1ο μέρος εργασίας του μαθήματος «Σχεδιασμός Ενσωματωμένων Συστημάτων» Ομάδα 10

Φοιτητές:

Ηλίας Παπαδέας 56989

Χριστόφορος Σπάρταλης 56785

Table of Contents

Εισαγωγή....2
Υψηπερατό φίλτρο στην επεξεργασία εικόνας....2
Πίνακας δεδομένων....5
    Μέγεθος....5
    Αριθμός Προσπελάσεων....5
1. Loop Unrolling....6
    Θεωρητικό σκέλος....6
    Παρατηρήσεις....6
    Συμπέρασμα....7
2. Loop Fusion-Fission....7
    Θεωρητικό σκέλος....8
    Παρατηρήσεις....8
    Συμπεράσματα....9
3. Loop Interchange....10
    Θεωρητικό σκέλος....10
    Παρατηρήσεις....11
    Συμπέρασμα....12
4. Loop Tiling....12
    Θεωρητικό σκέλος....12
    Παρατηρήσεις....13
    Συμπεράσματα....17
5. Loop Collapsing....18
    Θεωρητικό σκέλος....18
    Παρατηρήσεις....18
    Συμπεράσματα....19
6. Loop Inversion....20
    Θεωρητικό σκέλος....20
    Παρατηρήσεις....20
    Συμπεράσματα....22
Βιβλιογραφία....23

Εισαγωγή

Η παρούσα εργασία επαφίεται στη βελτιστοποίηση των βρόγχων επανάληψης που χρησιμοποιούνται για την επεξεργασία μιας εικόνας. Πιο συγκεκριμένα η διαδικασία που επιτελείται στην παρούσα εργασία είναι η υλοποίηση ενός υψηπερατού φίλτρου επεξεργασίας εικόνας σε γλώσσα προγραμματισμού C.

Η εργασία αποτελείται από μία σύντομη αναφορά/ περιγραφή σχετικά με τη λειτουργία και τη χρήση του φίλτρου, από την εξέταση του πίνακα δεδομένων και από 6 επιπλέον ενότητες. Κάθε ενότητα πραγματεύεται μία συγκεκριμένη τεχνική βελτιστοποίησης. Κάθε ενότητα διαρθρώνεται από 3 μέρη: μία θεωρητική προσέγγιση της τεχνικής βελτιστοποίησης, τα αποτελέσματα του armulator και τις παρατηρήσεις μας επ’ αυτών και τέλος τα συμπεράσματα μας.

Υψηπερατό φίλτρο στην επεξεργασία εικόνας

Στην επιστήμη της Ψηφιακής Επεξεργασίας Εικόνας, γίνεται χρήση υψηπερατών φίλτρων για την ανάδειξη και τον τονισμό των λεπτομερειών μιας εικόνας. Με άλλα λόγια, το υψηπερατό φίλτρο στην επεξεργασία εικόνας εφαρμόζεται στο sharpening της εικόνας, μέσω αύξησης του contrast των άκρων (edges). Για να το πετύχουμε αυτό, χρησιμοποιούμε μια μάσκα που αποτελείται από ένα μείγμα θετικών και αρνητικών συντελεστών. Οι θετικοί συντελεστές τοποθετούνται στο κέντρο, ενώ οι αρνητικοί κατανέμονται στην περιφέρεια. Επιπλέον, το άθροισμα των συντελεστών στη μάσκα που χρησιμοποιήθηκε είναι ίσο με μηδέν. Όταν η μάσκα εφαρμόζεται σε μια περιοχή σταθερής ή αργά μεταβαλλόμενης φωτεινότητας, το αποτέλεσμα θα είναι μηδέν ή κοντά στο μηδέν.

Παρ’ όλα αυτά, στις περιοχές γρήγορα μεταβαλλόμενης φωτεινότητας, το αποτέλεσμα θα είναι μεγάλος αριθμός, θετικός ή αρνητικός, μιας και η μάσκα περιέχει και θετικούς και αρνητικούς συντελεστές. Συνεπώς, οι τιμές που προκύπτουν πρέπει να βρίσκονται στο διάστημα [0,255].
Το 0 τοποθετείται στο μέσο της κλίμακας, δηλαδή στο 127. Αυτό σημαίνει πως οι αρνητικές τιμές κλιμακώνονται στην περιοχή [0,127], ενώ οι θετικές στην περιοχή [128,255]. Στις αρνητικές αντιστοιχούν οι σκοτεινοί τόνοι, ενώ στις θετικές οι φωτεινοί τόνοι.

Η υλοποίηση του κώδικα έγινε σε γλώσσα C. Χρησιμοποιήθηκε το εξής φίλτρο από το βιβλίο του κ. Παπαμάρκου, Ψηφακή Επεξεργασία και Ανάλυση Εικόνας:

```
// int highPassFilter [3][3] = {{-1, -1, -1} ,
//                              {-1,  8, -1} ,
//                              {-1, -1, -1} };
```

Οι αλγόριθμοι που αναπτύχθηκαν μελετήθηκαν ως προς το χώρο και το χρόνο που απαιτούν στο λογισμικό Metrowerks CodeWarrior for ARM Developer Suite v1.2 και στο AXD Debugger.

Η εικόνα που επεξεργαστήκαμε είναι η akiyoy.yuv:

Με την εφαρμογή της μάσκας, η εικόνα που εξάγαμε είναι η εξής:

Το τελικό αποτέλεσμα, μετά από sharpening της εικόνας θα ήταν το ακόλουθο (δεν υλοποιήθηκε στις τεχνικές βελτιστοποίησης):

Ουσιαστικά, είναι ο συνδυασμός της αρχικής-μη επεξεργασμένης εικόνας με τη μάσκα. Δηλαδή η πρόσθεση της μάσκας πάνω στην αρχική εικόνα

Αποτελέσματα του armulator για τον αρχικό κώδικα (highpass.c):

Image comonent sizes:

```
0   0   13   Errors and warnings for "highpass.mcp"
==============================================================================
Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

       940         60          0    1226792       4720   Object Totals
     10196        314          0        300       4548   Library Totals
==============================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     11136        374          0    1227092       9268   Grand Totals
==============================================================================

    Total RO  Size(Code + RO Data)                11510 (  11.24kB)
    Total RW  Size(RW Data + ZI Data)           1227092 (1198.33kB)
    Total ROM Size(Code + RO Data + RW Data)      11510 (  11.24kB)
==============================================================================
```

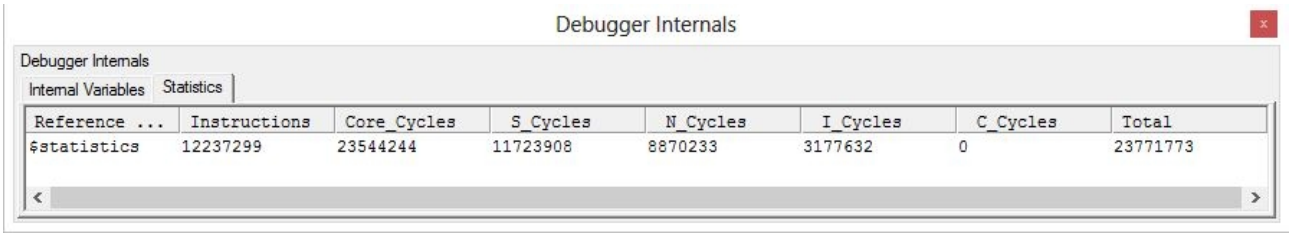
- RO data (Read Only data): Δεδομένα των οποίων την τιμή δεν μπορώ να μεταβάλλω. Για παράδειγμα οι σταθερές. Δηλαδή:
  #define N 288 / frame dimension for QCIF format */*
  #define M 352 / frame dimension for QCIF format */*
  #define filename "akiyoy.yuv"
  #define finalfilename "akiyo_high_y.yuv"

- RW data (Read/ Write data): Δεδομένα τα οποία μπορώ και να τα διαβάσω και να τους αναθέσω τιμή.

- ZI data (Zero Initialized data): Είναι οι global μεταβλητές. Τα δεδομένα τα οποία είναι έξω από τη main. Αυτά με το που «κατέβουν» στον μικροεπεξεργαστή θα πάρουν τιμή μηδέν. Για παράδειγμα:
  int current_y[N][M];
  int temparray[N+2][M+2];
  int newtemparray[N+2][M+2];
  int i,j;

- Code: Πρόκειται για τον κώδικά μας, οποίος καταλαμβάνει έναν συγκεκριμένο χώρο στη μνήμη.

Debugger Internals Statistics:



Reference ...	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	12237299	23544244	11723908	8870233	3177632	0	23771773

- Core_Cycles: Οι κύκλοι στον επεξεργαστή.
- S_Cycles (Sequential Cycles): Πλήθος κύκλων που προχωράμε σειριακά.
- N_Cycles (Non-Sequential Cycles): Όταν προχωράμε και χρειάζεται να μην εκτελεστεί ακολουθιακά η εντολή (π.χ. έχω μια διακλάδωση).
- C_Cycles: Οι κύκλοι στον συνεπεξεργαστή (π.χ. κάρτα γραφικών γενικού σκοπού)

Πίνακας δεδομένων

Μέγεθος

Ο πίνακας που περιέχει τα τελικά δεδομένα είναι ο current_y, ο οποίος έχει διαστάσεις M=352 επί N=288. Τα δεδομένα που αποθηκεύονται σ’ αυτόν τον πίνακα είναι τύπου integer (int). Κάθε ακέραιος τύπου int έχει μέγεθος 4 Bytes. Άρα το μέγεθος του πίνακα δεδομένων υπολογίζεται ως εξής:

Μέγεθος Πίνακα Δεδομένων = M * N * 4 Bytes = 405504 Bytes = 396 Kbytes

Αριθμός Προσπελάσεων

Στον αρχικό κωδικά, όπως επίσης στο loop unrolling, το loop fusion, το loop interchange, το loop collapsing και το loop inversion προσπελαύνουμε κάθε στοιχείο του πίνακα δεδομένων 5 φορές

1. Όταν αρχικοποιείται ο πίνακας current_y
2. Όταν εκχωρούμε τιμές στον πίνακα current_y με τη συνάρτηση fgetc()
3. Όταν εκχωρούμε τιμές από τον πίνακα current_y στον πίνακα temparray
4. Όταν εκχωρούμε τιμές στον πίνακα current_y από τον πίνακα newtemparray και κάνουμε την κλιμάκωση
5. Όταν λαμβάνουμε τιμές από τον current_y με τη συνάρτηση fputc()

Επομένως, ο συνολικός αριθμός προσπελάσεων είναι:

Συνολικός Αριθμός Προσπελάσεων = 5 * M * N = 506880

Στην περίπτωση του loop fission προσπελαύνουμε κάθε στοιχείο του πίνακα current_y 8 φορές, αφού χρησιμοποιούνται τρεις επιπλέον διακριτοί βρόγχοι επανάληψης για την κλιμάκωση. Επομένως:

Συνολικός Αριθμός Προσπελάσεων = 8 * M * N = 811008

Στην περίπτωση του loop tiling προσπελαύνουμε κάθε στοιχείο του πίνακα current_y 6 φορές, αφού χρησιμοποιούνται ένας επιπλέον διακριτός βρόγχος επανάληψης για την κλιμάκωση. Επομένως:

Συνολικός Αριθμός Προσπελάσεων = 8 * M * N = 608256

1. Loop Unrolling

Θεωρητικό σκέλος

Το loop unrolling είναι μια τεχνική μετασχηματισμού του loop που προσπαθεί να βελτιστοποιήσει την ταχύτητα εκτέλεσης του προγράμματος σε βάρος του δυαδικού μεγέθους του. Είναι μια προσέγγιση γνωστή ως αντάλλαγμα χώρου-χρόνου (space-time tradeoff). Ο μετασχηματισμός μπορεί να γίνει χειροκίνητα από τον προγραμματιστή ή από έναν μεταγλωττιστή βελτιστοποίησης.

Ο στόχος του loop unrolling είναι να αυξηθεί η ταχύτητα του προγράμματος μειώνοντας ή εξαλείφοντας τις εντολές που ελέγχουν τον βρόχο. Από την άλλη, όμως, όσο ξετυλίγουμε τους βρόγχους επανάληψης τόσο αυξάνεται το μέγεθος του κώδικα. Επομένως καταλαμβάνει περισσότερο χώρο στην μνήμη και μειώνεται ο διαθέσιμο χώρος για την αποθήκευση άλλων δεδομένων.

Συνοψίζοντας, υπάρχει ένα tradeoff με το πόσους ελέγχους (if ...) θα γλιτώσω και με το πόση διαθέσιμη μνήμη θα μου απομείνει.

Παρατηρήσεις

Αποτελέσματα armulator:

Image comonent sizes:

```
0   0   13   Errors and warnings for "unrolling.mcp"
==============================================================================
Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      1344         60          0    1226792       4892   Object Totals
     10196        314          0        300       4548   Library Totals
==============================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     11540        374          0    1227092       9440   Grand Totals
==============================================================================

    Total RO  Size(Code + RO Data)                11914 (  11.63kB)
    Total RW  Size(RW Data + ZI Data)           1227092 (1198.33kB)
    Total ROM Size(Code + RO Data + RW Data)      11914 (  11.63kB)
==============================================================================
```

Παρατηρούμε ότι αυξήθηκε το μέγεθος του κώδικα περίπου 400 Bytes

Debugger Internals Statistics:

Debugger Internals

Debugger Internals

Internal Variables | Statistics

Reference ...	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	9982638	19314571	9162723	7885456	2493921	0	19542100

Παρατηρούμε ότι μειώθηκαν οι συνολικοί κύκλοι περίπου 18%

Συμπέρασμα

Όπως περιμέναμε, αυξήθηκε το μέγεθος του κώδικα, επειδή ξετυλίξαμε τους βρόγχους επανάληψης και ταυτόχρονα μειώθηκαν οι συνολικοί κύκλοι, διότι χρειάστηκε να γίνουν λιγότεροι έλεγχοι (για το εάν το i και j ξεπερνούν τα όρια που θέσαμε).

2. Loop Fusion-Fission

Θεωρητικό σκέλος

Στην επιστήμη των υπολογιστών, loop fission (ή loop distribution) είναι μια βελτιστοποίηση εφαρμογών, στην οποία, ένα loop διασπάται σε πολλαπλά loops τα οποία έχουν κοινό index. Το κάθε ένα από αυτά τα loops αποτελεί ένα μέρος του αρχικού loop. Στόχος είναι να σπάσει ένα μεγάλο loop σε μικρότερα, ώστε να επιτευχθεί καλύτερη χρήση της τοπικότητας αναφοράς (locality of reference). Αυτή η βελτιστοποίηση είναι πιο αποτελεσματική σε πολυ-πύρηνους επεξεργαστές που μπορούν να χωρίσουν μια διεργασία σε πολλαπλές διεργασίες για κάθε επεξεργαστή.

Αντιστρόφως, loop fusion (ή loop jamming) είναι μια βελτιστοποίηση και ένας μετασχηματισμός του loop, που αντικαθιστά πολλαπλά loops σε ένα μοναδικό. Αυτή η μέθοδος είναι εφικτή όταν δύο loops επαναλαμβάνονται στο ίδιο εύρος και δεν αναφέρονται στα δεδομένα του άλλου. Η βελτιστοποίηση loop fusion δε βελτιώνει πάντα την ταχύτητα εκτέλεσης. Σε ορισμένες αρχιτεκτονικές, δύο loops μπορεί να έχουν καλύτερη απόδοση από ένα loop.

Παρατηρήσεις

Αποτελέσματα του armulator:

Image comonent sizes:

I. Loop Fusion

```
0    0    13    Errors and warnings for "loop_fusion.mcp"
==============================================================================
Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

       772         60          0    1226792       4740   Object Totals
     10196        314          0        300       4548   Library Totals
==============================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     10968        374          0    1227092       9288   Grand Totals
==============================================================================

    Total RO  Size(Code + RO Data)                11342 (  11.08kB)
    Total RW  Size(RW Data + ZI Data)           1227092 (1198.33kB)
    Total ROM Size(Code + RO Data + RW Data)      11342 (  11.08kB)
==============================================================================
```

II. Loop Fission

```
0   0   13   Errors and warnings for "loop-fission.mcp"
==============================================================================
Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      1076         60          0    1226792       4852   Object Totals
     10196        314          0        300       4548   Library Totals
==============================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     11272        374          0    1227092       9400   Grand Totals
==============================================================================

    Total RO  Size(Code + RO Data)                11646 (  11.37kB)
    Total RW  Size(RW Data + ZI Data)           1227092 (1198.33kB)
    Total ROM Size(Code + RO Data + RW Data)      11646 (  11.37kB)
==============================================================================
```

Παρατηρούμε ότι με το loop fusion μειώνεται το μέγεθος του κώδικα, ενώ με το loop fission αυξάνεται

Debugger Internals Statistics:

I. Loop Fusion

Debugger Internals

Debugger Internals

Internal Variables | Statistics

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	10201661	19578229	9995997	7140004	2668795	0	19804796

II. Loop Fission

Debugger Internals

Debugger Internals

Internal Variables | Statistics

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	13357343	25883102	12842796	9785499	3482976	0	26111271

Παρατηρούμε ότι στην πρώτη περίπτωση μειώθηκαν οι συνολικοί κύκλοι περίπου 16.7%, ενώ στη δεύτερη περίπτωση αυξήθηκαν περίπου 10%

Συμπεράσματα

I. Loop Fusion: Ο κώδικας μειώθηκε καθώς συνενώσαμε τους βρόγχους επανάληψης με κοινό index. Επίσης, ένας άμεσα αντιληπτός λόγος στον οποίο μπορεί να οφείλεται η μείωση των συνολικών κύκλων είναι το γεγονός ότι έχει μειωθεί το πλήθος των εντολών στον επεξεργαστή. Ακόμη, μία άλλη αιτία, στην οποία θα μπορούσαμε να αποδώσουμε αυτήν την βελτιστοποίηση είναι το ότι επεξεργαζόμαστε γίνονται ακολουθιακά πολλές μεταβολές/ έλεγχοι σε ένα συγκεκριμένο στοιχείο του πίνακα (βλέπε βρόγχο επανάληψης που υλοποιείται το φίλτρο – loop_fusion.c). Επομένως, περιορίζεται η πιθανότητα το δεδομένο αυτό να έρθει στην cache μετά να αντικατασταθεί και στη συνέχει να χρειαστεί να το ξανακαλέσουμε κ.ο.κ.

II. Loop Fission: Ο κώδικας σαφώς έχει αυξηθεί αφού καταλήξαμε να έχουμε περισσότερους βρόγχους επανάληψης. Πιστεύουμε ότι η αύξηση των κύκλων ενδέχεται να έγκειται στην τοπικότητα αναφοράς της cache. Δηλαδή αυτό που εξηγήσαμε παραπάνω. Με άλλα λόγια, εκχωρούμε μία τιμή σ’ ένα στοιχείο του πίνακα current_y (βλ. loop_fission.c), στη συνέχεια αυτό αντικαθίσταται από άλλα. Μετά χρειάζεται να εφαρμόσουμε την κλιμάκωση πάνω σ’ αυτό το στοιχείο, όμως πλέον δε βρίσκεται στην cache και χρειάζεται να το ξαναφέρουμε κ.ο.κ

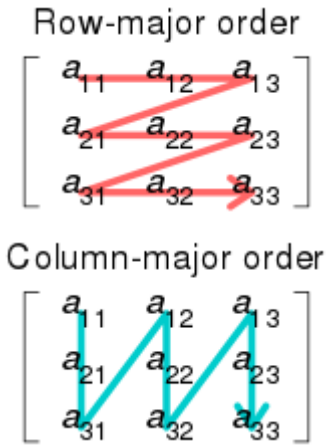
3. Loop Interchange

Θεωρητικό σκέλος

Είναι η διαδικασία ανταλλαγής της σειράς δύο μεταβλητών επανάληψης που χρησιμοποιούνται από έναν εμφωλευμένο βρόχο. Η μεταβλητή που χρησιμοποιείται στο εσωτερικό loop μεταβαίνει στο εξωτερικό και αντίστροφα. Συχνά, για να διασφαλιστεί ότι τα στοιχεία ενός πολυδιάστατου πίνακα είναι προσπελάσιμα με τη σειρά στην οποία βρίσκονται στη μνήμη, βελτιώνοντας της τοπικότητα αναφοράς.

Ο βασικός σκοπός του loop interchange είναι να εκμεταλλευτεί τη cpu cache κατά την πρόσβαση σε στοιχεία πίνακα. Όταν ένας επεξεργαστής προσπελαύνει ένα στοιχείο πίνακα για πρώτη φορά, θα ανακτήσει ένα ολόκληρο μπλοκ δεδομένων από την κύρια μνήμη στη μνήμη cache. Αυτό το μπλοκ είναι πιθανό να έχει πολλά περισσότερα διαδοχικά στοιχεία μετά την πρώτη φορά που θα κληθεί, έτσι στην επόμενη πρόσβαση σε στοιχείο του πίνακα, θα οδηγηθεί απευθείας από την cache (κάτι το οποίο είναι γρηγορότερο από το να οδηγούταν από την αργή κύρια μνήμη). Αποτυχίες της cache συμβαίνουν αν τα συνεχόμενα προσπελάσιμα στοιχεία του πίνακα μέσα στο loop προέρχονται από διαφορετικό μπλοκ της μνήμης cache. Το loop interchange μπορεί να βοηθήσει ώστε αυτό να αποτραπεί. Η αποδοτικότητα του loop interchange εξαρτάται και πρέπει να εξεταστεί υπό το πρίσμα του μοντέλου της cache, που χρησιμοποιείται από το βαθύτερο hardware και το μοντέλο πίνακα που χρησιμοποιείται από τον compiler.

Στη γλώσσα προγραμματισμού 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). Οι compilers βελτιστοποίησης μπορούν να εντοπίσουν την ακατάλληλη σειρά από τους προγραμματιστές και να αντικαταστήσει τη σειρά ώστε να επιτευχθεί καλύτερη απόδοση της κρυφής μνήμης(cache)



Ωστόσο, το loop interchange μπορεί να οδηγήσει σε χειρότερη επίδοση. Δεν είναι πάντα ασφαλές να ανταλλάσσονται οι μεταβλητές επανάληψης λόγω των εξαρτήσεων μεταξύ δηλώσεων για τη σειρά με την οποία πρέπει να εκτελούνται. Συνεπώς, για να προσδιοριστεί εάν ένας μεταγλωττιστής μπορεί να αλλάξει με ασφάλεια βρόχους, απαιτείται ανάλυση ανάλυσης εξάρτησης.

Παρατηρήσεις

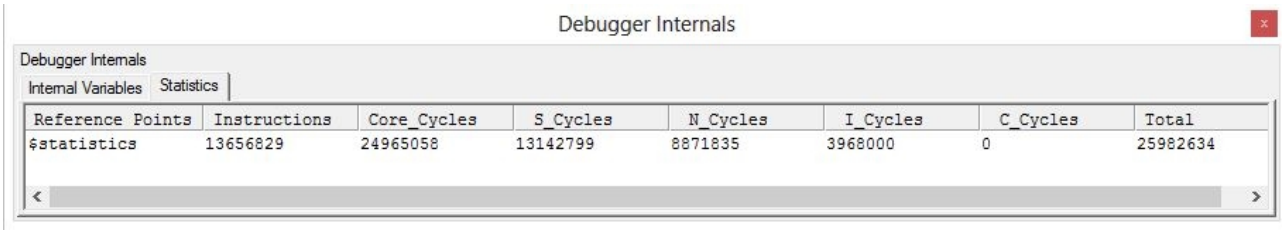
Αποτελέσματα armulator:

Image comonent sizes:

```
0   0   13   Errors and warnings for "loop_interchange.mcp"
==============================================================================
Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

       924         60          0    1226792       4768   Object Totals
     10196        314          0        300       4548   Library Totals
==============================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     11120        374          0    1227092       9316   Grand Totals
==============================================================================

    Total RO  Size(Code + RO Data)                11494 (  11.22kB)
    Total RW  Size(RW Data + ZI Data)           1227092 (1198.33kB)
    Total ROM Size(Code + RO Data + RW Data)      11494 (  11.22kB)
==============================================================================
```

Παρατηρούμε ότι το μέγεθος του κώδικα μειώθηκε ελάχιστα

Debugger Internals Statistics:



Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	13656829	24965058	13142799	8871835	3968000	0	25982634

Παρατηρούμε ότι οι συνολικοί κύκλοι αυξήθηκαν κατά 9.3%

Συμπεράσμα

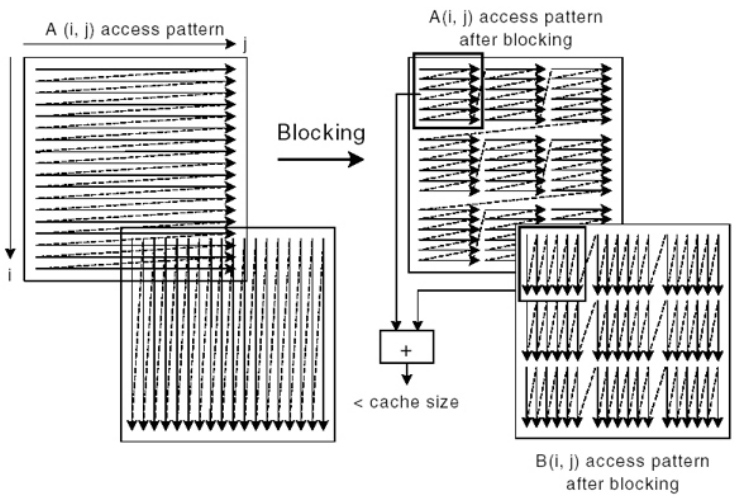
Η μείωση του μεγέθους του κώδικα είναι ελάχιστη και πιθανότατα οφείλεται σε κάποιες μικροδιαφορές που εντοπίζονται ανάμεσα στους δύο κώδικες. Η αύξηση των συνολικών κύκλων απορρέει πρώτα και κύρια από το γεγονός ότι στη C τα δεδομένα των πινάκων αποθηκεύονται κατά γραμμή και αυτό συνάδει με τον τρόπο με τον οποίο αποθηκεύονται στην μνήμη και διαβάζονται από αυτήν.

4. Loop Tiling

Θεωρητικό σκέλος

Ουσιαστικά αυτό που κάνουμε είναι να χωρίζουμε νοητά έναν πίνακα σε μικρότερους υποπίνακες (πλακάκια – tiles) και να τα επεξεργαζόμαστε κατά αυτόν τον τρόπο. Αυτό σαφώς αλλάζει και τον τρόπο με τον οποίο μεταβιβάζονται τα δεδομένα από την κύρια μνήμη στην μνήμη cache. Καθώς τα δεδομένα τα οποία χρησιμοποιούμε δεν ακολουθούν το συνηθισμένο μοτίβο προσπέλασης.

Υπάρχουν δύο τρόποι με τους οποίος μπορεί να επιτευχθεί το tiling. 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).



Σημείωση αναφορικά με τον τρόπο εργασίας μας:
Όσον αφορά την πρώτη περίπτωση (ο πιο εσωτερικός εμφωλευμένος βρόγχος μετράει τις στήλες), χωρίσαμε τους δισδιάστατους πίνακες σε tiles διαστάσεων:

- 288 x 22
- 288 x 32
- 288 x 44
- 288 x 88
- 288 x 176

Όσον αφορά τη δεύτερη περίπτωση (ο πιο εσωτερικός εμφωλευμένος βρόγχος μετράει τις γραμμές), χωρίσαμε τους δισδιάστατους πίνακες σε tiles διαστάσεων:

- 24 x 352
- 32 x 352
- 36 x 352
- 48 x 352
- 72 x 352
- 96 x 352
- 144 x 352

Παρατηρήσεις

Αποτελέσματα armulator:

Image comonent sizes:

I. Πρώτη περίπτωση

```
0   0   13   Errors and warnings for "loop_tilling.mcp"
==============================================================================
Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      1084         60          0    1226796       4908   Object Totals
     10196        314          0        300       4548   Library Totals
==============================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     11280        374          0    1227096       9456   Grand Totals
==============================================================================

    Total RO  Size(Code + RO Data)                11654 (  11.38kB)
    Total RW  Size(RW Data + ZI Data)           1227096 (1198.34kB)
    Total ROM Size(Code + RO Data + RW Data)      11654 (  11.38kB)
==============================================================================
```

Παρατηρούμε ότι για κάθε ξεχωριστό tile του συγκεκριμένου τρόπου προσέγγισης προκύπτει ακριβώς ο ίδιος πίνακας δεδομένων. Επίσης αυξήθηκε το μέγεθος του κώδικα και τα ZI Data κατά 4 Bytes.

II. Δεύτερη περίπτωση

```
0   0   13   Errors and warnings for "invertedN.mcp"
==============================================================================
Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      1024         60          0    1226796       4956   Object Totals
     10196        314          0        300       4548   Library Totals
==============================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     11220        374          0    1227096       9504   Grand Totals
==============================================================================

    Total RO  Size(Code + RO Data)                11594 (  11.32kB)
    Total RW  Size(RW Data + ZI Data)           1227096 (1198.34kB)
    Total ROM Size(Code + RO Data + RW Data)      11594 (  11.32kB)
==============================================================================
```

Παρατηρούμε ότι για κάθε ξεχωριστό tile του συγκεκριμένου τρόπου προσέγγισης προκύπτει ακριβώς ο ίδιος πίνακας δεδομένων. Επίσης αυξήθηκε το μέγεθος του κώδικα και τα ZI Data κατά 4 Bytes.

Debugger Internals Statistics:

288 x 22

Debugger Internals

Debugger Internals

Internal Variables | Statistics

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	12170641	23749783	11654172	9089736	3267463	0	24011371

288 x 32

Debugger Internals

Debugger Internals

Internal Variables | Statistics

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	12094146	23615608	11574827	9047891	3243233	0	23865951

288 x 44

Debugger Internals

Debugger Internals

Internal Variables | Statistics

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	12060429	23557723	11537660	9033224	3230435	0	23801319

288 x 88

Debugger Internals

Debugger Internals

Internal Variables | Statistics

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	12072313	23586103	11536824	9062388	3221491	0	23820703

288 x 176

Debugger Internals

Debugger Internals

Internal Variables | Statistics

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	12212235	23849113	11651246	9191810	3236159	0	24079215

24 x 352

Debugger Internals

Debugger Internals

Internal Variables | Statistics

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	13070792	24501148	12559057	8967472	3652481	0	25179010

32 x 352

Debugger Internals

Debugger Internals

Internal Variables | Statistics

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	13037990	24436630	12526273	8942083	3646136	0	25114492

36 x 352

Debugger Internals

Debugger Internals

Internal Variables | Statistics

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	13027056	24415124	12515345	8933620	3644021	0	25092986

48 x 352

Debugger Internals

Debugger Internals

Internal Variables | Statistics

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	13005188	24372112	12493489	8916694	3639791	0	25049974

72 x 352

Debugger Internals

Debugger Internals

Internal Variables | Statistics

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	12983320	24329100	12471633	8899768	3635561	0	25006962

96 x 352

Debugger Internals

Debugger Internals

Internal Variables | Statistics

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	12972386	24307594	12460705	8891305	3633446	0	24985456

144 x 352

Debugger Internals

Debugger Internals

Internal Variables | Statistics

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	12961452	24286088	12449777	8882842	3631331	0	24963950

Παρατηρούμε σε όλα αυξήθηκε ο συνολικός αριθμός των κύκλων. Η πρώτη περίπτωση έδωσε καλύτερα αποτελέσματα από τη δεύτερη.

Από την πρώτη περίπτωση την καλύτερη επίδοση είχε το tile 288 x 44, όπου ο ο συνολικός αριθμός των κύκλων αυξήθηκε περίπου 1.2%

Από τη δεύτερη περίπτωση την καλύτερη επίδοση είχε το tile 144 x 352, όπου ο συνολικός αριθμός των κύκλων αυξήθηκε περίπου 5%

Συμπεράσματα

Αρχικά, η πρώτη περίπτωση ,δηλαδή όταν ο πιο εσωτερικός εμφωλευμένος βρόγχος μετράει τις στήλες (προσπέλαση που μοιάζει με Z) δίνει μικρότερο αριθμό συνολικών κύκλων, γιατί όπως ξαναείπαμε τα δεδομένα είναι αποθηκευμένα στη μνήμη κατά γραμμή και όχι κατά στήλη.

Επίσης, η αύξηση του μεγέθους του κώδικα είναι προφανής, γιατί στους βρόγχους που εφαρμόστηκε το loop tiling χρησιμοποιήθηκαν τρίς και όχι δύο εμφωλευμένες for. Στην πρώτη περίπτωση το μέγεθος του κώδικα είναι μεγαλύτερο, επειδή εφαρμόσαμε αυτή την τεχνική σε περισσότερους βρόγχους επανάληψης.

Ακόμη, τα ZI data αυξήθηκαν κατά 4 Bytes, επειδή προστέθηκαν στον κώδικα οι ακέραιες (int) μεταβλητές jj (στην πρώτη περίπτωση) και ii (στη δεύτερη περίπτωση) αντίστοιχα, οι οποίες όπως είδαμε και στην εισαγωγή θα αρχικοποιηθούν με το μηδέν όταν «κατέβουν» στον επεξεργαστή.

5. Loop Collapsing

Θεωρητικό σκέλος

Ορισμένοι εμφωλευμένοι βρόχοι μπορούν να συρρικνωθούν σε έναν ενιαίο βρόχο για να μειωθεί το loop overhead(επιβάρυνση βρόχου) και να βελτιωθεί η απόδοση χρόνου εκτέλεσης.

Το loop collapsing μπορεί να βελτιώσει τις ευκαιρίες για περαιτέρω βελτιστοποιήσεις, όπως το loop unrolling. Δεν είναι συνηθισμένη βελτιστοποίηση στους C compilers, αλλά υποστηρίζεται σε μερικούς C compilers που στοχεύουν την επιστημονική αγορά.

Παρατηρήσεις

Αποτελέσματα armulator:

Image comonent sizes:

```
0   0   13   Errors and warnings for "loop_collapsing.mcp"
==============================================================================
Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

       852         60          0     816156       4656   Object Totals
     10196        314          0        300       4548   Library Totals
==============================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     11048        374          0     816456       9204   Grand Totals
==============================================================================

    Total RO  Size(Code + RO Data)                11422 (  11.15kB)
    Total RW  Size(RW Data + ZI Data)            816456 ( 797.32kB)
    Total ROM Size(Code + RO Data + RW Data)      11422 (  11.15kB)
==============================================================================
```

Παρατηρούμε ότι μειώθηκε ο κώδικας και τα ZI data.

Debugger Internals Statistics:

Debugger Internals

Debugger Internals

Internal Variables | Statistics

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	11620018	22820585	11005822	8662196	3491805	0	23159823

Παρατηρούμε ότι οι κύκλοι μειώθηκαν περίπου 2.6%.

Συμπεράσματα

Αρχικά, μείωση του κώδικα μπορεί να αποδοθεί στην μείωση των instructions, αφού κάποιοι εμφωλευμένοι βρόχοι, οι οποίο χρησιμοποιούσαν δύο δείκτες, άρα κάναν διπλούς ελέγχους για το αν ο δείκτης έχει ξεπεράσει το όριο, αντικαταστάθηκαν από βρόχους με έναν δείκτη.

Επίσης, η μεγάλη μείωση των ZI data μπορεί να αποδοθεί στο γεγονός, ότι ο δισδιάστατος πίνακας current_y αντιπροσωπεύεται από ένα integer δείκτη *c.

Τέλος, ένας προφανής λόγος για την μείωση των συνολικών κύκλων είναι το ότι μειώθηκαν τα instrctions περίπου 5%.

6. Loop Inversion

Θεωρητικό σκέλος

Το loop inversion αποτελεί βελτιστοποίηση και μετασχηματισμός του loop, στον οποίο, ένας βρόχος while αντικαθίσταται από ένα μπλοκ if που περιέχει ένα βρόχο do...while. Όταν χρησιμοποιείται σωστά, μπορεί να βελτιώσει την απόδοση λόγω του instruction pipelining.

Οι σύγχρονες CPU χρησιμοποιούν instruction pipeline. Εκ φύσεως. κάθε άλμα στον κώδικα προκαλεί pipeline stall, πράγμα το οποίο είναι επιζήμιο για την απόδοση.

Επιπρόσθετα, το loop inversion επιτρέπει ασφαλή κίνηση κώδικα με αμετάβλητο βρόχο.

Σημείωση αναφορικά με τον τρόπο εργασίας μας:
Αρχικά υλοποιήσαμε τον κώδικά μας με λούπες while αντί για for.
Στη συνέχεια αντικαταστήσαμε τις while με λούπες do...while εφαρμόζοντας την τεχνική loop inversion.
Δεν βάλαμε το έλεγχο if πριν την έναρξη κάθε βρόγχου do...while, αφού διασφαλίζεται ότι ένας τέτοιος έλεγχος θα επέστρεφε πάντα TRUE, επομένως θα ήταν περιττός άρα και επιζήμιος.

Παρατηρήσεις

Αποτελέσματα armulator:

Image comonent sizes:

Βρόγχοι while

```
0   0   13   Errors and warnings for "loop_inversion.mcp"
================================================================================
Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

       984         60          0    1226792       4864   Object Totals
     10196        314          0        300       4548   Library Totals
================================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     11180        374          0    1227092       9412   Grand Totals
================================================================================

    Total RO  Size(Code + RO Data)                11554 (  11.28kB)
    Total RW  Size(RW Data + ZI Data)           1227092 (1198.33kB)
    Total ROM Size(Code + RO Data + RW Data)      11554 (  11.28kB)
================================================================================
```

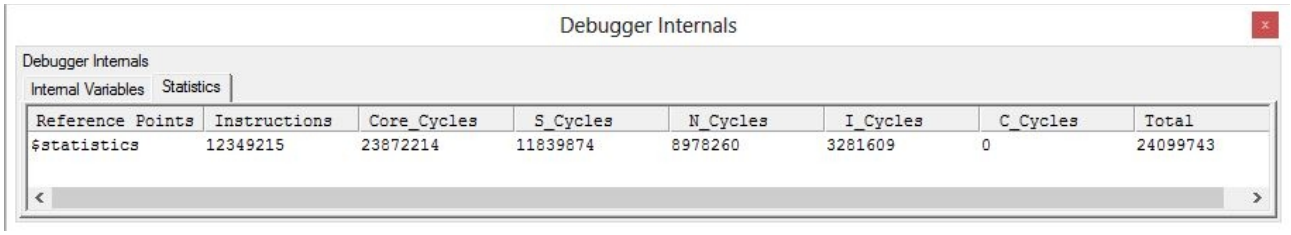
Βρόγχοι do...while

```
0   0   13   Errors and warnings for "loop_inversion.mcp"
================================================================================
Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

       944         60          0    1226792       4836   Object Totals
     10196        314          0        300       4548   Library Totals
================================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     11140        374          0    1227092       9384   Grand Totals
================================================================================

    Total RO  Size(Code + RO Data)                11514 (  11.24kB)
    Total RW  Size(RW Data + ZI Data)           1227092 (1198.33kB)
    Total ROM Size(Code + RO Data + RW Data)      11514 (  11.24kB)
================================================================================
```

Παρατηρούμε ότι και στις δύο περιπτώσεις ο κώδικας αυξήθηκε σε μικρό ποσοστό σε σχέση με τον αρχικό (for loops). Στην δεύτερη περίπτωση (do...while loops) υπήρξε μείωση του μεγέθους του κώδικα σε σύγκριση με την πρώτη περίπτωση (while loops).

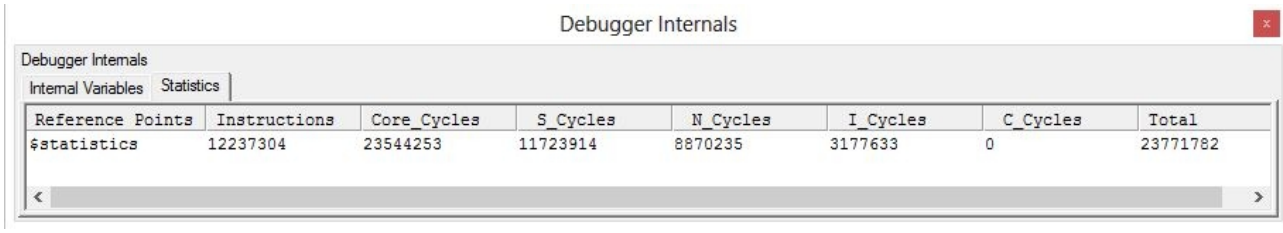
Debugger Internals Statistics:

Βρόγχοι while



Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	12349215	23872214	11839874	8978260	3281609	0	24099743

Βρόγχοι do...while



Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	12237304	23544253	11723914	8870235	3177633	0	23771782

Παρατηρούμε ότι ο συνολικός αριθμός των κύκλων αυξήθηκε κατά 1.3% στην πρώτη περίπτωση σε σχέση με τον αρχικό κώδικα, ενώ στη δεύτερη περίπτωση αυξάνεται κατά 0.00003%.

Το αξιοσημείωτο είναι ότι τα instructions είναι μειμένα στη δεύτερη περίπτωση σε σχέση με το αρχικό

Συμπεράσματα

Η αύξηση του μεγέθους του κώδικα είναι πολύ μικρή, οπότε μπορεί να οφείλεται σε σημαντικό βαθμό σε ορισμένες μικροδιαφορές όσον αφορά την ανάπτυξη του εκάστοτε κώδικα.

Η διαφορά στον αριθμό των συνολικών κύκλων μπορεί να οφείλεται στο γεγονός ότι κάθε λούπα επανάληψης μεταφράζεται με διαφορετικό τρόπο σε assembly. Επίσης το γεγονός ότι τα instructions στη δεύτερη περίπτωση είναι μειωμένα σε σχέση με την πρώτη μπορεί να οφείλεται σε σημαντικό βαθμό στο γεγονός ότι γίνεται ένας λιγότερος έλεγχος. Αφού, όπως αναφέραμε και στο θεωρητικό σκέλος, παραλείψαμε τον πρώτο έλεγχο των δεικτών i, j.

Βιβλιογραφία

Διαφάνειες εργαστηρίου από το μάθημα «Σχεδιασμός Ενσωματωμένων Συστημάτων»

Σημειώσει από το μάθημα «Σχεδιασμός Ενσωματωμένων Συστημάτων»

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://software.intel.com/en-us/articles/how-to-use-loop-blocking-to-optimize-memory-use-on-32-bit-intel-architecture

http://www.nullstone.com/htmls/category/collapse.htm

https://en.wikipedia.org/wiki/Loop_inversion