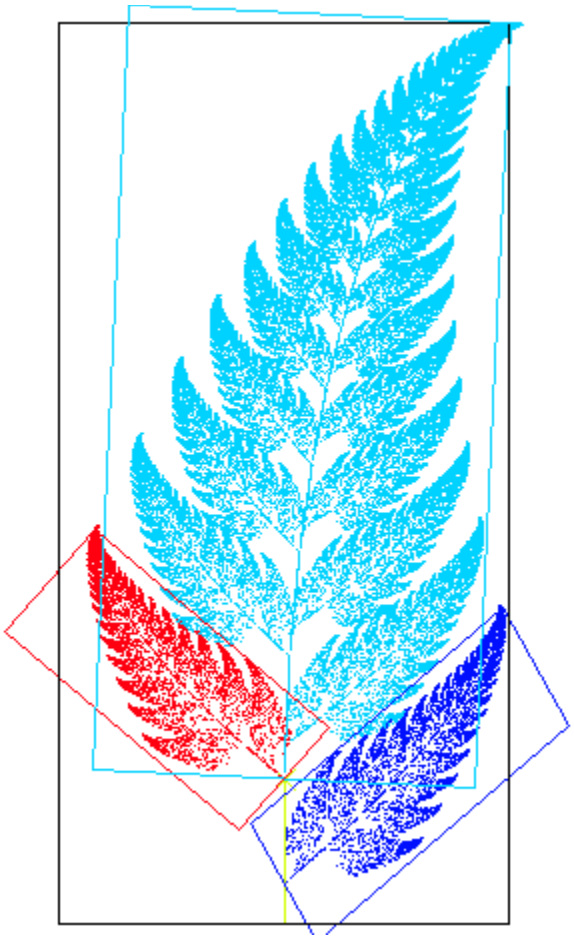
ΣΧΕΔΙΑΣΜΟΣ ΕΝΣΩΜΑΤΩΝ ΣΥΣΤΗΜΑΤΩΝ

Τρίτο 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

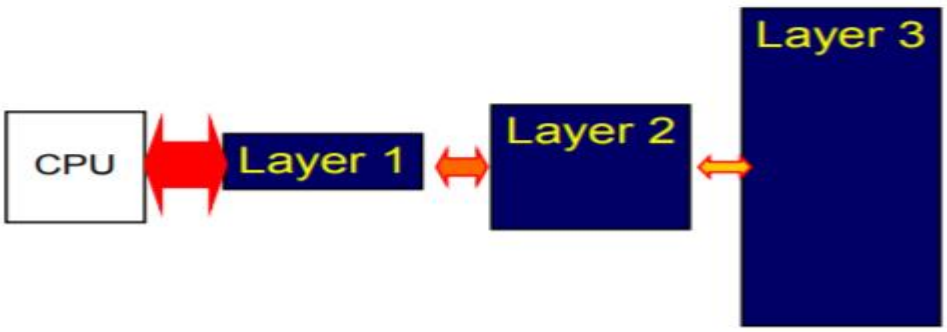
Εισαγωγή

Σκοπός της εργασίας είναι, αφού υλοποιηθεί σε γλώσσα C ο αλγόριθμος σε c εφαρμογής του συσχετισμένου μετασχηματισμού στην επεξεργασία εικόνας και βελτιστοποιηθεί με τεχνικές βελτιστοποίησης βρόγχων, να πραγματοποιηθούν οι κατάλληλοι μετασχηματισμοί επαναχρησιμοποίησης δεδομένων

Κατά την εφαρμογή των μετασχηματισμών επαναχρησιμοποίησης δεδομένων θα εξεταστεί μέσω του ARMulator του επεξεργαστή ARM τι αποτελέσματα έχει 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.

Θεωρητική Βάση

Με τους μετασχηματισμούς επαναχρησιμοποίησης δεδομένων στοχεύεται η εξοικονόμηση προσπελάσεων των πινάκων δεδομένων οι οποίοι έχουν μεγάλο μέγεθος. Αυτό επιτυγχάνεται με την χρήση buffers, δηλαδή μικρότερου μεγέθους πίνακες για την προσωρινή αποθήκευση δεδομένων που μπορεί να αποθηκευτούν πιο κοντά στον επεξεργαστή. Έτσι δημιουργείται μια ιεραρχία στην προσπέλασης της μνήμης στην οποία οι μικρές μνήμες (buffers) τοποθετούνται κοντύτερα στον επεξεργαστή και προσπελαύονται πολύ γρηγορότερα. Στα επίπεδα αυτά τοποθετούνται μεταβλητές που απαιτούν τον μεγαλύτερο αριθμό προσπελάσεων. Με αυτό τον τρόπο η μεταφορά δεδομένων από τη μνήμη στον επεξεργαστή -και αντίστροφα- γίνεται στα επίπεδα πλησίον του επεξεργαστή. Με αυτόν τον τρόπο δεν εξοικονομείται μόνο χρόνος αλλά και κατανάλωση ενέργειας



Υλοποίηση Πειράματος

Στην ουσία θέλουμε να πετύχουμε μείωση των προσπελάσεων των πινάκων current οι οποίοι έχουν μεγάλο μέγεθος, εισάγοντας πίνακες μικρότερου μεγέθους.

Σε κάθε εκτέλεση ενός βρόγχου τα δεδομένα που χρησιμοποιούνται θα μπορούσαν να αποθηκευτούν σε (3 πχ) buffers με διαστάσεις $1xN_{στηλών}$ (lines) ή με διαστάσεις KxK (Οπου Κ ένας μικρός αριθμός 3-5, συνήθως 3).

Στην πράξη να χρησιμοποιήσουμε την τεχνική επαναχρησιμοποίησης δεδομένων που μάθαμε στο εργαστήριο θα πρέπει στον κώδικάς μας και συγκεκριμένα στον αλγόριθμο που υλοποιούμε να υπάρχει μια μάσκα που εφαρμόζεται η ίδια κάθε φορά σε κάθε pixel η μπλοκ από pixel έτσι ώστε να μπορούμε να «σπάσουμε» σε υπό πίνακες εσωτερικά τους βρόγχους επανάληψης με σκοπό να χρησιμοποιηθούν μικρότερες και γρηγορότερες μνήμες για αυτούς.

Ωστόσο για τον δικό μας κώδικα οι μετασχηματισμοί επαναχρησιμοποίησης δεδομένων δεν μπορούν να έχουν ευρεία και ουσιαστική εφαρμογή (δηλαδή με buffers και blocks) διότι δεν χρησιμοποιείται κάποιος πίνακας kernel ούτε πραγματοποιούνται επαναλαμβανόμενες πράξεις με στοιχεία τα οποία δεν χρειάζεται να ανακληθούν. Πιο συγκεκριμένα κάθε φορά για κάθε συνδυασμό των i και j υπολογίζουμε ξεχωριστά τα (x,y) του κάθε πίξελ (κατ' επέκταση τα στοιχεία των current πινάκων που θα χρησιμοποιηθούν) και όχι με την χρήση μιας μάσκας.

Θα μπορούσε να προταθεί η χρήση ενός block με τα στοιχεία που χρειάζονται κάθε φορά xs, xs1,xs2 και ys, ys1,ys2 για να επιταχυνθούν κάποιες πράξεις. Στην ουσία όμως ούτε αυτό προσθέτει κάποια βελτιστοποίηση στον κώδικα, γιατί πολύ απλά σε κάθε επανάληψη η μόνη χρήση αυτών των στοιχείων είναι για να καταχωρηθούν στους νέους πίνακες new_y new_u new_v. Με άλλα λόγια δεν γίνεται επαναχρησιμοποίηση των ίδιων στοιχείων με προβλεπόμενο τρόπο.

Ωστόσο αυτό που θα μπορούσαμε να πετύχουμε με μετασχηματισμούς επαναχρησιμοποίησης δεδομένων είναι να τοποθετήσουμε στην γρήγορη sram τα στοιχεία που επαναχρησιμοποιούνται για να δούμε τι κέρδος σε ταχύτητα εκτέλεσης έχουμε.

Μετα το πέρας της πρώτης εργασίας, ασχοληθήκαμε, στην δεύτερη εργασία με την διαχείρηση της μνήμης του ενσωματομένου συστήματος που επιλέξαμε, είχαμε επιλέξει μία διεπίπεδη υλοποίηση με μία μνήμη ROM για τα read only data και μία γρήγορη sram για τα υπόλοιπα δεδομένα. Βέβαια αυτή η υλοποίηση ήταν πίο κοντα με μία ιδανική διάταξη, καθώς όταν πρέπει να ληφθουν υπόψην ο απαιτούμενος χώρος και το απαιτούμενο κόστος για αυτή η διάταξη δεν θα μπορούσε να είναι τόσο υλοποιήσημη.

αυτή τη φορά να βάλουμε τα δεδομένα που επαναχρησιμοποιούνται σε μια γρήγορη και μικρή μνήμη και τα υπόλοιπα δεδομένα σε μια σχετικά αργή και μεγάλη μνήμη.

1. Χωρίς ύπαρξη γρήγορης μνήμης :

Αρχεία

heap.c:

```
#include <rt_misc.h>

 __value_in_regs struct __initial_stackheap __user_initial_stackheap(
 unsigned R0, unsigned SP, unsigned R2, unsigned SL){

 struct __initial_stackheap config;

 //config.heap_limit  = 0x00724B40;

 //config.stack_limit = 0x00004000;
 //config.stack_limit = 0x00100000;

/* works */
config.heap_base   = 0x00260000;
config.stack_base  = 0x00460000;


/* factory defaults */
 //config.heap_base   = 0x00060000;
 //config.stack_base  = 0x00080000;
 return config;}
```

memory.map:

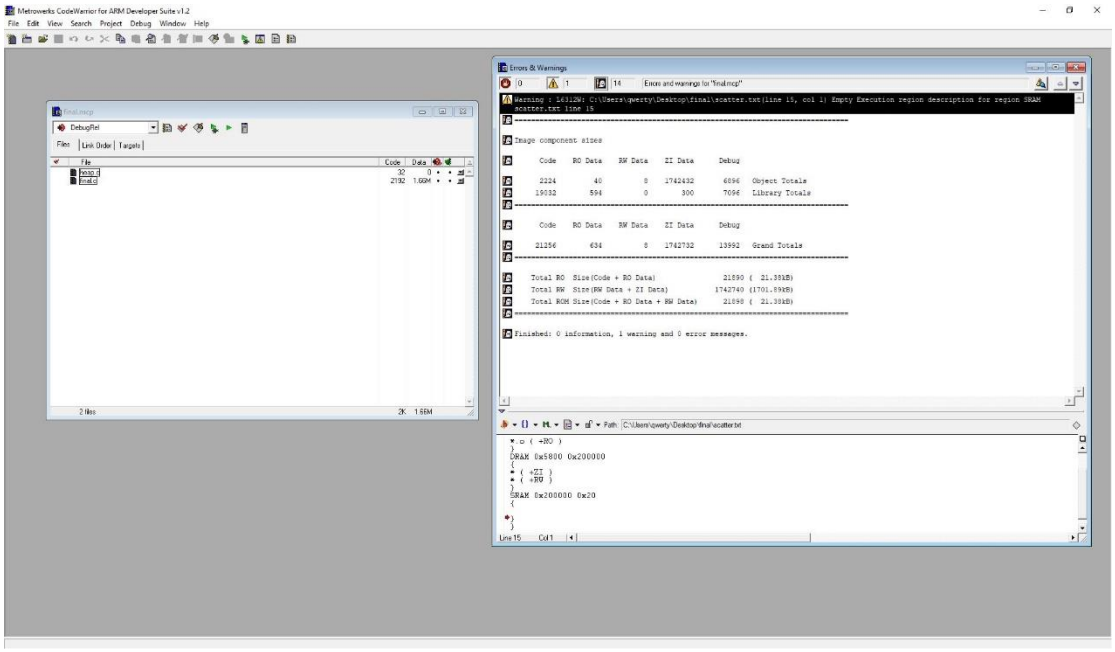
```
00000000 00005800 ROM 4 R 1/1 1/1
00005800 00200000 DRAM 4 RW 250/50 250/50
00200000 00000020 SRAM 4 RW 1/1 1/1
```

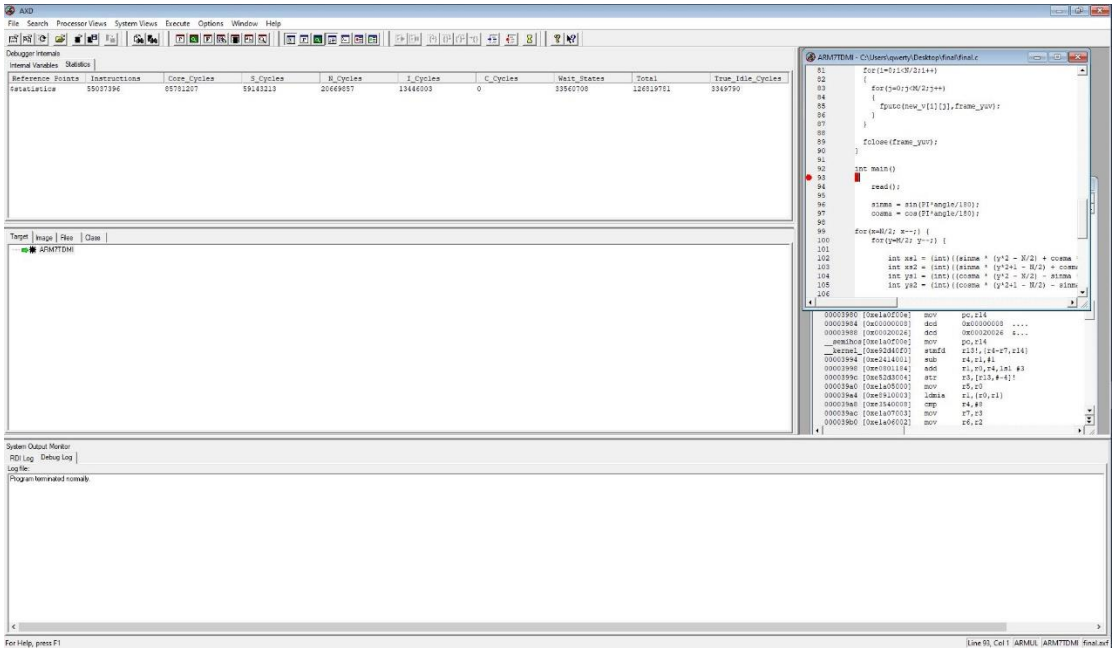
scatter.txt:

```
ROM 0x0 0x08000000
{
    ROM 0x0 0x5800
    {
        *.o ( +RO )
    }
    DRAM 0x5800 0x200000
    {
        * ( +ZI )
        * ( +RW )
    }
    SRAM 0x200000 0x20
    {

    }
}
```

Αποτελέσματα





2. Ύπαρξη γρήγορης μνήμης

Στην μικρή μνήμη θα συμπεριλάβουμε τα εξής :

```
1.  /* code for armulator*/
2.  #pragma arm section zidata="manual"
3.  int i;
4.  int j;
5.  int x;
6.  int y;
7.  double sinma;
8.  double cosma;
9.  double angle =54;
10. #pragma arm section
```

Τα δεδομένα που θα επαναχρησιμοποιήσουμε είναι οι δείκτες στις λούπες η γωνία και το ημίτονο και το συνημίτονο. Γνωρίζουμε ότι μία double μεταβλητή είναι 8 bytes και οι int μεταβλητές που θα χρησιμοποιήσουμε θα είναι από 2 bytes (κανονικά είναι 4 αλλά για ακεραίους στο εύρος -32767 εως 32767 είναι 2 bytes). Οπότε η χωρητικότητα που θα χρειαστούμε θα είναι :

4x 2 bytes + 3x 8 bytes= 32 bytes. Στο δεκαεξαδικό : 0x20

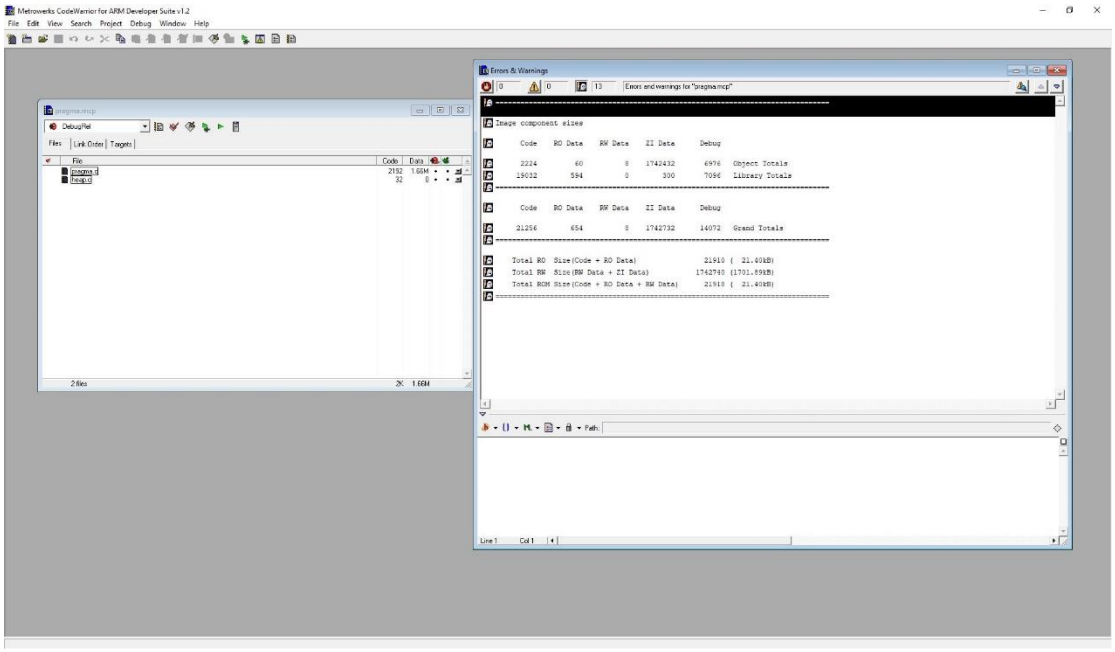
Αρχεία

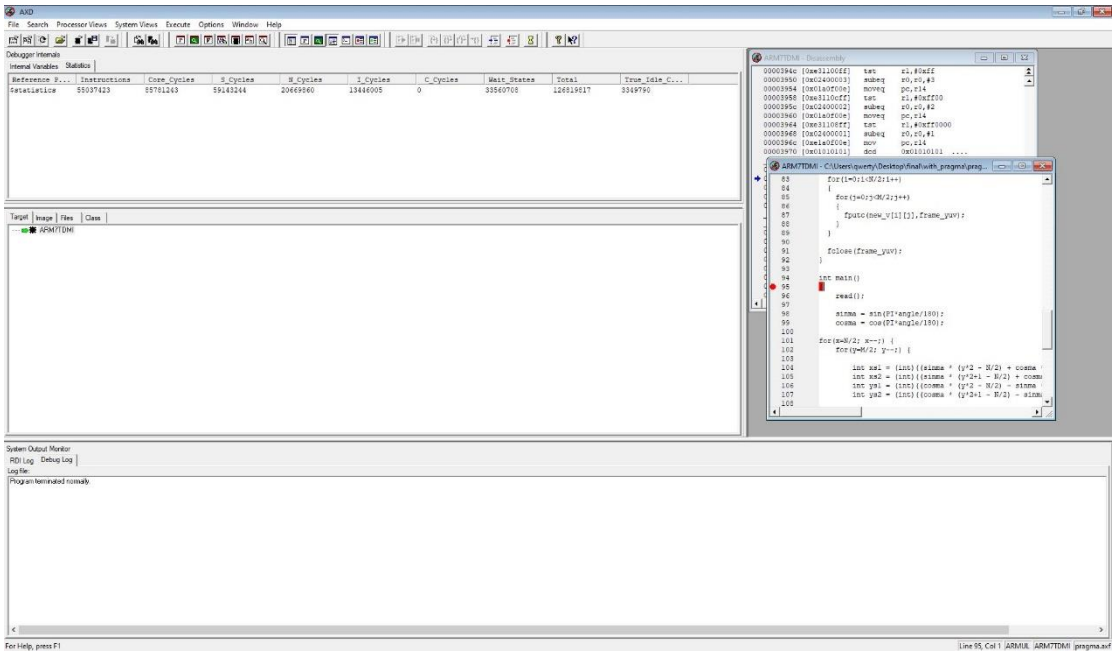
Το αρχείο scatter μεταβάλλεται ως εξής :

scatter.txt:

```
17. ROM 0x0 0x08000000
18. {
19.     ROM 0x0 0x5800
20.     {
21.         *.o ( +RO )
22.     }
23.     DRAM 0x5800 0x200000
24.     {
25.         * ( +ZI )
26.         * ( +RW )
27.     }
28.     SRAM 0x200000 0x20
29.     {
30.          * ( manual )
31.     }
32. }
```

Αποτελέσματα





Παρατηρησεις

Μετά τις δοκιμές που έγιναν με την χρήση και μη της γρήγορης μνήμης παρατηρούμε ότι δεν υπάρχει κάποια αισθητή διαφορά στον συνολικό αριθμό των κύκλων αλλα και στα wait states του επεξεργαστή μας.

Συμπεράσματα

Όπως ήταν αναμενόμενο η εφαρμογή αυτών των μετασχηματισμών δεν επιφέρει ουσιαστική βελτιστοποίηση, καθώς τα δεδομένα που επαναχρησιμοποιούνται και μπορούμε να τα βάλουμε στην γρήγορη μνήμη δεν απιτούν και ιδιαιτερη ισχύ ούτε εκτελούν κάποια χρονοβόρα διαδικασία (την οποία μπορούμε να ελαφρύνουμε τοποθετώντας τα σε μία μικρή μνήμη κοντά στον επεξεργαστή). Τα δεδομένα που απαιτούν την μεγαλύτερη ισχύ για την προσπέλασή τους και τον περισσότερο χρόνο είναι αυτά των πινάκων από τις εικόνες μας τα οποία όμως προσπελάυνονται και χρησιμοποιούνται με τέτοιο τρόπο που δεν είναι δυνατή η εφαρμογή του μετασχηματισμού επαναχρησιμοποίησης δεδομένων πάνω σε αυτά. Μπορούμε να καταλήξουμε στο γεγονός ότι ενώ αυτοί οι μετασχηματισμοί σαν γενικός κανόνας μπορούν να είναι ιδιαίτερα βοηθητικοί και να αυξήσουν τις αποδόσεις ενός προγράμματος δεν έχουν πάντα ικανοποιητικές εφαρμογές.