουσιάζεται το διάγραμμα ροής του φίλτρου.



Σχήμα 4.4: Διάγραμμα ροής φίλτρου ομαδοποίησης καμένων pixel

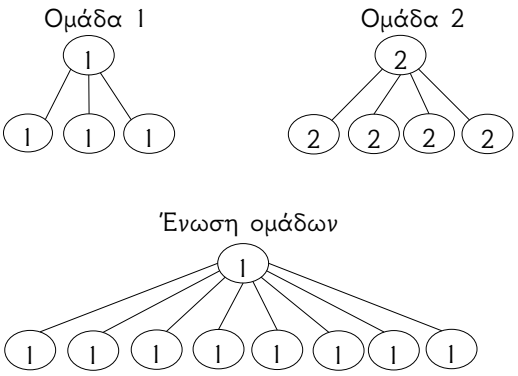
Φίλτρο ένωσης ομάδων καμένων pixel

Στο τελευταίο κατά σειρά φίλτρο η κάθε διεργασία εφαρμόζει το φίλτρο που υλοποιείται από την κλάση UnionCloseByFireFilter στο κομμάτι της εικόνας που της δίνεται. Ο σκοπός του σταδίου αυτού είναι να ομαδοποιηθούν τυχόν κοντινές σε απόσταση ομάδες καμένων περιοχών σε μεγαλύτερες ομάδες, όπως περιγράφεται στο Σχήμα 3.4. Η ήδη υπάρχουσα υλοποίηση της δομής ξένων συνόλων (union-find) δεν μπορεί να παραλληλοποιηθεί εύκολα. Έτσι υλοποιήθηκε με διαφορετικό τρόπο ώστε να ικανοποιεί τις ανάγκες της εφαρμογής. Η νέα υλοποίηση βασίζεται στη δομή quick-find union-find που προτείνεται από τους [Sedgewick and Wayne]. Η τελική υλοποίηση είναι μία παραλλαγή της υλοποίησης των [Sedgewick and Wayne] και υλοποιείται από την κλάση QuickFindUnionFind, όπως φαίνεται στο Παράδειγμα 4.2.

```
class QuickFindUnionFind(object):
  def __init__(self):
    self.__parents = {}

  def union(self, x, y):
    p_x = self.find(x)
    p_y = self.find(y)

    if p_x < p_y:
      self.__parents = dict(map(lambda (n, parent): (n, p_x) if parent == p_y
                                else (n, parent), self.__parents.iteritems()))
    elif p_x > p_y:
      self.__parents = dict(map(lambda (n, parent): (n, p_y) if parent == p_x
                                else (n, parent), self.__parents.iteritems()))

  def find(self, x):
    try:
      p = self.__parents[x]
    except:
      p = x
      self.__parents[x] = x

    return p
```

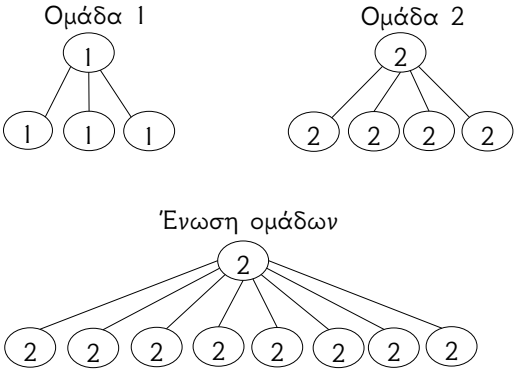
Παράδειγμα 4.2: Η υλοποίηση της δομής union-find που χρησιμοποιείται στην παράλληλη υλοποίηση

Η υλοποίηση της QuickFindUnionFind κατά την union διαδικασία τοποθετεί την ομάδα με το μεγαλύτερο αριθμό (id) υπό την ομάδα με το μικρότερο αριθμό (id). Η υλοποίηση αυτή δεν είναι η πιο αποδοτική, αλλά για να λειτουργήσει σωστά μία αποδοτική υλοποίηση πρέπει να διεξαχθεί περισσότερη επικοινωνία μεταξύ των διεργασιών που κάνει πιο αργό το συνολικό χρόνο εκτέλεσης του προγράμματος. Μία αποδοτική υλοποίηση είναι η χρήση της δομής weighted quick-union with path compression [Sedgewick and Wayne]. Το Σχήμα 4.5 παρουσιάζει τη χρήση της κλάσης QuickFindUnionFind κατά την *ένωση* (union) δύο ομάδων καμένων περιοχών.



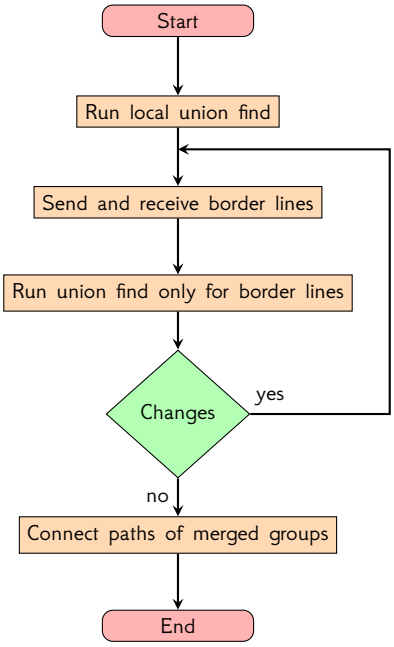
Σχήμα 4.5: Ένωση (union) ομάδων με χρήση της κλάσης QuickFindUnionFind

Η αποδοτική υλοποίηση της δομής μετανομάζει όλους τους κόμβους της ομάδας 1 σε κόμβους της ομάδας 2, λόγω του ότι η ομάδα 2 έχει περισσότερους κόμβους. Η παραλληλοποίηση δημιουργεί πρόβλημα στη λήψη της απόφασης για το πιο ξένο σύνολο-ομάδα έχει τους περισσότερους κόμβους. Το Σχήμα 4.6 παρουσιάζει την ιδανική υλοποίηση της δομής αυτής.



Σχήμα 4.6: Ιδανική ένωση (union) ομάδων με χρήση της κλάσης QuickFindUnionFind

Αρχικά, κάθε διεργασία εφαρμόζει συγχώνευση τοπικών ομάδων, έπειτα κάθε διεργασία στέλνει και λαμβάνει τις εξωτερικές γραμμές από τις γειτονικές της διαδικασίες και ελέγχει για τυχόν επιπλέον συγχωνεύσεις ομάδων. Η διαδικασία συνεχίζεται μέχρις ότου να μην υπάρχει καμία αλλαγή σε καμία διεργασία. Στο Σχήμα 4.7 παρουσιάζεται το διάγραμμα ροής του φίλτρου.



Σχήμα 4.7: Διάγραμμα ροής φίλτρου ένωσης ομάδων καμένων pixel

Κεφάλαιο 5

Αποτελέσματα

While programming, you can write fast programs, and you can write programs fast, but you can't write fast programs fast.

- Paul Lutus

Όπως αναφέρεται και πρηγουμένως, κύριος στόχος της παραλληλοποίησης είναι η μείωση του χρόνου εκτέλεσης της εφαρμογής. Αυτό επιτυγχάνεται σε ικανοποιητικό βαθμό.

Για την απόδειξη (validation) του παράλληλου αλγορίθμου χρησιμοποιήθηκε το εργαστήριο του 2ου ορόφου του Τμήματος Πληροφορίκης και Τηλεματικής του Χαροκόπειου Πανεπιστημίου. Σε αυτό το εργαστήριο υπήρχαν έως και 33 υπολογιστές ίδιου τύπου καθένας από τους οποίους είχε 4 πυρήνες, $4GB$ μνήμη RAM και επεξεργαστή Intel Core i5-2400 στα $3.10GHz$. Τα πειράματα έγιναν χρησιμοποιώντας λειτουργικό σύστημα Ubuntu. Λόγω των 4 πυρήνων, κάθε υπολογιστής μπορούσε να φιλοξενήσει 4 παράλληλες διεργασίες, χωρίς να επηρεάζει η μία την απόδοση της άλλης, θεωρητικά. Γι αυτόν το λόγο οι συνολικές διεργασίες που μπορούσαν να χρησιμοποιηθούν για να επιτευχθεί μέγιστη παραλληλοποίηση ήταν 132. Τα δεδομένα που συλλέχθηκαν αξιοποιούσαν είτε τους 30 υπολογιστές του εργαστηρίου είτε τους 33, ανάλογα με τη διαθεσιμότητά τους.

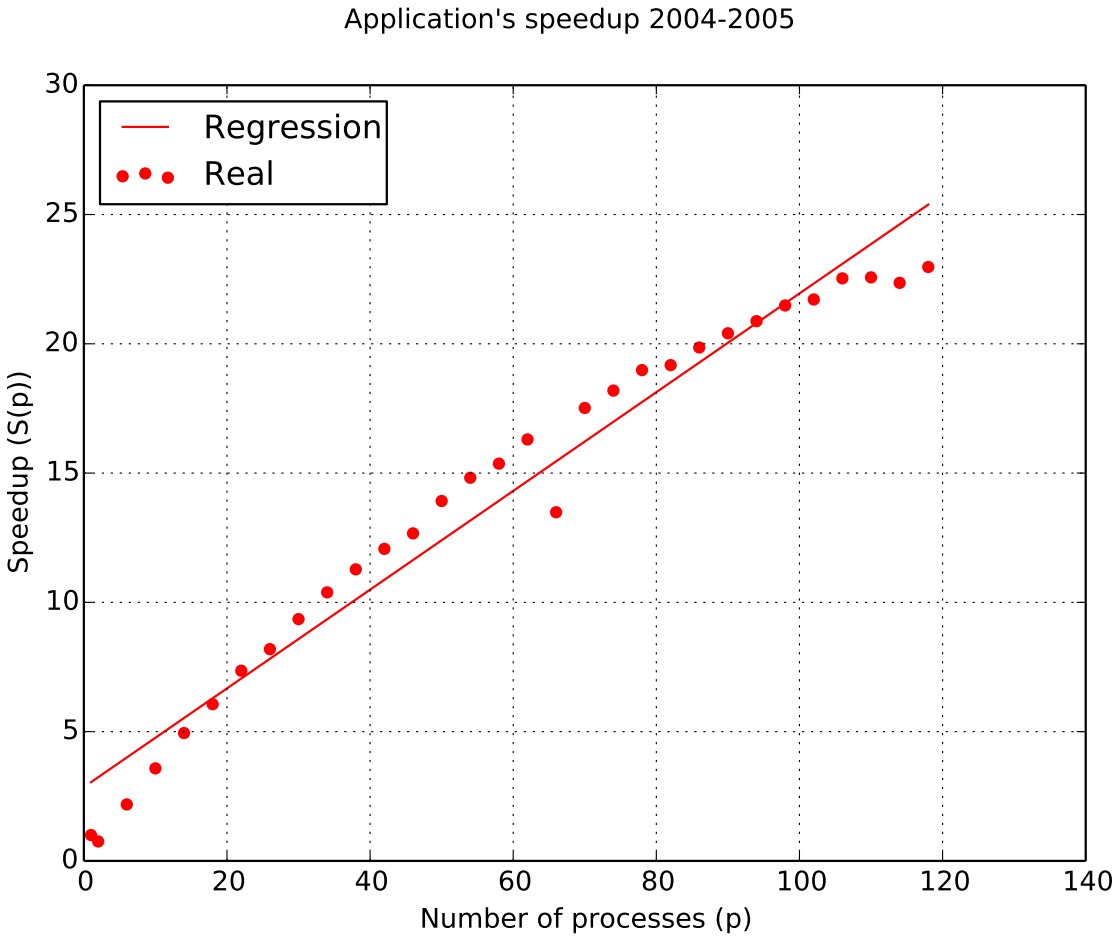
Η παραλληλοποίηση της εφαρμογής κατάφερε να αποδόσει ικανοποιητικά αποτελέσματα και να επιταχύνει τη συνολική διαδικασία. Η παρούσα πτυχιακή εργασία δεν μπόρεσε να μετρήσει τη μέγιστη επιτάχυνση και τον ελάχιστο χρόνο εκτέλεσης της εφαρμογής λόγω του περιορισμένου αριθμού υπολογιστών που υπήρχαν σε διαθεσιμότητα. Επομένως, οι ακόλουθες μετρήσεις δεν αναφέρονται στις μέγιστες αποδόσεις του συστήματος αλλά στις αποδόσεις που μετρήθηκαν. Οι μετρήσεις αναφέρονται στην εκτέλεση της παράλληλης εφαρμογής χρησιμοποιώντας δεδομένα από τα έτη 2004-2005 και 2011-2012. Για τα δεδομένα από τα έτη 2004-2005 χρησιμοποιήθηκαν 30 υπολογιστές, ενώ για τα έτη 2011-2012 χρησιμοποιήθηκαν 33 υπολογιστές.

Η κάθε Landsat εικόνα αποτελείται από ένα σύνολο εικόνων, οι οποίες είναι το στιγμιότυπο μίας περιοχής, στις διαφορετικές φασματικές μπάντες. Κάθε φασματική μπάντα χρησιμοποιείται για τη διεξαγωγή γνώσης, σε διάφορους τομείς, όπως περιγράφεται στον Πίνακα 2.4. Για τη χαρτογράφηση καμένων εκτάσεων χρησιμοποιούνται κυρίως οι μπάντες 3, 4, και 7. Επίσης χρησιμοποιούνται και εικόνες που περιγράφουν την ύπαρξη θορύβου από νερό ή σύννεφα. Έτσι, η κάθε Landsat εικόνα που επεξεργάζεται η εφαρμογή αποτελείται, κατά μέσο όρο, από 4 εικόνες. Η κάθε εικόνα έχει διαστάσεις $8781x9678$, είναι τύπου Tagged Image File Format (tiff) και έχει μέγεθος $82Mb$. Επίσης για κάθε εικόνα υπάρχουν και βοηθητικά δεδομένα τύπου High Dynamic Range (hdr) και αρχεία γλώσσας Extensible Markup Language (XML) όπου έχουν μέγεθος $328bytes$ και $1.6Kb$ αντίστοιχα. Έτσι, κάθε Landsat εικόνα έχει συνολικά μέγεθος, κατά μέσο όρο, $328.00752Mb$.

5.1 Επιτάχυνση εφαρμογής

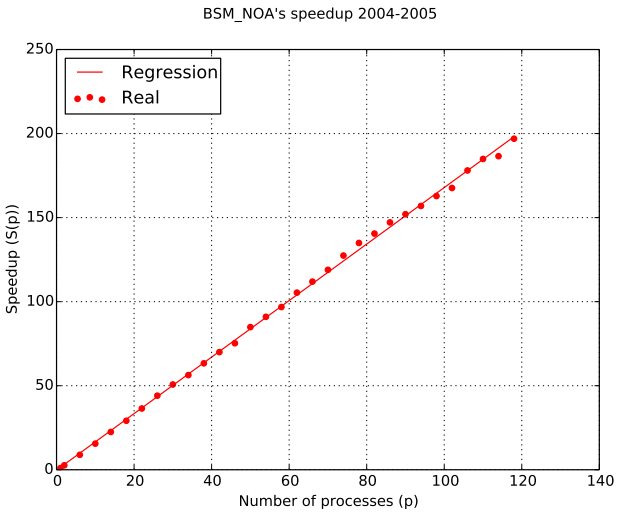
Τα Σχήματα που ακολουθούν παρουσιάζουν την επιτάχυνση των φίλτρων/αλγορίθμων ή της συνολικής εφαρμογής σε σχέση με τον αριθμό των διαδικασιών, επεξεργάζοντας δεδομένα από διάφορες χρονολογίες. Για τη μοντελοποίηση της σχέσης μεταξύ της επιτάχυνσης και των διεργασιών χρησιμοποιείται η προσέγγιση της γραμμικής παλινδρόμησης (linear regression) [Cohen, Cohen, West, *et al.*]

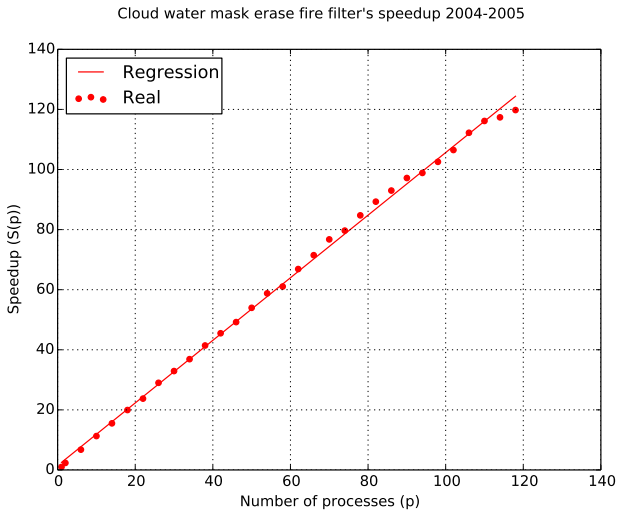
Το Σχήμα 5.1 παρουσιάζει τη συνολική επιτάχυνση της εφαρμογής σε σχέση με τον αριθμό των διαδικασιών, επεξεργάζοντας δεδομένα από τις χρονολογίες $2004 - 2005$.

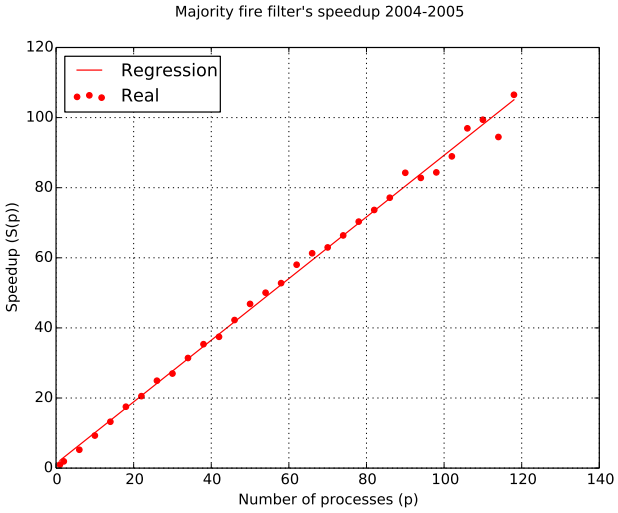


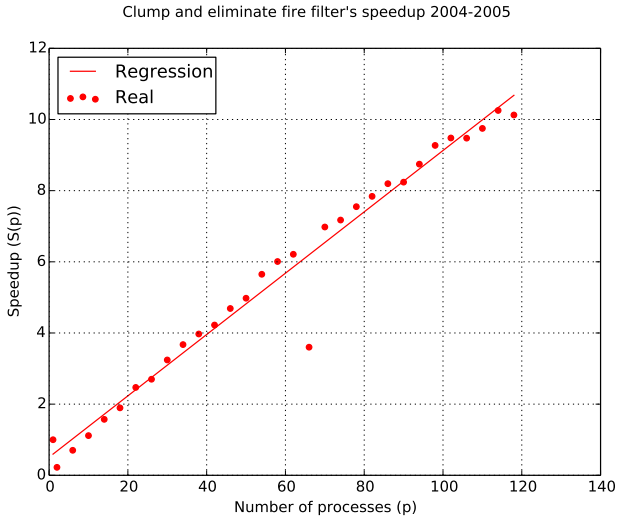
Σχήμα 5.1: Παρουσίαση της συνολικής επιτάχυνσης της εφαρμογής χρησιμοποιώντας δεδομένα από τα έτη 2004-2005

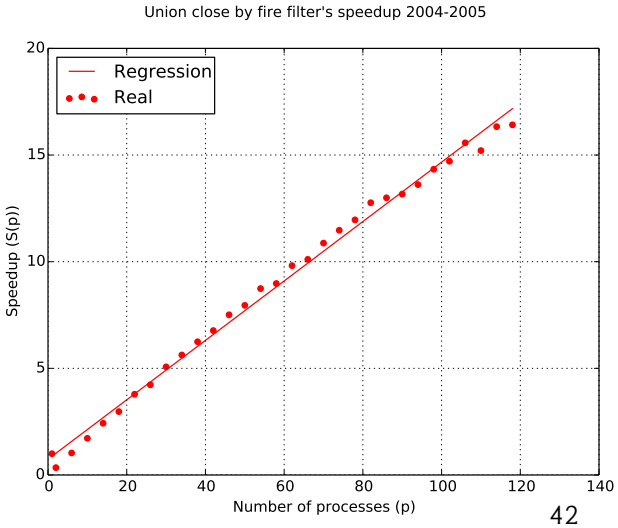
Οι γραφικές παραστάσεις στο Σχήμα 5.2 παρουσιάζουν την επιτάχυνση των φίλτρων σε σχέση με τον αριθμό των διαδικασιών, επεξεργάζοντας δεδομένα από τις χρονολογίες $2004-2005$.





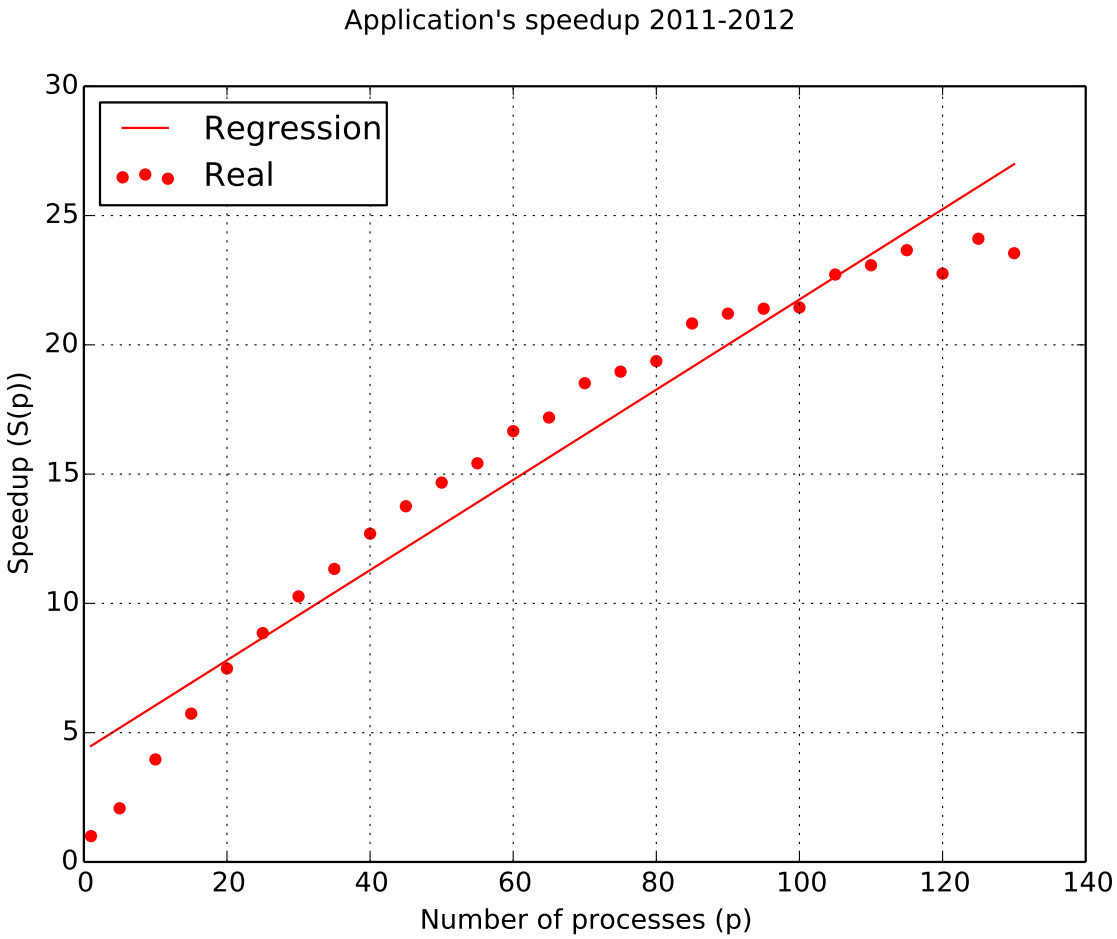






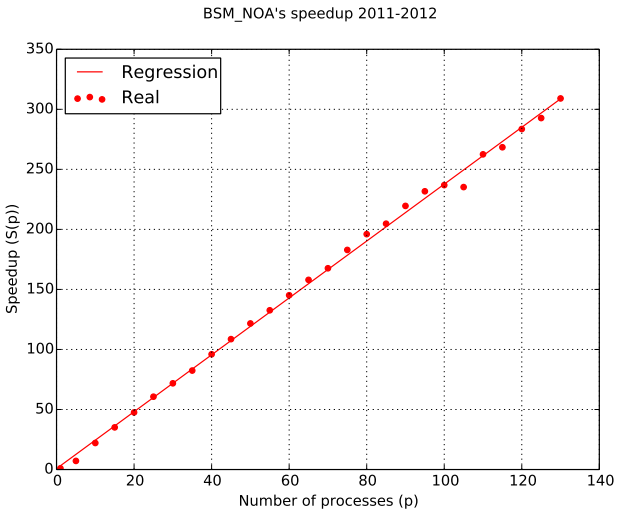
Σχήμα 5.2: Παρουσίαση της επιτάχυνσης των φίλτρων χρησιμοποιώντας δεδομένα από τις χρονολογίες 2004-2005

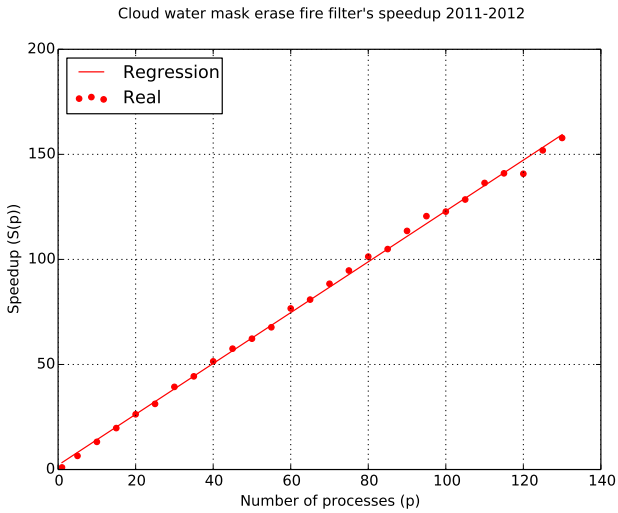
Το Σχήμα 5.3 παρουσιάζει τη συνολική επιτάχυνση της εφαρμογής σε σχέση με τον αριθμό των διαδικασιών, επεξεργάζοντας δεδομένα από τις χρονολογίες $2011 - 2012$.

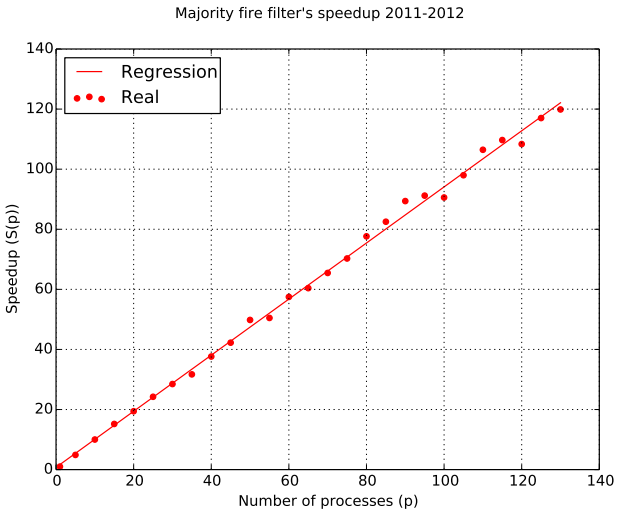


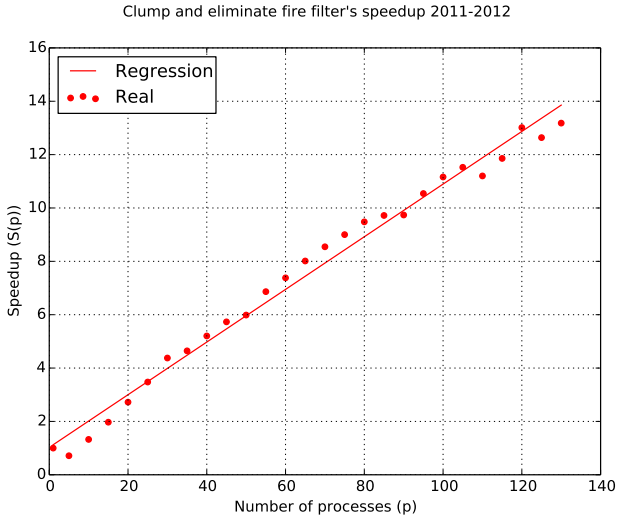
Σχήμα 5.3: Παρουσίαση της συνολικής επιτάχυνσης της εφαρμογής χρησιμοποιώντας δεδομένα από τα έτη 2011-2012

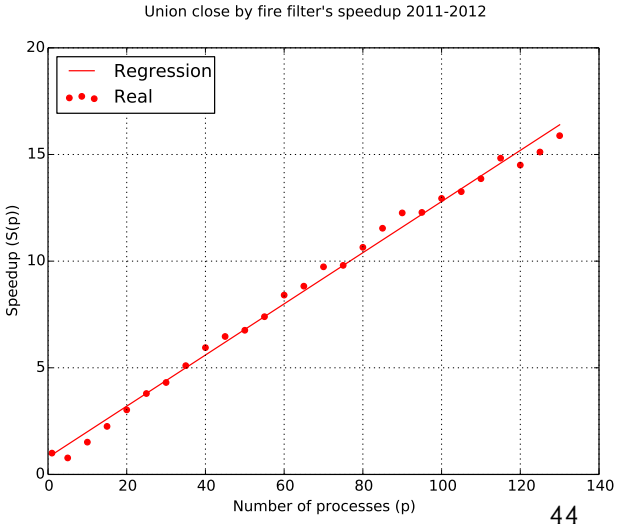
Οι γραφικές παραστάσεις στο Σχήμα 5.4 παρουσιάζουν την επιτάχυνση των φίλτρων σε σχέση με τον αριθμό των διαδικασιών, επεξεργάζοντας δεδομένα από τις χρονολογίες $2011-2012$.











Σχήμα 5.4: Παρουσίαση της επιτάχυνσης των φίλτρων χρησιμοποιώντας δεδομένα από τις χρονολογίες 2011-2012

Ανάλυση δεδομένων επιτάχυνσης

	120 διεργασίες σε 30 υπολογιστές	132 διεργασίες σε 33 υπολογιστές
BSM_NOA (Classifier)	200	300
Cloud/water mask filter	120	155
Majority filter	110	120
Clump and eliminate filter	11	14
Union close by filter	17	16
Συνολική	25	27

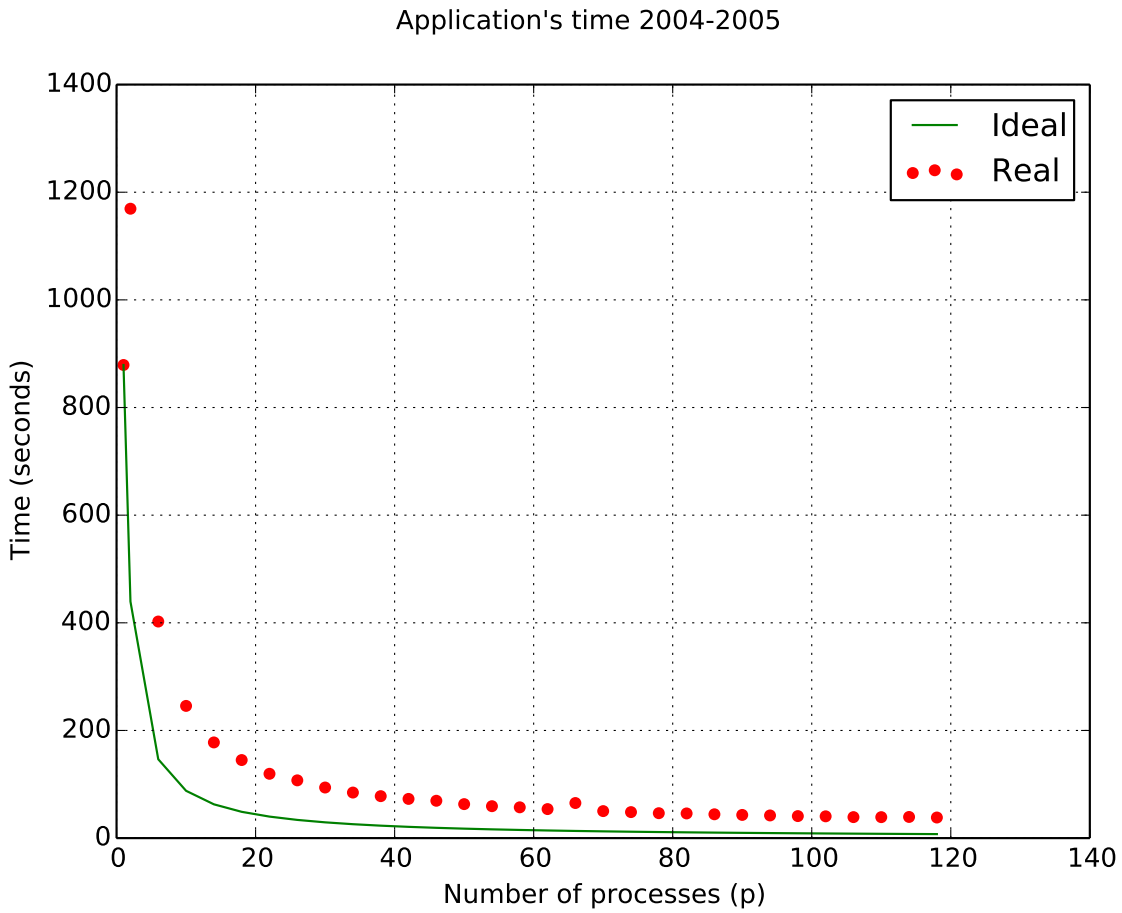
Πίνακας 5.1: Επιτάχυνση εργασιών σύμφωνα με τη συνάρτηση γραμμικής παλινδρόμησης με 120 διεργασίες με δεδομένα από τα έτη 2004-2005 και 132 διεργασίες με δεδομένα από τα έτη 2011-2012

Από τον Πίνακα 5.1 βγαίνει το συμπέρασμα πως η επιτάχυνση που έχει ο αλγόριθμος BSM_NOA και τα φίλτρα Cloud/water mask και majority είναι σχεδόν η ιδανική ή ακόμα και μεγαλύτερη από την ιδανική. Αντίθετα, η επιτάχυνση που έχουν τα φίλτρα clump and eliminate και union close by δεν είναι η επιθυμητή. Αυτό συμβαίνει λόγω αρκετής επικοινωνίας που χρειάζονται τα φίλτρα αυτά με σκοπό τη δημιουργία σωστού αποτελέσματος ή αποτελέσματος σχεδόν ίδιου με αυτό της σειριακής υλοποίησης.

5.2 Χρόνος εκτέλεσης

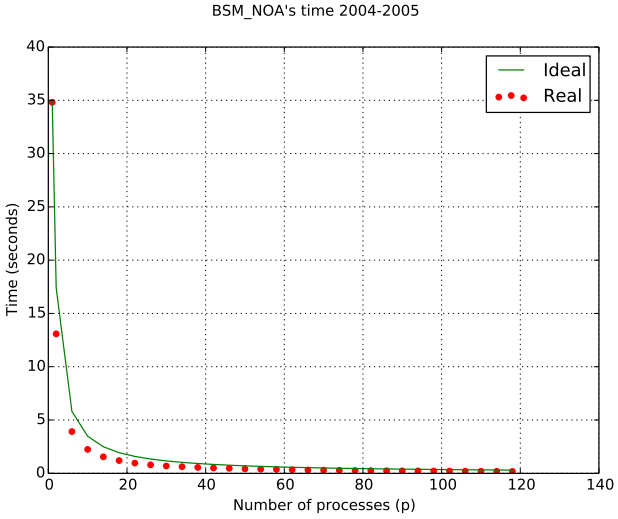
Τα Σχήματα που ακολουθούν παρουσιάζουν το χρόνο εκτέλεσης των φίλτρων/αλγορίθμων ή της συνολικής εφαρμογής σε σχέση με τον αριθμό των διαδικασιών, επεξεργάζοντας δεδομένα από διάφορες χρονολογίες. Επίσης παρουσιάζεται ο ιδανικός χρόνος εκτέλεσης, για λόγους σύγκρισης με τα πραγματικά δεδομένα. Από τα Σχήματα φαίνεται πως σε αρκετές περιπτώσεις ο χρόνος εκτέλεσης ενός φίλτρου είναι μικρότερος από τον αντίστοιχο ιδανικό χρόνο εκτέλεσης.

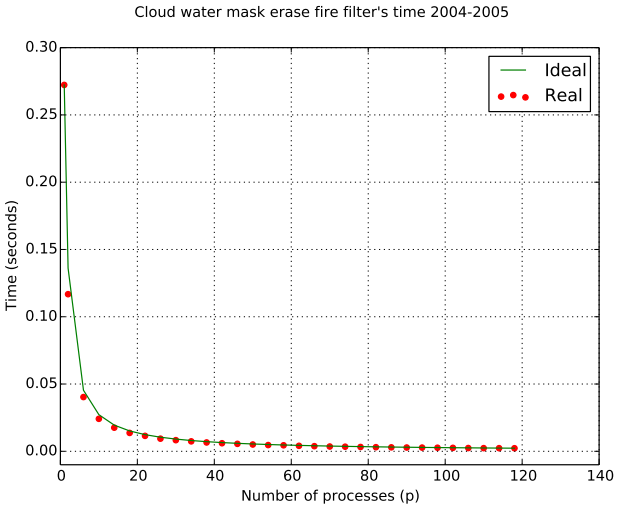
Το Σχήμα 5.5 παρουσιάζει το συνολικό χρόνο εκτέλεσης της εφαρμογής σε σχέση με τον αριθμό των διαδικασιών, επεξεργάζοντας δεδομένα από τις χρονολογίες $2004 - 2005$.

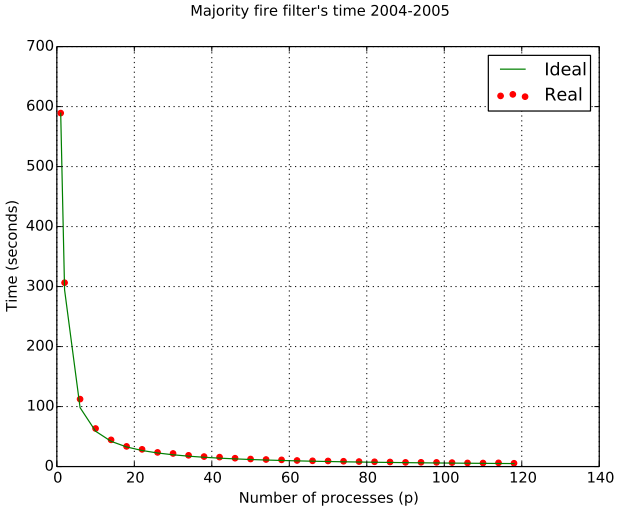


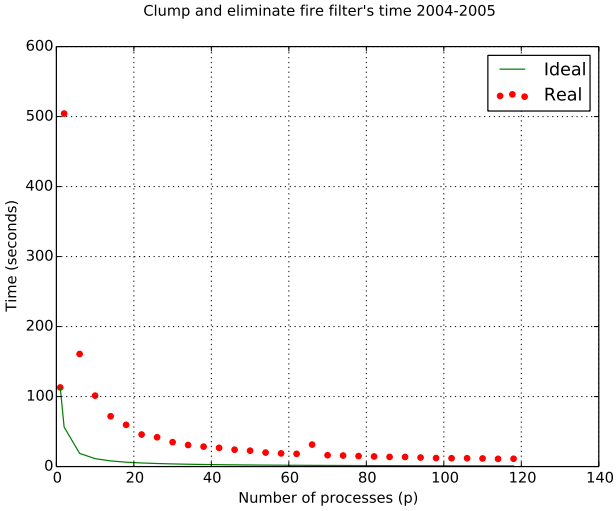
Σχήμα 5.5: Παρουσίαση του συνολικού χρόνου εκτέλεσης της εφαρμογής χρησιμοποιώντας δεδομένα από τις χρονολογίες 2004-2005

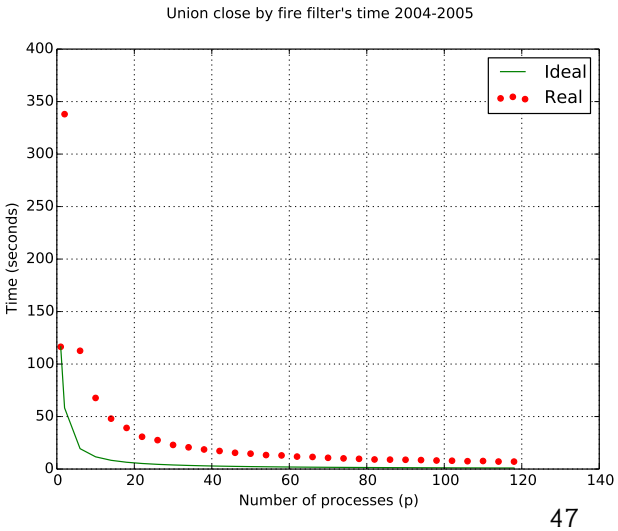
Οι γραφικές παραστάσεις στο Σχήμα 5.6 παρουσιάζουν το χρόνο εκτέλεσης των φίλτρων σε σχέση με τον αριθμό των διαδικασιών, επεξεργάζοντας δεδομένα από τις χρονολογίες $2004-2005$.





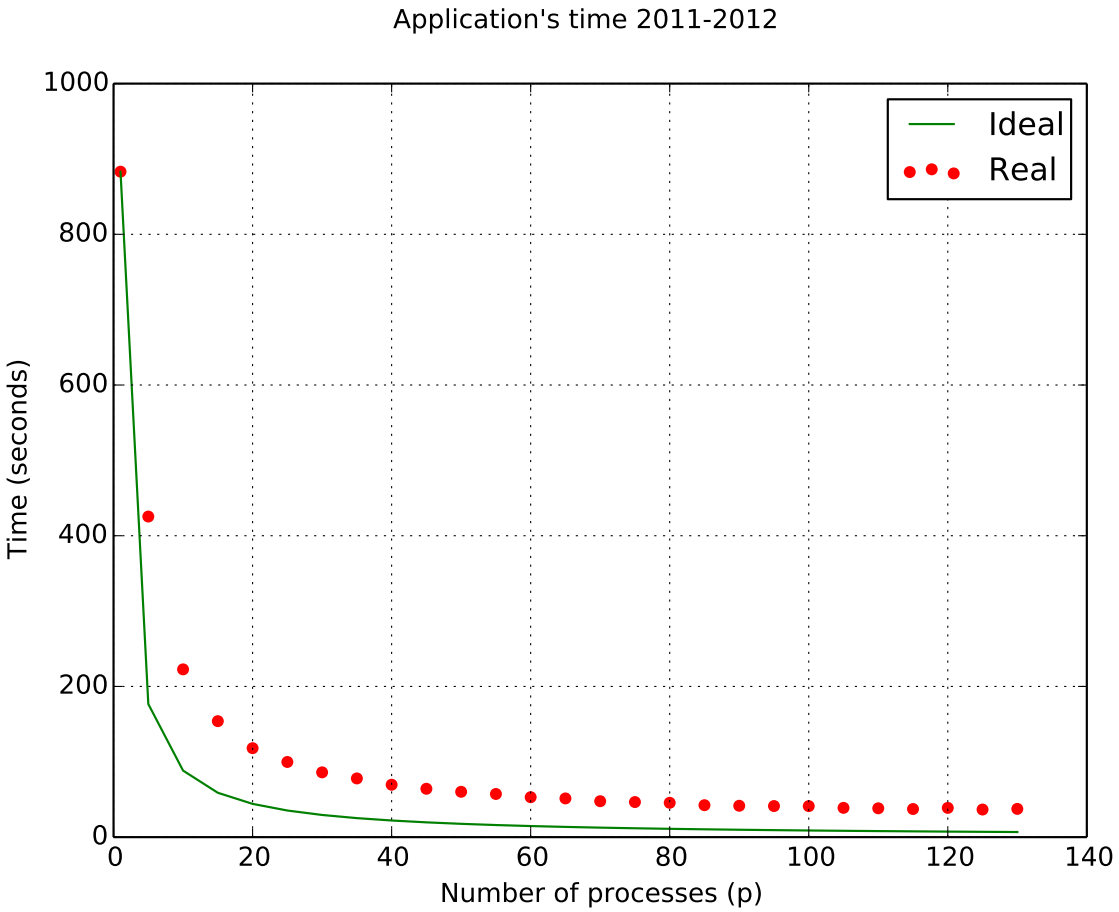






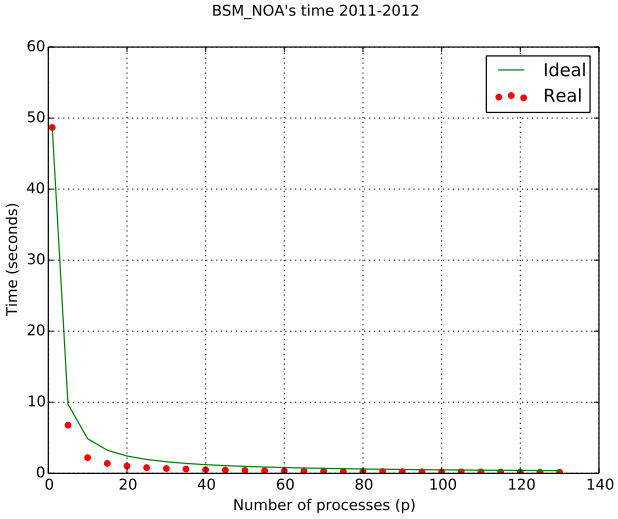
Σχήμα 5.6: Παρουσίαση του χρόνου εκτέλεσης των φίλτρων χρησιμοποιώντας δεδομένα από τις χρονολογίες 2004-2005

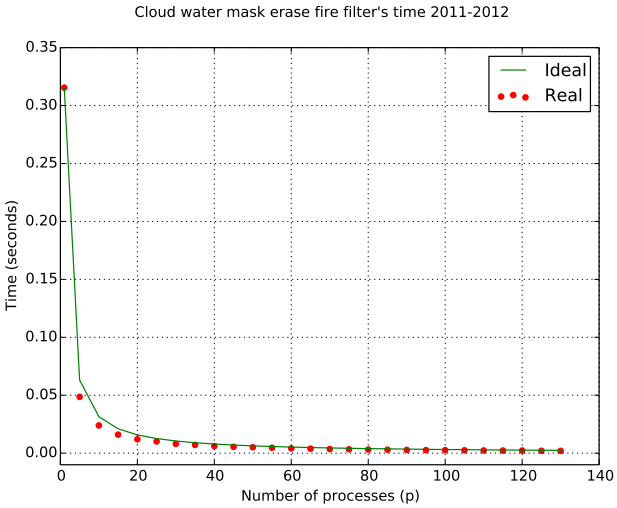
Το Σχήμα 5.7 παρουσιάζει το συνολικό χρόνο εκτέλεσης της εφαρμογής σε σχέση με τον αριθμό των διαδικασιών, επεξεργάζοντας δεδομένα από τις χρονολογίες $2011 - 2012$.

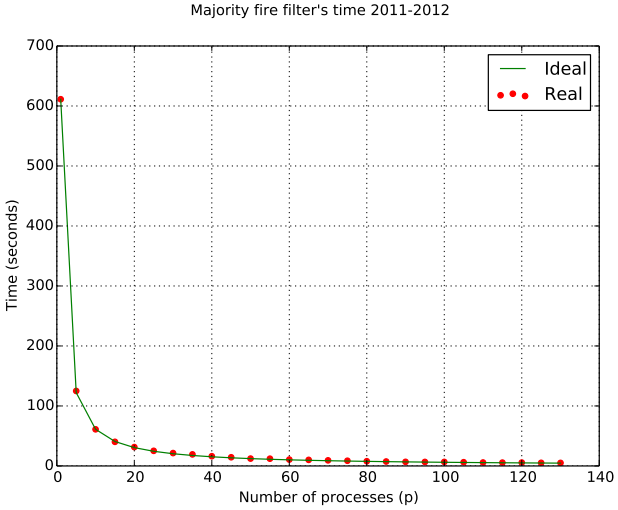


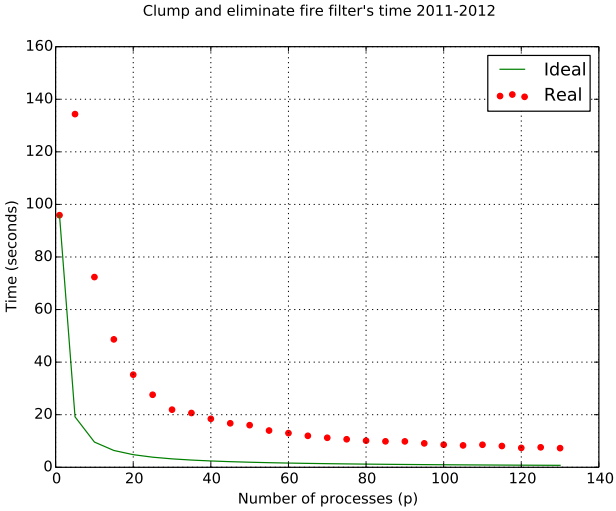
Σχήμα 5.7: Παρουσίαση του συνολικού χρόνου εκτέλεσης της εφαρμογής χρησιμοποιώντας δεδομένα από τις χρονολογίες 2011-2012

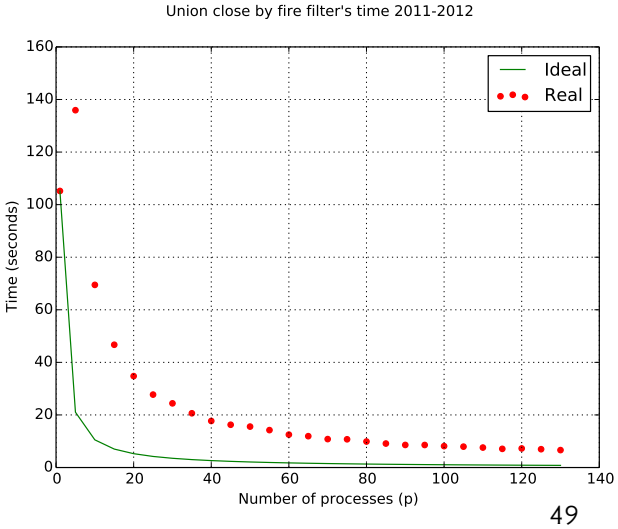
Οι γραφικές παραστάσεις στο Σχήμα 5.8 παρουσιάζουν το χρόνο εκτέλεσης των φίλτρων σε σχέση με τον αριθμό των διαδικασιών, επεξεργάζοντας δεδομένα από τις χρονολογίες $2011-2012$.











Σχήμα 5.8: Παρουσίαση του χρόνου εκτέλεσης των φίλτρων χρησιμοποιώντας δεδομένα από τις χρονολογίες 2011-2012

Ανάλυση δεδομένων χρόνου εκτέλεσης

	Σειριακός	Παράλληλος (120 διεργασίες σε 30 υπολογιστές)
BSM_NOA (Classifier)	34.811	0.176
Cloud/water mask filter	0.273	0.002
Majority filter	589.217	5.531
Clump and eliminate filter	113.031	11.162
Union close by filter	116.441	7.093
Συνολικός	878.999	37.265

Πίνακας 5.2: Σειριακός και παράλληλος χρόνος εκτέλεσης σε δευτερόλεπτα για τα δεδομένα από τα έτη 2004-2005

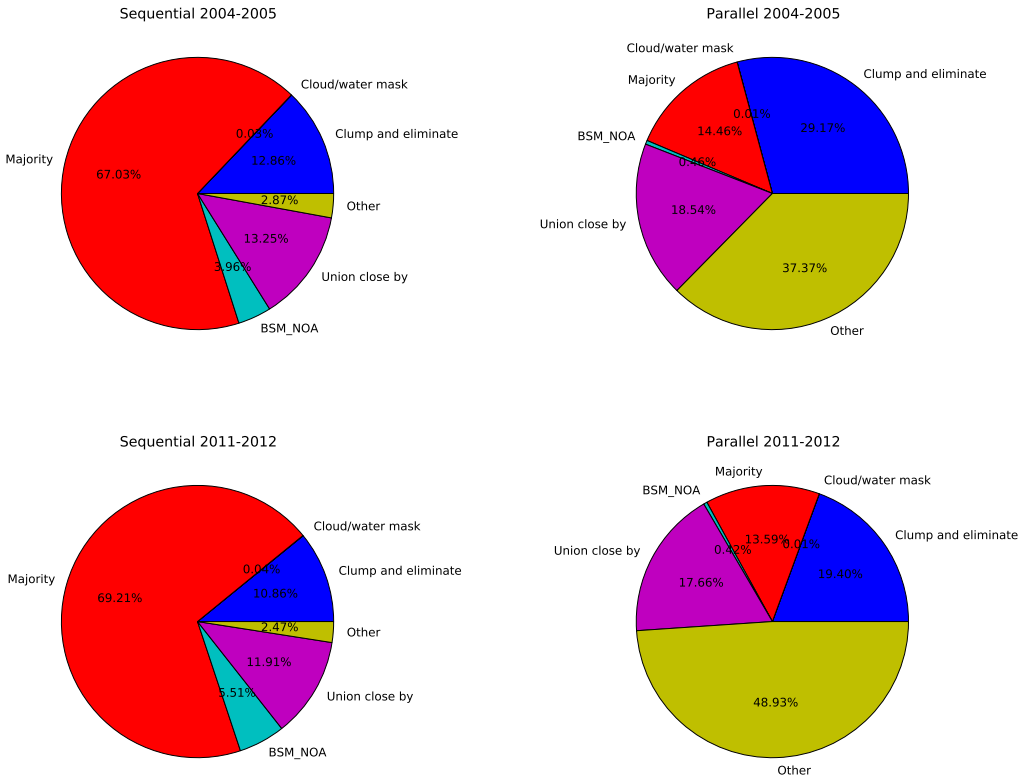
	Σειριακός	Παράλληλος (132 διεργασίες σε 33 υπολογιστές)
BSM_NOA (Classifier)	48.672	0.157
Cloud/water mask filter	0.315	0.001
Majority filter	611.241	5.098
Clump and eliminate filter	95.896	7.275
Union close by filter	105.203	6.624
Συνολικός	883.139	37.510

Πίνακας 5.3: Σειριακός και παράλληλος χρόνος εκτέλεσης σε δευτερόλεπτα για τα δεδομένα από τα έτη 2011-2012

Από τους Πίνακες 5.2 και 5.3 φαίνεται πως ο χρόνος εκτέλεσης κάθε εργασίας/φίλτρου όπως και ο συνολικός χρόνος της εφαρμογής, μειώνεται σε μεγάλο βαθμό, ανεξαρτήτως της επιτάχυνσης. Γενικά, η παραλληλοποίηση καταφέρνει να μειώσει το μέσο συνολικό χρόνο εκτέλεσης από τα 14.7 λεπτά στα 37.5 δευτερόλεπτα.

Το Σχήμα 5.9 παρουσιάζει το ποσοστό του συνολικού χρόνου που καταναλώνει κάθε εργασία/φίλτρο. Κατά τη σειριακή εκτέλεση της εφαρμογής ο περισσότερος χρόνος διατίθεται στο φίλτρο majority. Αντίθετα, κατά τη παράλληλη εκτέλεση της εφαρμογής ο περισσότερος χρόνος διατίθεται σε εργασίες όπως η ανάγνωση των δεδομένων εισόδου, η δημιουργία της εξόδου και η επικοινωνία μεταξύ των διεργασιών του συστήματος. Επομένως, για να βελτιωθεί η παράλληλη εφαρμογή είναι χρήσιμο να γίνει μείωση της επικοινωνίας και των δεδομένων που

στέλνονται μεταξύ των διεργασιών. Επίσης, η βελτιστοποίηση της υλοποίησης των παράλληλων φίλτρων clump and eliminate και union close by βελτιώνει την επιτάχυνση και μειώνει το συνολικό χρόνο εκτέλεσης.



Σχήμα 5.9: Ποσοστό του συνολικού χρόνου που καταναλώνει κάθε εργασία

5.3 Βελτιστοποίηση φίλτρου ομαδοποίησης καμένων pixel

Η μη αποδοτική υλοποίηση του φίλτρου ομαδοποίησης καμένων pixel δημιουργεί την ανάγκη για εφαρμογή του αλγόριθμου BFS με αποδοτικότερο τρόπο. Η νεα παράλληλη υλοποίηση που μπορεί να χρησιμοποιηθεί, είναι εκείνη που προτείνεται από τους [Buluç and Madduri]. Η υλοποίηση συνδυάζει τις αρχιτεκτονικές κατανεμημένης (distributed) και κοινής (shared) μνήμης, για την επίλυση του προβλήματος διάσχισης μεγάλων γραφημάτων. Για τον εύκολο διαμοιρασμό της εργασίας κατά την παραλληλοποίηση, οι [Buluç and Madduri] υλοποιούν με διαφορετικό τρόπο το σειριακό αλγόριθμο BFS, και αντί της δομής FIFO που χρησιμοποιείται συνήθως, χρησιμοποιούνται οι δύο στοίβες (FS, NS), όπως παρουσιάζεται στον Αλγόριθμο 1.

Έστω γράφημα $G(V, E)$ όπου το πλήθος των κόμβων ισούται με n και το πλήθος των ακμών

ισούται με m. Η απόσταση μεταξύ δύο κόμβων s και t του γραφήματος ισούται με $d(s,t)$. Κάθε ακμή του γραφήματος έχει βάρος ίσο με τη μονάδα. Σύμφωνα με τον αλγόριθμο BFS, όλοι οι κόμβοι σε απόσταση "k" από έναν αρχικό κόμβο s πρέπει να έχουν επισκεφθεί, προτού ο αλγόριθμος επισκεφθεί τους κόμβους σε απόσταση "$k+1$" . Έτσι, η στοίβα FS περιέχει όλους τους κόμβους που βρίσκονται στο επίπεδο "k" και αναμένεται να επισκεφθούν σύντομα, ενώ η στοίβα NS περιέχει κόμβους που πρόκεται να επισκεφθούν στο μέλλον.

Algorithm 1 Serial BFS algorithm. Πηγή: [Buluç and Madduri]

Input: $G(V,E)$, source vertex s.
Output: $d[1..n]$, where $d[v]$ gives the length of the shortest path from s to $v \in V$.

```
1: for all v ∈ V do
2:     d[v] ← ∞
3: end for
4: d[s] ← 0, level ← 1, FS ← φ, NS ← φ
5: push s → FS
6: while FS ≠ φ do
7:     for each u in FS do
8:         for each neighbor v of u do
9:             if d[v] = ∞ then
10:                push v → NS
11:                d[v] ← level
12:            end if
13:        end for
14:    end for
15:    FS ← NS, NS ← φ, level ← level + 1
16: end while
```

Οι [Buluç and Madduri] παρουσιάζουν δύο τρόπους για διαμοιρασμό (partitioning) των κόμβων. Ο πρώτος εφαρμόζει $1D$ διαμοιρασμό και ο δεύτερος εφαρμόζει $2D$. Η παρούσα πτυχιακή διευκολύνεται με το διαμοιρασμό $1D$, ο οποίος περιγράφεται παρακάτω. Στον $1D$ διαμοιρασμό, για σύστημα p διεργασιών, σε κάθε διεργασία ορίζονται n/p κόμβοι, καθώς και οι εξερχόμενες ακμές τους.

Για τη βελτίωση της απόδοσης, κάθε διεργασία μοιράζει το γράφημά της στα νήματα ή στους πυρήνες που διαθέτει. Ο πίνακας αποστάσεων d του συνολικού γραφήματος, διαμοιράζεται σε όλες τις διεργασίες. Κάθε διεργασία ανανεώνει την κατάσταση μόνο των κόμβων της. Συνεπώς, μόνο η ιδιοκτήτρια διεργασία γνωρίζει αν κάποιος κόμβος της είναι επεξεργασμένος. Για την επίλυση του προβλήματος, κάθε διεργασία που χρειάζεται να διαχειριστεί κόμβο άλλης διεργασίας, στέλνει τον κόμβο στην υπεύθυνη διεργασία, κατά το στάδιο της επικοινωνίας μεταξύ υπολογιστών. Στον Αλγόριθμο 2 παρουσιάζεται ο ψευδοκώδικας του παράλληλου BFS αλγόριθμου.

Algorithm 2 Hybrid parallel BFS with vertex partitioning. Πηγή: [Buluç and Madduri]

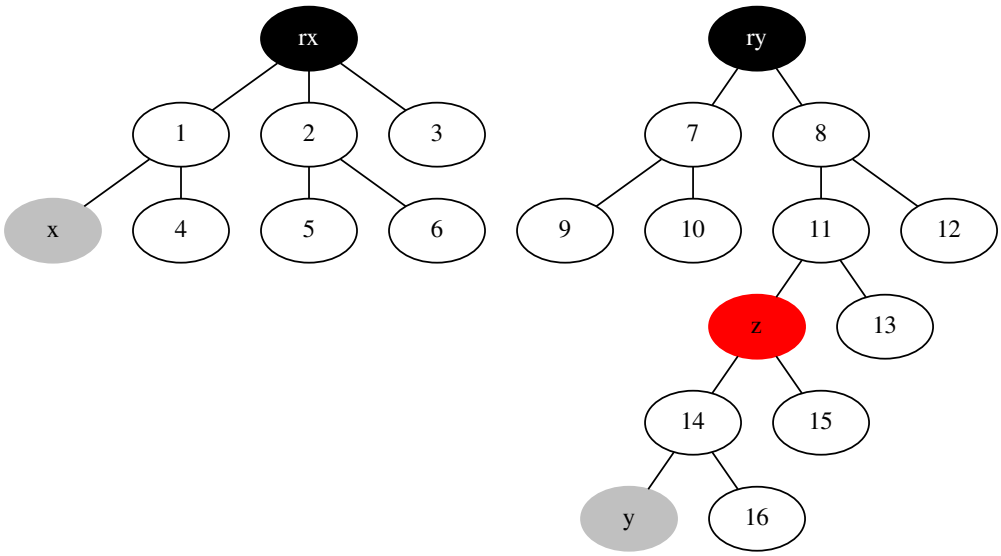
Input: $G(V, E)$, source vertex s.
Output: $d[1..n]$, where $d[v]$ gives the length of the shortest path from s to $v \in V$.

1: **for all** $v \in V$ **do**
2: $d[v] \leftarrow \infty$
3: **end for**
4: $level \leftarrow 1$, $FS \leftarrow \phi$, $NS \leftarrow \phi$
5: $op_s \leftarrow find\_owner(s)$
6: **if** $op_s = rank$ **then**
7: push $s \rightarrow FS$
8: $d[s] \leftarrow 0$
9: **end if**
10: **for** $0 \leq j < p$ **do**
11: $SendBuf_j \leftarrow \phi$ ▷ p shared message buffers
12: $RecvBuf_j \leftarrow \phi$ ▷ for MPI communication
13: $tBuf_{ij} \leftarrow \phi$ ▷ thread-local stack for thread i
14: **end for**
15: **while** $FS \neq \phi$ **do**
16: **for** each u in FS in parallel **do**
17: **for** each neighbor v of u **do**
18: $p_v \leftarrow find\_owner(v)$
19: push $v \rightarrow tBuf_{ip_v}$
20: **end for**
21: Thread Barrier
22: **for** $0 \leq j < p$ **do**
23: Merge thread-local $tBuf_{ij}$'s in parallel, form $SendBuf_j$
24: **end for**
25: Thread Barrier
26: All-to-all collective step with the master thread: Send data in $SendBuf$, aggregate newly-visited vertices into $RecvBuf$
27: Thread Barrier
28: **end for**
29: **for** each u in $RecvBuf$ in parallel **do**
30: **if** $d[u] = \infty$ **then**
31: $d[u] \leftarrow level$
32: push $u \rightarrow NS_i$
33: **end if**
34: **end for**
35: Thread Barrier
36: $FS \leftarrow \bigcup NS_i$ ▷ thread-parallel
37: Thread Barrier
38: **end while**

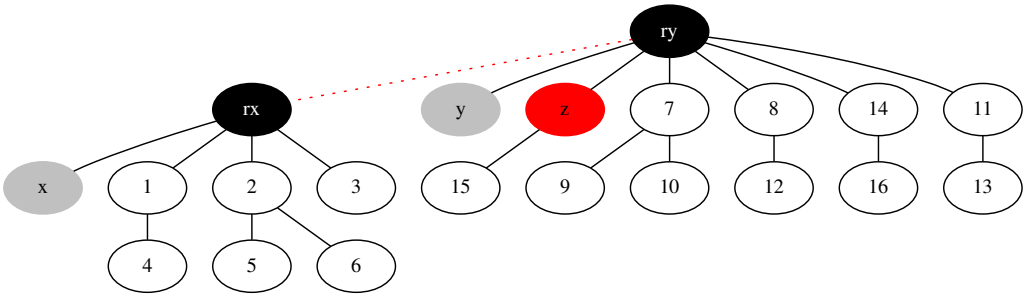
5.4 Βελτιστοποίηση φίλτρου ένωσης ομάδων καμένων pixel

Η μη αποδοτική υλοποίηση του φίλτρου ένωσης ομάδων καμένων pixel δημιουργεί την ανάγκη για εφαρμογή της δομής union-find με αποδοτικότερο τρόπο. Η νεα παράλληλη υλοποίηση που μπορεί να χρησιμοποιηθεί, είναι εκείνη που προτείνεται από τους [Manne and Patwary]. Η υλοποίηση συνδυάζει τις τεχνικές union-by-rank και path compression. Έτσι, η πολυπλοκότητα των λειτουργιών της δομής ανάγεται σε $O(na(m,n))$, όπου a είναι η συνάρτηση Ackerman [Cormen, Leiserson, Rivest, *et al.*] Η υλοποίηση βασίζεται στο γεγονός ότι για την ένωση δύο κόμβων x και y δεν είναι αναγκαία η εύρεση των ριζικών (root) κόμβων των δέντρων τους, έτσι ώστε να συγχωνευθούν, αλλά μπορούν να συγχωνευθούν με εξίσου σωστό τρόπο έχοντας ως κριτήριο τους αριθμούς rank των κόμβων των δέντρων.

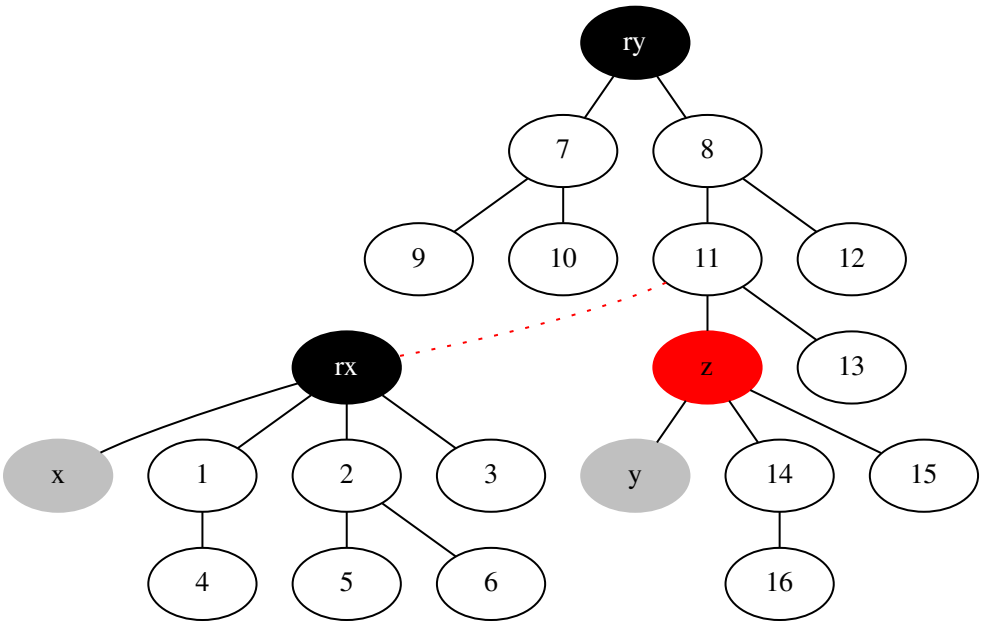
Έστω ότι ο αριθμός rank ενός κόμβου z συμβολίζεται ως $rank(z)$. Έστω δύο κόμβοι x και y οι οποίοι ανήκουν σε διαφορετικά σύνολα με ριζικούς πατέρες r_x και r_y αντίστοιχα, όπου $rank(r_x) < rank(r_y)$. Αν βρεθεί ο r_x πριν τον r_y τότε υπάρχει η δυνατότητα τερματισμού της αναζήτησης του r_y κόμβου, με την προϋπόθεση ότι η αναζήτηση στο δέντρο του y φτάνει σε έναν κόμβο z όπου $rank(z) = rank(r_y)$. Με αυτόν τον τρόπο, η αναζήτηση δεν συνεχίζεται μέχρι τον ριζικό πατέρα r_y και οι αριθμοί rank δεν αλλάζουν. Οι [Manne and Patwary] ονομάζουν την παραλλαγή της λειτουργίας Find ως zigzag Find. Το Σχήμα 5.11 παρουσιάζει το δέντρο που παράγεται κατά την εκτέλεση της λειτουργίας union με χρήση της κλασικής δομής union-find, ενώ το σχήμα 5.12 παρουσιάζει το δέντρο που παράγεται κατά την εκτέλεση της λειτουργίας union με χρήση της δομής union-find που χρησιμοποιεί τη zigzag Find λειτουργία. Τα σύνολα που δίνονται ως είσοδος και στις δύο περιπτώσεις αντιπροσωπεύονται από τα δέντρα του Σχήματος 5.10.



Σχήμα 5.10: Σύνολα που δίνονται ως είσοδος στη κλασική union-find δομή καθώς και στη zigzag Find union-find δομή



Σχήμα 5.11: Έξοδος κλασικής union-find δομής



Σχήμα 5.12: Έξοδος zigzag Find union-find δομής

Ο παράλληλος αλγόριθμος χωρίζεται σε δύο στάδια. Στο πρώτο στάδιο κάθε διεργασία επεξεργάζεται το υπογράφημα (subgraph) για το οποίο είναι υπεύθυνη, υπολογίζει ένα επικαλυπτικό δάσος (spanning forest) όπου και συλλέγονται οι ακμές που το ορίζουν. Έπειτα οι διεργασίες επικοινωνούν μεταξύ τους, για να οδηγηθούν στο τελικό αποτέλεσμα. Οι ακμές κάθε διεργασίας χρησιμοποιούνται για να δημιουργήσουν ένα επικαλυπτικό δάσος, το οποίο αντιπροσωπεύει τα σύνολα της δομής union-find.

Κεφάλαιο 6

Σύνοψη

Programming is one of the most difficult branches of applied mathematics; the poorer mathematicians had better remain pure mathematicians

- Edsger W. Dijkstra

Controlling complexity is the essence of computer programming

- Brian Kernighan

Η χαρτογράφηση καμένων εκτάσεων είναι σημαντική για τη φύλαξη της χλωρίδας και της φύσης του πλανήτη. Οι δορυφορικές εικόνες που επεξεργάζονται με σκοπό τη διεξαγωγή γνώσης για καμένες εκτάσεις είναι πολλές και μεγάλου μεγέθους, λόγω της υψηλής ανάλυσης που χρειάζεται μία τέτοια διαδικασία. Για τον λόγο αυτό η διαδικασία της χαρτογράφησης του Εθνικού Αστεροσκοπείου Αθηνών είχε ως μέσο χρόνο εκτέλεσης 14.7 λεπτά.

Η παρούσα πτυχιακή εργασία παραλληλοποίησε τη διαδικασία αυτή, με αποτέλεσμα να εκτελείται σε λιγότερο από 1 λεπτό, χρησιμοποιώντας 33 υπολογιστές ενός απλού εργαστηρίου, έχοντας λίγη έως καθόλου μείωση της ποιότητας της εξόδου της σε σχέση με τη σειριακή υλοποίηση της εφαρμογής.

Σύμφωνα με τα δεδομένα που παρουσιάζονται στο Κεφάλαιο 5 η παράλληλη εφαρμογή μπορεί να βελτιστοποιηθεί περαιτέρω με διάφορους τρόπους, όπως:

- Βελτιστοποίηση των παράλληλων φίλτρων στα οποία δαπανάται ο περισσότερος χρόνος εκτέλεσης
- Εύρεση τρόπων μείωσης επικοινωνίας και αποστολής δεδομένων μεταξύ των υπολογιστών του συστήματος

- Υλοποίηση των κρίσιμων σημείων του προγράμματος σε μία γλώσσα χαμηλότερου επιπέδου σε σχέση με τη γλώσσα Python, όπως η C ή C++
- Υλοποίηση των κρίσιμων σημείων του προγράμματος στην υβριδική γλώσσα προγραμματισμού Cython
- Υλοποίηση του αλγόριθμου BFS που χρησιμοποιείται στο φίλτρο ομαδοποίησης καμένων pixel με αποδοτικότερο τρόπο, καθώς και της δομής union-find που χρησιμοποιείται στο φίλτρο ένωσης ομάδων καμένων pixel
- Συνδυασμός παράλληλου προγραμματισμού κατανεμημένης (distributed) και κοινής (shared) μνήμης

Βιβλιογραφία

[1] Nasa, *Landsat program website*, 2016. [Online]. Available: http://landsat.gsfc.nasa.gov.

[2] ——, *Fire information for resource management system (firms)*. [Online]. Available: https://earthdata.nasa.gov/earth-observation-data/near-real-time/firms.

[3] Esa, *Copernicus program website*, 2016. [Online]. Available: http://www.esa.int/Our_Activities/Observing_the_Earth/Copernicus.

[4] ——, *Sentintels missions website*, 2016. [Online]. Available: http://www.esa.int/Our_Activities/Observing_the_Earth/Copernicus/Overview4.

[5] Noa, *Teleios program website*, 2016. [Online]. Available: http://www.earthobservatory.eu.

[6] ——, *Firehub program website*, 2016. [Online]. Available: http://ocean.space.noa.gr/FireHub.

[7] Nasa, *Moderate resolution imaging spectroradiometer (modis) website*, 2016. [Online]. Available: http://modis.gsfc.nasa.gov.

[8] L. C. Center, *Message passing interface (mpi) tutorial website*, 2016. [Online]. Available: https://computing.llnl.gov/tutorials/mpi.

[9] R. A. Schowengerdt, *Remote sensing: models and methods for image processing*, 3rd. Academic Press, 2007.

[10] Μ. Πλένιου, "Πολλαπλής κλίμακας πολυφασματική αξιολόγηση και χαρτογράφηση καμένων εκτάσεων με τη χρήση δορυφορικών δεδομένων," PhD thesis, Τμήμα Διαχείρισης Περιβάλλοντος και Φυσικών Πόρων, Πανεπιστήμιο Πατρών, 2013.

[11] *Landsat program wikipedia*, 2016. [Online]. Available: http://en.wikipedia.org/wiki/Landsat_program.

[12] *Enhanced thematic mapper*, 2016. [Online]. Available: http://landsat.gsfc.nasa.gov/?p=3225.

[13] *Landsat missions*, 2016. [Online]. Available: http://landsat.usgs.gov/best_spectral_bands_to_use.php.

[14] C. Kontoes, H. Poilvé, G. Florsch, I. Keramitsoglou, and S. Paralikidis, "A comparative analysis of a fixed thresholding vs. a classification tree approach for operational burn scar detection and mapping.," *Int. J. Applied Earth Observation and Geoinformation*, vol. 11, 2009.

[15] *Albedo wikipedia*, 2016. [Online]. Available: http://en.wikipedia.org/wiki/Albedo.

[16] *Normalized difference vegetation index website*, 2016. [Online]. Available: http://earthobservatory.nasa.gov/Features/MeasuringVegetation/measuring_vegetation_2.php.

[17] *Normalized burn ratio*, 2016. [Online]. Available: http://wiki.landscapetoolbox.org/doku.php/remote_sensing_methods:normalized_burn_ratio.

[18] L. Giglio, T. Loboda, D. P. Roy, B. Quayle, and C. O. Justice, "An active-fire based burned area mapping algorithm for the {modis} sensor," *Remote Sensing of Environment*, vol. 113, no. 2, 2009, ISSN: 0034-4257. DOI: http://dx.doi.org/10.1016/j.rse.2008.10.006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0034425708003180.

[19] P. Pacheco, *An Introduction to Parallel Programming*. Elsevier Inc., 2011.

[20] *Openmp website*, 2016. [Online]. Available: http://openmp.org/wp.

[21] Σ. Παπαδάκης and Κ. Διαμαντάρας, *Προγραμματισμός και Αρχιτεκτονική Συστημάτων Παράλληλης Επεξεργασίας*. Εκδόσεις Κλειδάριθμος, 2012.

[22] B. Wilkinson and M. Allen, *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*, 2nd. Pearson, 2005.

[23] A. Dimopoulos, *Lectures of parallel computers and algorithms*.

[24] C. Kontoes, I. Papoutsis, T. Herekakis, D. Michail, and E. Ieronymidi, *National scale burn scar mapping in greece*, 2013. [Online]. Available: http://www.earthobservatory.eu/publications/EARSEL2013.pdf.

[25] *Arop*, 2016. [Online]. Available: http://www.ars.usda.gov/services/software/download.htm?softwareid=364.

[26] E. E. Agency, *Corine land cover technical guide*. [Online]. Available: http://www.pedz.uni-mannheim.de/daten/edz-bn/eua/00/tech40add.pdf.

[27] *Gdal - geospatial data abstraction library*, 2016. [Online]. Available: http://www.gdal.org/.

[28] *Numpy*, 2016. [Online]. Available: http://www.numpy.org/.

[29] S. Skiena, *The Algorithm Design Manual*. Springer, 2008.

[30] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*, 3rd. The MIT Press, 2009.

[31] *Qgis - a free and open source geographic information system*, 2016. [Online]. Available: http://www.qgis.org/en/site/.

[32] *Opencv*, 2016. [Online]. Available: http://opencv.org/.

[33] *Scipy*, 2016. [Online]. Available: https://www.scipy.org/.

[34] L. Dalcín, R. Paz, M. Storti, and J. D'Elía, "{mpi} for python: performance improvements and mpi-2 extensions," *Journal of Parallel and Distributed Computing*, 2008.

[35] W. Lin, "A comparison of existing python modules of mpi," Master's thesis, Department of Mathematics, University of Oslo, Oslo, 2010.

[36] R. Sedgewick and K. Wayne, *Algorithms*, 4rd. Addison-Wesley, 2011.

[37] J. Cohen, P. Cohen, S. G. West, and L. S. Aiken, *Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences*, 2rd. Lawrence Erlbaum Associates, 2003.

[38] A. Buluç and K. Madduri, *Parallel breadth-first search on distributed memory systems*. [Online]. Available: http://gauss.cs.ucsb.edu/~aydin/sc11_bfs.pdf.

[39] F. Manne and M. M. A. Patwary, *A scalable parallel union-find algorithm for distributed memory computers*. [Online]. Available: http://users.eecs.northwestern.edu/~mpatwary/ppam09.pdf.