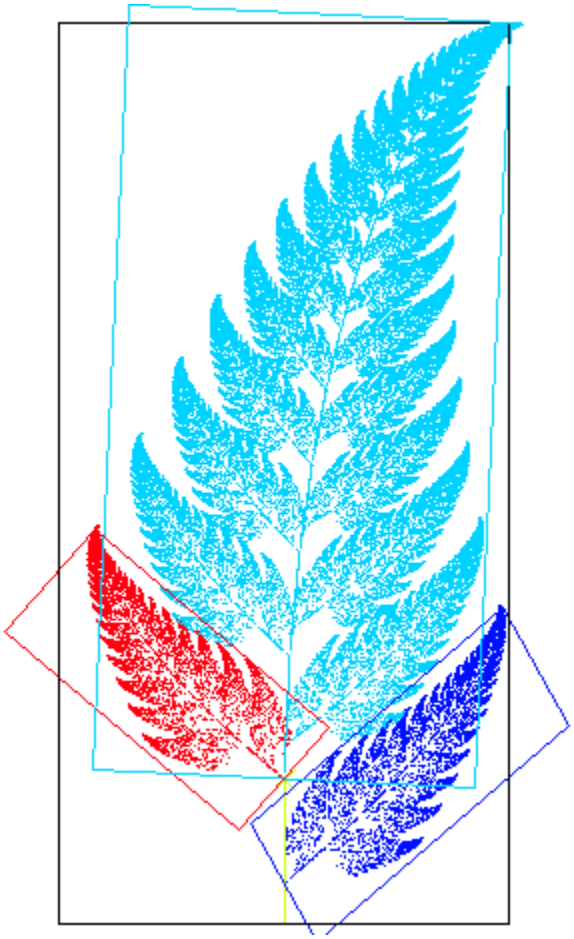
ΣΧΕΔΙΑΣΜΟΣ ΕΝΣΩΜΑΤΩΝ ΣΥΣΤΗΜΑΤΩΝ

Πρώτο μέρος της εργασίας 2020 -2021

Εφαρμογή του συσχετισμένου μετασχηματισμού στην επεξεργασία εικόνας

affine transformation in image processing



Υπεύθυνος καθηγητής

Συρακούλης Γεώργιος

Ομάδα 34

Καλτάκης Αναστάσιος 57271

Λιάπης Σάββας 57403

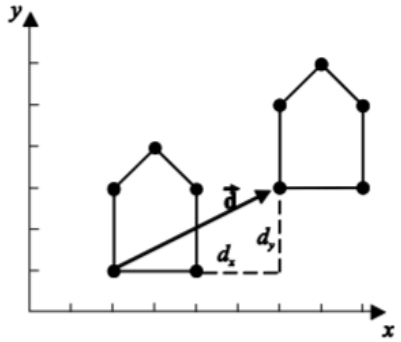
Εισαγωγή

Σκοπός της εργασίας είναι, αφού υλοποιηθεί σε γλώσσα C ο αλγόριθμος σε c εφαρμογής του συσχετισμένου μετασχηματισμού στην επεξεργασία εικόνας, να εφαρμοστεί η μεθοδολογία βελτιστοποίησης μεταφοράς και αποθήκευσης δεδομένων στον αλγόριθμο αυτό. Κατά την βελτιστοποίηση θα εξεταστεί μέσω του ARMulator του επεξεργαστή ARM τι αποτελέσματα έχει η εφαρμογή μετασχηματισμών βρόγχων για την κανονικοποίηση της δομής του αλγορίθμου στον πίνακα δεδομένων και στον αριθμό προσπελάσεων που γίνονται σε αυτόν, καθώς και στην ταχύτητα εκτέλεσης του αλγορίθμου

Περιγραφή του συσχετισμένου μετασχηματισμού στην επεξεργασία εικόνας (affine transformation in image processing)

Ο συσχετισμένος μετασχηματισμός είναι μια σημαντική κατηγορία γραμμικών 2-D γεωμετρικών μετασχηματισμών που χαρτογραφούν μεταβλητές (π.χ. τιμές pixel που βρίσκονται στη θέση (x1,y1) σε μια εικόνα εισόδου) σε νέες μεταβλητές (π.χ. (x2,y2) σε μια εικόνα εξόδου) εφαρμόζοντας έναν γραμμικό συνδυασμό μετατόπισης, περιστροφής, κλιμάκωσης και / ή διάτμησης (δηλαδή μη ομοιόμορφης κλίμακας σε ορισμένες κατευθύνσεις).Πιο συγκεκριμένα οι συσχετισμένοι μετασχηματισμοί εκφράζονται ως :

Μετατόπιση :



η μετατόπιση στις 2 διαστάσεις ενός σημείου $p = \begin{bmatrix} x \\ y \end{bmatrix}$ κατά διάνυσμα $\boldsymbol{d} = \begin{bmatrix} d_x \\ d_y \end{bmatrix}$ δίνει ένα νέο σημείο $p' = \begin{bmatrix} x' \\ y' \end{bmatrix}$ σύμφωνα με την σχέση :

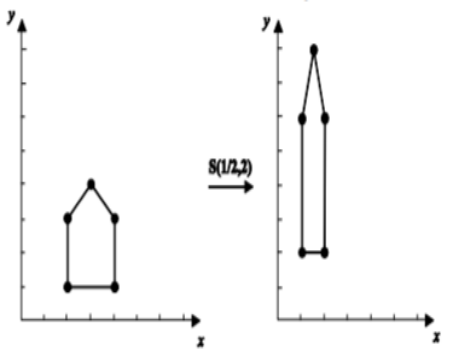
$$p' = p + d$$

Αλλαγή κλίμακας :

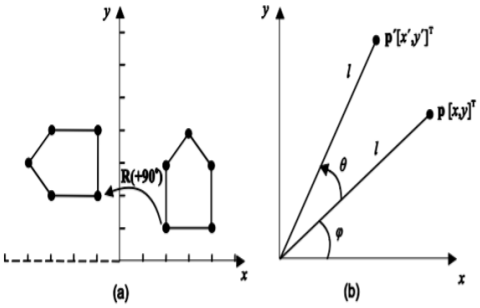
η αλλαγή κλίμακας σε 2 διαστάσεις ενός σημείου $p = \begin{bmatrix} x \\ y \end{bmatrix}$ κατά διάνυσμα το $p' = \begin{bmatrix} x' \\ y' \end{bmatrix}$ σύμφωνα με την σχέση :

$p' = \boldsymbol{S}(s_x, s_y)p$, όπου : $\boldsymbol{S}(s_x, s_y) = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$

Σημείωση : η αλλαγή κλίμακας σε σημείο δεν είναι παρατηρήσιμη



Περιστροφή :

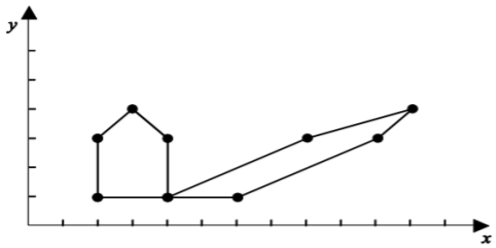


Η περιστροφή στρέφει ένα αντικείμενο γύρω από το κέντρο των συντεταγμένων.
η αλλαγή κλίμακας σε 2 διαστάσεις ενός σημείου $p = \begin{bmatrix} x \\ y \end{bmatrix}$ κατά διάνυσμα το $p' = \begin{bmatrix} x' \\ y' \end{bmatrix}$ σύμφωνα με την σχέση :
$p' = \boldsymbol{R}(\theta)p$, όπου : $\boldsymbol{R}(\theta) = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$

Στρέβλωση:

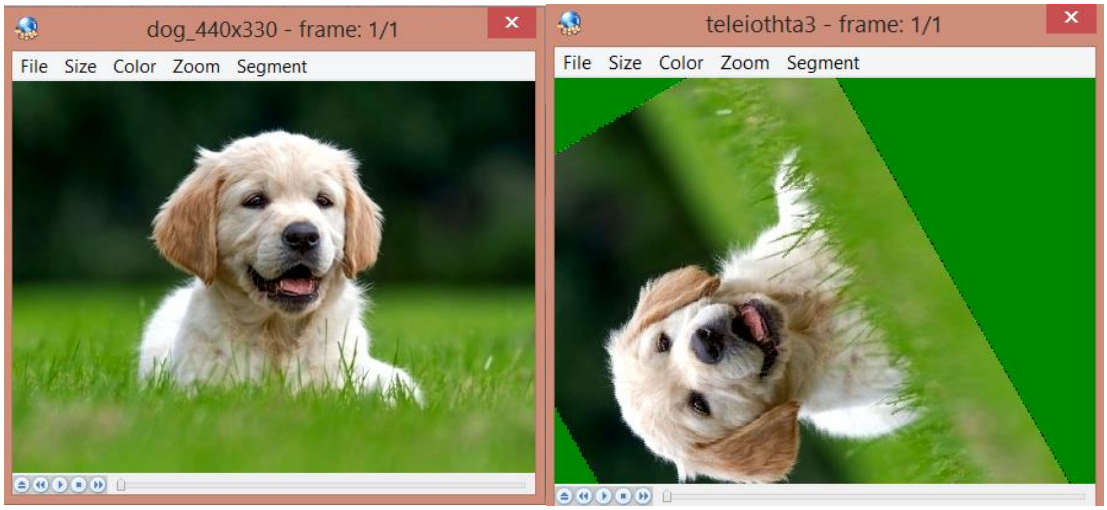
η στρέβλωση σε 2 διαστάσεις ενός σημείου $p = \begin{bmatrix} x \\ y \end{bmatrix}$ κατά διάνυσμα το $p' = \begin{bmatrix} x' \\ y' \end{bmatrix}$ σύμφωνα με την σχέση :

- στρέβλωση στον άξονα x : $p' = SH_x(a) * p$, όπου : $SH_x(a) = \begin{bmatrix} 1 & \alpha \\ 0 & 1 \end{bmatrix}$
- στρέβλωση στον άξονα y : $p' = SH_x(b) * p$, όπου : $SH_x(b) = \begin{bmatrix} 1 & \alpha \\ b & 1 \end{bmatrix}$



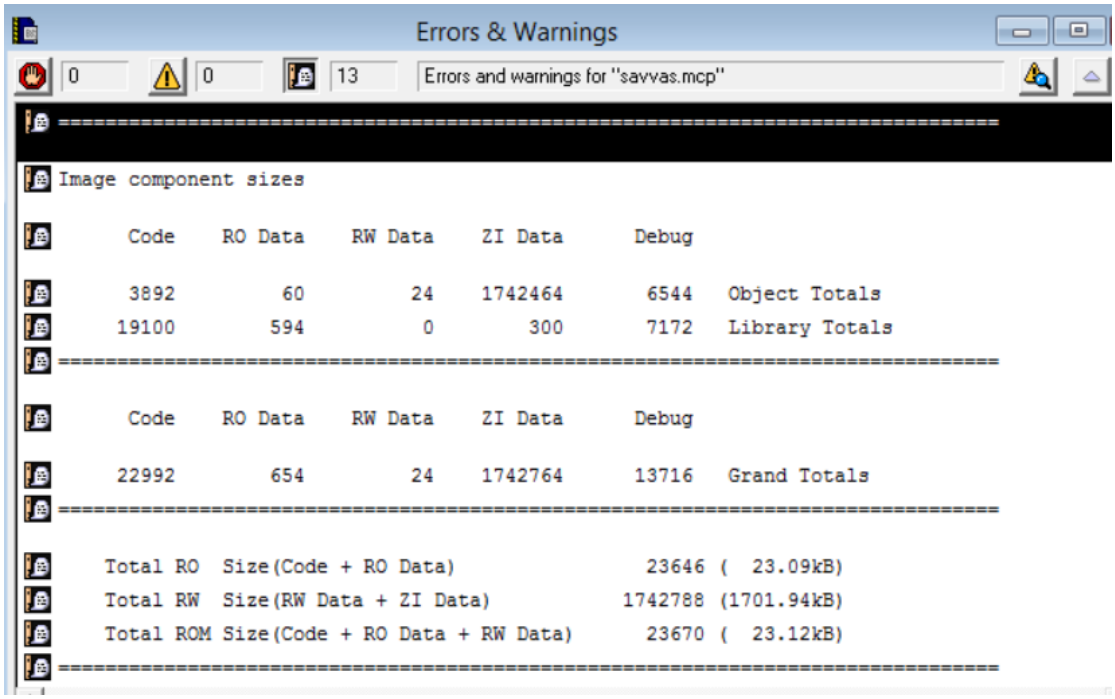
Περίληψη Αλγορίθμου

Η υλοποίηση του αλγορίθμου έγινε ως εξής: Εφαρμόστηκε ο affine transformation της περιστροφής. Η υλοποίηση γίνεται μέσα στην main. Δηλώνουμε τις μεταβλητές xt και yt οι οποίες στην ουσία είναι η κανονικοποίηση ως προς το κέντρο της εικόνας. Έπειτα οι μεταβλητές xs ys δίνουν την μετασχηματισμένη μορφή . δηλαδή πολλαπλασιάζονται τα xt και yt με τον πίνακα περιστροφής και με την πρόσθεση του κέντρου δίνεται η κανονική τους θέση.



αποτελέσματα :

τρέχοντας τον κώδικα στον armulator βλέπω τα εξής αποτελέσματα :



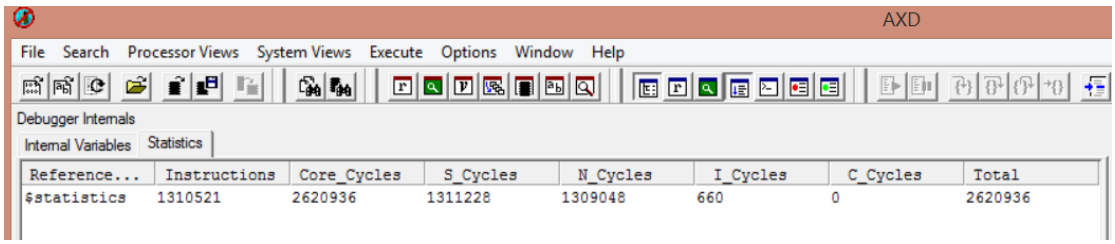
στο παραπανω tab αναγράφονται :

- **Code**: είναι ο χώρος σε byte που καταλαμβάνει ο κώδικας που γράψαμε στην μνήμη.
- **RO data (Read Only data)**: ο χώρος σε bytes που καταλαμβάνουν τα δεδομένα των οποίων την τιμή δεν είναι δυνατόν να μεταβληθεί δηλαδή προορίζονται για ανάγνωση. Στον προκείμενο κώδικα έχουμε σαν RO data:
    - N=330
    - M=440
    - Το αρχείο εικόνα που θα εισάγουμε
    - Την τιμή του π
- **RW data (Read/ Write data):** ο χώρος σε bytes που καταλαμβάνουν τα δεδομένα για τα οποία υπάρχει η δυνατότητα επαναγραφής και είναι αρχικοποιημένα.

- **ZI data (Zero Initialized data):** ο χώρος σε bytes που καταλαμβάνουν πάλι επαναγράψημα δεδομένα αλλά μη αρχικοποιημένα. Αυτά με το που αρχικοποιηθούν στον μικροεπεξεργαστή θα πάρουν τιμή μηδέν. Στον προκείμενο κώδικα κάποια ZI data είναι ενδεικτικά :
    - current_y[N][M]
    - i, j, x, y, hM, hN, sinma, cosma, xt, yt, xs, ys (μεταβλητές που αθ αχρησιμοποιήσω στην main)
- **Debug :** ο χώρος σε bytes που καταλαμβάνουν τα δεδομένα που οφείλονται στο debugging. Δεν μας αφορούν γιατί δεν θα καταλήξουν στο ενσωματωμένο και χρησιμοποιούνται μόνο κατά την περίοδο σχεδιασμού.

Παρακάτω απεικονίζονται και κάποια totals (Total RO, Total RW , Total ROM) τα οποία είναι πολύ σημαντικά καθώς δείχνουν την αναγκαία μνήμη που θα χρειαστεί το ενσωματωμένο σύστημα.

Εκτός από το Internal Variables υπάρχει και το tab statistics μέσω του οποίου θα εξετάζεται ο βαθμός επιτυχίας των βελτιστοποιήσεων. Για να ανοίξει αυτό το tab πατάω το εικονίδιο play και αφού επιλεγεί το go 2 φορές και system views -> debagger internals επιλέγεται το tab statistics.



Reference...	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	1310521	2620936	1311228	1309048	660	0	2620936

Στο tab statistics αναγράφονται :

- Instructions: οι εντολές (σε assembly) που χρειάστηκαν για να τρέξει ο κώδικας
- Core_Cycles: οι κύκλοι που χρειάστηκαν για να εκτελεστούν οι εντολές στον επεξεργαστή

- S_Cycles (Sequential Cycles): οι κύκλοι κατά τους οποίους οι διαδοχικές εντολές εντελλόντουσαν σε διαδοχικές θέσεις μνήμης (δηλαδή ο program counter αυξανόταν κατά 1)
- N_Cycles (Non-Sequential Cycles): οι κύκλοι κατά τους οποίους ο επεξεργαστής μεταπηδά κάποιες διευθύνσεις μνήμης (δηλαδή ο programm counter δεν αυξάνεται κατά 1). Αυτοί οι κύκλοι περιγράφουν συνήθως τους κύκλους εκτελέσεων των branches. Αυτά μας νοιάζουν και περισσότερο, γιατί εξαιτίας αυτών των cycles δεν μπορεί να γίνει σωστό pipelining, αφού ο επεξεργαστής δεν μπορεί εύκολα να «προβλέψει» τα επόμενα δεδομένα που θα χρειαστούν από την μνήμη, οπότε είναι περισσότερο αναγκαίο να μειωθούν σε σχέση με τα sequential.
- I_Cycles: οι κύκλοι κατά τους οποίους δεν απαιτείται κάποιο call από την μνήμη (κάποια πράξη κτλ)
- C_Cycles: οι κύκλοι κατά τους οποίους ο επεξεργαστής δουλεύει παράλληλα με έναν extra επεξεργαστή (δεν θα μας απασχολήσει τώρα)

Βελτίωση Απόδοσης Αλγορίθμου

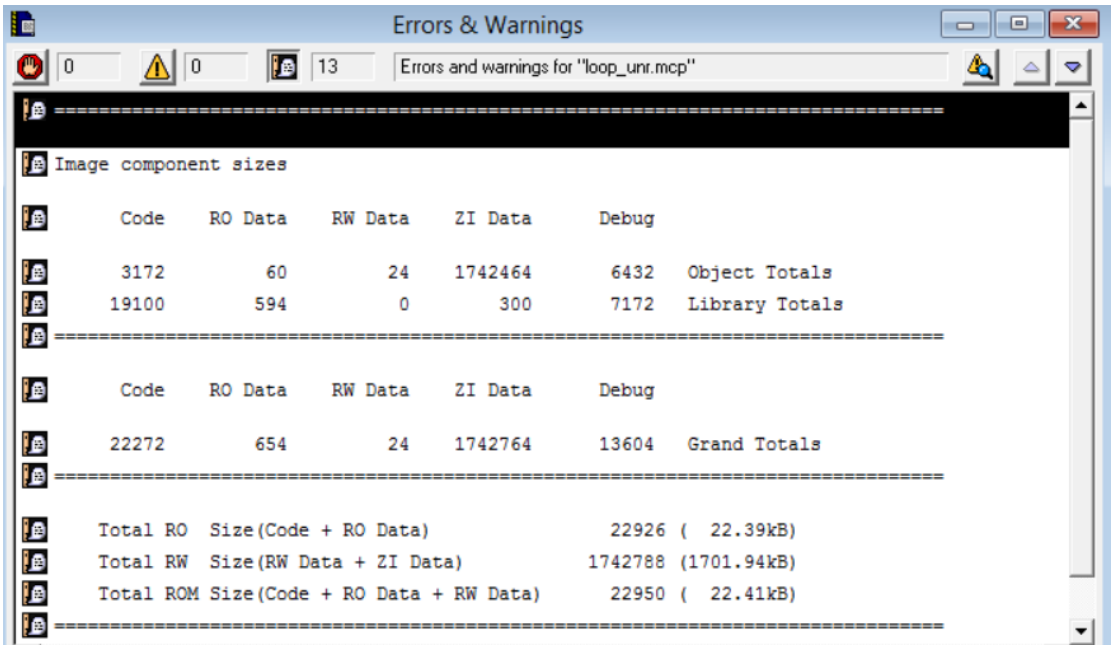
Σε αυτό το κομμάτι θα εξεταστούν οι συνέπειες που έχουν διάφοροι μετασχηματισμοί βρόγχων στην απόδοση του παρόντος κώδικα. Είναι προφανές ότι βελτιστοποίηση σημαίνει μείωση των κύκλων, οπότε μετά την εκτέλεση διαφόρων μεθόδων βελτιστοποίησης μετράται η μεταβολή στους κύκλους εκτέλεσης του κώδικα. Πέρα από αυτό θα εξετάζονται και οι συνέπειες των αλλαγών που θα παρατηρούνται σε άλλα αποτελέσματα. Οι αλγόριθμοι βελτιστοποίησης που θα εξεταστούν θα είναι :

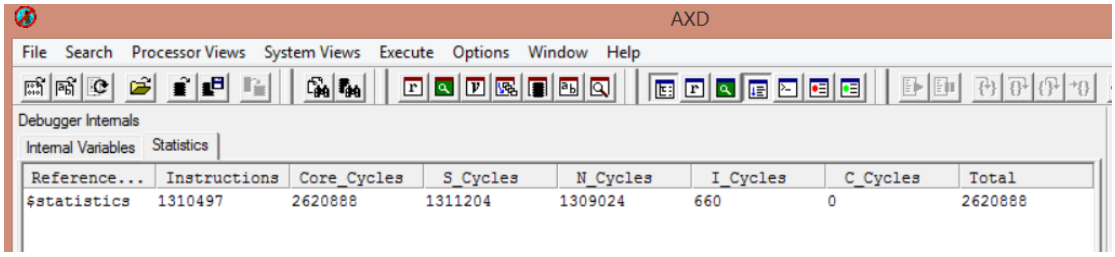
1. Loop unrolling
2. Loop fusion
3. Loop fission
4. Loop interchange
5. Loop tiling
6. Loop collapsing
7. Loop inversion

1.Loop Unrolling

Είναι μια τεχνική μετασχηματισμού της επαναληπτικής διαδικασίας που προσπαθεί να βελτιστοποιήσει την ταχύτητα εκτέλεσης του προγράμματος σε βάρος του δυαδικού μεγέθους του. Ο στόχος του loop unrolling είναι η αύξηση της ταχύτητας του προγράμματος μειώνοντας ή εξαλείφοντας τις εντολές ελέγχου του βρόγχου. Από την άλλη, όμως, ξετυλίγοντας τους βρόγχους επανάληψης αυξάνεται το μέγεθος του κώδικα, με αποτέλεσμα να απαιτείται περισσότερος χώρος στην μνήμη και μειώνεται ο διαθέσιμος χώρος για την αποθήκευση άλλων δεδομένων.

Εφαρμόζοντας την τεχνική loop unrolling παίρνουμε τα παρακάτω αποτελέσματα :





Συμπεραίνεται ότι με την τεχνική αυτή ο χώρος, που θα καταλαμβάνει πλέον ο κώδικας στην μνήμη, θα αυξηθεί άλλα θα έχουμε σημαντική μείωση στους συνολικούς κύκλους

2. Loop Fusion

Είναι μια βελτιστοποίηση και ένας μετασχηματισμός της επαναληπτικής μεθόδου, ώστε να αντικατασταθούν πολλαπλοί βρόγχοι σε έναν. Αυτή η μέθοδος είναι εφικτή όταν δύο βρόγχοι επαναλαμβάνονται στο ίδιο εύρος και ο ένας δεν επεμβαίνει στα δεδομένα του άλλου. Είναι αναγκαίο να σημειωθεί ότι αυτή η μέθοδος δεν έχει πάντα σαν αποτέλεσμα την βελτίωση της ταχύτητας εκτέλεσης.

```
Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      1436         60         24    1742448       5660   Object Totals
     19100        594          0        300       7172   Library Totals
================================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     20536        654         24    1742748      12832   Grand Totals
================================================================================

    Total RO  Size(Code + RO Data)                21190 (  20.69kB)
    Total RW  Size(RW Data + ZI Data)           1742772 (1701.93kB)
    Total ROM Size(Code + RO Data + RW Data)      21214 (  20.72kB)
================================================================================
```

File Search Processor Views System Views Execute Options Window Help

Debugger Internals

Internal Variables | Statistics

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	1310509	2620912	1311216	1309036	660	0	2620912

3. Loop Interchange

Είναι η διαδικασία ανταλλαγής της σειράς σάρωσης δύο δεικτών επανάληψης που χρησιμοποιούνται από έναν εμφωλευμένο βρόχο. Ο δείκτης του εσωτερικού βρόγχου μεταβαίνει στον εξωτερικό και αντίστροφα. Ο βασικός σκοπός είναι η αποδοτική διαχείρηση της cpu cache κατά την πρόσβαση σε στοιχεία πίνακα. Η προσπέλαση ενός στοιχείου πίνακα για πρώτη φορά, απαιτεί την ανάκτηση ενός μπλοκ δεδομένων από την κύρια μνήμη στη μνήμη cache. Αυτό το μπλοκ είναι πιθανό να έχει πολλά περισσότερα διαδοχικά στοιχεία, έτσι στην επόμενη πρόσβαση σε στοιχείο του πίνακα, θα οδηγηθεί απευθείας από την cache (εξοικονόμιση χρόνου). Η αποδοτικότητα του loop interchange εξαρτάται από το μοντέλο της cache και το μοντέλο πίνακα που χρησιμοποιείται από τον compiler. Συγκεκριμένα στην γλώσσα C, τα στοιχεία πίνακα της ίδιας σειράς αποθηκεύονται διαδοχικά στη μνήμη (a [1,1], a [1,2], a [1,3]), σε σειρά γραμμής (row-major order). Πρέπει να παρατηρηθεί ότι η ανταλλαγή δεικτών δεν είναι πάντα δυνατή, λόγω των εξαρτήσεων μεταξύ δηλώσεων για τη σειρά με την οποία πρέπει να εκτελούνται και απαιτείται ανάλυση εξάρτησης πριν την εφαρμογή του .

Εφαρμόζοντας την τεχνική loop interchange παίρνουμε τα παρακάτω αποτελέσματα :

```
Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      2416         60         24    1742448       6076   Object Totals
     19100        594          0        300       7172   Library Totals
================================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     21516        654         24    1742748      13248   Grand Totals
================================================================================

    Total RO  Size(Code + RO Data)                 22170 (  21.65kB)
    Total RW  Size(RW Data + ZI Data)            1742772 (1701.93kB)
    Total ROM Size(Code + RO Data + RW Data)       22194 (  21.67kB)
================================================================================
```

File Search Processor Views System Views Execute Options Window Help

Debugger Internals

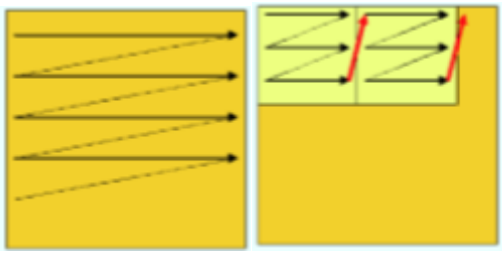
Internal Variables | Statistics

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	1310509	2620912	1311216	1309036	660	0	2620912

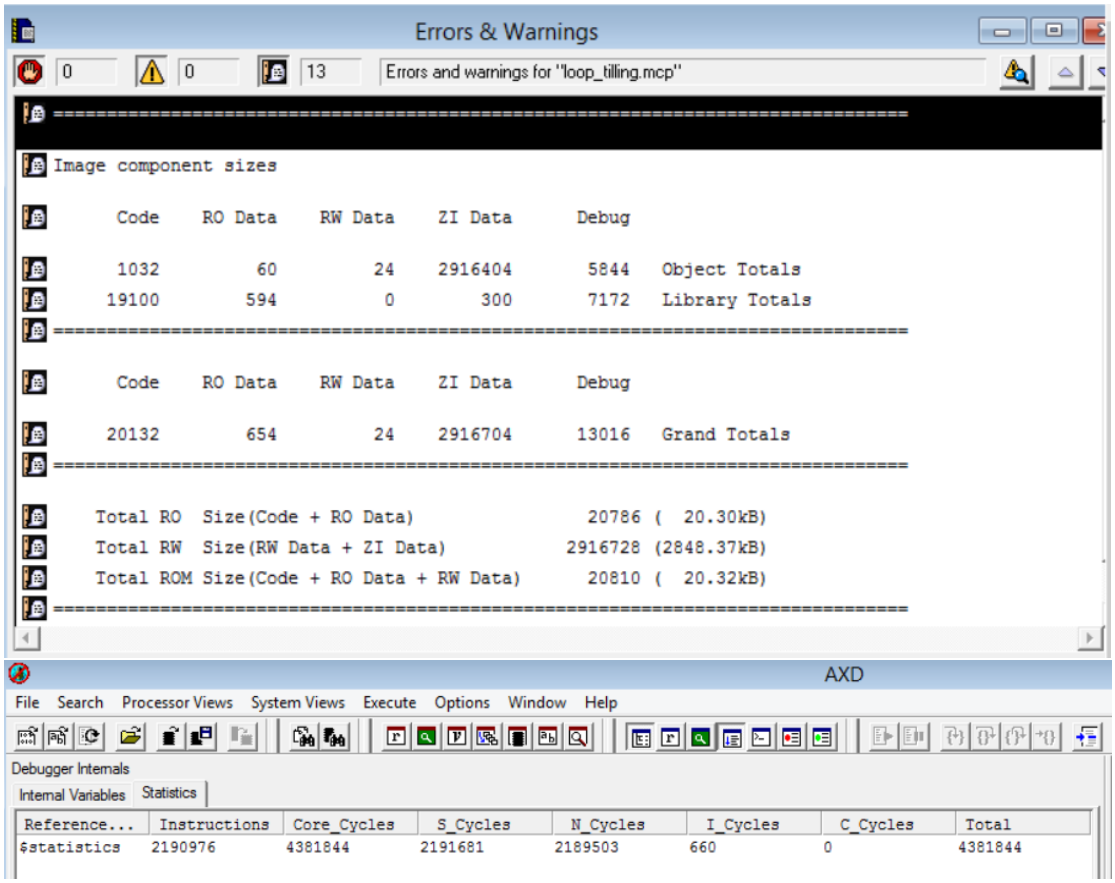
Από τα παραπάνω παρατηρούμε ότι η αλλαγή στην σειρά που εκτελούνται οι λούπες δεν έχει καμία συνέπεια.

4. Loop Tiling

Εκτελείται νοητός διαχωρισμός ενός πίνακα σε μικρότερους υποπίνακες (πλακάκια – tiles). Έτσι αλλάζει ο τρόπος μεταβίβασης δεδομένων μεταξύ κύριας μνήμης και cache αφού δεν ακολουθείται το ίδιο μοτίβο προσπέλασης. Το tiling εφαρμόζεται με 2 τρόπους. Οπτικά στον ένα τρόπο οι προσπελάσεις σχηματίζουν πολλά νοητά Z (δηλαδή ο πιο εσωτερικός εμφωλευμένος βρόγχος μετράει τις στήλες) και στον δεύτερο να σχηματίζουν πολλά αντίστροφα N (δηλαδή ο πιο εσωτερικός εμφωλευμένος βρόγχος μετράει τις γραμμές).



Εφαρμόζοντας την τεχνική loop tilling παίρνουμε τα παρακάτω αποτελέσματα :

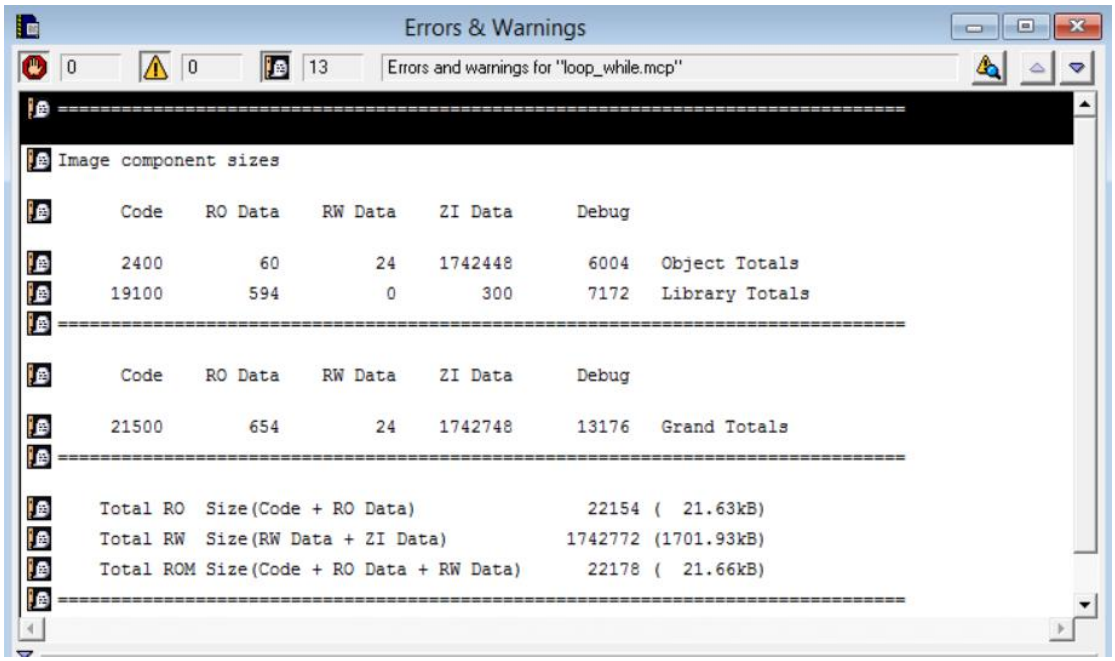


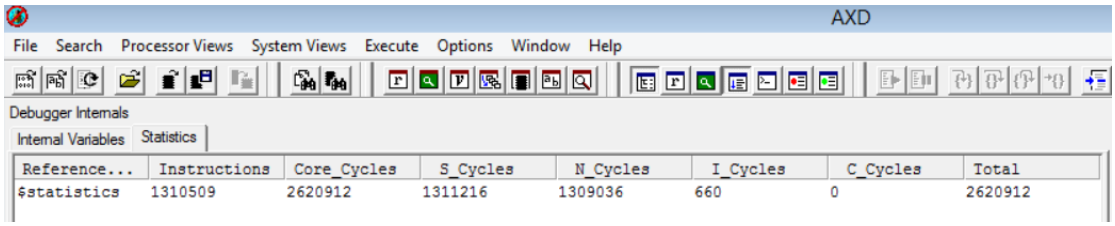
Από τα παραπάνω παρατηρούμε ότι η τεχνική αυτή για τα συγκεκριμένα tiles (22χ440) στην προσπέλαση του πίνακα current_y επιβαρύνει την απόδοση του κώδικα . Ωστόσο με την κατάλληλη επιλογή των tiles μπορεί να επιφέρονταν και ένα καλύτερο αποτέλεσμα.

5. Loop inversion

Το loop inversion αποτελεί βελτιστοποίηση και μετασχηματισμός στον οποίο, ένας βρόχος while αντικαθίσταται από ένα μπλοκ if που περιέχει ένα βρόχο do...while. με σωστή χρήση επιτυγχάνεται βελτίωση της απόδοσης λόγω του instruction pipelining. Επιπρόσθετα, το loop inversion επιτρέπει ασφαλή κίνηση κώδικα με απαράλλαχτο βρόχο.

Αρχικά ο κώδικά υλοποιήθηκε με λούπες while αντί για for.





Στη συνέχεια αντικαταστήσαμε τις while με λούπες do...while εφαρμόζοντας την τεχνική loop inversion. Ο έλεγχος if πριν την έναρξη κάθε βρόγχου do...while, δεν τοποθετήθηκε αφού διασφαλίζεται ότι ένας τέτοιος έλεγχος θα ικανοποιούνταν πάντα επομένως θα ήταν περιττός).

Εφαρμόζοντας την τεχνική loop inversion παίρνουμε τα παρακάτω αποτελέσματα :

```
Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      2380         60         24    1742448       6076   Object Totals
     19100        594          0        300       7172   Library Totals
================================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     21480        654         24    1742748      13248   Grand Totals
================================================================================

    Total RO  Size(Code + RO Data)                 22134 (  21.62kB)
    Total RW  Size(RW Data + ZI Data)            1742772 (1701.93kB)
    Total ROM Size(Code + RO Data + RW Data)       22158 (  21.64kB)
================================================================================
```

File Search Processor Views System Views Execute Options Window Help

Debugger Internals

Internal Variables | Statistics

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	1310509	2620912	1311216	1309036	660	0	2620912

Από τα παραπάνω παρατηρούμε ότι αυτή η τεχνική δεν προσθέτει κάποια βελτιστοποίηση στο κώδικά. Αντιθέτως ενός χρειάζονται οι ίδιοι κύκλοι απαιτεί λίγο παραπάνω μνήμη οπότε δεν μας συμφέρει.

Βιβλιογραφία :

- *https://images.app.goo.gl/1cBmT549PoskqLmv7https://eclass.duth.gr/modules/document/file.php/TMA182/%CE%95%CF%81%CE%B3%CE%B1%CF%83%CF%84%CE%AE%CF%81%CE%B9%CE%BF/%CE%95%CF%81%CE%B3%CE%B1%CF%83%CF%84%CE%B7%CF%81%CE%B9%CE%B1%CE%BA%CE%AE%20%CE%86%CF%83%CE%BA%CE%B7%CF%83%CE%B7%201/%CE%95%CE%BD%CF%83%CF%89%CE%BC%CE%B1%CF%84%CF%89%CE%BC%CE%AD%CE%BD%CE%B1%201%CE%B7%20%CE%95%CF%81%CE%B3%CE%B1%CF%83%CF%84%CE%B7%CF%81%CE%B9%CE%B1%CE%BA%CE%AE.pdf*
- *https://en.wikipedia.org/wiki/Affine_transformation*
- *http://homepages.inf.ed.ac.uk/rbf/HIPR2/affine.htm*
  http://archive.eclass.uth.gr/eclass/modules/document/file.php/DIB139/Geometric_Transformations_v2.pdf
- *http://graphics.cs.aueb.gr/cgvizbook/greek/slides/CGVIZ_Chapter_3_GR.pdf*
- *https://en.wikipedia.org/wiki/Loop_unrolling*
- *https://en.wikipedia.org/wiki/Loop_fission_and_fusion*
- *https://en.wikipedia.org/wiki/Loop_interchange*
- *https://en.wikipedia.org/wiki/Loop_nest_optimization*
- *https://en.wikipedia.org/wiki/Loop_splitting*
- *nullstone.com/htmls/category/collapse.htm*
- *https://en.wikipedia.org/wiki/Loop_inversion*