ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

Ενσωματωμένα Συστήματα

Ενότητα: ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ No 6

Δρ. Μηνάς Δασυγένης

mdasyg@ieee.org

Τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών

Εργαστήριο Ψηφιακών Συστημάτων και Αρχιτεκτονικής Υπολογιστών

http://arch.icte.uowm.gr/mdasyg

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ψηφιακά Μαθήματα του Πανεπιστημίου Δυτικής Μακεδονίας**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Περιεχόμενα

1.Σκοπός της άσκησης....4
2.Παραδοτέα....4
3.Μετασχηματισμός επαναχρησιμοποίησης δεδομένων με εισαγωγή του πίνακα previous_line……4
4.Μετασχηματισμός επαναχρησιμοποίησης δεδομένων με την εισαγωγή του πίνακα RW....5
5.Μετασχηματισμός επαναχρησιμοποίησης δεδομένων με την εισαγωγή των πινάκων previous_line και rw....6
6.Χρήση εργαλείων για βελτιστοποίηση αλγορίθμων....6
6.1Εκμάθηση εργαλείων....6
6.1.1Το πρόγραμμα gprof....9
6.2Εφαρμογή εργαλείων στο phods....10
7.Χρήση ενός προσομοιωτή κρυφής μνήμης για την ανάλυση προγράμματος....11

1. Σκοπός της άσκησης

- Αλγοριθμικοί Μετασχηματισμοί για βελτιστοποίηση κώδικα – **Μέρος Β**.
- Σχεδίαση κρυφής μνήμης.

(Α) 18 ερωτήσεις

(C) 4 ασκήσεις/προγράμματα

+ 1 bonus

2. Παραδοτέα[i]

- **Παραδοτέο C1:** Ο κώδικας και screenshot με το SNR
- **Παραδοτέο C2:** Ο κώδικας και screenshot με το SNR
- **Παραδοτέο C3:** Ο κώδικας και screenshot με το SNR
- **Παραδοτέο C4:** Σχολιασμός 3-4 γραμμών

3. Μετασχηματισμός επαναχρησιμοποίησης δεδομένων με εισαγωγή του πίνακα previous_line

Με τον ίδιο τρόπο γίνεται η εφαρμογή των μετασχηματισμών επαναχρησιμοποίησης δεδομένων με τον πίνακα δεδομένων `previous`. Πρώτη μας επιλογή είναι η εισαγωγή του πίνακα `previous_line`. Η διάσταση του πίνακα αυτού είναι `M×(2*p+B)`, `p=B+S+S/2+S/4`. Η εισαγωγή του πίνακα αυτού είναι όμοια με την αντίστοιχη του `current_line` για τον `current` πίνακα. Ο αλγόριθμος μετασχηματίζεται στον ακόλουθο.

Οδηγίες:

- Να δηλωθεί ο πίνακας previous_line
- Μέσα στο j βρόχο να τοποθετήσετε τον παρακάτω κώδικα αφού το μελετήσετε και κατανοήσετε τη λειτουργία του:

```
if(x==0)

{

   if(i<p) previous_line[i][j]=0;

         else previous_line[i][j]=previous[i-p][j];
         /* Copy from previous*/

}
```

```
else

{

   if(i<2*p) previous_line[i][j]=previous_line[i+B][j];
   /* Reuse from  array*/

         else

   {

         if(x==N/B-1 && i>B+p) previous_line[i][j]=0;

         else previous_line[i][j]=previous[B*x-p+i][j];

   }
```

- Τροποποιήστε τις γραμμές που γράφουμε το p2 ή το q2, ώστε να διαβάζουμε από το previous_line

Παραδοτέο C1: Ο κώδικας και screenshot με το SNR

4. Μετασχηματισμός επαναχρησιμοποίησης δεδομένων με την εισαγωγή του πίνακα RW

Ο μετασχηματισμός αυτός εισάγει τον πίνακα `RW` διαστάσεων `(B+2*p)×(B+2*p)` για την προσωρινή αποθήκευση τμήματος δεδομένων του πίνακας `previous`. Ο αρχικός αλγόριθμος PHODS μετασχηματίζεται στον

Οδηγίες:

- Δηλώστε τον πίνακα RW με διαστάσεις [B+2p][B+2p]

- Μέσα στο y βρόχο, τοποθετήστε τον κώδικα για να αρχικοποιείται το rw, χρησιμοποιώντας την παρακάτω βοήθεια. Τοποθετείστε τον κώδικα που λείπει στα μαύρα πλαίσια:

```
for(k=0;k<B+2*p;k++) /* Copy data from previous to RW */
 for(l=0;l<[redacted])
    {
     if((B*x+k-p)<0 || (B*x+k-p)>(N-1) || (B*y+l-p)<0 || (B*y+l-p)>(M-1))
        rw[k][l]=0;
     else
       {
        if(l>2*p-1 || y==0) rw[k][l]=previous[B*x+k-p][[redacted]];
        else rw[k][l]=rw[k][redacted];
       }
    }
```

- Τροποποιήστε την πρώτη γραμμή που γράφουμε στο p2, ώστε να γίνεται η ανάγνωση από το rw, ως εξής:

```
p2=rw[vectors_x[x][y]+i+k+p][vectors_y[x][y]+l+p];   /*   Read
from RW */
```

- Τροποποιήστε ανάλογα τη δεύτερη γραμμή που γράφουμε στο p2.

Παραδοτέο C2: Ο κώδικας και screenshot με το SNR

5. Μετασχηματισμός επαναχρησιμοποίησης δεδομένων με την εισαγωγή των πινάκων previous_line και rw

Ο τελευταίος μετασχηματισμός επαναχρησιμοποίησης εισάγει και τους δύο πίνακες `previous_line` και `RW` στον κλάδο του πίνακα `previous`.

Οδηγίες:

- Δηλώστε τον πίνακα previous_line
- Δηλώστε τον πίνακα RW
- Μέσα στο βρόχο x αρχικοποιήστε το previous_line
- Μέσα στο βρόχο y αρχικοποιήστε το rw, όπως προηγουμένως, όμως θα πρέπει να διαβάζει από τον πίνακα previous_line και όχι από το previous.
- Στα σημεία που τοποθετείται η τιμή στο p2 να γίνεται πρόσβαση στον πίνακα rw.

Παραδοτέο C3: Ο κώδικας και screenshot με το SNR

6. Χρήση εργαλείων για βελτιστοποίηση αλγορίθμων

6.1 Εκμάθηση εργαλείων

Πολλές φορές υπάρχει η λανθασμένη εντύπωση ότι περισσότερος χρόνος απαιτείται για την ανάπτυξη ενός προγράμματος παρά για οτιδήποτε άλλο. Αυτή η ιδέα είναι λάθος επειδή συνήθως όσο χρόνο απαιτεί η ανάπτυξη ενός προγράμματος τόσο χρόνο απαιτείται για τον έλεγχο, την αποσφαλμάτωση και για τη βελτίωση. Η βελτίωση επιτυγχάνεται με μια πληθώρα εργαλείων. Σε αυτή την ενότητα θα γνωρίσουμε το εργαλείο ανάλυσης κάλυψης *(code coverage analysis)* **gcov**.

Το εργαλείο αυτό μας επιτρέπει να δούμε πόσες φορές εκτελέστηκε κάθε γραμμή, πόσες φορές έγινε αλλαγή ροής *(branch)* και αν υπάρχουν κομμάτια κώδικα που ποτέ δεν εκτελέστηκαν. Τα κριτήρια που σχετίζονται με την ανάλυση **code coverage** είναι τα παρακάτω:

- **function coverage:** πόσες φορές εκτελέστηκε η κάθε συνάρτηση.
- **statement coverage:** πόσες φορές εκτελέστηκε η κάθε εντολή.

- **decision coverage:** στις δομές ελέγχου *(if)* έχουν εκτελεστεί όλες οι περιπτώσεις *(if/elseif/else)*
- **condition coverage:** έχουν εκτελεστεί και οι true και οι false εκφράσεις bool;
- **path coverage:** έχει εκτελεστεί με όλες τις δυνατές πιθανότητες ροής ελέγχου;
- **entry/exit coverage:** έχει γίνει η είσοδος και η έξοδος από όλες τις δυνατές τοποθεσίες;

Το πρόγραμμα **gcov** είναι ένα εργαλείο **coverage analysis** που συνεργάζεται στενά με το εργαλείο gprof. Τα προγράμματα αυτά έρχονται μαζί με το μεταφραστή gcc και βρίσκονται (σχεδόν) σε κάθε σύστημα. Για να χρησιμοποιηθούν τα εργαλεία βελτιστοποίησης θα πρέπει να ισχύουν τα παρακάτω:

- ο κώδικας θα πρέπει να έχει γίνει compile με το gcc
- θα πρέπει να έχουν χρησιμοποιηθεί οι παράμετροι `-fprofile-arcs -ftest-coverage` στο gcc
- δε θα πρέπει να χρησιμοποιείται καμία παράμετρος βελτιστοποίησης (παράμετροι `-O,-O1`...)
- να τοποθετήσετε μια εντολή C ανά γραμμή, επειδή τα στατιστικά αναφέρονται ανά γραμμή.

Θα δούμε τη διαδικασία ανάλυσης παρακάτω.

Κατασκευάστε ένα φάκελο coverage. Εισέλθετε μέσα στο φάκελο coverage. Δημιουργήστε μέσα στο φάκελο το πηγαίο αρχείο test1.c με τις παρακάτω εντολές:

```
#include <stdio.h>

int main (void)

{

int i;

for(i=1;i<13;i++)

{

if(i%2 == 0)

printf("%d is divisible by 2 \n",i);

if(i%5==0)

printf("%d is divisible by 5 \n",i);

if (i%14==0)

printf("%d is divisible by 14 \n",i);

}
```

```
return 0;

}
```

Κάντε compile το ανωτέρω αρχείο με τις παραμέτρους που ενεργοποιούν το **coverage analysis** στο εκτελέσιμο αρχείο **test1** .

Εκτελέστε μια φορά το ανωτέρω αρχείο. Μόλις το εκτελέσετε θα δείτε ότι έχουν δημιουργηθεί κάποια επιπρόσθετα αρχεία που δεν υπήρχαν πριν.

Ποια αρχεία έχουν δημιουργηθεί μετά την εκτέλεση;
_____**(A1)**

(το ένα αρχείο έχει το δένδρο κλήσεων συναρτήσεων και το άλλο τους αριθμούς κλήσεων όλα σε κωδικοποιημένη μορφή).

Εκτελέστε το gcov: Σύνταξη **gcov όνομα_πηγαίου_αρχείου.c**

Δηλαδή, **gcov test1.c**

Μόλις το εκτελέσετε θα δείτε ότι δημιουργείται το αρχείο **test1.c.gcov**. Ανοίξτε το με έναν επεξεργαστή κειμένου. Στην αριστερή στήλη αναφέρεται πόσες φορές έχει εκτελεστεί η γραμμή (αν έχει – δεν είναι γραμμή που εκτελείται, αν έχει #### τότε δεν εκτελέστηκε καθόλου. Μετά ακολουθεί ο αριθμός γραμμής του πηγαίου αρχείου).

Πόσες φορές έχει εκτελεστεί η γραμμή:
printf("%d is divisible by 5 \n",i);_____ **(A2)**

Ποια γραμμή έχει εκτελεστεί τις περισσότερες φορές; _____ **(A3)**

Υπάρχει κάποια γραμμή που εκτελείται, η οποία δεν έχει εκτελεστεί καθόλου;
_____ **(A4)**

Χρησιμοποιήστε την παράμετρο **-b** στο gcov προκειμένου να σας αναφερθούν οι πιθανότητες εκτέλεσης των διαφορετικών βρόχων *(branch probabilities)* και εκτελέστε πάλι το gcov.

Παρατηρήστε τι αναφέρει το gcov.

Πόσο ποσοστό των γραμμών εκτελέστηκαν; _____ **(A5)**

Πόσοι βρόχοι εκτελέστηκαν τουλάχιστον μια φορά *(Taken at least once)*;
_____**(A6)**

Ανοίξτε το αρχείο test1.c.gcov.

Υπάρχει κάποια κλήση συνάρτησης (call) που δεν έχει εκτελεστεί ποτέ; Σε ποια γραμμή είναι και ποια είναι αυτή;_____**(A7)**

Παρατηρώντας τις διακλαδώσεις και τους αριθμούς φορών εκτέλεσης του κάθε branch, και γνωρίζοντας ότι στο pipeline μειώνεται η απόδοση όταν έχουμε αλλαγή ροής εκτέλεσης λόγω διαφορετικού branch, ποιο κομμάτι if θα προκαλέσει τη μεγαλύτερη επιβάρυνση στο pipeline;
_____**(A8)**

6.1.1 Το πρόγραμμα gprof

Μαζί με το gcov συνήθως χρησιμοποιείται το πρόγραμμα gprof το οποίο μας αναφέρει πόσο ποσοστό του χρόνου του επεξεργαστή ξοδεύτηκε σε κάθε συνάρτηση. Για να χρησιμοποιηθεί το gprof απαιτείται να χρησιμοποιηθεί η παράμετρος -pg στο gcc κατά τη δημιουργία του compile. Στη συνέχεια εκτελείται το αρχείο τουλάχιστον μια φορά και δημιουργείται ένα νέο αρχείο στον κατάλογο.

Κάντε compile το αρχείο collatz.c (βρίσκεται στα αρχεία του εργαστηρίου) χρησιμοποιώντας την παράμετρο ενεργοποίησης στατιστικών για το gprof και με την επιλογή να αποθηκευτεί το αρχείο που θα δημιουργηθεί στο collatz.

Εκτελέστε μια φορά το collatz.

Δώστε την εντολή: **gprof -l -u -a -b collatz | more**

(σε περίπτωση που κάποιοι παράμετροι δεν υποστηρίζονται στο σύστημα που βρίσκεστε αγνοήστε τις) (Οι παράμετροι αυτοί είναι έγκυροι για το FreeBSD).

Θα σας εμφανιστεί μια λίστα όπου η πρώτη στήλη θα είναι % (% time) του χρόνου του επεξεργαστή για τη συγκεκριμένη συνάρτηση (το όνομα της συνάρτησης αναφέρεται στο τέλος), η

δεύτερη στήλη (cumulative seconds) το χρόνο που χρειάστηκε για να εκτελεστεί αυτή η συνάρτηση και οι κλήσεις που έγιναν από αυτή, η τρίτη στήλη (self seconds) ο χρόνος που δαπανήθηκε μόνο σε αυτή τη συνάρτηση, η τέταρτη στήλη (calls) πόσες φορές έγινε κλήση η συνάρτηση, ενώ η τελευταία στήλη είναι το όνομα της συνάρτησης.

Ποια συνάρτηση δαπάνησε τον περισσότερο χρόνο (sec) του επεξεργαστή;_____ **(A9)**

Ποια συνάρτηση είχε τις περισσότερες κλήσεις;

Πόσες ήταν αυτές; _____ **(A10)**

Η παραπάνω έξοδος ονομάζεται flat-profile.

Αν αντί για την παράμετρο **-l** έχουμε **-L** τότε θα έχουμε ένα **call-graph profile**.

Εκτελέστε το και δείτε την ιεραρχία κλήσεων, δηλαδή πως η μια συνάρτηση έκανε κλήση την επόμενη κ.ο.κ.

Στο διάγραμμα call-graph πόσο % του χρόνου του επεξεργαστή η _start κάλεσε την main η οποία κάλεσε την nseq και η οποία κάλεσε την printf; _____**(A11)**

Ποια συνάρτηση εκτελέστηκε το περισσότερο;_____ **(A12)**

Αφού ολοκληρωθεί το εργαστήριο απαντήστε: Ποια είναι η διαφορά των gprof/gcov; _____ **(A13)**

6.2 Εφαρμογή εργαλείων στο phods

Μεταφέρετε τα αρχικά (χωρίς τροποποίηση) αρχεία του phods στο διακομιστή του Τμήματος, σύμφωνα με τις οδηγίες που σας έχουν δοθεί *(Για παράδειγμα **(α)** με winSCP ή **(β)** με αποθήκευση στο NAS του Τμήματος και μετά αντιγραφή των αρχείων από τον κατάλογο του διακομιστή /mnt/public).*

Κάντε **compile** το **phods** με τις κατάλληλες παραμέτρους για να χρησιμοποιηθούν το **gcov** και **gprof**.

Εκτελέστε μια φορά το πρόγραμμα phods για να δημιουργηθούν τα αρχεία εξόδου, και επιβεβαιώστε με το **ls -l** ότι έχουν δημιουργηθεί.

Χρησιμοποιώντας το **gprof**, απαντήστε:

Ποια συνάρτηση και σε ποιο ποσοστό είχε τον περισσότερο % χρόνο χρήσης του CPU_____ ; **(A14)**

Ποια συνάρτηση είχε τον περισσότερο αριθμό κλήσεων _____ ; **(A15)**

Χρησιμοποιώντας το **gcov**, απαντήστε:

Στη γραμμή 93 του πηγαίου κώδικα, υπάρχει η εντολή:

```
93: p1 = current[B*x+k][B*y+l];
```

πόσες φορές εκτελέστηκε αυτή η εντολή, δηλαδή πόσες φορές έγινε πρόσβαση ανάγνωσης στον πίνακα **current [ ][ ]** ; _____ **(A16)**

Στη γραμμή 102 του πηγαίου κώδικα, υπάρχει η εντολή:

```
102: p2 = previous[B*x+vectors_x[x][y]+i+k][B*y+vectors_y[x][y]
+l];
```

πόσες φορές εκτελέστηκε αυτή η εντολή, δηλαδή πόσες φορές έγινε πρόσβαση ανάγνωσης στον πίνακα **previous [ ][ ]** ; _____ **(A17)**

Χρησιμοποιήστε τον τελευταίο κώδικα με τους μετασχηματισμούς *(που έχετε προσθέσει RW και prev_line)* και συγκρίνετε τις τιμές που εκτυπώνονται από τους μετρητές που έχετε τοποθετήσει και τις τιμές που αναφέρει το gcov, και σημειώστε τα συμπεράσματά σας.

Παραδοτέο C4: Σχολιασμός 3-4 γραμμών

7. Χρήση ενός προσομοιωτή κρυφής μνήμης για την ανάλυση προγράμματος

Ένα σημαντικό υποσύστημα του ενσωματωμένου συστήματος είναι η κρυφή μνήμη. Η επιλογή της κρυφής μνήμης επηρεάζει τόσο την κατανάλωση ισχύος, όσο και την απόδοση του συστήματος. Μια πολύ μικρή κρυφή μνήμη για την εφαρμογή μας, μπορεί να μειώσει την απόδοση, ενώ μια πολύ μεγάλη κρυφή μνήμη θα αυξήσει σημαντικά την κατανάλωση ενέργειας. Προκειμένου να επιλέξουμε τη βέλτιστη κρυφή μνήμη για το σύστημά μας, εξάγουμε ένα ίχνος εκτέλεσης της εφαρμογής *(ή των τυπικών εφαρμογών που θα εκτελέσουμε)* και εξάγουμε πειραματικές μετρήσεις με έναν προσομοιωτή κρυφής μνήμης.

Σε αυτή την άσκηση θα ασχοληθείτε με τον προσομοιωτή κρυφής μνήμης **dineroIV** προκειμένου να αναλύσουμε τα ίχνη εκτέλεσης που έχουν ήδη δημιουργηθεί για 3 τυπικές εφαρμογές. Τα ίχνη εκτέλεσης για ένα εκατομμύριο εντολές βρίσκονται στο **zafora** στη διαδρομή **/mnt/public** και έχουν ονόματα **cc1.din, spice.din, tex.din** *(αν δε βρίσκονται εκεί μπορείτε να τοποθετήσετε εσείς τα αρχεία στο NAS)* Το dineroIV μπορεί να προσομοιώσει ένα πολύ μεγάλο πλήθος κρυφών μνημών δεδομένων και εντολών. Διαβάζει κάθε γραμμή του trace file, που αναφέρει αν είναι instruction fetch, data read *(load)*, data write *(store)*. Στο τέλος, αναφέρονται τα στατιστικά της κρυφής μνήμης που έχουμε επιλέξει.

Η χρήση του προγράμματος dineroIV είναι η εξής:

```
dineroIV [cache options] < tracefile
```

Μπορείτε να δείτε τις παραμέτρους του προγράμματος με την εντολή
```
dineroIV-help
```

Συνδεθείτε στο server **zafora.icte.uowm.gr** με το όνομα χρήστη που έχετε ***(ictest00XXX)*** χρησιμοποιώντας το πρωτόκολλο SSH (putty)

(Πληροφορίες: http://zafora.icte.uowm.gr).

Να προσομοιώσετε για instruction & data cache για μεγέθη: 1K, 4k, 16K με πιθανά μεγέθη block 8, 16 και 64 *(συνολικά 9 συνδυασμοί για κάθε trace file)*. Μπορείτε να δημιουργήσετε τα κατάλληλα **/bin/sh** script, ώστε να αυτοματοποιηθούν οι μετρήσεις, όπως:

```
for trace in cc1.din spice.din tex.din
    for cache in 1 4 16 ; do
        for blocksize in 8 16 64 ; do
εκτέλεση dinero με κατάλληλες παραμέτρους $trace $cache $blocksize>>outputfile
        done
    done
done
```

Από τα αποτελέσματα να βρείτε για κάθε εφαρμογή ποια έχει τα λιγότερα misses και το λιγότερο memory traffic._____**(A18)**

BONUS +0.5 Διαβάζοντας στο Internet να βρείτε τον τρόπο να εξάγετε το trace file από το PHODS και να επαναλάβετε την ανωτέρω διαδικασία. Θα πρέπει να καταγράψετε τα βήματα που ακολουθήσατε *(με αντίστοιχα screenshots)* για το πως εξάγατε το trace file

TIP: **(a)** τρόπος, υπάρχει κατάλληλη επιλογή στο ARMulator για εξαγωγή trace file **(b)** χρήση *(runtime)* instrumentation, όπως Intel Atom Tool.

i **Παραδοτέα αυτού του εργαστηρίου:**

- Κώδικες για κάθε βήμα βελτιστοποίησης

- Προσομοίωση κάθε βήματος στο ARMulator (ARM7TDMI) και εξαγωγή μετρήσεων ως προς εντολές που εκτελέστηκαν, κύκλοι εκτέλεσης, αριθμό προσβάσεων σε πίνακες και τιμή SNR. Αν μπορείτε, να μετρήσετε μόνο τις εντολές και τους κύκλους από τη συνάρτηση phods_motion_estimation(), αφού αυτή είναι η βασική συνάρτηση της εφαρμογής μας.

- Για να μετρήσουμε το συνολικό αριθμό των προσβάσεων σε κάθε πίνακα τοποθετήστε μετρητές προσβάσεων σε κάθε πίνακα. Για παράδειγμα, θα χρησιμοποιήσουμε τη μεταβλητή counter_current_read counter_current_write για να μετρήσουμε τις προσβάσεις στον πίνακα counter_current. Κάθε φορά που γίνεται εγγραφή στον πίνακα counter_currrent

  (δηλαδή βρίσκεται στην αριστερή πλευρά της εντολής C), τότε θα πρέπει εμείς από κάτω να γράψουμε **counter_current_write++;** Κάθε φορά που γίνεται ανάγνωση στον πίνακα counter (δηλαδή, βρίσκεται στη δεξιά πλευρά της εντολής C), τότε θα πρέπει εμείς από κάτω να γράφουμε **counter_current_read++;** Στο τέλος, έξω από τη συνάρτηση phods_motion_estimation(), θα πρέπει να εκτυπώνονται όλες αυτές οι τιμές, όπως παρακάτω

  “Number of read accesses on current: XXXXX”

  “Number of write accesses on current: XXXXX”

  για κάθε πίνακα.

➔ Πρέπει να σημειωθεί ότι όλοι οι μετασχηματισμοί του εργαστηρίου ανήκουν στην κατηγορία των αλγοριθμικών μετασχηματισμών ανεξαρτήτου αρχιτεκτονικής. Δηλαδή, οι μετασχηματισμοί αυτοί βελτιστοποιούν τον αλγόριθμο, είτε εκτελεστεί σε επεξεργαστή ARM, είτε σε επεξεργαστή Intel. Για αυτό το λόγο μπορούμε να χρησιμοποιήσουμε εργαλεία σε άλλα λειτουργικά συστήματα.