ΔΗΜΟΚΡΙΤΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΡΑΚΗΣ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Τεχνολογία Παράλληλης Επεξεργασίας

Singular Value Decomposition

Παπαιωάννου Νικόλαος

Σανδάλης Γεώργιος Μιλτιάδης

Τοκατλίδης Γεώργιος

Εργαστήριο Αρχιτεκτονικής Υπολογιστών

και Συστημάτων Υψηλών Επιδόσεων

4ο έτος

Θεωρητική Ανάλυση του SVD

Στην γραμμική άλγεβρα πινάκων αν υποθέσουμε ότι A είναι ένας $N \times N$ πίνακας εισόδου τότε μπορεί να αναλυθεί σε ένα γινόμενο τριών πινάκων.

$$\boldsymbol{A = U\Sigma V^T}$$

Αυτή η ανάλυση ονομάζεται Singular Value Decomposition (SVD). Ας δούμε λεπτομερώς τον ρόλο του κάθε πίνακα :

Ορθογώνιος Πίνακας U :

- **Περιγραφή:** Ο πίνακας U είναι ένας ορθογώνιος πίνακας $N \times N$. Ένας πίνακας είναι ορθογώνιος όταν οι στήλες του αποτελούν ορθοκανονικά διανύσματα, δηλαδή έχουν μοναδιαίο μέγεθος και είναι κάθετες μεταξύ τους.

- **Ιδιότητες:** Οι στήλες του U είναι τα ιδιοδιανύσματα του πίνακα AA^T.

- **Εφαρμογή:** Ο πίνακας U χρησιμοποιείται για να περιστρέψει το αρχικό σύστημα συντεταγμένων έτσι ώστε οι στήλες του να ευθυγραμμίζονται με τις κύριες κατευθύνσεις του A

Διαγώνιος Πίνακας Σ :

- **Περιγραφή:** Ο πίνακας Σ είναι ένας $N \times N$ διαγώνιος πίνακας. Αυτό σημαίνει ότι όλα τα στοιχεία εκτός της κύριας διαγωνίου είναι μηδενικά.

- **Ιδιότητες:** Τα στοιχεία της κύριας διαγωνίου του Σ είναι οι Singular Values του πίνακα A, οι οποίες είναι μη αρνητικές και συνήθως ταξινομημένες κατά φθίνουσα σειρά.

- **Εφαρμογή:** Οι Singular Values εκφράζουν την κλίμακα κατά μήκος των κύριων αξόνων μετά την περιστροφή και την αναδιάταξη που εφαρμόζεται από τους πίνακες U και V^T.

Ορθογώνιος Πίνακας V^T:

- **Περιγραφή:** Ο πίνακας V^T είναι ο ανάστροφος του $N \times N$ ορθογώνιου πίνακα V.

- **Ιδιότητες:** Οι στήλες του V είναι τα ιδιοδιανύσματα του πίνακα $A^T A$.

- **Εφαρμογή:** Ο πίνακας V χρησιμοποιείται για να περιστρέψει το αρχικό σύστημα συντεταγμένων στο χώρο των χαρακτηριστικών του A.

Εφαρμογές του SVD

1. **Συμπίεση εικόνας με τη χρήση του SVD**

   Τι είναι η Συμπίεση Εικόνας;
   Η συμπίεση εικόνας είναι η διαδικασία μείωσης του μεγέθους μιας εικόνας, με σκοπό την αποθήκευση ή τη μετάδοση της με λιγότερα δεδομένα, διατηρώντας όσο το δυνατόν περισσότερο την ποιότητά της.

   Πώς χρησιμοποιείται το SVD για τη Συμπίεση Εικόνας;

   1. **Μετατροπή της Εικόνας σε Πίνακα:**

- Μια εικόνα μπορεί να αναπαρασταθεί ως ένας πίνακας όπου κάθε στοιχείο του πίνακα αντιπροσωπεύει την ένταση του φωτός ή το χρώμα ενός εικονοστοιχείου (pixel).

   2. **Υπολογισμός του SVD:**

- Εκτελούμε την ανάλυση SVD στον πίνακα της εικόνας A , δηλαδή $A = U\Sigma V^T$. Αυτό διασπά την εικόνα σε τρεις πίνακες: τον U, τον Σ, και τον V^T.

3. **Περιορισμός των Singular Values:**

- Οι Singular Values που βρίσκονται στον πίνακα Σ συχνά ταξινομούνται κατά φθίνουσα σειρά. Για τη συμπίεση, κρατάμε μόνο τις μεγαλύτερες $\boldsymbol{k}$ Singular values και μηδενίζουμε τις υπόλοιπες. Αυτό έχει ως αποτέλεσμα έναν νέο, μειωμένων διαστάσεων πίνακα Σ'.

4. **Ανακατασκευή της Εικόνας:**

- Χρησιμοποιώντας τους μειωμένων διαστάσεων πίνακες U, Σ' και V^T, ανακατασκευάζουμε τον πίνακα της εικόνας: $A' = U\Sigma'V^T$. Αυτή η νέα εικόνα είναι μια προσέγγιση της αρχικής αλλά με πολύ λιγότερα δεδομένα.

Παράδειγμα:

Ας υποθέσουμε ότι έχουμε μια εικόνα 270×270 pixels. Μετά την ανάλυση SVD, επιλέγουμε να διατηρήσουμε μόνο τις πρώτες 20 singular values αντί για 270, μειώνοντας σημαντικά το μέγεθος της εικόνας. Αυτό , έχει ως αποτέλεσμα την **μείωση χρόνου αποθήκευσης** και την **μείωση της ταχύτητας μετάδοσης.** Παρακάτω παρουσιάζονται δύο εικόνες, η αριστερή πλευρά δείχνει την αρχική εικόνα, ενώ η δεξιά δείχνει την εικόνα μετά από συμπίεση με χρήση 20 Singular values.

Original ($k = 270$)

$k = 20$

2. Ανάλυση Κύριων Συνιστωσών (PCA)

Η Ανάλυση Κύριων Συνιστωσών (Principal Component Analysis, PCA) είναι μια τεχνική στατιστικής που χρησιμοποιείται για τη μείωση διαστάσεων, δηλαδή για τη μείωση του αριθμού των παραμέτρων σε ένα σύνολο δεδομένων χωρίς σημαντική απώλεια της πληροφορίας. Το SVD είναι ένα κεντρικό εργαλείο σε αυτή τη διαδικασία, παρέχοντας την αριθμητική μέθοδο για την εξαγωγή των κύριων συνιστωσών.

Βασικές Έννοιες της PCA

- **Στόχος:** Η PCA στοχεύει στην εύρεση των κατευθύνσεων (κύριων συνιστωσών) στις οποίες τα δεδομένα παρουσιάζουν τη μέγιστη διακύμανση και στη μείωση του διαστασιακού χώρου των δεδομένων, διατηρώντας το μεγαλύτερο δυνατό ποσοστό της συνολικής διακύμανσης.

- **Κύριες Συνιστώσες:** Οι κύριες συνιστώσες είναι οι νέες μεταβλητές (γραμμικοί συνδυασμοί των αρχικών μεταβλητών) που προκύπτουν μέσω της PCA και είναι διατεταγμένες κατά φθίνουσα σειρά διακύμανσης.

Πώς Εφαρμόζεται η PCA με Χρήση του SVD

Η διαδικασία της PCA μέσω SVD περιλαμβάνει τα εξής βήματα:

1. **Κεντροποίηση Δεδομένων:**

   - Αρχικά, αφαιρούμε τον μέσο όρο κάθε χαρακτηριστικού από τα δεδομένα, ώστε να μετασχηματίσουμε τα δεδομένα σε έναν κεντροποιημένο πίνακα X.

2. **Υπολογισμός του SVD:**

   - Εφαρμόζουμε SVD στον κεντροποιημένο πίνακα δεδομένων X :

$$\boldsymbol{X = U\Sigma V^T}$$

3. **Εξαγωγή των Κύριων Συνιστωσών:**

- Οι κύριες συνιστώσες είναι οι στήλες του πίνακα V, και κάθε κύρια συνιστώσα είναι μια γραμμική συνδυαστική των αρχικών μεταβλητών.

- Οι Singular values στον πίνακα Σ δίνουν την ποσότητα διακύμανσης που εξηγείται από κάθε κύρια συνιστώσα.

4. **Επιλογή των Σημαντικότερων Κύριων Συνιστωσών:**

- Για μείωση διαστάσεων, επιλέγουμε τις πρώτες k κύριες συνιστώσες που εξηγούν το μεγαλύτερο ποσοστό της συνολικής διακύμανσης.
- Τα δεδομένα προβάλλονται στον χώρο των πρώτων k κύριων συνιστωσών, δημιουργώντας έναν νέο πίνακα δεδομένων μειωμένης διάστασης.

Συμπερασματικά, Το Singular Value Decomposition (SVD) είναι εξαιρετικά χρήσιμο στην Ανάλυση Κύριων Συνιστωσών (PCA) λόγω της αριθμητικής του σταθερότητας και της ικανότητάς του να χειρίζεται μεγάλους πίνακες δεδομένων παρέχοντας ακριβή αποτελέσματα.

Όπως γίνεται αντιληπτό, η ανάλυση SVD έχει πάρα πολλές εφαρμογές σε διάφορα πεδία. Αυτή η μέθοδος δεν είναι μόνο ουσιαστική για την συμπίεση εικόνων, μειώνοντας το μέγεθος αρχείων χωρίς σημαντική απώλεια ποιότητας, αλλά και στην Ανάλυση Κύριων Συνιστωσών (PCA), που χρησιμοποιείται για τη μείωση διαστάσεων σε σύνολα δεδομένων. Επιπλέον, το SVD είναι πολύτιμο για την επίλυση προβλημάτων ελαχίστων τετραγώνων, την ανάκτηση πληροφοριών από θορυβώδεις πίνακες και γενικά για την αποδοτική διαχείριση και ανάλυση μεγάλων δεδομένων, κάνοντάς το ένα ανεκτίμητο εργαλείο στη σύγχρονη υπολογιστική επιστήμη.

Σύγρχονοι Αλγόριθμοι Υπολογισμού του SVD

Υπάρχουν πολλές διαφορετικές υλοποιήσεις για τον υπολογισμό του Singular Value Decomposition (SVD), καθεμία με τα δικά της πλεονεκτήματα και μειονεκτήματα. Οι διάφοροι αλγόριθμοι επιτρέπουν την προσαρμογή της μεθόδου στις εκάστοτε ανάγκες και περιορισμούς των δεδομένων και της εφαρμογής. Ορισμένοι αλγόριθμοι παρέχουν καλύτερη υπολογιστική σταθερότητα, ενώ άλλοι πλεονεκτούν στην επεξεργασία μεγάλων πινάκων. Μερικοί απαιτούν συγκεκριμένη μορφή εισόδου, προϋποθέτοντας περαιτέρω προ-επεξεργασία, ενώ άλλοι είναι πιο απλοί στην κατανόηση και χρήση. Οι κύριοι αλγόριθμοι περιλαμβάνουν:

1. Κλασικός Αλγόριθμος SVD (Golub-Kahan-Reinsch Algorithm)

Ο παραδοσιακός αλγόριθμος αποτελείται από δύο στάδια:

- **Μείωση σε Τριγωνική Μορφή**: Ο πίνακας μειώνεται σε αμφιδιαγώνια μορφή χρησιμοποιώντας αλγόριθμο Householder.
- **Διαγώνιωση**: Η αμφιδιαγώνια μορφή διαγωνιοποιείται χρησιμοποιώντας επαναληπτικούς μετασχηματισμούς.

2. Αλγόριθμος Lanczos

Αποδοτικός για μεγάλους και αραιούς πίνακες. Χρησιμοποιεί επαναληπτική διαδικασία για τη μείωση του πίνακα σε τριγωνική μορφή και εξαγωγή των singular values.

- **Πλεονεκτήματα**: Ιδανικός για μεγάλους αραιούς πίνακες.
- **Μειονεκτήματα**: Μπορεί να παρουσιάσει αριθμητική αστάθεια

3. Αλγόριθμος Randomized SVD

Χρησιμοποιεί τυχαίους προβολείς για να επιταχύνει τη διαδικασία υπολογισμού του SVD.

- **Βήματα**:
  - Εφαρμογή τυχαίου προβολέα στον πίνακα.
  - Μείωση διάστασης του πίνακα.
  - Υπολογισμός SVD στον μειωμένο πίνακα.
  - Ανακατασκευή του SVD από τον μειωμένο πίνακα.

- **Πλεονεκτήματα**: Ταχύτερος για μεγάλους πίνακες με μικρό αριθμό σημαντικών singular values.
- **Μειονεκτήματα**: Απαιτεί παραμετροποίηση και μπορεί να είναι λιγότερο ακριβής από τις παραδοσιακές μεθόδους.

4. Truncated SVD

Χρησιμοποιείται όταν ενδιαφερόμαστε μόνο για τις πρώτες k singular values.

- **Πλεονεκτήματα**: Αποδοτικός για αναλύσεις όπου οι πρώτες k συνιστώσες είναι αρκετές.
- **Μειονεκτήματα**: Περιορίζεται σε αναλύσεις που αφορούν τις πρώτες k συνιστώσες.

5. Divide and Conquer SVD

Διασπά τον πίνακα σε μικρότερα υποπροβλήματα, υπολογίζει το SVD των υποπροβλημάτων και συνδυάζει τα αποτελέσματα.

- **Πλεονεκτήματα**: Αποδοτικός για μεγάλους πίνακες.
- **Μειονεκτήματα**: Περίπλοκη υλοποίηση.

6. QR Αλγόριθμος

Επαναληπτική μέθοδος που διασπά έναν πίνακα σε ορθογώνια και ανώτερη τριγωνική μορφή και επαναλαμβάνει τη διαδικασία μέχρι να συγκλίνει.

- **Πλεονεκτήματα**: Σταθερότητα και ακρίβεια.
- **Μειονεκτήματα**: Μπορεί να είναι αργός για πολύ μεγάλους πίνακες.

7. Jacobi SVD Αλγόριθμος

Χρησιμοποιεί περιστροφές για να μετασχηματίσει έναν πίνακα σε διαγώνια μορφή.

- **Πλεονεκτήματα**: Υψηλή ακρίβεια.
- **Μειονεκτήματα**: Μπορεί να απαιτεί μεγάλο χρόνο υπολογισμού για μεγάλα σύνολα δεδομένων.

Μέθοδος Jacobi για Singular Value Decomposition

Ο αλγόριθμος Jacobi ξεχωρίζει ως η ιδανική επιλογή λόγω των σημαντικών πλεονεκτημάτων του έναντι άλλων μεθόδων. Συγκεκριμένα, προσφέρει εξαιρετική ακρίβεια και αριθμητική σταθερότητα, καθιστώντας τον ιδανικό για την ανάλυση μεγάλων και κακοπροσδιορισμένων πινάκων. Η εφαρμογή διαδοχικών περιστροφών Givens μειώνει τα εκτός διαγωνίου στοιχεία, διασφαλίζοντας αξιόπιστα αποτελέσματα. Επιπλέον, η σχετικά απλή υλοποίησή του συγκριτικά με άλλες τεχνικές καθιστά τον αλγόριθμο Jacobi προσιτό και αποτελεσματικό, χωρίς να απαιτεί εξειδικευμένες γνώσεις στην αριθμητική ανάλυση.

Βήματα της μεθόδου Jacobi

Αρχικοποίηση

1. Έστω ο πίνακας A με διαστάσεις $m \times n$:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

2. Αρχικοποίηση των ορθογώνιων πινάκων U και V ως μοναδιαίους πίνακες:

$$U = I_m , \quad V = I_n$$

Υπολογισμός περιστροφών Givens

Για κάθε ζεύγος (i, j) όπου $i \neq j$, υπολογίζουμε την περιστροφή $G(i, j, \theta)$ που θα μηδενίσει τα εκτός διαγώνιού στοιχεία.

1. Υπολογισμός της γωνίας περιστροφής θ :

$$\theta = \frac{1}{2}\arctan\left(\frac{2_{aij}}{a_{ii} - a_{jj}}\right)$$

2. Σχηματισμός του πίνακα περιστροφής Givens :

$$G(i,j,\theta) = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix}$$

3.Ενημέρωση των πινάκων $A, U\ και\ V$ με την περιστροφή :

$$A' = G^T A G, \quad U' = UG, \quad V' = VG$$

Επαναληπτική Διαδικασία

Η διαδικασία αυτή επαναλαμβάνεται για όλα τα ζεύγη (i,j) μέχρι τα εκτός διαγωνίου στοιχεία του πίνακα Aνα είναι αρκετά μικρά (κοντά στο μηδέν) σύμφωνα με ένα προκαθορισμένο κατώφλι ανοχής.

$$\text{Επανάληψη μέχρι } \mathbf{max_{i \neq j} |a_{ij}| < \varepsilon}$$

<u>Παράδειγμα υλοποίησης αλγορίθμου Jacobi</u>

Έστω ότι έχουμε έναν πίνακα A :

$$A = \begin{pmatrix} 4 & 0 & 2 \\ 3 & 1 & 0 \\ 0 & 1 & 3 \end{pmatrix}$$

Αρχικοποιούμε τους πίνακες $U\ και\ V$ ως μοναδιαίους :

$$\mathrm{U} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad \mathrm{V} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Πρώτη Επανάληψη

- Για το στοιχείο α_{12} υπολογίζουμε την γωνία θ :

$$\theta = \frac{1}{2}\arctan\left(\frac{2 \times 0}{4-1}\right) = 0$$

Παρατηρούμε ότι δεν χρείαζεται περιστροφή γιαυτό το στοιχείο καθώς είναι ήδη μηδέν.

- Για το στοιχείο α_{13} υπολογίζουμε την γωνία :

$$\theta = \frac{1}{2}\arctan\left(\frac{2\times 2}{4-3}\right) = \frac{1}{2}\arctan(4)$$

Στη συνέχεια αφού υπολογίσουμε το $\cos(\theta)$ $και$ $\sin(\theta)$ σχηματίζουμε τον πίνακα περιστροφής Givens :

$$G(1,3,\theta) = \begin{pmatrix} \cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{pmatrix}$$

Τέλος, εφαρμόζουμε την περιστροφή:

$$A' = G^T A G, \quad U' = UG, \quad V' = VG$$

Αυτή η διαδικασία επαναλαμβάνεται για όλα τα ζεύγη (i,j) μέχρι να μηδενιστούν όλα τα στοιχεία που δεν βρίσκονται στην διαγώνιο.

Παρουσίαση του σειριακού κώδικα

Ο κώδικας περιλαμβάνει τις εξής κύριες λειτουργίες:

1. **Δημιουργία Πινάκων**
2. **Αρχικοποίηση Πίνακα A**
3. **Υπολογισμός SVD με τη Μέθοδο Jacobi**
4. **Ανακατασκευή του Πίνακα A**
5. **Απελευθέρωση Μνήμης**

Ακολουθεί η ανάλυση του κάθε βήματος:

1. Δημιουργία Πινάκων

Αυτό το κομμάτι του κώδικα δεσμεύει την μνήμη για τους πίνακες που θα χρησιμοποιηθούν στην SVD.

```
double **U,**V, *S,**U_t, **V_t, **A, **temp, **S_mat,
**reconA;
    double alpha, beta, gamma, c, zeta, t,s,sub_zeta,
converge;
      int *I1, *I2;

   int acum = 0;
    int temp1, temp2;
   converge = 1.0;

   S_mat = new double*[N];
   reconA = new double*[N];
   temp = new double*[N];
   U = new double*[N];
   V = new double*[N];
   U_t = new double*[N];
   V_t = new double*[N];
   A = new double*[N];
   S = new double[N];
   I1= new int[N];
   I2= new int[N];

   for(int i =0; i<N; i++){
      S_mat[i]= new double [N];
      temp[i]= new double[N];
      reconA[i]= new double[N];
      U[i] = new double[N];
      V[i] = new double[N];
      U_t[i] = new double[N];
      V_t[i] = new double[N];
      A[i] = new double[N];
   }
```

2. Αρχικοποίηση Πίνακα A

Για να εξασφαλίσουμε ότι η μέθοδος πρέπει να συγκλίνει στον ίδιο αριθμό επαναλήψεων πρέπει κάνουμε τα πειράματα σε σταθερό πίνακα εισόδου. Αυτό το πετυχαίνουμε δημιουργώντας αρχικά τους πίνακες με τυχαίο τρόπο με την συνάρτηση $'fillMatrixWithRandomValues'$. Η οποία γεμίζει τον πίνακα A με τυχαίες τιμές ανάμεσα στις τιμές -49 και 49. Αυτό τον πίνακα έπειτα τον αποθηκεύουμε σε αρχεία τύπου .data τα οποία επιτρέπουν εύκολη είσοδο έξοδο για πίνακες.

```
srand(static_cast < unsigned int > (time(0)));
fillMatrixWithRandomValues(A);
```

Αφού έχουμε στην διάθεση μας τους τυχαίους πίνακες, τους περνάμε στον ίδιο φάκελο όπου βρίσκεται το εκτελέσιμο του κώδικα και χρησιμοποιούμε την παρακάτω συνάρτηση για να τους διαβάσουμε.

```
int readMatrixFromMemory(double ** A, char * filename){

    // Open the file in read mode
    ifstream inFile(filename);
    if (!inFile.is_open()) {
        cerr << "Error opening file for reading" << endl;
    return 1;
    }

    // Read the matrix from the file

    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            inFile >> A[i][j];
        }
    }

    // Close the file
    inFile.close();
}
```

3. Υπολογισμός SVD με τη Μέθοδο Jacobi

Σε αυτό το τμήμα, πραγματοποιείται ο υπολογισμός του SVD χρησιμοποιώντας τη μέθοδο Jacobi. Η διαδικασία είναι επαναληπτική και εφαρμόζει διαδοχικές περιστροφές Givens στους πίνακες μέχρι η σύγκλιση να είναι μικρότερη από μια προκαθορισμένη τιμή (epsilon).

3.1 Αρχικοποίηση Πινάκων και Μεταβλητών

Αρχικά, οι πίνακες U^T και V^T αρχικοποιούνται. Ο πίνακας U^T παίρνει τις τιμές του A^T, ενώ ο πίνακας V^T αρχικοποιείται ως μοναδιαίος πίνακας.

```
transpose(A, U_t);

for (int i = 0; i < M; i++) {
    for (int j = 0; j < N; j++) {
        if (i == j) {
            V_t[i][j] = 1.0;
        } else {
            V_t[i][j] = 0.0;
        }
    }
}
```

Η αρχικοποίηση του U^T με τον ανάστροφο του A σημαίνει ότι ο πίνακας U^T ξεκινά με μια αντιπροσωπευτική δομή των δεδομένων του A. Αυτό παρέχει μια καλή αρχική εκτίμηση για τον υπολογισμό της SVD, επιτρέποντας στις περιστροφές Givens να συγκλίνουν γρηγορότερα προς τις ορθογώνιες βάσεις.

3.2 Υπολογισμός Παραμέτρων

Στο τμήμα του κώδικα που ακολουθεί, οι παράμετροι alpha, beta, και gamma :

```
for(int i = 1; i<M; i++){
    for(int j = 0; j<i; j++){

        alpha = 0.0;
        beta = 0.0;
        gamma = 0.0;

        for(int k = 0; k<N ; k++){
            alpha = alpha + (U_t[i][k] * U_t[i][k]);
            beta = beta + (U_t[j][k] * U_t[j][k]);
            gamma = gamma + (U_t[i][k] * U_t[j][k]);
        }
```

Αναλυτική Επεξήγηση των Παραμέτρων

- $alpha$: Το άθροισμα των τετραγώνων των στοιχείων της στήλης i του πίνακα U_t.

$$\boldsymbol{alpha} = \sum_{\boldsymbol{k=0}}^{\boldsymbol{N-1}} (\boldsymbol{U_t[i][k]})^2$$

Η παράμετρος $alpha$ υπολογίζεται ως το άθροισμα των τετραγώνων των στοιχείων της στήλης i του πίνακα U_t. Αυτός ο υπολογισμός αντιπροσωπεύει το μέτρο της στήλης i, και είναι σημαντικός για τον προσδιορισμό της συμβολής της στήλης στη συνολική δομή του πίνακα.

- $beta$: Το άθροισμα των τετραγώνων των στοιχείων της στήλης j του πίνακα U_t.

$$beta = \sum_{k=0}^{N-1} (U_t[j][k])^2$$

Η παράμετρος $beta$ υπολογίζεται αντίστοιχα με την $alpha$, αλλά για τη στήλη j του πίνακα U_t. Όπως και η $alpha$, η $beta$ μετράει τη συμβολή της στήλης j στη συνολική δομή του πίνακα.

- $gamma$: Το εσωτερικό γινόμενο των στοιχείων των στηλών i και j του πίνακα U_t.

$$gamma = \sum_{k=0}^{N-1} (U_t[i][k] \cdot U_t[j][k])$$

Η παράμετρος $gamma$ είναι το εσωτερικό γινόμενο (dot product) των δύο στηλών i και j του πίνακα U_t. Αυτός ο υπολογισμός μας δίνει μια μέτρηση του πόσο ορθογώνιες (ή μη) είναι οι δύο στήλες μεταξύ τους. Όταν οι στήλες είναι ορθογώνιες, το εσωτερικό τους γινόμενο θα είναι μηδέν. Όταν οι στήλες είναι γραμμικά εξαρτώμενες, το γινόμενο θα είναι μέγιστο.

Ρόλος των παραμέτρων

- **Alpha και Beta**: Οι παράμετροι alpha και beta χρησιμοποιούνται για να κανονικοποιήσουν τη συνεισφορά κάθε στήλης, επιτρέποντας την ορθή εκτίμηση της γωνίας περιστροφής και της αναλογίας των στοιχείων τους.
- **Gamma**: Η παράμετρος gamma είναι κρίσιμη για τον υπολογισμό της γωνίας περιστροφής μεταξύ των στηλών. Η τιμή της gamma σε σχέση με alpha και beta καθορίζει πόσο μακριά είναι οι στήλες από το να είναι ορθογώνιες και ως εκ τούτου πόσο πρέπει να περιστρέφουν για να επιτευχθεί ορθογωνιότητα.

  Η χρήση αυτών των παραμέτρων διασφαλίζει ότι οι περιστροφές Jacobi οδηγούν σε έναν πίνακα που είναι πιο κοντά στην ορθογώνια μορφή του, επιταχύνοντας έτσι τη σύγκλιση της μεθόδου προς τη λύση SVD.

3.3 Υπολογισμός Σύγκλισης και Γωνιών Περιστροφής

Σε αυτό το σημείο του κώδικα, υπολογίζεται η τιμή της σύγκλισης και οι γωνίες περιστροφής, οι οποίες είναι απαραίτητες για την ενημέρωση των πινάκων U_t και V_t:

```
converge = max(converge, abs(gamma)/sqrt(alpha*beta));
//compute convergence

//basicaly is the angle

//between column i and j

zeta = (beta - alpha) / (2.0 * gamma);
t = sgn(zeta) / (abs(zeta) + sqrt(1.0 + (zeta*zeta)));
//compute tan of angle
c = 1.0 / (sqrt (1.0 + (t*t)));
//extract cos
s = c*t;
//extract sin
```

Εξήγηση Παραμέτρων και υπολογισμών

- **converge**: Η μεταβλητή **converge** ενημερώνεται με τη μέγιστη τιμή του $|\gamma|/\sqrt{\alpha \cdot \beta}$. Αυτός ο λόγος αντιπροσωπεύει τη γωνία μεταξύ των στηλών i και j και χρησιμοποιείται για να καθορίσει πόσο κοντά είναι ο πίνακας στη σύγκλιση.

$$converge = \max\left(converge, \frac{|\gamma|}{\sqrt{\alpha \cdot \beta}}\right)$$

- **zeta**: Ο ενδιάμεσος υπολογισμός **zeta** βοηθά στον υπολογισμό της γωνίας περιστροφής. Εκφράζεται ως:

$$\zeta = \frac{\beta - \alpha}{2 \cdot \gamma}$$

- **t**: Η εφαπτομένη της γωνίας περιστροφής, t, υπολογίζεται χρησιμοποιώντας το πρόσημο της zeta και το απόλυτο της τιμής της. Αυτή η προσέγγιση επιτρέπει την εύρεση της μικρότερης γωνίας περιστροφής που απαιτείται για την ορθογώνια θέση των στηλών.

$$t = \frac{\mathrm{sgn}(\zeta)}{|\zeta| + \sqrt{1+\zeta^2}}$$

Όπου '$\boldsymbol{sgn(zeta)}$' είναι η συνάρτηση πρόσημου, που επιστρέφει 1 για θετικές τιμές, -1 για αρνητικές και 0 για μηδέν.

- **c** : Το συνημίτονο της γωνίας περιστροφής, c, υπολογίζεται από την εφαπτομένη t. Εκφράζεται ως:

$$c = \frac{1}{\sqrt{1+t^2}}$$

Το **c** αναπαριστά το ποσοστό διατήρησης της αρχικής τιμής της στήλης κατά την περιστροφή.

- **s**: Το ημίτονο της γωνίας περιστροφής, s, υπολογίζεται ως το γινόμενο του c με την εφαπτομένη t:

$$s = c \cdot t$$

Το s αναπαριστά το ποσοστό της τιμής της άλλης στήλης που προστίθεται στη στήλη κατά την περιστροφή.

Ρόλος των παραμέτρων στη Σύγκλιση

- **converge**: Παρακολουθεί την πρόοδο της σύγκλισης. Όταν η converge είναι μικρότερη από ένα προκαθορισμένο όριο **epsilon**, ο αλγόριθμος θεωρείται ότι έχει συγκλίνει.
- **zeta**: Καθορίζει την ακρίβεια και την απόδοση της γωνίας περιστροφής. Μικρότερες τιμές του zeta οδηγούν σε μικρότερες γωνίες, μειώνοντας την πιθανότητα μεγάλων αριθμητικών σφαλμάτων.
- **t, c, s**: Οι τιμές αυτές χρησιμοποιούνται για την πραγματική περιστροφή των πινάκων. Το c και το s ειδικά εξασφαλίζουν ότι οι περιστροφές διατηρούν την ορθογώνια μορφή των πινάκων όσο το δυνατόν περισσότερο.
  Ο ακριβής υπολογισμός και η εφαρμογή αυτών των παραμέτρων

επιταχύνει τη σύγκλιση του αλγορίθμου, διασφαλίζοντας ταυτόχρονα την ακρίβεια των αποτελεσμάτων.

3.4 Εφαρμογή Περιστροφών Givens

Οι περιστροφές Givens εφαρμόζονται σειριακά στους πίνακες U_t και V_t, ενημερώνοντας τις τιμές τους βάσει των υπολογισμένων γωνιών περιστροφής. Αυτό το βήμα είναι κρίσιμο για την επίτευξη της ορθογωνιοποίησης και της σύγκλισης των πινάκων.

Κώδικας και ανάλυση :

```
//Apply rotations on U and V
        for(int k=0; k<N; k++){
            t = U_t[i][k];
            U_t[i][k] = c*t - s*U_t[j][k];
            U_t[j][k] = s*t + c*U_t[j][k];

            t = V_t[i][k];
            V_t[i][k] = c*t - s*V_t[j][k];
            V_t[j][k] = s*t + c*V_t[j][k];
        }
    }
}
```

Εφαρμογή των Περιστροφών

Οι περιστροφές Givens ενημερώνουν τις τιμές των πινάκων U_t και V_t χρησιμοποιώντας τους υπολογισμένους παράγοντες c (συνημίτονο της γωνίας περιστροφής) και s (ημίτονο της γωνίας περιστροφής). Αυτές οι περιστροφές βοηθούν στην επίτευξη της ορθογωνιοποίησης των πινάκων.

Για κάθε στοιχείο k των πινάκων U_t και V_t, πραγματοποιούνται οι εξής ενημερώσεις:

Ενημέρωση Πίνακα U_t :

- Για το στοιχείο $U_t[i][k]$:

$$U_t[i][k] \leftarrow c \cdot U_t[i][k] - s \cdot U_t[j][k]$$

- Για το στοιχείο $U_t[j][k]$:

$$U_t[j][k] \leftarrow s \cdot U_t[i][k] - c \cdot U_t[j][k]$$

Ενημέρωση Πίνακα V_t :

- Για το στοιχείο $V_t[i][k]$:

$$V_t[i][k] \leftarrow c \cdot V_t[i][k] - s \cdot V_t[j][k]$$

- Για το στοιχείο $V_t[j][k]$:

$$V_t[j][k] \leftarrow s \cdot V_t[i][k] - c \cdot V_t[j][k]$$

Σημασία της Σύγκλισης και των Περιστροφών

Η ακριβής εφαρμογή των περιστροφών είναι ουσιώδης για τη σύγκλιση του αλγορίθμου. Οι τιμές c και s επιλέγονται με τέτοιο τρόπο ώστε να ελαχιστοποιούν την απόκλιση μεταξύ των στηλών, εξασφαλίζοντας την προοδευτική ορθογωνιοποίηση των πινάκων U_t και V_t .

Η συνεχής ενημέρωση των πινάκων U_t και V_t με αυτές τις περιστροφές οδηγεί σταδιακά στη μείωση της τιμής σύγκλισης converge, έως ότου αυτή γίνει μικρότερη από μια προκαθορισμένη τιμή epsilon, υποδεικνύοντας ότι ο αλγόριθμος έχει συγκλίνει.

3.5 Δημιουργία του Πίνακα S

Σε αυτό το βήμα, ο πίνακας S δημιουργείται από τις τιμές του πίνακα U_t μετά την εφαρμογή των περιστροφών Givens. Ο πίνακας S είναι ένας διαγώνιος πίνακας που περιέχει τις μοναδιαίες τιμές (singular values) του αρχικού μητρικού πίνακα A.

Επεξήγηση της Διαδικασίας

1.Υπολογισμός της Ευκλείδειας Νορμας των Στηλών του Πίνακα U_t :

Για κάθε στήλη i του πίνακα $\boldsymbol{U_t}$, υπολογίζεται η ευκλείδεια νόρμα, η οποία αντιστοιχεί στη μοναδιαία τιμή $\boldsymbol{\sigma_i}$ (singular value).

$$\sigma_i = \sqrt{\sum_{k=0}^{N-1} U_t[i][k]^2}$$

Εδώ, σ_i είναι η μοναδιαία τιμή για την i-στήλη και $U_t[i][k]$ είναι το στοιχείο στη γραμμή i και στήλη k του πίνακα U_t.

2. Κανονικοποιήση των Στηλών του Πίνακα U_t :

Κάθε στοιχείο της i-στήλης του πίνακα U_t διαιρείται με την αντίστοιχη μοναδιαία τιμή σ_i για να κανονικοποιηθεί η στήλη.

$$U_t[i][k] = \frac{U_t[i][k]}{\sigma_i}$$

Αυτό διασφαλίζει ότι οι στήλες του πίνακα U_t είναι ορθοκανονικές.

3. Ενημέρωση του Διαγώνιου Πίνακα S :

Ο πίνακας S είναι ένας διαγώνιος πίνακας που περιέχει τις μοναδιαίες τιμές σ_i στις διαγώνιες θέσεις.

$$S[i][i] = \sigma_i$$

Όλα τα άλλα στοιχεία του πίνακα S είναι μηδενικά.

Υλοποίηση κώδικα

```
// Create matrix S
for (int i = 0; i < M; i++) {
    t = 0;
    for (int j = 0; j < N; j++) {
        t = t + pow(U_t[i][j], 2);
    }
    t = sqrt(t);

    for (int j = 0; j < N; j++) {
        U_t[i][j] = U_t[i][j] / t;
        if (i == j) {
            S[i] = t;
        }
```

```
    }
}
```

- **Υπολογισμός της Νόρμας (t):** Η νόρμα της i-στήλης του πίνακα U_t υπολογίζεται ως το άθροισμα των τετραγώνων των στοιχείων της και έπειτα λαμβάνεται η τετραγωνική ρίζα.
- **Κανονικοποιήση της Στήλης:** Κάθε στοιχείο της i-στήλης διαιρείται με την υπολογισμένη νόρμα t, κανονικοποιώντας έτσι τη στήλη.
- **Ενημέρωση του Διαγώνιου Πίνακα S:** Η νόρμα t αποθηκεύεται στη διαγώνια θέση i του πίνακα S.

$$S = \begin{pmatrix} \sigma_1 & 0 & 0 & \cdots & 0 \\ 0 & \sigma_2 & 0 & \cdots & 0 \\ 0 & 0 & \sigma_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \sigma_n \end{pmatrix}$$

3.6 Διόρθωση του τελικού αποτελέσματος

```
// fix final result
for(int i =0; i<M; i++){
    for(int j =0; j<N; j++){
        U[i][j] = U_t[j][i];
        V[i][j] = V_t[j][i];
    }
}
```

Στο τελευταίο αυτό τμήμα, τα στοιχεία των πινάκων U και V ενημερώνονται με τις τιμές των πινάκων U_t και V_t αντίστοιχα. Αυτή η αντιστροφή των δεικτών i και j εξασφαλίζει τη σωστή διάταξη των στοιχείων στους πίνακες U και V.

Με αυτόν τον τρόπο, ολοκληρώνεται η σειριακή εφαρμογή των περιστροφών Jacobi για την υπολογισμό της SVD ενός πίνακα.

4. Ανακατασκευή του πίνακα A :

Οι πίνακες U , S και V χρησιμοποιούνται για να ανακατασκευαστεί ο αρχικός πίνακας A.

```
matrixMultiply(U, S_mat, temp);
```

```
matrixMultiply(temp, V_t, reconA);
```

5. Απελευθέρωση μνήμης :
Στο τέλος, ο κώδικας απελευθερώνει τη μνήμη που είχε δεσμεύσει.

```
delete [] S;
for(int i = 0; i<N;i++){
    delete[] S_mat[i];
    delete [] A[i];
    delete [] U[i];
    delete [] V[i];
    delete [] U_t[i];
    delete [] V_t[i];
    delete [] reconA[i];
    delete [] temp[i];
}
delete [] S_mat;
delete [] A;
delete [] U;
delete [] U_t;
delete [] V_t;
delete [] V;
delete [] reconA;
delete [] temp;
```

Παρουσίαση Παράλληλου κώδικα

Η παράλληλη εκδοχή του αλγορίθμου Jacobi για τον Υπολογισμό Μοναδιαίων Πινάκων (SVD) είναι σχεδιασμένη να αξιοποιεί την ικανότητα των πολυπύρηνων επεξεργαστών να εκτελούν πολλαπλές διεργασίες ταυτόχρονα. Αυτό επιτρέπει την ταχύτερη σύγκλιση του αλγορίθμου και την αποδοτικότερη επεξεργασία μεγάλων δεδομένων.

Παρακάτω παρουσιάζεται ο σειριακός κώδικας πρωτού παραλληλοποιηθεί :

```
for (int i = 1; i < M; i++){
    for (int j = 0; j < i; j++){

        alpha = 0.0;
        beta = 0.0;
        gamma = 0.0;
```

```
        for (int k = 0; k < N; k++){
            alpha = alpha + (U_t[i][k] * U_t[i][k]);
            beta = beta + (U_t[j][k] * U_t[j][k]);
            gamma = gamma + (U_t[i][k] * U_t[j][k]);
        }

    converge = max(converge, abs(gamma) / sqrt(alpha *
beta)); // compute convergence

    zeta = (beta - alpha) / (2.0 * gamma);
    t = sgn(zeta) / (abs(zeta) + sqrt(1.0 + (zeta *
zeta))); // compute tan of angle
    c = 1.0 / (sqrt (1.0 + (t * t))); // extract cos
    s  = c * t; // extrac sin

    // Apply rotations on U and V

    for (int k=0; k < N; k++){
        t = U_t[i][k];
        U_t[i][k] = c * t - s * U_t[j][k];
        U_t[j][k] = s * t + c * U_t[j][k];

        t = V_t[i][k];
        V_t[i][k] = c * t - s * V_t[j][k];
        V_t[j][k] = s * t + c * V_t[j][k];

        }

    }
}
```

Καθώς και ο παραλληλοποιημένος :

```
#pragma omp parallel for num_threads(num)
for (int p = 1; p <= r1; p++){
    int k = omp_get_thread_num();
    int i = I1[p], j = i + 1;
    double alpha = 0, beta = 0, gamma = 0;
    double zeta, t, c, s;
    for (int k = 0; k < N; k++) {
        alpha = alpha + (U_t[i][k] * U_t[i][k]);
        beta = beta + (U_t[j][k] * U_t[j][k]);
        gamma = gamma + (U_t[i][k] * U_t[j][k]);
    }
    C[k] = max(C[k], abs(gamma)/sqrt(alpha*beta));

    zeta = (beta - alpha) / (2.0 * gamma);
    t = sgn(zeta) / (abs(zeta) + sqrt(1.0 + (zeta*zeta)));
    c = 1.0 / (sqrt (1.0 + (t*t)));              //extract cos
    s = c*t;                                     //extrac sin
    for(int k=0; k<N; k++){
        t = U_t[i][k];
        U_t[i][k] = c*t - s*U_t[j][k];
        U_t[j][k] = s*t + c*U_t[j][k];

        t = V_t[i][k];
        V_t[i][k] = c*t - s*V_t[j][k];
        V_t[j][k] = s*t + c*V_t[j][k];

    }
}
```

```
#pragma omp parallel for num_threads(num)
for (int p = 1; p <= r2; p++){
    int k = omp_get_thread_num();
    int i = I2[p], j = i + 1;
    double alpha = 0, beta = 0, gamma = 0;
    double zeta, t, c, s;
    for (int k = 0; k < N; k++) {
        alpha = alpha + (U_t[i][k] * U_t[i][k]);
        beta = beta + (U_t[j][k] * U_t[j][k]);
        gamma = gamma + (U_t[i][k] * U_t[j][k]);
    }
    C[k] = max(C[k], abs(gamma)/sqrt(alpha*beta));

    zeta = (beta - alpha) / (2.0 * gamma);
    t = sgn(zeta) / (abs(zeta) + sqrt(1.0 + (zeta*zeta)));
    c = 1.0 / (sqrt (1.0 + (t*t)));              //extract cos
    s = c*t;                                     //extrac sin
    for(int k=0; k<N; k++){
        t = U_t[i][k];
        U_t[i][k] = c*t - s*U_t[j][k];
        U_t[j][k] = s*t + c*U_t[j][k];

        t = V_t[i][k];
        V_t[i][k] = c*t - s*V_t[j][k];
        V_t[j][k] = s*t + c*V_t[j][k];

    }
}
```

Ο παραλληλοποιημένος κώδικας χωρίζει το φόρτο εργασίας τις διαδικασίας Jacobi σε 2 περιοχές βάση των μεταβλητών **I1** και **I2**. Η παράλληλη υλοποίηση της συγκεκριμένης περιοχής μέσα στο βρόχο *while* επιλέχθηκε στρατηγικά για να βελτιστοποιηθεί η απόδοση και να εκμεταλλευτούν πλήρως οι δυνατότητες των σύγχρονων πολυπύρηνων επεξεργαστών καθώς είναι το τμήμα του κώδικα που χρησιμοποιείται περισσότερο από όλα. Η διαίρεση των υπολογισμών σε ανεξάρτητα τμήματα και η χρήση του OpenMP για την παράλληλη εκτέλεσή τους επιτυγχάνουν σημαντική μείωση του χρόνου εκτέλεσης και βελτιώνουν τη συνολική απόδοση του αλγορίθμου.

Ανάλυση των Μεταβλητών I1 και I2 στον Παράλληλο Κώδικα SVD

Οι μεταβλητές I1 και I2 χρησιμοποιούνται για να κατανείμουν τις γραμμές του πίνακα σε δύο ομάδες, οι οποίες θα επεξεργαστούν παράλληλα κατά τη διάρκεια της εφαρμογής του αλγόριθμου Jacobi. Ας δούμε αναλυτικά πώς λειτουργεί αυτό και γιατί επιλέγεται αυτή η προσέγγιση.

Το παρακάτω τμήμα κώδικα είναι υπεύθυνο για την κατανομή των γραμμών του πίνακα στις δύο ομάδες **I1** και **I2**:

```
for (int l = 1; l < M; l++) {
    int r1 = 0, r2 = 0;
    for (int i = 0; i + l < M; i++) {
        if (i % (2 * l) < l)
            I1[++r1] = i;
        else
            I2[++r2] = i;
    }
}
```

Λειτουργία Κατανομής:

1. **Επανάληψη για κάθε l από 1 έως M-1**:

- Ο L χρησιμοποιείται για να καθορίσει την απόσταση μεταξύ των ζευγών γραμμών που θα επεξεργαστούν.

2. **Εσωτερική Επανάληψη για κάθε i από 0 έως M-l-1**:

- Η συνθήκη i % (2 * L) < L χρησιμοποιείται για να καθοριστεί αν το i θα προστεθεί στο I1 ή στο I2.

3. **Διανομή στα I1 και I2**:

- Αν η συνθήκη είναι αληθής, το i προστίθεται στο I1. Αλλιώς, το i προστίθεται στο I2.

Γιατί Επιλέγονται Αυτές οι Περιοχές;

Ο λόγος που επιλέγονται αυτές οι περιοχές για να επεξεργαστούν παράλληλα είναι ότι δεν έχουν αλληλεξαρτήσεις μεταξύ τους. Αυτό σημαίνει ότι κάθε γραμμή σε ένα ζεύγος (i, j) μπορεί να υπολογιστεί ανεξάρτητα από τις υπόλοιπες, αποφεύγοντας έτσι τις ανταγωνιστικές συνθήκες (race conditions).

Συνοψίζοντας, οι μεταβλητές I1 και I2 διευκολύνουν την παράλληλη επεξεργασία των γραμμών του πίνακα A, εξασφαλίζοντας την αποδοτική και ασφαλή εκτέλεση του παράλληλου αλγορίθμου Jacobi.

Πολυπλοκότητα του αλγορίθμου Jacobi για SVD

Η πολυπλοκότητα του αλγορίθμου Jacobi για την SVD εξαρτάται από το μέγεθος του πίνακα και τον αριθμό των επαναλήψεων που απαιτούνται για τη σύγκλιση. Για έναν πίνακα διαστάσεων $m \times n$ η πολυπλοκότητα κάθε περιστροφής είναι $O(n^2)$. Δεδομένου ότι η μέθοδος συνήθως απαιτεί έναν αριθμό επαναλήψεων που είναι ανάλογος του n, η συνολική πολυπλοκότητα της μεθόδου Jacobi για την SVD εκτιμάται ως $O(n^3)$. Αυτή η πολυπλοκότητα καθιστά τη μέθοδο Jacobi λιγότερο αποδοτική σε σχέση με άλλες μεθόδους όπως οι μέθοδοι που βασίζονται στην QR διάσπαση, ειδικά για πολύ μεγάλους πίνακες. Παρόλα αυτά, η απλότητα της υλοποίησής της και η δυνατότητα να παρέχει ακριβή αποτελέσματα την καθιστούν χρήσιμη σε πολλές εφαρμογές.

Σημειογραφία

- **M**: Ο εξωτερικός βρόχος εκτελείται για l από 1 έως $M-1$.
- **N**: Οι εσωτερικοί υπολογισμοί εξαρτώνται από το N, τη διάσταση των διανυσμάτων U_t και V_t
- **num**: Ο αριθμός των νημάτων που χρησιμοποιούνται στις παράλληλες ενότητες του OpenMP.
- **r1, r2**: Τα μεγέθη των πινάκων δεικτών I_1 και I_2.

Εξωτερικός Βρόχος

Ο εξωτερικός βρόχος εκτελείται από $l = 1$ έως $M-1$. Αυτό δίνει $O(M)$ επαναλήψεις.

Εσωτερικοί Βρόχοι

Για κάθε τιμή του l:

Βρόχος Δεικτών: Ο πρώτος εσωτερικός βρόχος που συναντάται εκτελείται $M-1$ φορές. Εφόσον το l κυμαίνεται από 1 έως $M-1$ κατά μέσο όρο, ο βρόχος εκτελείται $O\left(\frac{M}{2}\right)$φορές. Επομένως, η πολυπλοκότητα για αυτό το μέρος είναι $O(M)$.

Εσωτερικοί Υπολογισμοί: Μέσα στους παράλληλους βρόχους, πραγματοποιούνται οι εξής λειτουργίες:

Ο υπολογισμός των α, β, γ χρειάζεται $O(N)$ επαναλήψεις.

Οι τριγωνομετρικοί υπολογισμοί και οι ενημερώσεις των U_t και V_t χρειάζονται επίσης $O(N)$ επαναλήψεις.

Αυτοί οι υπολογισμοί είναι $O(N)$ για κάθε ζεύγος (i, j) οδηγώντας σε $O(M \times N)$ επαναλήψεις για κάθε l.

Συνολική Πολυπλοκότητα

Συνδυάζοντας αυτές τις παρατηρήσεις:

Ο εξωτερικός βρόχος εκτελείται $O(M)$φορές.

Για κάθε επανάληψη του εξωτερικού βρόχου, οι εσωτερικές λειτουργίες έχουν πολυπλοκότητα $O(M \times N)$

Επομένως, η συνολική πολυπλοκότητα $O(M^2 N)$ σε περιπτώσεις όπου οι πίνακες είναι τετραγωνικοί.

Αποτελέσματα και Συμπεράσματα

Η εγκυρότητα των αποτελεσμάτων ελέγχθηκε με τα αποτελέσματα του SVD του Matlab. Ο έλεγχος έγινε πάνω σε πίνακες μικρών διαστάσεων, ώστε να μπορούμε να παρατηρήσουμε πιο εύκολα τους πίνακες εξόδου. Αναφορικά ένα παράδειγμα είναι ένας πίνακας 4x4 όπως φαίνεται πιο κάτω. Αυτός είναι ο κώδικας για το Matlab και τα αποτελέσματα του.

```
A = [
    10 3 7 1;
    4 9 -6 15;
    47 -13 -2 1;
    -1 35 4 6;];

[U,S,V] = svd(A);

A_recon= U*S*(V');
```

51.2580	0	0	0
0	36.1783	0	0
0	0	14.6276	0
0	0	0	5.2702

Πίνακας των Singular τιμών.

-0.1397	-0.2124	0.3489	-0.9019
0.0135	-0.3773	-0.8894	-0.2573
-0.9306	-0.2842	0.0408	0.2268
0.3380	-0.8553	0.2921	0.2621

Πίνακας U.

-0.8861	-0.4460	0.1065	0.0668
0.4610	-0.8368	0.1871	0.2282
0.0420	-0.0570	0.6061	-0.7921
0.0226	-0.3120	-0.7656	-0.5620

Πίνακας V.

```
U

0.139709  0.212423  0.901989  -0.348961
-0.013501  0.377371  0.257342  0.889485
0.930619  0.28426  -0.226886  -0.0408328
-0.338005  0.855373  -0.262134  -0.29219
```

```
V

0.886109  0.446085  -0.0668317  -0.106553
-0.461013  0.836864  -0.22827  -0.187135
-0.0420283  0.0573745  0.792153  -0.606165
-0.0226348  0.312051  0.562068  0.765631
```

Πίνακες U και V που παράγονται από των κώδικα.

```
S

51.258  0.0  0.0  0.0
0.0  36.1783  0.0  0.0
0.0  0.0  5.27059  0.0
0.0  0.0  0.0  14.6276

10 3 7 1
4 9 -6 15
47 -13 -2 1
-1 35 4 6
```

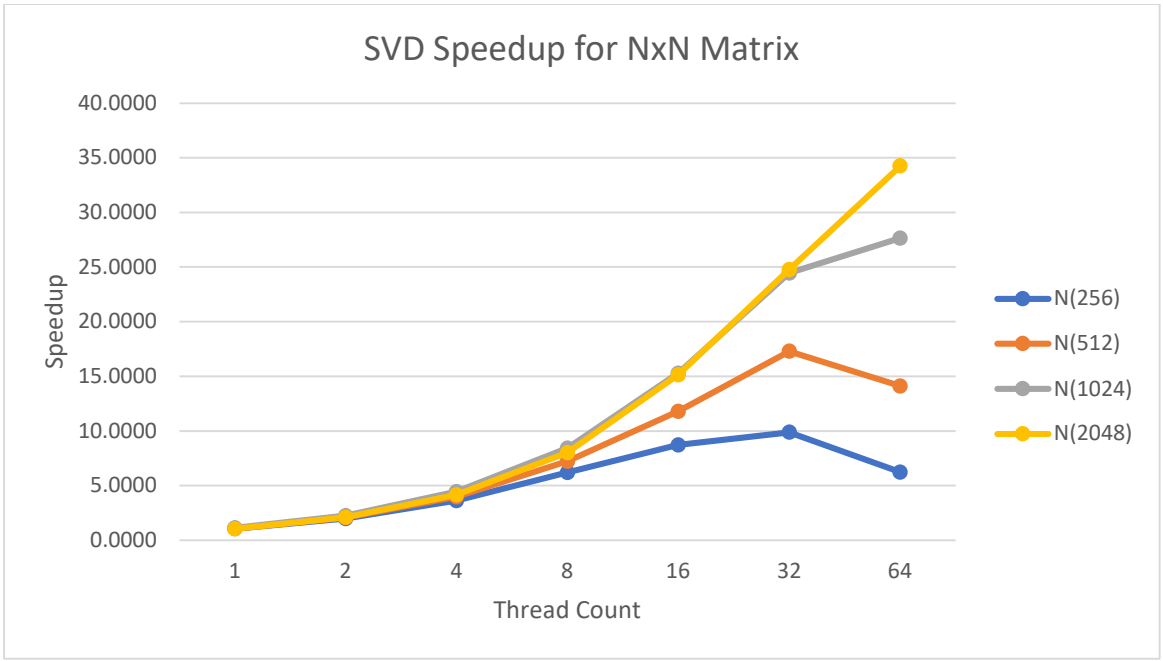
Ο πίνακας των Singular τιμών που παράγεται και η επαλήθευση του ορισμού του SVD που θέτει ότι το γινόμενο των παραγόμενων πινάκων πρέπει να επιστρέφει τον αρχικό πίνακα.

Όπως φαίνεται τα πρόσημα είναι ανεστραμμένα στους πίνακες των Eigenvectors. Αυτό δεν αποτελεί πρόβλημα αρκεί να ισχύει για όλα τα στοιχεία των vectors κάτι που βλέπουμε ότι ισχύει. Αυτός ο έλεγχος έγινε και για τυχαίους πίνακες, ενώ τα αποτελέσματα διασταυρώθηκαν και με μια ιστοσελίδα υπολογισμού των Eigenvectors και Eigenvalues για μεγαλύτερη σιγουριά.

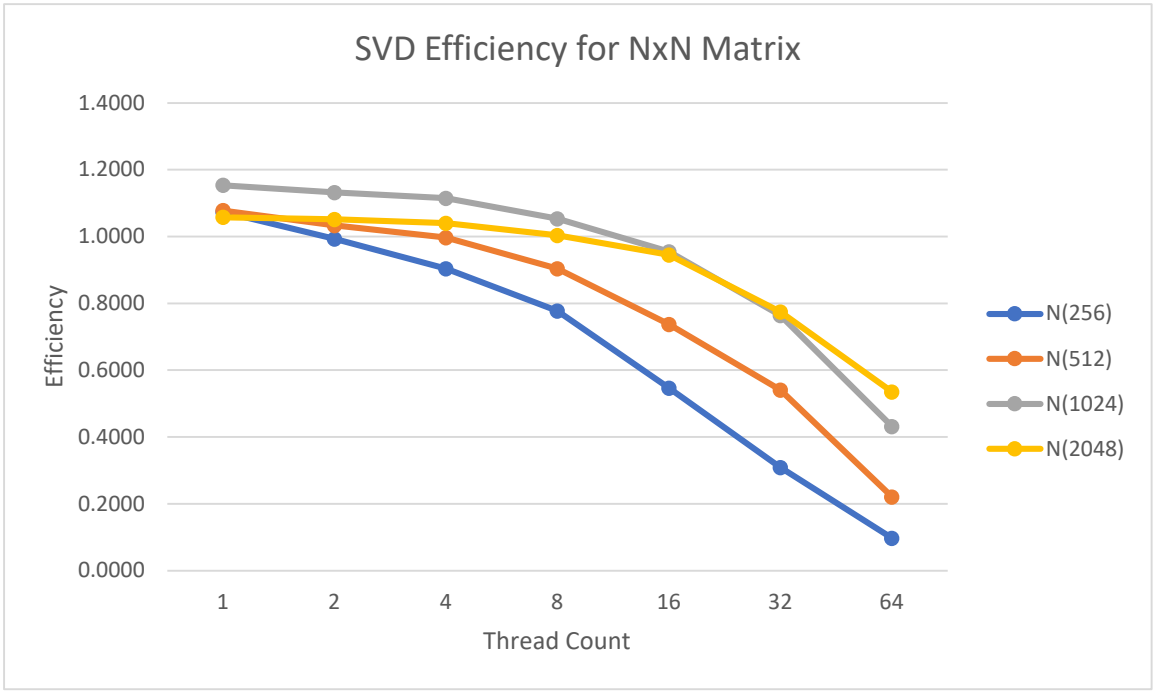
Έπειτα, οι κώδικες του SVD δοκιμάστηκαν για ένα πλήθος τετραγωνικών πινάκων. Χρησιμοποιήσαμε διαστάσεις επαρκής για να μας δώσουν μια ικανοποιητική οπτική της απόδοσης του αλγορίθμου και της συμπεριφοράς του στην παραλληλοποίηση. Συγκεκριμένα, πήραμε δεδομένα για τις περιπτώσεις όπου το N του Πίνακα είναι: 256, 512, 1024, 2048. Τα αποτελέσματα αυτών των μετρήσεων φαίνονται στον παρακάτω πίνακα.

N	Serial	P=1	P=2	P=4	P=8	P=16	P=32	P=64
256	1.66727	1.55264	0.83977	0.46105	0.26800	0.19078	0.16852	0.26767
Speedup		1.07383	1.98540	3.61628	6.22106	8.73914	9.89360	6.22885
Efficiency		1.07383	0.99270	0.90407	0.77763	0.54620	0.30918	0.09733
512	14.27630	13.24160	6.90676	3.58055	1.97469	1.21064	0.82525	1.01045
Speedup		1.07814	2.06700	3.98718	7.22964	11.79236	17.29934	14.12866
Efficiency		1.07814	1.03350	0.99680	0.90371	0.73702	0.54060	0.22076
1024	121.94900	105.69000	53.88170	27.34470	14.46640	7.98018	4.98813	4.41116
Speedup		1.15384	2.26327	4.45969	8.42981	15.28148	24.44784	27.64556
Efficiency		1.15384	1.13164	1.11492	1.05373	0.95509	0.76399	0.43196
2048	1034.79000	977.82500	491.80700	248.63500	128.91100	68.41330	41.74410	30.21040
Speedup		1.05826	2.10406	4.16188	8.02717	15.12557	24.78889	34.25277
Efficiency		1.05826	1.05203	1.04047	1.00340	0.94535	0.77465	0.53520

Μια πρώτη παρατήρηση είναι ότι σε όλες τις περιπτώσεις ο παράλληλος κώδικας με χρήση ενός νήματος καταλήγει να είναι πιο γρήγορος από τον σειριακό. Αυτό είναι κάτι αναμενόμενο στην δική μας περίπτωση. Ο χωρισμός των δεδομένων που αναφέραμε στην παραλληλοποίηση του κώδικα οδηγεί σε καλύτερη χρήση των cache μνημών και πιθανότατα σε λιγότερη επικοινωνία με τα μακρινότερα επίπεδα μνήμης που αυξάνουν των χρόνο σημαντικά. Πέρα από αυτό όμως, είναι πιθανό η βελτίωση να σχετίζεται και με τον τρόπο που ο compiler χειρίζεται τις λούπες που βρίσκονται στην παράλληλη περιοχή. Εκεί, η δυνατότητα του prefetch των δεδομένων μπορεί να αυξήσει την απόδοση. Ας δούμε όμως, τα γραφήματα για να κατανοήσουμε καλύτερα το performance.



Με μια πρώτη ματιά συμπεραίνουμε ότι οι πίνακες με N ίσο 256 και 512 είναι σχετικά μικροί για την χρήση 64 νημάτων και έτσι η απόδοση για αυτούς μειώνεται. Ωστόσο, γίνεται εύκολα αντιληπτό ότι όσο μεγαλύτερο το πλήθος δεδομένων τόσο πιο κοντά στο ιδανικό βρίσκεται στο Speedup.



Η πορεία που διαγράφει το Efficiency μας δείχνει ότι ο κώδικας αξίζει να γίνει παράλληλος, ενώ για τον σωστό συνδυασμό δεδομένων και πυρήνων επεξεργασίας είναι πολύ επικερδής.

Βιβλιογραφία

- Επιστημονικοί Υπολογισμοί ‘Γραββάνης Γεώργιος
- https://skim.math.msstate.edu/LectureNotes/Numer_Lin_Algebra.pdf
- https://www.irisa.fr/sage/bernard/publis/SVD-Chapter06.pdf
- Αναλυτικός Υπολογισμός του SVD (video) https://www.youtube.com/watch?v=uOzMM13iElw&t=352s
- “A mixed precision Jacobi SVD algorithm “ by Weiguo Gao, Yuxin Ma, Meiyue Shao
- https://www.mathworks.com/help/fixedpoint/ref/fixed.jacobisvd.html
- https://web.stanford.edu/class/cme335/lecture7.pdf
- https://github.com/lixueclaire/Parallel-SVD
- https://devdocs.io/eigen3-chapter-dense-matrix-and-array-manipulation/
- https://matrixcalc.org/
- https://www.andreinc.net/2021/01/25/computing-eigenvalues-and-eigenvectors-using-qr-decomposition