ΣΧΕΔΙΑΣΜΟΣ ΕΝΣΩΜΑΤΩΜΕΝΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΕΡΓΑΣΙΑ 1η

Καζαντζίδης Ηλίας  Α.Ε.Μ. 6141

Πέτρου Χρήστος      Α.Ε.Μ. 6178

Καθηγητής

Γεώργιος Συρακούλης

Ξάνθη 2015

1.Εισαγωγή

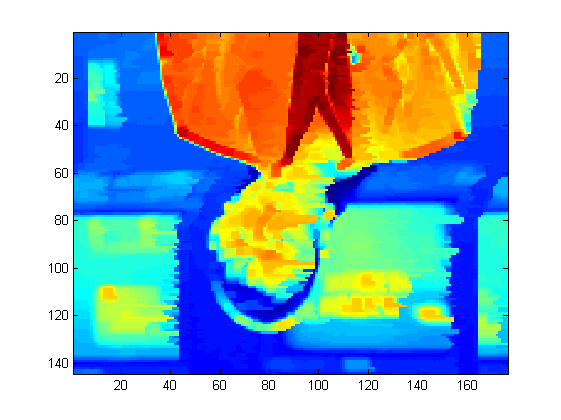
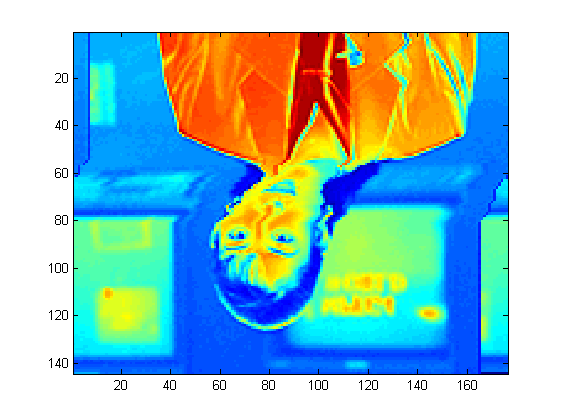
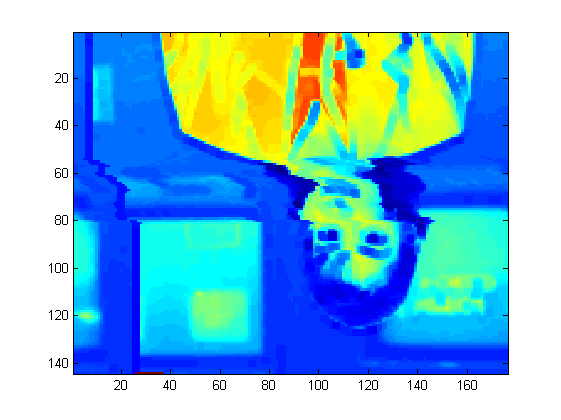
Σκοπός της εργασίας ήταν να υλοποιήσουμε σε γλώσσα C τον αλγόριθμο του “*μη γραμμικού φίλτρου min/max για την ψηφιακή επεξεργασία εικόνων*” και στη συνέχεια να εφαρμόσουμε βελτιστοποιήσεις μεταφοράς και αποθήκευσης δεδομένων στον αλγόριθμο αυτόν. Ύστερα από τις βελτιστοποιήσεις αυτό που θέλαμε να δούμε είναι επίδρασή του στην του επεξεργαστή ARM με τη βοήθεια του προσομοιωτή ARMulator.

Με το φίλτρο min/max το οποίο υλοποιήσαμε, μπορούμε να επεκτείνουμε τις περιοχές εικόνας με μαύρο/άσπρο χρώμα και να συρρικνώνουμε τις περιοχές με άστρο/μαύρο, αντίστοιχα. Αυτό γίνεται χωρίζοντας την εικόνα, την οποία διαβάζουμε σε ένα 2-D πίνακα, σε μικρές περιοχές και θέτοντας το κεντρικό σημείο της κάθε μίας ίσο με το ελάχιστο/μέγιστο αυτής της περιοχής.

Διαλέξαμε ένα φίλτρο 3x3, καθώς με ένα ζυγό αριθμό δεν θα υπήρχε κεντρικό στοιχείο και επίσης διότι με μεγαλύτερο αριθμό (π.χ. 5x5) α) θα χρειαζόμασταν περισσότερους κύκλους επεξεργασίας και β) θα ήταν λιγότερο λεπτομερές (<<πιξελιάζει λιγότερο>>).

Εδώ θα έπρεπε να πούμε ότι σε κάθε αρχείο .c εξηγείται με σχόλια η λειτουργία του κώδικα, επομένως σε αυτό το κείμενο θα ασχοληθούμε κυρίως με τις βελτιστοποιήσεις, οι οποίες αποτελούν το σημαντικό θέμα της εργασίας. Κάτι που επίσης πρέπει να αναφέρουμε είναι ότι θέσαμε στο πρόγραμμα AXD θέσαμε τη ταχύτητα του ρολογιού ίσης με 50MHz έτσι ώστε τα αποτελέσματα που θα πάρουμε στη συνέχεια για τους κύκλους επεξεργασίας να μπορούν να είναι ορθώς συγκρίσιμα.

Πριν παρουσιάσουμε όμως τις βελτιστοποιήσεις της απόδοσης θα πρέπει να σκεφτούμε ποιες είναι οι βασικές *μετρικές της απόδοσης*:  
 Πρώτα από όλα πρέπει ο κώδικας να είναι *λειτουργικός* δηλαδή να δίνει σωστά αποτελέσματα. Αυτό το είδαμε αρχικά μέσω των printf σε κάποια ενδεικτικά σημεία του αρχικού και των επεξεργασμένων min/max πινάκων. Παρατηρήσαμε δηλαδή ότι σε κάθε 3x3 περιοχή του αρχικού πίνακα το κεντρικό στοιχείο έπαιρνε στο νέο πίνακα την ελάχιστη/μέγιστη τιμή της περιοχής εκείνης . Φυσικά οι printf μπήκαν σε σχόλιο στη τέλος αφού ανέβαζαν πάρα πολύ τους κύκλους επεξεργασίας(της τάξης 107 ). Στη συνέχεια το ίδιο το παρατηρήσαμε και οπτικά μέσω του matlab κάνοντας εξαγωγή την αρχική και τις επεξεργασμένες εικόνες. Βλέπουμε στην εικόνες παρακάτω ότι με τον min αλγόριθμο έχει πέσει η φωτεινότητα ενώ με το max αλγόριθμο έχει ανέβει, Παρόλα αυτά με τον min αλγόριθμο από τη μέση περίπου και κάτω γίνεται κάποιο shift προς τα δεξιά, το οποίο δε μπορέσαμε να εξηγήσουμε (το μόνο που αλλάξαμε από τον κώδικα max στον min ήταν ένας τελεστής σύγκρισης).



      επεξεργασμένη με min αρχική    επεξεργασμένη με max

Επόμενο κριτήριο είναι να μην χρησιμοποιούνται πολλοί *καταχωρητές*. Εφόσον όμως ξέρουμε ότι θα τρέξει σε συγκεκριμένο επεξεργαστή με συγκεκριμένο αριθμό καταχωρητών αυτό μπορούμε να το αφήσουμε στην άκρη προς στιγμήν.

Το τελευταίο, με το οποίο και θα ασχοληθούμε είναι ο *αριθμός των εντολών σε Assembly* (σε C μπορεί και να μη παίζει ρόλο)  που εκτελούνται, όπως και των *κύκλων επεξεργασίας*. Αυτό είναι το κριτήριο, το οποίο θέλουμε να βελτιστοποιήσουμε.

2.Βελτιστοποιήσεις

Για κάθε αρχείο .c αυτά που βλέπαμε πατώντας make στο CodeWarrior ήταν:  
•Code : bytes που καταλαμβάνονται από το κώδικα.

•RO Data : bytes που καταλαμβάνονται από δεδομένα της ROM.

•RW Data : bytes που καταλαμβάνονται από δεδομένα της RAM.

•ZI Data : bytes που καταλαμβάνονται από δεδομένα που τα έχουμε δηλώσει ως 0 ή δεν τα έχουμε δηλώσει.

•Debug : bytes που καταλαμβάνονται από debug δεδομένα.

•Object Totals : bytes που καταλαμβάνονται από τα αντικείμενα που συνδέονται (object code) για να μας δώσουν την εικόνα.

•Library Totals : bytes που καταλαμβάνονται από αντικείμενα των βιβλιοθηκών.

•Grand Totals : πραγματικό μέγεθος φωτογραφίας.

•Total ROM : ελάχιστο μέγεθος της ROM που απαιτείται για να αποθηκευτεί η εικόνα.

Επίσης αυτά που βλέπαμε όταν τρέχαμε το πρόγραμμα στο AXD ήταν:

•Instructions : αριθμός ARM εντολών

•Core\_Cycles : κύκλοι του πυρήνα (επεξεργαστή).

•Sequential\_Cycles :κύκλοι λόγω σειριακών κλήσεων από τη μνήμη.

•Non-Sequential\_Cycles : κύκλοι λόγω μη-σειριακών κλήσεων από τη μνήμη.

•Internal\_Cycles : κύκλοι όταν δε χρειάζεται κλήση από τη μνήμη.

•Co\_Cycles : κύκλοι μια προαιρετική μονάδα προστίθεται στη CPU και είναι υπεύθυνη για την εκτέλεση κάποιων εντολών της CPU.

•Total\_Cycles : Άθροισμα όλων των παραπάνω πέρα από τους core.

Σαν μέτρο σύγκρισης θα χρησιμοποιούμε τους Total cycles οι οποίοι είναι πάντα κοντά στους Core\_Cycles.

Τέλος αν θέλουμε να βρούμε σε κάθε περίπτωση το χρόνο εκτέλεσης του προγράμματος παίρνουμε το τύπο :

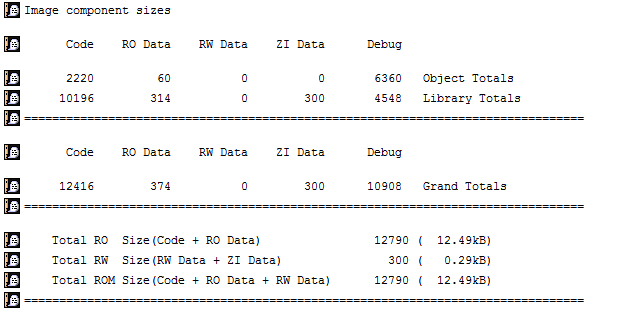
 ,όπου

-: ο αριθμός των ARM εντολών

- : ο μέσος αριθμός των κύκλων ρολογιού ανά εντολές

- : συχνότητα επεξεργαστή (50MHz)

Πριν τη 1η βελτιστοποίηση δίνονται τα αποτελέσματα του **αρχικού κώδικα (v2)** πατώντας make:

 v2

Τον αριθμό των εντολών ARM εντολών και συνολικών κύκλων τους βλέπουμε στο πίνακα μετά από όλες τις βελτιστοποιήσεις. Εκεί θα φαίνεται ξεκάθαρα πως θα πέφτουν μετά από κάθε βελτιστοποίηση αυτοί οι δύο αριθμοί. Επίσης εκεί παρουσιάζεται και η συνολική ROM που απαιτείται δηλαδή το μέγεθος του κώδικα μαζί με τα RO και τα RW data.

Οι πρώτες δύο βελτιστοποιήσεις είναι γενικές ενώ οι δύο επόμενες έχουν σχέση με μετασχηματισμούς βρόχων.

**1η βελτιστοποίηση (v3.c) : Οριακές Συνθήκες**

Η 1η βελτιστοποίηση έχει σχέση με τις τιμές γύρω από τον 2-D πίνακα. Συγκεκριμένα αντί να ελέγχουμε όλες τις περιπτώσεις στα άκρα του πίνακα δηλαδή στις 4 γωνίες, στις 4 πλευρές και εσωτερικά (9 περιπτώσεις), δημιουργούμε ένα νέο πίνακα ίδιο με τον αρχικό στον οποίο προσθέτουμε όμως περιμετρικά κατάλληλη τιμή (0 ή 255). Έτσι γλιτώνουμε 8 if με πολύ μικρότερο κόστος λόγω των for. Το κομμάτι του κώδικα με αυτή την βελτίωση για τον min αλγόριθμο είναι το παρακάτω :

//gemisma tou "megalou" pinaka min me 225

for(i=0;i<(N+2);++i){

for(j=0;j<(M+2);++j){

temp\_min\_image[i][j]=255;}}

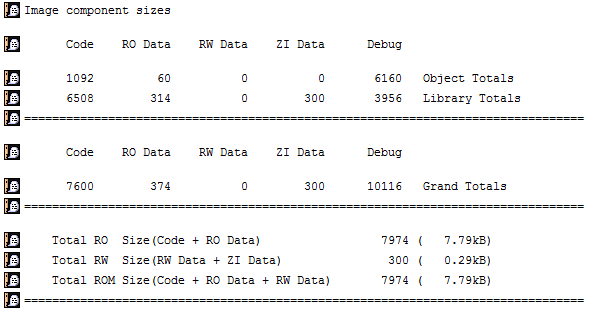
//gemisma twn eswterikwn shmeiwn tou "megalou" pinaka min me tis arxikes times

for (i=1; i<N+1; i++){

for(j=1;j<M+1;j++){

temp\_min\_image[i][j]=image[i-1][j-1];}}

Τα αποτελέσματα με make είναι τα εξής :

   
 v3

Παρατηρούμε ότι η συνολική ROM έπεσε σημαντικά ενώ οι συνολικοί κύκλοι πέφτουν κατά μισό περίπου εκατομμύριο.  
  
**2η βελτιστοποίηση (v4.c) : Απαλοιφή πίνακα k**

Στην 2η βελτίωση αυτό που κάνουμε είναι να γλιτώσουμε από την δημιουργία ενός πίνακα (του k) κι έτσι να ρίξουμε τόσο την ROM όσο και τους κύκλους για άλλο περίπου μισό εκατομμύριο. Παρακάτω δίνεται το σημείο με τον παλιό (v3) και τον καινούργιο (v4) κώδικα :

v3:  
for (i=1;i<N+1;i++){

for(j=1;j<M+1;j++){

k[0][0]=temp\_min\_image[i-1][j-1

k[0][1]=temp\_min\_image[i-1][j];

k[0][2]=temp\_min\_image[i-1][j+1];

k[1][0]=temp\_min\_image[i][j-1];

k[1][1]=temp\_min\_image[i][j];

k[1][2]=temp\_min\_image[i][j+1];

k[2][0]=temp\_min\_image[i+1][j-1];

k[2][1]=temp\_min\_image[i+1][j];

k[2][2]=temp\_min\_image[i+1][j+1];

min=255;

row=0;

line=0;

for (x=0;x<3;x++){

for (y=0;y<3;y++){

if (k[x][y]<min){

min=k[x][y];

row=x;

line=y; }

}

}

v4:

for (i=1;i<N+1;i++){

for(j=1;j<M+1;j++){

max=temp\_max\_image[i-1][j-1];

for(x=i-1; x<i+2; x++){ for(y=j-1; y<j+2; y++){

if (temp\_max\_image[x][y]>max){

max=temp\_max\_image[x][y];

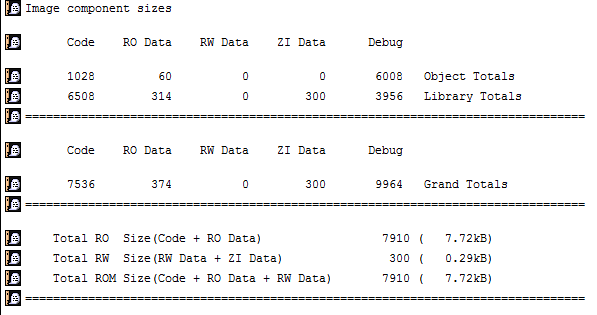
row=x;

line=y;}

}

}

και τα αποτελέσματα μετά το make :



v4

**3η βελτιστοποίηση (v6.c) : Loop\_unrolling**

Με αυτήν την βελτιστοποίηση μειώνεται η πρόσθετη επιβάρυνση των βρόγχων. Αυτό γιατί ο έλεγχος αντί να πηγαίνει μία φορά στη μνήμη και να φέρνει ένα στοιχείο, πηγαίνει και φέρνει περισσότερα τα οποία αποθηκεύονται προσωρινά πιο κοντά. Ένα από τα δύο κομμάτια κώδικα, που εφαρμόστηκε φαίνεται παρακάτω :

for (i=1; i<(N+1); i+=4){

for(j=1;j<(M+1);j+=4){

temp\_min\_image[i][j]=image[i-1][j-1];

temp\_min\_image[i][j+1]=image[i-1][j];

temp\_min\_image[i][j+2]=image[i-1][j+1];

temp\_min\_image[i][j+3]=image[i-1][j+2];

temp\_min\_image[i+1][j]=image[i][j-1];

temp\_min\_image[i+1][j+1]=image[i][j];

temp\_min\_image[i+1][j+2]=image[i][j+1];

temp\_min\_image[i+1][j+3]=image[i][j+2];

temp\_min\_image[i+2][j]=image[i+1][j-1];

temp\_min\_image[i+2][j+1]=image[i+1][j];

temp\_min\_image[i+2][j+2]=image[i+1][j+1];

temp\_min\_image[i+2][j+3]=image[i+1][j+2];

temp\_min\_image[i+3][j]=image[i+2][j-1];

temp\_min\_image[i+3][j+1]=image[i+2][j];

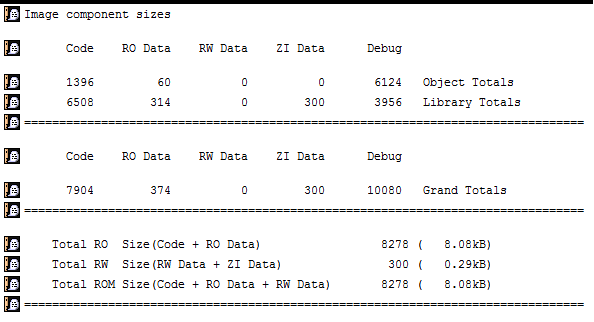
temp\_min\_image[i+3][j+2]=image[i+2][j+1];

temp\_min\_image[i+3][j+3]=image[i+2][j+2];

}

}  
  
 Καταλαβαίνουμε ότι κάθε φορά φέρνει 16 στοιχεία από τη μνήμη (θα μπορούσαμε να το βάλουμε και για παραπάνω και να βελτιωνόταν ακόμα περισσότερο - ωστόσο κάποια στιγμή μας έτυχε πέσει σε exception). Δοκιμάσαμε την ίδια μέθοδο και για τον κυρίως αλγόριθμο αλλά είδαμε ότι δεν δούλευε σωστά.

Περιμένουμε γενικά σημαντική βελτίωση στους κύκλους επεξεργασίας εφόσον γνωρίζουμε ότι η μεταφορά αποτελεί την πιο κοστοβόρα διαδικασία. Όντως πέφτουν άλλο μισό εκατομμύριο. Ωστόσο με make βλέπουμε ότι το total ROM αυξάνεται κάτι που ήταν όμως επίσης αναμενόμενο :



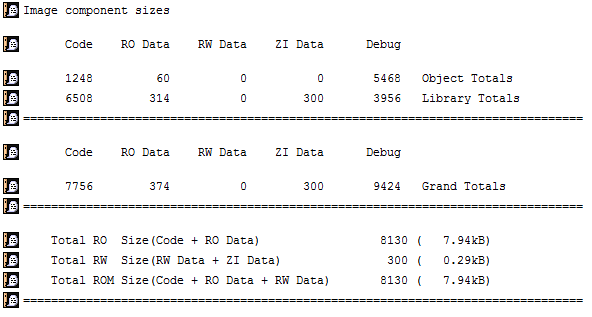
v6

**4η βελτιστοποίηση (v6.c) : Loop\_merging**

Η τελευταία βελτιστοποίηση που εφαρμόσαμε μειώνει την πρόσθετη επιβάρυνση των βρόχων και κανονικοποιεί τη δομή του αλγορίθμου. Πολλαπλά μειώνει τις περιττές προσπελάσει στη μνήμη.

Αυτή η βελτιστοποίηση εφαρμόστηκε με το να ενώσουμε τις δύο συναρτήσεις min και max σε μία. Αυτό βέβαια προϋποθέτει ότι αυτός που θα χρησιμοποιήσει τον το πρόγραμμα για να επεξεργαστεί την εικόνα του, όντως θα θέλει τα αποτελέσματα και από τους δύο αλγορίθμους.

Ο αντίστοιχος κώδικας που θα έπρεπε να παρουσιαστεί εδώ είναι ολόκληρη η συνάρτηση min\_max( ). Ωστόσο αποτελεί όντως ένα loop\_merging κι αυτό μπορούμε να το αντιληφθούμε αν σκεφτούμε ότι αντί για συναρτήσεις γράφαμε το κώδικα μέσα στη main. Τότε θα ήταν δυνατό ένα κανονικό loop\_merging. Οι κύκλοι μειώνονται κατά πολύ και τα αποτελέσματα με make φαίνονται παρακάτω :



v8

Έτσι ολοκληρώσαμε το πίνακα με τις μετρήσεις μας. Δείξαμε ότι με 4 βελτιστοποιήσεις μπορούμε να ρίξουμε τους κύκλους του επεξεργαστή κατά περισσότερο από 3 εκατομμύρια.  
Στο πίνακα στην επόμενη σελίδα φαίνεται η σταδιακή αυτή μείωση. Κάναμε επομένως το πρόγραμμά μας αποδοτικότερο.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Βελτιστοποίηση | Total Rom | Instructions | Total Cycles |
| v2.c | - | 12,790 | 7,997,961 | 14,063,053 |
| v3.c | Οριακές συνθήκες | 7,974 | 7,858,309 | 13,431,651 |
| v4.c | Απαλοιφή πίνακα k | 7,910 | 7,355,365 | 13,080,355 |
| v6.c | Loop\_unrolling | 8,278 | 7,275,617 | 12,506,081 |
| v8.c | Loop\_unrolling | 8,130 | 6,175,124 | 10,791,797 |

Μετρήσεις από κάθε αρχείο .c