Πανεπιστήμιο Θεσσαλίας

Διπλωματική Εργασία

Μελέτη και υλοποίηση αλγορίθμων για την αριθμητική επίλυση γραμμικών συστημάτων

Study and implementation of linear systems solvers

Αδαμαντίδη Αικατερίνη

ΑΕΜ: 1146

Επιβλέπουσα Καθηγήτρια:

Τσομπανοπούλου Παναγιώτα

Βόλος 2015

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τους επιβλέποντες καθηγητές της διπλωματικής μου εργασίας κα. Τσομπανοπούλου Παναγίωτα και κ. Μποζάνη Παναγιώτη, οι οποίοι υπήρξαν καθοδηγητές μου καθ' όλη την διάρκεια της εκπόνησης διπλωματικής εργασίας χωρίς τους οποίους δεν θα μπορούσε να έχει φτάσει στο πέρας της . Η άψογη συνεργασία και επικοινωνία καθώς και η καθοδήγηση τους αποτελούν στοιχεία τα οποία συντέλεσαν σημαντικά στην ολοκλήρωση αυτής της μελέτης.

Επίσης, ευχαριστώ τους φίλους που μου συμπαραστάθηκαν και ήταν συνοδοιπόροι σε αυτό το ταξίδι γνώσης τα τελευταία έξι χρόνια.

Τέλος, ένα μεγάλο ευχαριστώ στην οικογένεια μου η οποία με στήριζε και μου συμπαραστεκόταν με κάθε δυνατό τρόπο καθ' όλη την διάρκεια της ακαδημαϊκής μου πορείας.

Περιεχόμενα

Περίληψη

Η παρακάτω διπλωματική εργασία έχει σαν αντικείμενο μελέτης τα γραμμικά συστήματα και τους διαφόρους τρόπους επίλυσης τους. Πιο αναλυτικά, σε αυτήν την εργασία θα ασχοληθούμε με τους τρόπους επίλυσης και απλοποίησης αυτών των συστημάτων δίνοντας έμφαση στην παραγοντοποίηση Cholesky, και την μέθοδο επίλυσης γραμμικών συστημάτων που προκύπτουν από διαφορικές εξισώσεις conjugate gradient. Αξίζει να σημειώσουμε ότι στην συγκεκριμένη εργασία επικεντρωνόμαστε κυρίως σε πίνακες οι οποίοι χαρακτηρίζονται από συγκεκριμένες ιδιότητες, όπως η συμμετρία, και το αν είναι θετικά ορισμένοι. Η υλοποίηση των μεθόδων έγινε σε python και περιλαμβάνει σειριακή και παράλληλη υλοποίηση μεθόδων προκειμένου να επιλυθούν τα συστήματα. Απώτερος σκοπός της διπλωματικής αυτής εργασίας είναι η εξαγωγή συμπερασμάτων σχετικά με τον χρόνο επίλυσης γραμμικών συστημάτων όταν οι μέθοδοι αυτοί υλοποιούνται σειριακά και παράλληλα.

Εισαγωγή

Είναι γνωστό ότι τα περισσότερα προβλήματα που συναντάμε είτε είναι γραμμικά είτε τα αναγάγουμε σε γραμμικά προκειμένου να προσεγγίσουμε την λύση τους. Για την προσέγγιση της επιθυμητής λύσης μπορούμε να χρησιμοποιήσουμε είτε άμεσες είτε επαναληπτικές μεθόδους. Σε ότι αφορά τις άμεσες μεθόδους εστιάσαμε την μελέτη μας στις μεθόδους LU και παραγοντοποίησης Cholesky του πίνακα του συστήματος μας . Ακόμα σχετικά με τις επαναληπτικές μεθόδους έγινε εκτενής μελέτης της μεθόδου απότομης καθόδου (conjugate gradient). Σε αυτό το σημείο αξίζει να σημειώσουμε ότι η παρακάτω μελέτη επικεντρώνεται κυρίως σε αραιούς πίνακες, δηλαδή σε πίνακες με αρκετά μηδενικά στοιχεία. Η μέθοδος conjugate gradient υλοποιήθηκε σε γλώσσα προγραμματισμού python σειριακά αλλά και παράλληλα. Επίσης, γίνεται αναλυτική περιγραφή των διαθέσιμων πακέτων που διαθέτει η python για παράλληλο προγραμματισμό και παρουσίαση των αποτελεσμάτων και συμπερασμάτων που προέκυψαν από την παράλληλη εκτέλεση του κώδικα.

Κεφάλαιο 1: Γραμμική Άλγεβρα

Όπως αναφέραμε και στη εισαγωγή τα περισσότερα προβλήματα είτε μοντελοποιούνται με την βοήθεια γραμμικών εξισώσεων είτε γίνεται η αναγωγή τους σε γραμμικά προκειμένου να μοντελοποιηθούν μέσω ενός γραμμικού συστήματος ώστε να καταφέρουμε να τα επιλύσουμε. Το πιο σημαντικό στοιχείο του γραμμικού συστήματος είναι ο πίνακας Α εάν θεωρήσουμε ότι ένα γραμμικό σύστημα έχει την μορφή Ax = b γιατί από αυτό καθορίζονται αρκετά πράγματα που αφορούν τον τρόπο με τον οποίο θα περιέλθουμε στην επιθυμητή λύση του προβλήματος. Συνεπώς, συμπεραίνουμε ότι η μέθοδος επίλυσης που θα ακολουθήσουμε εξαρτάται σε μεγάλο βαθμό από τις ιδιότητες που χαρακτηρίζουν τον πίνακα.

Οι ιδιότητες του πίνακα έχουν να κάνουν τόσο με την πυκνότητα του, δηλαδή με το ποσοστό των μηδενικών στοιχείων ως προς τον συνολικό αριθμό των στοιχείων του, όσο με την συμμετρία και την θέση των στοιχείων. Έχει παρατηρηθεί ότι όταν οι πίνακες συστημάτων έχουν συγκεκριμένη μορφή μπορεί να γίνει ειδική επεξεργασία των συστημάτων αυτών που είναι πιο συμφέρουσα ως προς τον χρόνο επεξεργασίας των πινάκων αλλά και τον συνολικό χρόνο επίλυσης του προβλήματος. Οι ιδιότητες που θα μας απασχολήσουν περισσότερο είναι η συμμετρία και το αν είναι θετικά ορισμένοι. Ωστόσο, ιδιαίτερο ενδιαφέρον παρουσιάζουν και οι τρισδιαγώνιοι πίνακες στους οποίους γίνεται και αναφορά.

Λαμβάνοντας λοιπόν τις παραπάνω ιδιότητες γίνεται παρουσίαση των μεθόδων παραγοντοποίησης του πίνακα LU και Cholesky, γνωστή και ως LL^T. Οι μέθοδοι παραγοντοποίησης προηγούνται της εφαρμογής μίας μεθόδου επίλυσης του συστήματος και εφαρμόζονται με σκοπό να μετατρέψουν το σύστημα σε ένα ισοδύναμο του προκειμένου να είναι πιο εύκολη η εύρεση της τελικής λύσης.

Ιδιότητες πινάκων

Όπως έχουμε ήδη αναφέρει παραπάνω, οι πίνακες ενός συστήματος διέπονται από κάποιες ιδιότητες. Μία από τις πιο σημαντικές ιδιότητες που αποτελεί κριτήριο εφαρμογής αρκετών μεθόδων επίλυσης συστήματος είναι αυτή της συμμετρίας. Το κριτήριο το οποίο αν ικανοποιείται ισχύει η συμμετρία είναι ότι ο ανάστροφος πίνακας του Α είναι ίσος με τον πίνακα Α, δηλαδή $A^T=A$. Συνεπώς προκειμένου να τον επεξεργαστούμε αρκεί να πάρουμε τα στοιχεία είτε από την διαγώνιο και πάνω είτε από την διαγώνιο και κάτω.

Ακόμα, ιδιαίτερο ενδιαφέρον παρουσιάζουν οι πίνακες οι οποίοι είναι τρισδιαγώνιοι ή πενταδιαγώνιοι. Στη πλειοψηφία τους είναι μεγάλοι πίνακες οι οποίοι έχουν όλα τα στοιχεία τους μηδενικά πλην αυτών που βρίσκονται στην διαγώνιο και σε μια-δυο διαγώνιες πάνω και κάτω της κύριας διαγωνίου μη μηδενικά αντίστοιχα. Επίσης, σε ότι αφορά το πλήθος των συνολικών μηδενικών στοιχείων ενός πίνακα μπορούμε να χωρίσουμε τους πίνακες σε πυκνούς και αραιούς. Οι πυκνοί πίνακες είναι αυτοί οι οποίοι έχουν μικρό ποσοστό των συνολικών στοιχείων τους μηδενικά ενώ οι αραιοί πίνακες έχουν μεγάλο ποσοστό στοιχείων μηδενικά.

Ο διαχωρισμός των πινάκων σε πυκνούς και αραιούς γίνεται γιατί μπορούμε να επεξεργαστούμε διαφορετικά τα συστήματα με αραιούς πίνακες διότι αρχικά απαιτούν λιγότερο χώρο αποθήκευσης τους και κατά συνέπεια και μικρότερο χρόνο επίλυσης των συστημάτων που τους περιέχουν. Οι διάφοροι τρόποι αποθήκευσης ενός αραιού πίνακα είναι σαν dictionary of keys(DOK), σαν list of lists (LIL), σαν coordinate list(COO), Yale και τέλος Compressed Sparse Column ή Compressed Sparse Row (CSC ή CSR). Προκειμένου να χρησιμοποιήσουμε κάποια από τις παραπάνω μεθόδους αποθήκευσης του πίνακα είναι απαραίτητο να γνωρίζουμε την ακριβή θέση των μη μηδενικών στοιχείων στον πίνακα. Για τους σποραδικούς πίνακες χρησιμοποιούμε επαναληπτικές μεθόδους επίλυσης γραμμικών συστημάτων ενώ για τους πυκνούς άμεσες μεθόδους.

Dictionaries of Keys (DOK)

Σε αυτήν την μορφή έχουμε ένα dictionary το οποίο έχει σαν κλειδί ένα ζεύγος (γραμμή, στήλη) και σαν τιμή την τιμή του στοιχείου. Όπως είναι αναμενόμενο λείπουν τα ζευγάρια που η τιμή του στοιχείου είναι 0. Η μέθοδος αυτή είναι βολική όταν θέλουμε να δημιουργήσουμε σταδιακά έναν πίνακα με τυχαία σειρά στοιχείων αλλά όχι για επαναλαμβανόμενη πρόσβαση στα μη μηδενικά στοιχεία του πίνακα με λεξικογραφική σειρά. Συνήθως, όταν κάποιος χρησιμοποιεί αυτή την μορφή κατασκευής του πίνακα εφόσον την χρησιμοποιήσει θα χρειαστεί να προβεί σε μία άλλη μορφή αποθήκευσης προκειμένου να επεξεργαστεί πιο εύκολα και αποτελεσματικά τον πίνακα του.

Lists of Lists (LIL)

Σε αυτήν την μορφή έχουμε μία λίστα από λίστες με την κάθε μία από τις τελευταίες να αντιπροσωπεύει μία γραμμή του πίνακα. Όπως και η προαναφερθείσα μορφή αποθήκευσης DOK είναι μία μέθοδος που μπορεί να είναι χρήσιμη για την σταδιακή κατασκευή του πίνακα αλλά μετά την χρήση της είναι αναγκαστικό να χρησιμοποιηθεί είτε η CSC ή CRS. Συνεπώς, ούτε αυτός ο τρόπος είναι ιδανικός για την αποθήκευση ενός αραιού πίνακα.

Coordinate List (COO)

Ο πίνακας A διασπάται σε τρεις επιμέρους λίστες. Η μία αντιπροσωπεύει τις μη μηδενικές τιμές των στοιχείων του πίνακα η δεύτερη την γραμμή στην οποία βρίσκεται το κάθε μη μηδενικό στοιχείο του πίνακα και ο τρίτος την γραμμή στην οποία βρίσκεται το μη μηδενικό στοιχείο. Είναι σημαντικό να τονίσουμε ότι τα στοιχεία αποθηκεύονται κατά γραμμή, δηλαδή για να κατασκευάσουμε αυτούς τους 3 πίνακες διατρέχουμε τον αρχικό πίνακα A κατά γραμμή, δηλαδή από αριστερά προς τα δεξιά και έπειτα από επάνω προς τα κάτω. Επίσης, υπάρχει και η αντίστοιχη μέθοδος αποθήκευσης για διαπέραση του πίνακα κατά στήλες δηλαδή από πάνω προς τα κάτω και έπειτα από αριστερά προς τα δεξιά.

Παράδειγμα 1.1

Αν έχουμε τον πίνακα A :

$$A=\begin{pmatrix} 11 & 0 & 0 & 14 & 0 & 0 \\ 0 & 22 & 0 & 0 & 0 & 26 \\ 31 & 0 & 33 & 0 & 0 & 0 \\ 41 & 0 & 0 & 0 & 45 & 0 \\ 0 & 0 & 53 & 0 & 55 & 0 \\ 0 & 0 & 0 & 64 & 0 & 66 \end{pmatrix}$$

Τότε στην COO μορφή αποθήκευσης οι λίστες θα είναι οι εξής:

$a\_\_values$=[11, 14, 22, 26, 31, 33, 41, 45, 53, 55, 64, 66]

$i\_\_index$=[1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6]

$j\_\_index$=[1, 4, 2, 6, 1, 3, 1, 5, 3, 5, 4, 6]

Yale

Όπως και στην COO έτσι και στην Yale ο αρχικός πίνακας αναπαρίσταται από τρεις επιμέρους διανύσματα/λίστες. Η πρώτη λίστα περιέχει μόνο τις τιμές των μη μηδενικών στοιχείων του πίνακα ενώ η δεύτερη (IA) έχει μέγεθος όσο και η A λίστα συν ένα διότι περιέχει την θέση του πρώτου μη μηδενικού στοιχείου της κάθε γραμμής και τελευταίο της στοιχείο είναι ο συνολικός αριθμός των μη μηδενικών στοιχείων του πίνακα. Επιπροσθέτως, η τελευταία λίστα περιέχει την κάθετη συντεταγμένη του στοιχείου. Τέλος αξίζει να σημειωθεί ότι το πρώτο στοιχείο της λίστας IA είναι πάντα 0.

Παράδειγμα 1.2

Για τον πίνακα του παραδείγματος 1.1 προκύπτουν οι εξής λίστες¨

A = [11, 14, 22, 26, 31, 33, 41, 45, 53, 55, 64, 66]

IA = [0, 1, 0, 0, 2, 3 ,12]

JA = [0, 3, 1, 5, 0, 3, 0, 4, 2, 4, 3, 5]

Compressed Row Storage (CRS) & Compressed Column Storage (CCS)

Η συγκεκριμένη μέθοδος αποθήκευσης ενός αραιού πίνακα έχει αρκετά κοινά σημεία με την μέθοδο Yale. Πάλι χρειαζόμαστε 3 καινούριους πίνακες για την αποθήκευση των στοιχείων που μας αφορούν και όπως και στη προηγούμενη μέθοδο κρατάμε ως έχουν τους πίνακες a_values και j_index και αντί του i_index θα χρησιμοποιήσουμε έναν διαφορετικό πίνακα τον i_ptr ο οποίος σηματοδοτεί την αρχή της κάθε γραμμής του πίνακα έτσι χρησιμοποιώντας το παραπάνω παράδειγμα θα δείξουμε τα περιεχόμενα του καινούριο πίνακα i_ptr. Η αντίστοιχη μέθοδος για διαπέραση κατά στήλες είναι η Compressed Sparse Column (CSC or CCS) και στην περίπτωση όπου την υλοποιήσουμε θα έχουμε όπως στην COO τους πίνακες a_values και i_index και αντί του j_index θα χρησιμοποιούσαμε τον j_ptr όπου θα σηματοδοτούσε την αρχή της κάθε στήλης. Ουσιαστικά, δηλαδή οι πίνακες i_ptr και j_ptr περιέχουν τον αριθμό πλήθους του πρώτου μη μηδενικού στοιχείου του πίνακα στην καινούρια γραμμή ή στήλη. Η διαφορά ανάμεσα στην CRS και στην CCS είναι ότι ο τρόπος με τον οποίο διαβάζουμε τα στοιχεία, στην πρώτη τα διαβάζουμε πρώτα από αριστερά προς τα δεξιά και έπειτα από πάνω προς τα κάτω ενώ στην δεύτερη πρώτα από πάνω προς τα κάτω και έπειτα από αριστερά προς τα δεξιά

Παράδειγμα 1.3

Ο πίνακας A του παραδείγματος 1.1 με την CRS

$a\_\_values$=[11, 14, 22, 26, 31, 33, 41, 45, 53, 55, 64, 66]

$i\_\_ptr$=[1, 3, 5, 7, 9, 11, 13]

$j\_\_index$=[1, 4, 2, 6, 1, 3, 1, 5, 3, 5, 4, 6]

Παράδειγμα 1.4

Ο πίνακας A του παραδείγματος 1.1 με την CCS

$a\_\_values$=[11, 31, 41, 22, 33, 53, 14, 64, 45, 55, 26, 66]

$i\_\_ptr$ =[1, 3, 4, 2, 3, 5, 1, 6, 4, 5, 2, 6]

$j\_\_index$ =[1, 4, 5, 7, 9, 11, 13]

Σαφώς υπάρχουν και άλλοι τρόποι αποθήκευσης ενός αραιού πίνακα άλλα δεν χρειάζεται για την μελέτη μας να προβούμε σε περαιτέρω ανάλυση του συγκεκριμένου θέματος. Οι μορφές αυτές υποστηρίζονται κάποια συγκεκριμένα file formats όπως το matrix market format το οποίο υποστηρίζει την COO μέθοδο αποθήκευσης και το Harwell-Boeing Exchange Format , που υποστηρίζει το CSR format. Τέλος αξίζει να αναφέρουμε πως υπάρχει πληθώρα δεδομένων στα sites https://www.cise.ufl.edu/research/sparse/matrices/ και www.netlib.org από όπου αντλήσαμε και εμείς τα δεδομένα μας προκειμένου να καταλήξουμε στα συμπεράσματα μας για τα οποία θα υπάρξει ξεχωριστό κεφάλαιο.

Τετραγωνικές Μορφές Πίνακα

Στα γραμμικά συστήματα κάθε πίνακας ο οποίος είναι τετραγωνικός, συμμετρικός και τα στοιχεία του είναι πραγματικοί αριθμοί θεωρείται θετικά ορισμένος. Αρκεί όλες οι ιδιοτιμές του πίνακα που εξετάζουμε να είναι θετικές προκειμένου να καταλήξουμε στο συμπέρασμα ότι ένας πίνακας είναι θετικά ορισμένος. Αντίστοιχα ένας πίνακας είναι αρνητικά ορισμένος όταν όλες οι ιδιοτιμές του είναι αρνητικές. Ακόμα θετικά και αρνητικά ημι-ορισμένοι θεωρούνται οι πίνακες οι οποίοι έχουν τις ιδιοτιμές του μεγαλύτερες ή ίσες με το μηδέν και αρνητικές ή ίσες με το 0 αντίστοιχα.

Προκειμένου να αποφύγουμε την διαδικασία εύρεσης των ιδιοτιμών αρκεί να δείξουμε ότι οι ορίζουσες όλων των κύριων υποπινάκων του πίνακα μας συμπεριφέρονται αντίστοιχα με το πώς είναι ορισμένος ο πίνακας μας, πχ. για θετικά ορισμένους πρέπει όλες να είναι θετικές κοκ. Όταν ένας πίνακας δεν ανήκει σε καμία από τις παραπάνω τετραγωνικές μορφές τότε είναι αόριστος.

Σύνοψη

Ένας συμμετρικός πίνακας A και η τετραγωνική μορφή x^TAx θα ονομάζονται:

Θετικά ορισμένοι εάν:	$x^TAx>0$
Θετικά ημιορισμένοι εάν:	$x^TAx\geq 0$
Αρνητικά ημιορισμένοι εάν:	$x^TAx\leq 0$
Αρνητικά ορισμένοι εάν:	$x^TAx<0$
Αόριστοι εάν:	x^TAx παίρνει και θετικές και αρνητικές τιμές

Άμεσες Μέθοδοι (Direct Methods)

Οι άμεσες μέθοδοι είναι κατάλληλες για γραμμικά συστήματα με πυκνούς πίνακες όπως έχουμε ήδη αναφέρει. Μία από τις πιο σημαντικές άμεσες μεθόδους είναι η απαλοιφή Gauss(Gauss Elimination). Προκειμένου να λύσουμε το σύστημα μας χρησιμοποιούμε την A = LU παραγοντοποίηση διότι είναι πολύ πιο εύκολη η επίλυση ενός συστήματος με τριγωνικό πίνακα. Οπότε επιλέγουμε να λύσουμε δύο επιμέρους συστήματα για καταλήξουμε στην επιθυμητή λύση. Το σύστημα Ax= b ανάγεται μετά τη LU παραγοντοποίηση στο Ly=b και Ux =y.

Παραγοντοποίηση A = LU

Είναι γνωστό πως είναι πολύ πιο εύκολο να επιλύσουμε ένα γραμμικό σύστημα εξισώσεων όταν ο πίνακας του συστήματος είναι σε τριγωνική μορφή δηλαδή είτε άνω τριγωνικός είτε κάτω τριγωνικός. Στην συγκεκριμένη παραγοντοποίηση ο αρχικός πίνακας αναλύεται σε δύο επιμέρους πίνακες, έναν άνω τριγωνικό U και έναν κάτω τριγωνικό L όπως περιγράφεται στην παραπάνω φωτογραφία:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}.$$

Ο αλγόριθμος για την παραγοντοποίηση A = LU είναι ο ακόλουθος:

for $i = 1,\ldots,n$

  for $j = 1,\ldots,i-1$

$$L_{ij} = \frac{A_{ij} - \sum_{k=1}^{j-1} L_{ik} U_{kj}}{U_{jj}}$$

  end

  $L_{ii} = 1$

  for $j = i,\ldots,n$

$$U_{ij} = A_{ij} - \sum_{k=1}^{i-1} L_{ik} U_{kj}$$

  end

end

Έπειτα από την παραπάνω παραγοντοποίηση του πίνακα Α μπορούμε να χρησιμοποιήσουμε την προς τα πίσω αντικατάσταση για την εύρεση της λύσης.

Λύση συστήματος με άνω τριγωνικό πίνακα:

$$\begin{aligned} u_{11}x_1 + u_{12}x_2 + \ldots + u_{1n}x_n &= b_1 \\ u_{22}x_2 + \ldots + u_{2n}x_n &= b_2 \\ &\vdots \\ u_{nn}x_n &= b_n \end{aligned}$$

Λύση συστήματος με κάτω τριγωνικό πίνακα:

$$u_{nn}x_n = b_n \Leftrightarrow x_n = \frac{b_n}{u_{nn}}$$

$$u_{n-1,n-1}x_{n-1} + u_{n-1,n}x_n = b_{n-1} \Leftrightarrow u_{n-1,n-1}x_{n-1} = b_{n-1} - u_{n-1,n}x_n \Leftrightarrow x_{n-1} = \frac{b_{n-1} - u_{n-1,n}x_n}{u_{n-1,n-1}}$$

$$\vdots$$

$$u_{11}x_1 + u_{12}x_2 + \ldots + u_{1n}x_n = b_1 \Leftrightarrow u_{11}x_1 = b_1 - (u_{12}x_2 + \ldots + u_{1n}x_n) \Leftrightarrow$$

$$x_1 = \frac{b_1 - (u_{12}x_2 + \ldots + u_{1n}x_n)}{u_{11}}$$

Cholesky factorization A = LL^T

Η ανάλυση Cholesky είναι εφαρμόσιμη εφόσον έχουμε να κάνουμε με συστήματα των οποίων οι πίνακες είναι τετραγωνικοί, συμμετρικοί και τα στοιχεία τους ανήκουν όλα στο σύνολο των πραγματικών αριθμών. Αν μάλιστα συνδυάσουμε όλα όσα αναφέραμε παραπάνω συμπεραίνουμε όρι είναι απαιτούμενο οι πίνακες να είναι συμμετρικοί και θετικά ορισμένοι. Σε αυτήν την παραγοντοποίηση ανάγεται η παραγοντοποίηση A=LU εάν ο πίνακας ικανοποιεί τις παραπάνω συνθήκες, οπότε και ισχύει ότι U = L^T. Αξίζει να σημειωθεί ότι αυτή η παραγοντοποίησης είναι δύο φορές αποδοτικότερη της LU παραγοντοποίησης. Επίσης, τα στοιχεία της διαγωνίου του πίνακα L επιτρέπεται να είναι και μηδέν. Τέλος, από την συγκεκριμένη μέθοδο ανάλυσης προκύπτουν και οι παραγοντοποιήσεις LD L^T και LDM^T στις οποίες δεν θα επεκταθούμε στην συγκεκριμένη διπλωματική εργασία.

Ο αλγόριθμος αυτής της μεθόδου ανάλυσης του πίνακα είναι ο ακόλουθος:

$$\text{for } i = 1,\ldots,n$$
$$\quad \text{for } j = 1,\ldots,i-1$$
$$\quad\quad L_{ij} = \frac{A_{ij} - \sum_{k=1}^{j-1} L_{ik} L_{jk}}{L_{jj}}$$
$$\quad \text{end}$$
$$\quad L_{ii} = \left(A_{ii} - \sum_{k=1}^{i-1} (L_{ik})^2 \right)^{1/2}$$
$$\text{end}$$

Επαναληπτικές Μέθοδοι

Στην αρχή αυτού του κεφαλαίου διατυπώσαμε την άποψη ότι οι επαναληπτικές μέθοδοι είναι κατάλληλοι για συστήματα όπου οι πίνακες τους είναι σποραδικοί, έχουν δηλαδή αρκετά μηδενικά στοιχεία. Συνήθως, τέτοια συστήματα προκύπτουν στην προσπάθεια εύρεσης λύσης σε μερικές διαφορικές εξισώσεις ή όταν έχουμε να κάνουμε με προβλήματα βελτιστοποίησης. Επίσης, αυτές οι μέθοδοι επιτρέπουν και τον παραλληλισμό των υπολογισμών πιο εύκολα από ότι οι άμεσες μέθοδοι.

Οι πιο γνωστές επαναληπτικές μέθοδοι για γραμμικά συστήματα είναι η Jacobi , η Gauss-Seidel και η SOR. Η γενική ιδέα στις επαναληπτικές μεθόδους είναι ότι έχουμε μία αρχική προσέγγιση της λύσης, συνήθως ισούται με το μηδενικό διάνυσμα και καταλήγουμε πιο γρήγορα από τις άμεσες μεθόδους. Προκειμένου να εφαρμόσουμε τις επαναληπτικές μεθόδους ο πίνακας του συστήματος οφείλει να είναι αντιστρέψιμος, δηλαδή η ορίζουσα του να μην ισούται με το μηδέν. Η βασική διάσπαση του αρχικού πίνακα είναι A= L+D+U.

Στις επαναληπτικές μεθόδους για την επίλυση γραμμικών συστημάτων περιλαμβάνονται και οι μέθοδοι ελαχιστοποίησης. Οι μέθοδοι αυτοί ονομάζονται έτσι διότι προσπαθούν να περιέλθουν στην λύση του συστήματος κατασκευάζοντας την $f(x) = \frac{1}{2}x^TAx - x^Tb$ και αναζητώντας το ελάχιστο αυτής το οποίο θεωρείται και η λύση του συστήματος. Πιο γενικευμένα το σκεπτικό είναι πως προσπαθούμε να βρούμε ένα σταθερό σημείο ενός ισοδύναμου προβλήματος του αρχικού συστήματος.

Στην συνέχεια αυτής της παραγράφου θα εντρυφήσουμε περεταίρω στο τις επαναληπτικές μεθόδους καθώς και στο πως καταλήγουμε στα σποραδικά συστήματα μέσω των μερικών διαφορικών εξισώσεων καθώς και την επίλυσης συστημάτων αραιών πινάκων με την μέθοδο ελαχιστοποίησης συζυγών κλίσεων, γνωστή ως conjugate gradient καθώς και με την μέθοδο αυτή όταν γίνεται και εφαρμογή ενός preconditioner (προρυθμιστή).

Jacobi Method

Η διάσπαση σε αυτήν την μέθοδο είναι η : $Dx = (L+U)x + b$ και το διάνυσμα της προσέγγισης του επόμενου σημείου επέρχεται από την εξίσωση : $x^{(k+1)} = D^{-1}(L+U)x^{(k)} + D^{-1}b$. Η μέθοδος συγκλίνει όταν $\rho(D^{-1}(L+U)<1$, όπου $\rho(A)=\max\{|\lambda_1|,|\lambda_2|,\ldots, |\lambda_n|\}$, όταν δηλαδή η νόρμα όλων των διανυσμάτων των ιδιοτιμών του πίνακα $D^{-1}(L+U)$ είναι μικρότερη του 1 ενώ η σύγκλιση της μεθόδου είναι βέβαιη όταν ο A έχει κυρίαρχη διαγώνιο, δηλαδή όταν όλα τα στοιχεία της κύριας διαγωνίου είναι μεγαλύτερα από τα υπόλοιπα στοιχεία του πίνακα.

Τέλος, αξίζει να σημειωθεί ότι το η επόμενη προσέγγιση του x εξαρτάται μόνο από την προηγούμενη προσέγγιση του x συνεπώς καταλήγουμε στο συμπέρασμα ότι η συγκεκριμένη μέθοδος είναι πλήρως παραλληλίσιμη.

Gauss-Seidel

Η διάσπαση σε αυτήν την μέθοδο είναι η : $(D+ L)x = Ux +b$ και το διάνυσμα της προσέγγισης του επόμενου σημείου επέρχεται από την εξίσωση : $x^{(k+1)} = (D-L)^{-1} Ux^{(k)} + (D-L)^{-1}b$. Η μέθοδος συγκλίνει όταν $\rho((D-L)^{-1}U)<1$, όπου $\rho(A)=\max\{|\lambda_1|,|\lambda_2|,\ldots, |\lambda_n|\}$, όταν δηλαδή η νόρμα όλων των διανυσμάτων των ιδιοτιμών του πίνακα $(D-L)^{-1}U$ είναι μικρότερη του 1 ενώ η σύγκλιση της μεθόδου είναι βέβαιη όταν ο A έχει κυρίαρχη διαγώνιο, δηλαδή όταν όλα τα στοιχεία της κύριας διαγωνίου είναι μεγαλύτερα από τα υπόλοιπα στοιχεία του πίνακα. Τέλος, αξίζει να σημειωθεί ότι το η επόμενη προσέγγιση του x εξαρτάται από όλες τις προηγούμενες προσέγγισεις του x καθώς και από όλες τις επόμενες συνεπώς καταλήγουμε στο συμπέρασμα ότι η συγκεκριμένη μέθοδος είναι σειριακή οπότε δεν προτιμάται έναντι της Jaco/bi method.

SOR

Έχοντας ήδη μία αρχική προσέγγιση και x^k καθώς και την προσέγγιση της Gauss-Seidel τότε το η επόμενη προσέγγιση του $x^{(k+1)}$ προκύπτει ως εξής : $(D-\omega L)x^{(k+1)} = [(1-\omega)D +\omega U]x^{(k)} +\omega b$. Η μέθοδος συγκλίνει εφόσον $0<\omega<2$. Τέλος, όταν $\omega=1$ τότε η SOR ταυτίζεται με την Gauss-Seidel.

Μέθοδοι Ελαχιστοποίησης

Όπως αναφέραμε στην εισαγωγή αυτής της παραγράφου οι μερικές διαφορικές εξισώσεις αποτελούν την κύρια πηγή μεγάλων γραμμικών συστημάτων με σποραδικούς πίνακες. Η πιο γνωστή μέθοδος προκειμένου να ανάγουμε το σύστημα μας σε ένα ισοδύναμο, να γίνει δηλαδή διακριτοποίηση μιας μερικής διαφορικής εξίσωσης είναι η μέθοδος πεπερασμένων στοιχείων.

Μέθοδος Πεπερασμένων Στοιχείων (Finite Elements Method)

Πρόκειται για μία μέθοδο εύρεσης προσεγγιστικής λύσης σε προβλήματα συνοριακών συνθηκών για μερικές διαφορικές εξισώσεις. Το βασικό σκεπτικό που πρέπει να ακολουθήσουμε στην μέθοδο αυτή είναι διακριτοποιούμε το πρόβλημα σε επιμέρους μικρότερα προβλήματα και έπειτα η εύρεση της λύσης προκύπτει από το συνδυασμό των λύσεων των επιμέρους αυτών προβλημάτων. Η τελική λύση προέρχεται από ένα σύνολο γραμμικών εξισώσεων για προβλήματα σταθερής κατάστασης ή/και από ένα σύνολο συνήθων διαφορικών εξισώσεων για προβλήματα κατάστασης μετάβασης. Στην παρούσα μελέτη θα εμβαθύνουμε στην πρώτη κατηγορία προβλημάτων.

Σε αυτά τα προβλήματα που περιγράφουμε η κάθε εξίσωση έχει ένα πεδίο Ω για το οποίο ισχύει η μερική διαφορική εξίσωση, γι’ αυτό εξίσου σημαντικές είναι και οι συνοριακές συνθήκες που πρέπει να ικανοποιούνται στα άκρα του Ω. Υπάρχουν τριών ειδών συνοριακές συνθήκες ενώ είναι δυνατόν μία διαφορική εξίσωση να χρειάζεται να ικανοποιεί ανάμεικτη συνοριακή συνθήκη. Τα είδη των συνοριακών συνθηκών είναι τα εξής:

Dirchlet Boundary Condition	$u(x)=\phi(x)$
Neumann Boundary Condition	$\frac{\partial u}{\partial n}(x) = 0$
Cauchy Boundary Condition	$\frac{\partial u}{\partial n}(x) + \alpha(x)u(x)=\gamma(x)$

Μέθοδος Συζυγών Κλίσεων (Conjugate Gradient Method)

Αυτή η τεχνική είναι κατάλληλη για μεγάλα και αραιά συστήματα των οποίων ο χειρισμός είναι δύσκολος με άμεσες μεθόδους. Όπως και σε κάθε επαναληπτική μέθοδο έχουμε μία αρχική προσέγγιση της λύσης x_0 η οποία είναι συνήθως το μηδενικό διάνυσμα και προσπαθούμε να προσεγγίσουμε την λύση της quadratic equation $f(x) = \frac{1}{2}x^TAx - x^Tb$. Το μέτρο με το οποίο ελέγχουμε εάν είμαστε πλησιέστερα στην λύση σε κάθε επανάληψη είναι εάν η τιμή του f(x) είναι μικρότερη από την τιμή του σε σχέση με την προηγούμενη προσέγγιση. Το χαρακτηριστικό αυτής της μεθόδου είναι ότι εάν έχουμε επιλέξει σωστά τα συζυγή διανύσματα κατεύθυνσης μπορεί να μην τα χρειαζόμαστε στο σύνολο τους προκειμένου να καταλήξουμε σε μία καλή προσέγγιση της λύσης του x. Σε κάθε βήμα το διάνυσμα r_k είναι συζυγές με το p_k και σε κάθε βήμα της μεθόδου απαιτείται μόνο ένας πολλαπλασιασμός πίνακα διανύσματος (Ap_k). Ακολουθεί ο αλγόριθμος της μεθόδου:

$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$
$\mathbf{p}_0 := \mathbf{r}_0$
$k := 0$
repeat
  $\alpha_k := \frac{\mathbf{r}_k^\mathsf{T}\mathbf{r}_k}{\mathbf{p}_k^\mathsf{T}\mathbf{A}\mathbf{p}_k}$
  $\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k\mathbf{p}_k$
  $\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k\mathbf{A}\mathbf{p}_k$
  if r_{k+1} is sufficiently small then exit loop
  $\beta_k := \frac{\mathbf{r}_{k+1}^\mathsf{T}\mathbf{r}_{k+1}}{\mathbf{r}_k^\mathsf{T}\mathbf{r}_k}$
  $\mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k\mathbf{p}_k$
  $k := k + 1$
end repeat
The result is $\mathbf{x}_{k+1}$

Μέθοδος Συζυγών Κλίσεων με Προρυθμιστή(Preconditioned Conjugate Gradient Method)

Η μέθοδος συζυγών κλίσεων στο σύνολο της θεωρείται ότι είναι μία άμεση μέθοδος που ακόμα και σαν επαναληπτική η προσεγγιστική λύση του συστήματος επιτυγχάνεται έπειτα από n πεπερασμένες επαναλήψεις οι οποίες δεν ξεπερνούν το μέγεθος του πίνακα. Επίσης, δεν θεωρείται ότι είναι μία σταθερή μέθοδος, δηλαδή δεν είναι σίγουρο ότι θα συγκλίνει μιας και τα p_k διανύσματα κατεύθυνσης στην πραγματικότητα δεν είναι συζυγή. Προκειμένου, λοιπόν να έχουμε μεγαλύτερη βεβαιότητα ως προς την σύγκλιση της μεθόδου και ως προς την ικανοποίηση των συνθηκών του κάθε επαναληπτικού βήματος εφαρμόζουμε έναν προρυθμιστή πίνακα.

Αρχικά είναι ιδιαίτερα σημαντικό το γεγονός ότι ο πίνακας που θα χρησιμοποιήσουμε ως προρυθμιστή του συστήματος θα πρέπει να είναι συμμετρικός και θετικά ορισμένος. Ο καλύτερος πίνακας που μπορεί να χρησιμοποιηθεί ως preconditioner είναι ο αντίστροφος του αρχικού πίνακα του συστήματος μας όμως η εύρεση του δεν είναι πάντα εύκολη. Υπάρχει πληθώρα τέτοιον πινάκων που μπορούμε να χρησιμοποιήσουμε. Αυτός που φέρνει τα καλύτερα αποτελέσματα είναι ο αντίστροφος του πίνακα αλλά ο υπολογισμός του είναι ιδιαίτερα ακριβός.

Ακόμη ως τέτοιος θα μπορούσε να χρησιμοποιηθεί και ο ταυτοτικός ο οποίος όμως δεν θα επέφερε αλλαγή στο σύστημα μιας και αυτό θα παρέμενε ακριβώς ίδιο. Επόμενος πίνακας που θα εξετάσουμε και είναι ένας από αυτούς που θα χρησιμοποιήσουμε στην υλοποίηση της μεθόδου είναι ο Jacobi : M =diag(A) του οποίου ο υπολογισμός είναι αρκετά φθηνός. Παρόλο όμως που μπορεί επιφέρει κάποια βελτίωση συνήθως δεν συμβαίνει αυτό.

Επίσης, θα μπορούσαμε να εφαρμόσουμε και ένα αποτέλεσμα από τις επαναληπτικές μεθόδους που αναφέραμε παραπάνω. Τέλος, θα μπορούσαμε να εφαρμόσουμε έναν προρυθμιστή που προκύπτει από τις μη ολοκληρωμένες παραγοντοποιήσεις(incomplete factorizations) όπως την incomplete Cholesky factorization, preconditioner τον οποίο επίσης χρησιμοποιούμε μιας και είναι ο περισσότερο αποδοτικός. Με την εφαρμογή του preconditioner ο αλγόριθμος διαμορφώνεται ως εξής:

$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$
$\mathbf{z}_0 := \mathbf{M}^{-1}\mathbf{r}_0$
$\mathbf{p}_0 := \mathbf{z}_0$
$k := 0$
repeat
$\alpha_k := \frac{\mathbf{r}_k^\mathsf{T}\mathbf{z}_k}{\mathbf{p}_k^\mathsf{T}\mathbf{A}\mathbf{p}_k}$
$\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k\mathbf{p}_k$
$\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k\mathbf{A}\mathbf{p}_k$
if $\mathbf{r}_{k+1}$ is sufficiently small **then** exit loop **end if**
$\mathbf{z}_{k+1} := \mathbf{M}^{-1}\mathbf{r}_{k+1}$
$\beta_k := \frac{\mathbf{z}_{k+1}^\mathsf{T}\mathbf{r}_{k+1}}{\mathbf{z}_k^\mathsf{T}\mathbf{r}_k}$
$\mathbf{p}_{k+1} := \mathbf{z}_{k+1} + \beta_k\mathbf{p}_k$
$k := k + 1$
end repeat
The result is $\mathbf{x}_{k+1}$

Τέλος ο παραπάνω αλγόριθμος ισοδυναμεί με την επίλυση του συστήματος χωρις προρυθμιστή:

$E^{-1}A\,(E^{-1})^T x^* = E^{-1}b$, όπου $EE^T = M$ και $x^* = E^T x$

Incomplete Cholesky Factorization

Η κλασική Cholesky factorization LL^T ανάγεται πλέον στην KK^T όπου K ένας πίνακας αρκετά κοντά στον L. Ο τρόπος με τον οποίο υπολογίζεται ο K είναι να βρούμε την ακριβή Cholesky factorization θέτοντας ίσο με μηδέν κάθε στοιχείο αν στην αντίστοιχη θέση του πίνακα A του συστήματος μας είναι στοιχείο ίσο με μηδέν. Ο αλγόριθμος για τον υπολογισμό της είναι ο εξής:

For i from 1 to N:
$$L_{ii} = \left(a_{ii} - \sum_{k=1}^{i-1} L_{ik}^2\right)^{\frac{1}{2}}$$
For j from $i+1$ to N:
$$L_{ji} = \frac{1}{L_{ii}}\left(a_{ij} - \sum_{k=1}^{i-1} L_{ik}L_{jk}\right)$$

Κεφάλαιο 2 : Python Και Παράλληλοι Υπολογισμοί

Η γλώσσα στην οποία επιλέξαμε να υλοποιήσουμε και την απλή conjugate gradient και την preconditioned conjugate gradient είναι η python. Είναι μία γλώσσα κατάλληλη για scientific computing, scripting αλλά και για αντικειμενοστραφή προγραμματισμό. Είναι ιδιαίτερα γρήγορη στους υπολογισμούς της και διαθέτει πλήθος πακέτων και βιβλιοθηκών για όλα τα παραπάνω είδη προγραμματισμού. Στην υλοποίηση μας χρησιμοποιήσαμε κυρίως built in συναρτήσεις της για τους υπολογισμούς που αφορούν το εσωτερικό γινόμενο διανυσμάτων, τον πολλαπλασιασμό πίνακα με διάνυσμα, τον υπολογισμό του Jacobi preconditioner πίνακα που χρειαστήκαμε κατά την συγγραφή κώδικα για την μέθοδο preconditioned conjugate gradient. Ακόμα, όποτε χρειάστηκε χρησιμοποιήσαμε την τις ενσωματωμένες με την γλώσσα συναρτήσεις για τν υπολογισμό του ανάστροφου ενός πίνακα ή ενός διανύσματος. Επιπροσθέτως, η Python διαθέτει πακέτα για το άνοιγμα αρχείων σε Matrix Market Format (.mtx αρχεία) ενώ έχει αρκετές βιβλιοθήκες για παράλληλους υπολογισμούς. Στο κεφάλαιο αυτό θα αναλύσουμε τα πακέτα που διαθέτει αυτή η γλώσσα προγραμματισμού για γραμμική άλγεβρα στον βαθμό που μας ενδιαφέρει καθώς θα και θα γίνει αναφορά στα διαθέσιμα πακέτα για παράλληλους υπολογισμούς.

Numpy

Η βιβλιοθήκη numpy είναι μία βιβλιοθήκη για προγραμματισμό προβλημάτων που σχετίζονται με γραμμική άλγεβρα. Μέσω αυτού και μέσω του Scipy πακέτου μπορούμε να επεξεργαστούμε πίνακες, να προβούμε σε πράξεις μεταξύ τους και να υπολογίσουμε τους αντιστρόφους τους ή τους ανάστροφους τους. Είτε ο χρήστης χρησιμοποιήσει το Numpy είτε το Scipy είναι σχεδόν το ίδιο διότι περιέχουν πάρα πολλές κοινές συναρτήσεις. Οι παρακάτω συναρτήσεις είναι ίδιες με μόνη διαφορά το όνομα του πακέτου και στα δύο python modules και όπως είναι φανερό εμείς χρησιμοποιήσαμε το numpy πακέτο.

Δημιουργία Δυσδιάστατου Πίνακα → A = numpy.array([[a, b], [c,d]]),
A = numpy.matrix([[a, b], [c,d]])

Δημιουργία Μονοδιάστατου Πίνακα → A = numpy.array([a, b, c,d]),
A = numpy.matrix([a, b, c,d])

Εύρεση Αντίστροφου Πίνακα → invert_A = numpy.linalg.inv(A)

Εύρεση Αντίστροφου Πίνακα →transpose_A = A.T

Εσωτερικό Γινόμενο Πίνακα-Διάνυσμα/Διάνυσματων → numpy.dot(A, b)

Διάνυσμα με Διαγώνια Στοιχεία Ενός Πίνακα → numpy.diag(A)

Εύρεση Πίνακα έχοντας σαν στοιχεία μόνο αυτά της Διαγωνίου → numpy.diag (numpy.diag(A))

Δημιουργία Μηδενικού Πίνακα → numpy.zeros((a, b)), όπου α,β ακέραιοι

Δημιουργία Μηδενικού Διανύσματος → numpy.zeros(a), όπου α ακέραιος

Εύρεση Νόρμας Πίνακα →numpy.linalg.norm(A), by default δίνει την δεύτερη νόρμα ενώ δίπλα από τον πίνακα μπορούμε να περάσουμε σαν όρισμα το ποιας τάξης θέλουμε να είναι, εάν είναι άπειρη τότε η συνάρτηση καλείται ως εξής:

numpy.linalg.norm(A, numpy.inf)

Όπως έχουμε ήδη αναφέρει, υπάρχει το .mtx file format το οποίο υποστηρίζει την μέθοδο αποθήκευσης αραιού πίνακα COO. Τα αρχεία αυτά είναι προσβάσιμα από την Python μέσω της βιβλιοθήκης Scipy.io που περιέχει συναρτήσεις για την πρόσβαση μας στο αρχείο, την φόρτωση κατευθείαν του πίνακα, τις πληροφορίες για το μέγεθος και τον χώρο αποθήκευσης του πίνακα και την εγγραφή στο αρχείο.

Επιστρέφει πληροφορίες για μέγεθος και χώρο αποθήκευσης → scipy.io.mminfo(filename)

Δημιουργεί πίνακα κατευθείαν από .mtx αρχείο → a =scipy.io.mmread(filename)

Εγγραφή στο αρχείο σε COO forma έναν αραιό πίνακα → scipy.io.mmwrite(filename, a)

Όλα τα παραπάνω είναι πολύ εύκολο να τα βρει κανείς στο διαδίκτυο. Το αξιοσημείωτο είναι όμως το γεγονός ότι τα πακέτα αυτά μπορεί να έχουν συνδεθεί με κάποια από τις βιβλιοθήκες lapack, blas και mkl που κάνουν τους υπολογισμούς της python ακόμα γρηγορότερους απ'ότι είναι όταν δεν έχει γίνει η σύνδεση των βιβλιοθηκών αυτών με τα πακέτα επιστημονικού υπολογισμού και αριθμητικής ανάλυσης. Πρόκειται για βιβλιοθήκες οι οποίες περιέχουν αλγορίθμους οι οποίοι βελτιστοποιούν τις πράξεις. Μπορούμε να δούμε αν έχει γίνει η συγκεκριμένη διασύνδεση μέσω της εντολής numpy.show_config() όπου εάν έχει γίνει η διασύνδεση θα δούμε τις βιβλιοθήκες με τις οποίες θα έχει γίνει η διασύνδεση ενώ αν δεν έχει γίνει θα δείχνει ότι δεν είναι διαθέσιμη η βιβλιοθήκη.

Επίσης ένας άλλος τρόπος για να δούμε εάν μία πράξη χρησιμοποιεί τις βιβλιοθήκες αυτές είναι να δώσουμε σε μία python κονσόλα την εντολή numpy.dot.__module__ η οποία εάν η εντολή εκτελείτε μέσω των βιβλιοθηκών βελτιστοποίησης της πράξης θα επιστρέψει ‘numpy.core._dotblas’ ενώ στην αντίθετη περίπτωση 'numpy.core.multiarray'. Όπως είναι αναμενόμενο υπάρχουν εντολές οι οποίες ενεργοποιούν την διασύνδεση εφόσον είναι διαθέσιμες οι βιβλιοθήκες και που την απενεργοποιούν. Ωστόσο, κάτι τέτοιο δεν φαίνεται να ισχύει στην πράξη μιας και η εναλλαγή δεν φαίνεται να λαμβάνει χώρα.

Παρακάτω δείχνουμε το αποτέλεσμα που επιστρέφει η εντολή numpy.show_config()

```
lapack opt info:
    libraries = ['mkl lapack95 lp64', 'mkl blas95 lp64', 'mkl intel lp64',
'mkl intel thread', 'mkl core', 'libiomp5md', 'libifportmd',
'mkl lapack95 lp64', 'mkl blas95 lp64', 'mkl intel lp64', 'mkl intel thread',
'mkl core', 'libiomp5md', 'libifportmd']
    library dirs = ['C:/Program Files (x86)/Intel/Composer
XE/mkl/lib/intel64']
    define macros = [('SCIPY MKL H', None)]
    include dirs = ['C:/Program Files (x86)/Intel/Composer XE/mkl/include']
blas opt info:
    libraries = ['mkl lapack95 lp64', 'mkl blas95 lp64', 'mkl intel lp64',
'mkl intel thread', 'mkl core', 'libiomp5md', 'libifportmd']
    library dirs = ['C:/Program Files (x86)/Intel/Composer
XE/mkl/lib/intel64']
    define macros = [('SCIPY MKL H', None)]
    include dirs = ['C:/Program Files (x86)/Intel/Composer XE/mkl/include']
openblas lapack info:
  NOT AVAILABLE
lapack mkl info:
    libraries = ['mkl lapack95 lp64', 'mkl blas95 lp64', 'mkl intel lp64',
'mkl intel thread', 'mkl core', 'libiomp5md', 'libifportmd',
'mkl lapack95 lp64', 'mkl blas95 lp64', 'mkl intel lp64', 'mkl intel thread',
'mkl core', 'libiomp5md', 'libifportmd']
    library dirs = ['C:/Program Files (x86)/Intel/Composer
XE/mkl/lib/intel64']
    define macros = [('SCIPY MKL H', None)]
    include dirs = ['C:/Program Files (x86)/Intel/Composer XE/mkl/include']
blas mkl info:
    libraries = ['mkl lapack95 lp64', 'mkl blas95 lp64', 'mkl intel lp64',
'mkl intel thread', 'mkl core', 'libiomp5md', 'libifportmd']
    library dirs = ['C:/Program Files (x86)/Intel/Composer
XE/mkl/lib/intel64']
    define macros = [('SCIPY MKL H', None)]
    include dirs = ['C:/Program Files (x86)/Intel/Composer XE/mkl/include']
mkl info:
    libraries = ['mkl lapack95 lp64', 'mkl blas95 lp64', 'mkl intel lp64',
'mkl intel thread', 'mkl core', 'libiomp5md', 'libifportmd']
    library dirs = ['C:/Program Files (x86)/Intel/Composer
XE/mkl/lib/intel64']
    define macros = [('SCIPY MKL H', None)]
    include_dirs = ['C:/Program Files (x86)/Intel/Composer XE/mkl/include']
```

```
>>> numpy.show_config()
lapack_info:
  NOT AVAILABLE
lapack_opt_info:
  NOT AVAILABLE
openblas_lapack_info:
  NOT AVAILABLE
blas_info:
  NOT AVAILABLE
atlas_3_10_blas_threads_info:
  NOT AVAILABLE
atlas_threads_info:
  NOT AVAILABLE
blas_src_info:
  NOT AVAILABLE
atlas_3_10_threads_info:
  NOT AVAILABLE
atlas_blas_info:
  NOT AVAILABLE
atlas_3_10_blas_info:
  NOT AVAILABLE
lapack_src_info:
  NOT AVAILABLE
atlas_blas_threads_info:
  NOT AVAILABLE
openblas_info:
  NOT AVAILABLE
blas_mkl_info:
  NOT AVAILABLE
blas_opt_info:
  NOT AVAILABLE
atlas_info:
  NOT AVAILABLE
atlas_3_10_info:
  NOT AVAILABLE
lapack_mkl_info:
  NOT AVAILABLE
mkl_info:
  NOT AVAILABLE
>>>
```

Εικόνα 2.1: Στιγμιότυπο το οποίο δείχνει ότι δεν είναι διαθέσιμες οι βιβλιοθήκες βελτιστοποίησης αλγορίθμων.

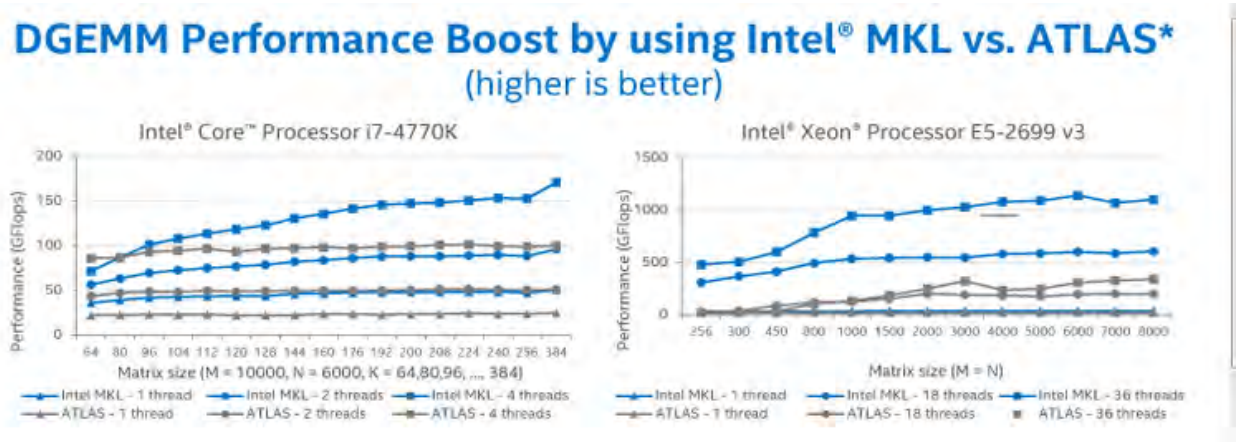
Βιβλιοθήκες Βελτιστοποίησης

Όπως είπαμε έχουμε την δυνατότητα να χρησιμοποιήσουμε τις πράξεις που υπάρχουν ήδη υλοποιημένες στην Python με στην βελτιστοποιημένη τους υλοποίηση όταν έχει γίνει η διασύνδεση του πακέτου που χρησιμοποιούμε με τις αντίστοιχες βιβλιοθήκες βελτιστοποίησης. Οι βιβλιοθήκες αυτές είναι η LAPACK, BLAS και MKL.

Όλες οι προαναφερθείσες βιβλιοθήκες είναι βιβλιοθήκες για γραμμική άλγεβρα και περιέχουν συναρτήσεις που υλοποιούν βελτιστοποιημένα συναρτήσεις για την εύρεση εσωτερικού γινομένου, την εύρεση νόρμας πινάκων και άλλες συχνά χρησιμοποιούμενες πράξεις στις μεθόδους επίλυσης και ελαχιστοποίησης συστημάτων γραμμικής άλγεβρας οι οποίες εν τέλει βελτιώνουν τον χρόνο εκτέλεσης του κώδικα.

Η βιβλιοθήκη LAPACK στηρίζεται πάνω στην βιβλιοθήκη BLAS η οποία περιέχει αυτές τις βασικές πράξεις που αναφέραμε. Η LAPACK περιέχει διαδικασίες για την επίλυση συστημάτων ταυτόχρονων γραμμικών εξισώσεων, εύρεση λύσης προβλημάτων ελαχίστων τετραγώνων για υπερπροσδιορισμένα συστήματα, εύρεση λύσης σε προβλήματα ιδιοτιμών και ιδιαζουσών τιμών.

Η MKL (Math Kernel Library) της Intel ουσιαστικά δίνει καλύτερα αποτελέσματα σε σχέση με τις προηγούμενες δύο βιβλιοθήκες όταν δουλεύουμε σε υπολογιστικό σύστημα που είναι συμβατό με την αρχιτεκτονική της Intel. Οι τελευταία είναι βιβλιοθήκη στην οποία πρόκειται να προστεθούν και άλλοι αλγόριθμοι και θα υποστηρίζεται και από μελλοντικές αρχιτεκτονικές της Intel με μικρές προσαρμογές.



Εικόνα 2.3: Performance Boost by using Intel MKL vs ATLAS

Παράλληλη Python

Είναι ευρέως γνωστό ότι τρία πράγματα στην ζωή είναι αναπόφευκτα, οι φόροι, ο θάνατος και ο παραλληλισμός. Αυτό που προσπαθούμε να επιτύχουμε με την παράλληλη υλοποίηση μια μεθόδου είναι λιγότερος χρόνος για την επεξεργασία ίδιων δεδομένων. Εφόσον κάποια διαδικασία είναι επαναλαμβανόμενη μπορεί να γίνει και παράλληλα σπάζοντας το σύνολο των δεδομένων σε επιμέρους.

Η Python διαθέτει πληθώρα πακέτων για παράλληλο προγραμματισμό, τόσο για νήματα με διαμοιραζόμενη μνήμη όσο και για ξεχωριστή μνήμη. Γενικά, υποστηρίζει τα είδη παραλληλισμού ένα πρόγραμμα πολλά δεδομένα (SPMD- Single Program Multiple Data), πολλά προγράμματα πολλαπλά δεδομένα (MPMD- Multiple Program Multiple Data), παραλληλισμό μέσω ανταλλαγής μηνυμάτων (MPI), και task framing. Επίσης, είναι δυνατό ο χρήστης να επιλέξει έναν συνδυασμό των παραπάνω τεχνικών παράλληλου προγραμματισμού. Για τα συστήματα διαμοιραζόμενης μνήμης υπάρχουν τα πακέτα multiprocessing, αυτό επιλέξαμε στην υλοποίηση μας, threading και joblib. Ενώ για παράλληλο προγραμματισμό χωρίς κοινή μνήμη χρησιμοποιείται το IPython.parallel και το mpi4py που είναι το MPI για Python.

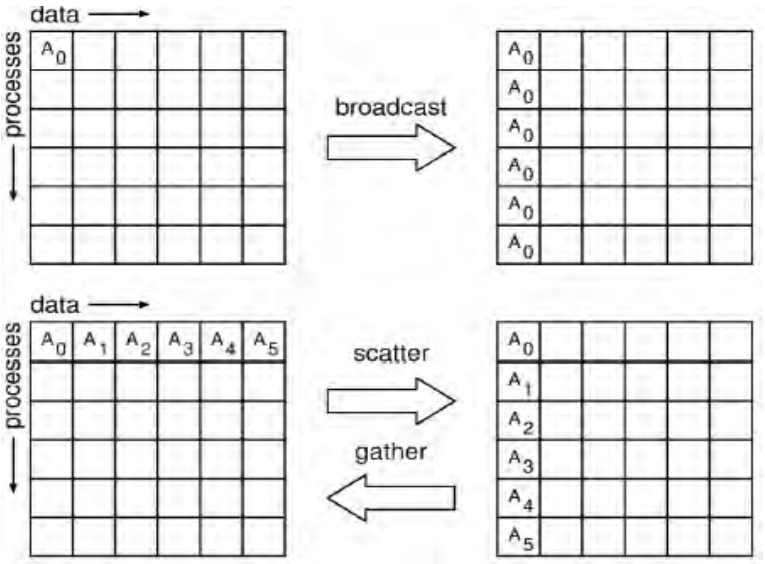
IPython

Το module αυτό είναι κατάλληλο για παράλληλα συστήματα χωρίς κοινή μνήμη. Ιδιαίτερο πλεονέκτημα αυτού του Python πακέτου είναι το ότι υπάρχει μπορεί ο χρήστης ταυτόχρονα να γράφει τον κώδικα και να τον αποσφαλματώνει. Αρχικά, γίνεται έναρξη κάποιων μηχανών (slaves), οι μηχανές που μπορούμε να έχουμε είναι τόσες όσοι και οι πυρήνες του υπολογιστή/συστήματος υπολογιστών μας. Στην συνέχεια ο προγραμματιστής πρέπει να υλοποιήσει μία συνάρτηση στην οποία θα υλοποιείται η λειτουργία που επιθυμούμε για μία μονάδα από αυτές στις οποίες θέλουμε να εφαρμοστεί. Έπειτα γίνεται map της συνάρτησης σε όλα τα δεδομένα, τα οποία είναι σε μορφή λίστας επιστρέφοντας τα αποτελέσματα επίσης σε μορφή λίστας. Επίσης, υπάρχει και η δυνατότητα διαμοίρασης των δεδομένων σε όλες τις μηχανές και με το πέρας τις διαδικασίας η συλλογή των αποτελεσμάτων στην αρχική διεργασία με την λειτουργίες scatter και gather αλλά δεν θα τις εξετάσουμε στην συγκεκριμένη μελέτη.

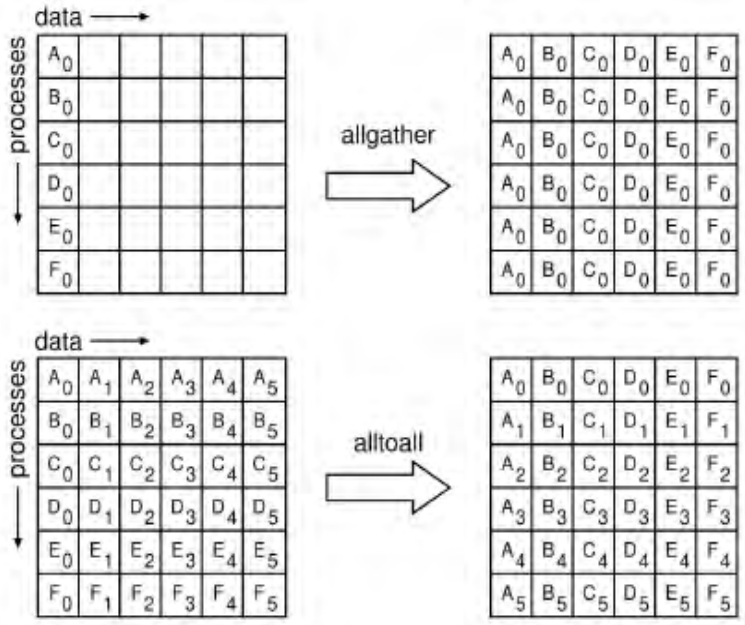
Mpi4py

Πρόκειται για το ευρέως γνωστό και χρησιμοποιούμενο για υλοποίηση παράλληλων διαδικασιών message parsing interface(MPI) σε Python.Αξίζει να σημειώσουμε ότι υποστηρίζει συστήματα με ξεχωριστή μνήμη, υποστηρίζεται από τις περισσότερες αρχιτεκτονικές για παράλληλα συστήματα και υλοποιεί το μοντέλο παραλληλισμού SPMD. Έχουμε τον κόσμο όλων των διεργασιών MPI.COMM_WORLD από τον οποίο μπορούμε να δούμε πόσες διεργασίες έχουμε επί συνόλου, και το προσωπικό αναγνωριστικό της κάθε διεργασίας.

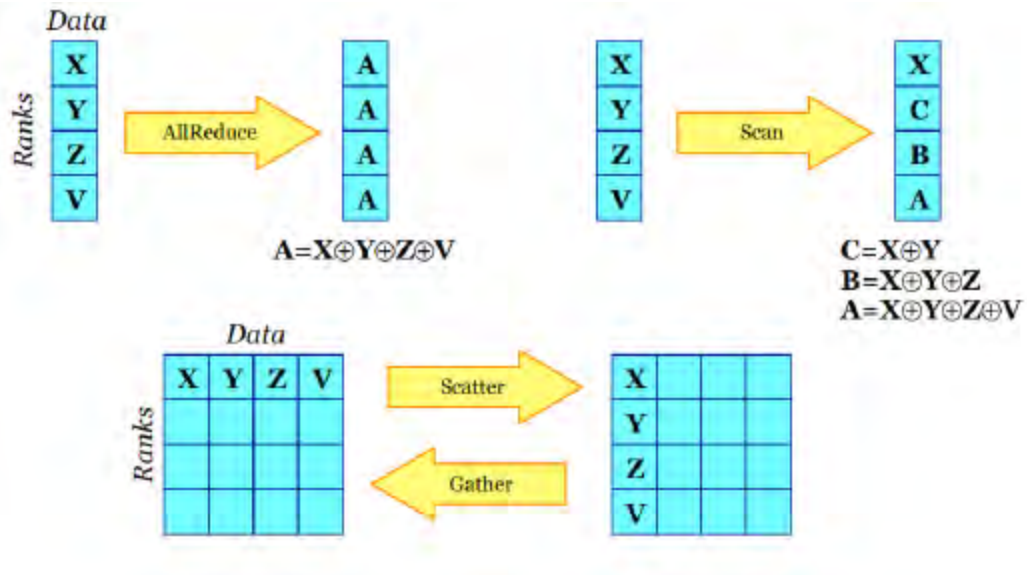
Επιπλέον, υποστηρίζονται πλήρως οι λειτουργίες send, recv καθώς και οι Send , Recv για επικοινωνία σημείο προς σημείο. Οι τελευταίες χρησιμοποιούνται όταν κώδικας μας πρόκειται να περιέχει διανύσματα και είναι πιο γρήγορες από τις send και recv αλλά με την σειρά τους προσθέτουν overhead στην επικοινωνία διότι από απλά Python αντικείμενα μετατρέπονται σε arrays από bytes όταν στέλνονται και το ανάποδο όταν λαμβάνονται . Επίσης, οι λειτουργίες bcast, reduse, reduseAll, scan, scatter, gather, AllGather, AllToAll για περιπτώσεις συλλογικής επικοινωνίας.



Εικόνα 2.4: Λειτουργίες Scatter, gather και broadcast του MPI



Εικόνα 2.5: Λειτουργίες AllGather και AlltoAll του MPI



Εικόνα 2.6: Παραδείγματα λειτουργιών του MPI

Multiprocessing

Είναι ιδανικό για cpu bound προβλήματα, εκμεταλλεύεται τους πολλαπλούς πυρήνες του συστήματος μας και χειρίζεται το λειτουργικό τον προγραμματισμό των εργασιών. Το βασικό σκεπτικό που πρέπει να έχουμε υπόψη όταν θέλουμε να χρησιμοποιήσουμε αυτό το πακέτο της Python είναι ότι δημιουργούμε μία συνάρτηση η οποία να κάνει αυτό που θέλουμε και περνάμε σαν λίστα το διαφορετικό όρισμα που θέλουμε να δώσουμε κάθε φορα. Αν δηλαδή θέλω να κάνω μία πράξη για 150 ακεραίους δημιουργώ την συνάρτηση που κάνει για τον έναν την επιθυμητή ενέργεια και όταν χρησιμοποιήσω το multiprocessing δίνω σαν όρισμα μία λίστα με όλους τους ακεραίους στους οποίους θέλω να κάνω την επιθυμητή ενέργεια.

Παράδειγμα 2.1 Τρέξιμο μιας συνάρτησης με το multiprocessing

```
def  plus_1(x):

   return x+1
```

```
list_of_integers=[1,2,3,…,150]

pool= multiprocessing.Pool(processes = n)

result = pool.map(plus_1, list_of_integers)

pool.terminate()

pool.join()
```

Στο παραπάνω παράδειγμα ανοίγουμε ένα σύνολο διεργασιών στις οποίες κάνουμε map στην συνάρτηση την τιμή του ορίσματος που θέλουμε κάθε φορά. Το result είναι επίσης μία λίστα που κάθε στοιχείο της αντιστοιχεί στην ανάλογη τιμή που δώσαμε σαν όρισμα κάθε φορά στην συνάρτηση και υπάρχει πλήρης αντιστοιχία διότι σε κάθε διεργασία που ανοίγουμε της ανατίθεται ένα μοναδικό ακέραιο αναγνωριστικό. Τέλος, για κάθε pool που ανοίγουμε πρέπει να το κλείνουμε και να κάνουμε join τα νήματα εφόσον έχουν τερματιστεί όλες οι διεργασίες. Το συγκεκριμένο Python module περιέχει και άλλες κλάσεις στις οποίες δεν θα επεκταθούμε γιατί δε χρησιμοποιήσαμε στην υλοποίηση μας.

Σύνοψη

Και τα τρία αυτά Python πακέτα είναι κατάλληλα για την υλοποίηση μας και τα τρία περιμένουμε να έχουν σημαντικό overhead άρα εν τέλει να μην έχουμε ελάττωση του συνολικού χρόνου εκτέλεσης του προγράμματος μας. Το IPython και το Multiprocessing απαιτούν μετατροπή των δεδομένων σε λίστα κάτι που δεν είναι καθόλου φθηνό υπολογιστικά ενώ στο MPI έχουμε επιβάρυνση των πράξεων λόγω της ανταλλαγής διανυσμάτων ή πινάκων μέσω μηνυμάτων. Εμείς επιλέξαμε στην συγκεκριμένη διπλωματική εργασία να το υλοποιήσουμε με το multiprocessing.

Κεφάλαιο 3 : Υλοποίηση, Πειράματα και Συμπεράσματα

Υλοποίηση

Όπως έχουμε ήδη αναφέρει στην υλοποίηση μας χρησιμοποιήσαμε τα πακέτα numpy, scipy και multiprocessing. Η σειριακές μέθοδοι υλοποιήθηκαν στο πλαίσιο μίας συνάρτησης η κάθε μία όπως ακριβώς περιγράφηκαν στο πρώτο κεφάλαιο με τους αλγορίθμους τους. Ως προρυθμιστή πίνακα χρησιμοποιήσαμε τον Jacobi preconditioner που δεν περιμένουμε να βελτιώσει τα αποτελέσματα κατά πολύ αλλά και τον ευρέως χρησιμοποιούμενο που προκύπτει έπειτα από την εφαρμογή την incomplete cholesky παραγοντοποίησης. Για την εξαγωγή του πίνακα για την πρώτη περίπτωση χρησιμοποιήσαμε την ήδη ενσωματωμένη στην Python συνάρτηση M=numpy.diag(numpy.diag(A)) ενώ για την δεύτερη περίπτωση υλοποιήσαμε μία συνάρτηση ακολουθώντας τον αλγόριθμο για τον υπολογισμό του προρυθμιστή πίνακα που παρουσιάσαμε στον πρώτο κεφάλαιο.

Σε ό,τι αφορά την υλοποίηση του παράλληλου κομματιού χρησιμοποιήσαμε το module cProfile προκειμένου να διαπιστώσουμε ποια πράξη είναι αυτή που απαιτεί μεγαλύτερο χρόνο στην υλοποίηση μας και συμπεράναμε ότι η πιο συχνά εκτελούμενη πράξη είναι αυτή του εσωτερικού γινομένου και του γινόμενο πίνακα με διάνυσμα. Όμως η πράξη αυτή στην συγκεκριμένη γλώσσα προγραμματισμού που έχουμε επιλέξει είναι η ίδια πράξη οπότε με μία. Συνεπώς η βελτιστοποίηση μας ως προς την εκτέλεση της πράξης του dot του numpy πακέτου της Python.

Όπως έχουμε ήδη αναφέρει η πράξη αυτή είναι ήδη πάρα πολύ γρήγορη λόγω του ότι έχει γίνει διασύνδεση της υλοποίησης της με τις βιβλιοθήκες blas και lapack οι οποίες υλοποιούν τον αλγόριθμο βελτιστοποιημένα. Από αυτό και μόνο μπορούμε να συμπεράνουμε ότι δεν υπάρχουν αρκετά περιθώρια βελτίωσης. Ωστόσο, η βελτιστοποίηση την οποία εμείς επιχειρούμε στην υλοποίηση μας σχετίζεται με το μέγεθος του πίνακα και κατά πόσο το μέγεθος του πίνακα επηρεάζει τον χρόνο εκτέλεσης τόσο του συνολικού προγράμματος όσο και της πράξης αυτής καθεαυτής. Επιχειρούμε λοιπόν στην περίπτωση που έχουμε πράξη πίνακα με διάνυσμα να ανάγουμε την πράξη σε αυτή του εσωτερικού γινομένου, δηλαδή διάνυσμα με διάνυσμα. Αντίστοιχα, καταβάλλεται προσπάθεια ώστε να αναχθεί η πράξη εσωτερικού γινομένου σε απλό γινόμενο μεταξύ αριθμών και στην συνέχεια άθροιση αυτών. Λόγω του ότι χρησιμοποιήσαμε το πακέτο multiprocessing δεν είχαμε και μεγάλα περιθώρια δοκιμών ως προς την διαμέλιση του πίνακα, δηλαδή ως προς τον χωρισμό του σε μικρότερους πίνακες, οπότε και τον σπάσαμε σε διανύσματα δηλαδή ανά γραμμές.

Έχουμε ήδη εξηγήσει πως δουλεύει το multiprocessing και είναι εύκολο κανείς να κατανοήσει πως έχουμε κατασκευάσει δύο συναρτήσεις της μορφής:

```
def matrix_vector((max, vec)):          # Συνάρτηση για παράλληλη εκτέλεση
  return dot(max,vec)                   # πολλαπλασιασμού πίνακα με διάνυσμα
```

Προπαρασκευαστικό Στάδιο

Για την υλοποίηση με το συγκεκριμένο πακέτο για εκτέλεση παράλληλων πράξεων χρειάζεται πρώτα μία μετατροπή των δεδομένων σε λίστες με τα διανύσματα ή/και τις ακέραιες τιμές που θα δίνουμε σαν όρισμα κάθε φορά που καλούμε την συνάρτηση. Αυτή είναι μία αρκετά χρονοβόρα διαδικασία και παρακάτω όπου θα παρουσιάσουμε τα αποτελέσματα αλλά και τα συμπεράσματα που απορρέουν από αυτή την μελέτη αυτό θα γίνει ιδιαίτερα εμφανές. Επίσης, το ίδιο χρονοβόρα είναι και η αντίστροφη διαδικασία, δηλαδή η μετατροπή των αποτελεσμάτων από λίστες σε διανύσματα. Τέλος, στο προπαρασκευαστικό στάδιο ανήκει και η διαδικασία έναρξης και τερματισμού του συνόλου των διεργασιών. Όλα τα παραπάνω αναλύονται εκτενώς στις παραγράφους που θα ακολουθήσουν.

Πειράματα

Για τα πειράματα μας δυστυχώς δεν είχαμε στην διάθεση μας ένα σύστημα με τους διαθέσιμους πόρους που θα επιθυμούσαμε προκειμένου να δούμε διαφορές σε μεγάλα γραμμικά συστήματα ανάμεσα στους χρόνους εκτέλεσης. Τα δεδομένα που χρησιμοποιήσαμε είναι διαθέσιμα στις ιστοσελίδες του https://www.cise.ufl.edu/research/sparse/matrices/ και www.netlib.org . Οι πίνακες που χρησιμοποιήσαμε ήταν περίπου μεγέθους 2000 x 2000. Παρατηρήθηκε ότι η μεγαλύτερη βελτίωση χρονισμού επήλθε με την δημιουργία τεσσάρων νημάτων και προκειμένου να καταλήξουμε σε αυτό το συμπέρασμα έγιναν δοκιμές με 2, 4, 8 και 16 νήματα. Το υπολογιστικό σύστημα στο οποίο εκτελέστηκαν οι αλγόριθμοι που έχουμε περιγράψει μέχρι στιγμής έχει τα εξής χαρακτηριστικά:

CPU: Intel(R) Core(TM) i5-4300M CPU @2.6Ghz

2 cores - 4 physical threads

RAM: 4,00GB

System Type: 64-bit Operating System

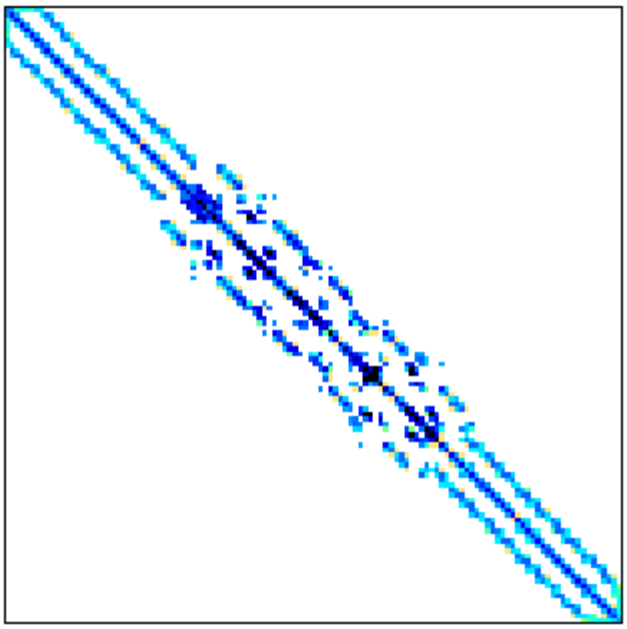
GPU: Intel® HD Graphics 4600

Αξίζει να επισημάνουμε σε αυτό το σημείο πως η χρονομέτρηση των πινάκων έγινε με την χρήση του πακέτου time της Python. Η τεχνική που εφαρμόσαμε είναι ότι κρατάμε τον αρχικό χρόνο δηλαδή το σημείο από το οποίο θέλουμε να αρχίσουμε την χρονομέτρηση της πράξης μας και στο τέλος τον αφαιρούμε από τον χρόνο κατά τον οποίο τερματίστηκε η εκτέλεση του προγράμματος μας ή της πράξης της οποίας θέλουμε να χρονομετρίσουμε.

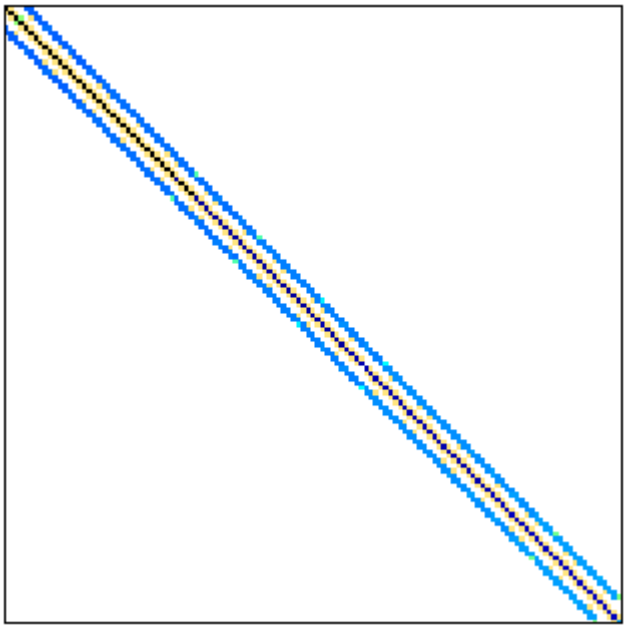
Τέλος, χρησιμοποιήσαμε το λογισμικό Pycharm για την ανάπτυξη του κώδικα. Πρόκειται για ένα πάρα πολύ χρήσιμο εργαλείο για κάθε προγραμματιστή Python. Διαθέτει Python κονσόλα, linux terminal, είναι συμβατό και με τα όλα τα λειτουργικά συστήματα (Windows, Linux, etc). Ακόμα, με αυτό το λογισμικό είναι δυνατή η χρήση κάποιου εργαλείου version control όπως svn ή git η άμεση αποσφαλμάτωση του κώδικα καθώς προσφέρει στον χρήστη ένα ενσωματωμένο εργαλείο για αυτήν την δουλειά που στο σύνολο του είναι πολύ εύχρηστο και ιδιαίτερα χρήσιμο. Επίσης, είναι εύκολη η εγκατάσταση νέων πακέτων της Python και η ενσωμάτωση αρχείων φακέλων και πακέτων, τα οποία εμείς δημιουργούμε στο project μας. Τέλος, προσφέρει στον προγραμματιστή την δυνατότητα refactoring του κώδικα με πολύ εύκολο τρόπο.

Πίνακες που χρησιμοποιήσαμε για τα πειράματα μας

Οι πίνακες που χρησιμοποιήσαμε για την διεξαγωγή των πειραμάτων είναι από δεδομένα της NASA, πρόκειται για πίνακες οι οποίοι είναι συμμετρικοί και θετικά ορισμένοι. Πιο συγκεκριμένα, ο πίνακας nasa1824 είναι διαστάσεων 1824 x 1824 και περιέχει 39208 μη μηδενικά στοιχεία ενώ ο πίνακας nasa2146 είναι διαστάσεων 2146 x 2146 και περιέχει 72250 μη μηδενικά στοιχεία. Τα δύο αυτά σύνολα δεδομένων που χρησιμοποιήθηκαν αποτελούν δεδομένα από structural προβλήματα.



Εικόνα 3.1: Μορφή πίνακα nasa1824



Εικόνα 3.2 Μορφή πίνακα nasa2146

Πίνακας αποτελεσμάτων – Χρόνοι Εκτέλεσης Πράξεων και Προγραμμάτων		
Method	Iterations	Total Time (seconds)
Conjugate Gradient	2	0.118000030518
Preconditioned Conjugate Gradient (Jacobi)	2	1.84200000763
Preconditioned Conjugate Gradient (Cholesky)	1	2.41600012779
Parallel Conjugate Gradient	2	0.2442639034880
Preconditioned Conjugate Gradient (Jacobi)	2	3.16299986839
Preconditioned Conjugate Gradient (Cholesky)	1	4.46499991417
Multiply Time	-	0.0019998550415
Parallel Multiplication	-	0.522000074387
Single Multiplication	-	3.947510739202966e-07

Πίνακας 3.1: nasa1824

Πίνακας αποτελεσμάτων – Χρόνοι Εκτέλεσης Πράξεων και Προγραμμάτων		
Method	Iterations	Total Time (seconds)
Conjugate Gradient	1	0.176000118256
Preconditioned Conjugate Gradient (Jacobi)	1	1.89800000191
Preconditioned Conjugate Gradient (Cholesky)	1	3.61599993706
Parallel Conjugate Gradient	1	0.36562094578
Preconditioned Conjugate Gradient (Jacobi)	1	3.81299996376
Preconditioned Conjugate Gradient (Cholesky)	1	5.75900006294
Multiply Time	-	0.0019998550415
Parallel Multiplication	-	0.580999851227
Single Multiplication	-	1.973755369601483e-06
Χρόνος για έναρξη Νημάτων		0.03
Χρόνος για λήξη Νημάτων	-	0.08

Πίνακας 3.2: nasa2146

Συμπεράσματα

Τα σύνολα δεδομένων τα οποία είχαμε την δυνατότητα να τρέξουμε στον υπολογιστή μας δεν επαρκούν ώστε να έχουμε μία πληρέστερη εικόνα για το ποσοστό βελτίωσης σε ότι αφορά τον απαιτούμενο αριθμό επαναλήψεων με κάθε μέθοδο. Ωστόσο τα αποτελέσματα είναι αναμενόμενα σε ότι αφορά τον αριθμό των επαναλήψεων που απαιτείται με κάθε μία από τις σειριακές μεθόδους. Τα συμπεράσματα είναι αντίστοιχα με αυτά των σειριακών μεθόδων ως προς τον συνολικό αριθμό επαναλήψεων απορρέουν και από την εφαρμογή των παράλληλων αλγορίθμων. Όπως έχουμε αναλύσει ήδη στα κεφάλαια που προηγήθηκαν η βελτίωση που αναμένεται να παρατηρηθεί σχετίζεται σε μεγάλο βαθμό με τον προρυθμιστή πίνακα που εφαρμόζουμε στο γραμμικό μας σύστημα. Επίσης, έχουμε ήδη επισημάνει ότι έχουμε εφαρμόσει ως προρυθμιστή τον καλύτερο και τον χειρότερο πίνακα. Όταν αυτός είναι ο πίνακας με τα στοιχεία της διαγωνίου του αρχικού τότε δεν περιμένουμε να σημειωθεί μεγάλη βελτίωση ενώ όταν εφαρμόζεται ένας καλύτερος πίνακας περιμένουμε να έχουμε μείωση στο σύνολο των επαναλήψεων όπως και φαίνεται να έχουμε. Βέβαια δεν θεωρούμε ότι το ποσοστό βελτίωσης που φαίνεται να έχουμε είναι αντιπροσωπευτικό της αναμενόμενης βελτίωσης διότι είναι πολύ υψηλό.

Το επόμενο μέτρο ως προς το οποίο σκοπεύουμε να σχολιάσουμε τα αποτελέσματα που λάβαμε είναι αυτό της χρονικής διάρκειας της εκτέλεσης του κώδικα ανάμεσα στην σειριακή και παράλληλη εκτέλεση των αλγορίθμων. Σε αυτή την περίπτωση δεν φαίνεται να έχουμε βελτίωση, αντίθετα έχουμε χειροτέρευση. Αυτό είναι κάτι το οποίο επίσης αναμέναμε, και συμβαίνει διότι είναι ιδιαίτερα χρονοβόρο το προπαρασκευαστικό στάδιο για την παράλληλη εκτέλεση, δηλαδή η μετατροπή των πινάκων σε λίστα από διανύσματα κάθε φορά. Ακόμα, σε ό,τι αφορά τον συνολικό χρόνο εκτέλεσης της πράξης βλέπουμε πως αυτός αυξάνεται σημαντικά αν όμως χρονομετρήσουμε την πράξη που έχουμε να εκτελέσουμε κάθε φορά στην παράλληλη υλοποίηση παρατηρούμε ότι αυτή καθεαυτή η πράξη απαιτεί κάποια μseconds. Αυτό που επιβαρύνει περεταίρω την συνολική πράξη είναι η κλήση της συνάρτησης αλλά και η επιστροφή από αυτή καθώς και η έναρξη και η λήξη, μαζί με την επιστροφή στο master νήμα.

Συμπερασματικά λοιπόν, δεν θεωρούμε ότι είναι δυνατή η παράλληλη υλοποίηση αυτών των αλγορίθμων και ταυτόχρονα η βελτίωση του χρόνου εκτέλεσης με αυτό το Python module διότι είναι ακριβές οι μετατροπές από λίστες σε διανύσματα, συνεπώς και η αντίστροφη αυτής. Επίσης, οι βελτιστοποιήσεις που μπορούν να εφαρμοστούν στους αλγορίθμους συζυγών κλήσεων αφορούν πράξεις οι οποίες ήδη τρέχουν βέλτιστα λόγω της διασύνδεσης τους με τις βιβλιοθήκες blas, lapack, mkl κτλ. Σαν μελλοντική συνέχιση της συγκεκριμένης μελέτης θα μπορούσε να επιχειρηθεί πάλι η παράλληλη υλοποίηση των συγκεκριμένων

μεθόδων με κάποιο άλλο πακέτο της Python. Ωστόσο επίσης δεν αναμένονται ιδιαίτερες βελτιστοποιήσεις λόγω της διασύνδεσης των πράξεων με τις προαναφερθείσες βιβλιοθήκες και εξαιτίας του ότι πάντα θα έχουμε κάποια χρονική επιβάρυνση λόγω του προπαρασκευαστικού σταδίου της εφαρμογής παράλληλων τεχνικών υλοποίησης. Τέλος, πιο συγκεκριμένα, με το IPython πακέτο αναμένονται παρόμοια αποτελέσματα όπως με το multiprocessing λόγω του ότι έχουν παρόμοια τεχνική εφαρμογής, ενώ με το MPI έχουμε επιβάρυνση λόγω της συνεχούς μετακίνησης των δεδομένων που στην περίπτωση μας είναι διανύσματα, κάτι που καθιστά την διαδικασία ιδιαίτερα ακριβή χρονικά.

Βιβλιογραφία

[1] Y. Saad “Iterative Methods for Sparse Linear Systems”, Second Edition (http://www-users.cs.umn.edu/~saad/IterMethBook_2ndEd.pdf)

[2]G.H.Golub-C.F Van Loan “Matrix Computations”, Third Edition (http://web.mit.edu/ehliu/Public/sclark/Golub%20G.H.,%20Van%20Loan%20C.F.-%20Matrix%20Computations.pdf)

[3] www.stackoverflow.com

[4] http://persson.berkeley.edu/228A/Fall10/doc/lec41-2x3.pdf

[5] http://stanford.edu/class/ee364b/lectures/conj_grad_slides.pdf

[6] http://www.eecs.berkeley.edu/~binetude/course/cs267/assignment3/assignment3.htm

[7] www.netlib.org

[8] https://www.cise.ufl.edu/research/sparse/matrices/

[9] http://inf-server.inf.uth.gr/courses/CE213/

[10] http://inf-server.inf.uth.gr/courses/CE410/

[11] https://docs.python.org/

[12] Introduction to MPI with MPI4Py (http://coco.sam.pitt.edu/~emeneses/wp-content/uploads/2013/11/introMPIwithMPI4Py.pdf)

[13] Concurrency Kung-fu: Python Style (https://python.g-node.org/python-summerschool-2012/_media/parallel_talk.pdf)

[14] https://software.intel.com/en-us/intel-mkl

[15] http://ark.intel.com