SOBEL OPTIMISATIONS

ΟΝΟΜΑΤΕΠΩΝΥΜΟ:	ΓΕΩΡΓΑΚΙΔΗΣ ΧΡΗΣΤΟΣ
ΑΕΜ:	1964
ΚΑΘΗΓΗΤΗΣ:	ΧΡΗΣΤΟΣ ΑΝΤΩΝΟΠΟΥΛΟΣ
ΜΑΘΗΜΑ:	ΣΥΣΤΗΜΑΤΑ ΥΠΟΛΟΓΙΣΜΟΥ ΥΨΗΛΩΝ ΕΠΙΔΟΣΕΩΝ
ΕΤΟΣ:	2017-2018

SOBEL OPTIMISATIONS

Τροπος Υπολογισμού Τιμών: Έγιναν 12 επαναλήψεις εκτελέσεων του προγράμματος μετά από κάθε βελτιστοποίηση και καταγράφηκε ο χρόνος εκτέλεσης και το PSNR. Έπειτα αφαιρούμε την καλύτερη και την χειρότερη μέτρηση και βγάλαμε τη μέση τιμή και τυπική απόκλιση.

Η ανίχνευση ακμών είναι μια πολύ συνηθισμένη διαδικασία στην επεξεργασία εικόνας. Στα πλαίσια αυτής της εργασίας σας δίνεται έτοιμος κώδικας ο οποίος ανιχνεύει ακμές σε εικόνα σε τόνους του γκρι (grayscale) με χρήση του φίλτρου Sobel. Ο κώδικας διαβάζει μια εικόνα από αρχείο εισόδου, την επεξεργάζεται και παράγει την εικόνα εξόδου την οποία επίσης αποθηκεύει σε αρχειο. Τέλος, συγκρίνει το αποτέλεσμα που παρήγαγε σε σχέση με αυτό που παράγεται από μια σωστή υλοποίηση (επίσης σας δίνεται ως υπόδειγμα) και υπολογίζει το PSNR.

Εσείς θα πρέπει να υλοποιήσετε διαδοχικά μια σειρά πολύ συνηθισμένων βελτιστοποιήσεων στον κώδικα που σας δίνεται. Για κάθε βελτιστοποίηση που εφαρμόζετε θα πρέπει να μετράτε το χρόνο εκτέλεσης και να τον συγκρίνετε με αυτόν της αμέσως προηγούμενης έκδοσης. Αυτή η διαδικασία θα πρέπει να επαναληφθεί τόσο με απενεργοποιημένες όσο και με ενεργοποιημένες τις βελτιστοποιήσεις του μεταγλωττιστή. Εξυπακούεται ότι μετά από κάθε βήμα θα πρέπει να επαληθεύετε την ορθότητα της υλοποίησής σας.

ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΥΠΟΛΟΓΙΣΤΗ & ΛΕΙΤΟΥΡΓΙΚΟΥ ΣΥΣΤΗΜΑΤΟΣ

Όνομα μοντέλου:	MacBook Pro Retina 13” 2015
Όνομα επεξεργαστή:	Intel Core i5-5257U
Ταχύτητα επεξεργαστή:	2,7 GHz
Αριθμός επεξεργαστών:	1
Συνολικός αριθμός πυρήνων:	2 (με hyperthreading 4)
L2 Cache (ανά πυρήνα):	256 KB
L3 Cache:	3 MB
Μνήμη:	8 GB 1867 MHz DDR3
Γραφικά:	Intel Iris Graphics 6100 1536 MB
Λειτουργικό Σύστημα:	macOS High Sierra
Έκδοση λειτουργικού συστήματος:	10.13
Compiler:	icc (ICC) 18.0.0 20170811

(Δεν υπήρχε δυνατότητα εγκατάστασης λειτουργικού συστήματος Linux. Ωστόσο εγκατέστησα τον icc και δούλεψα κανονικά χωρίς πρόβλημα)

Επιπλέον χαρακτηριστικά:

Pagesize:	4096
bus frequency:	100000000
cache line size:	64
L1i cache size:	32768
L1d cache size:	32768
L2 cache size:	262144
L3 cache size:	3145728

-O0 OPTIMISATIONS

Οι βελτιστοποιήσεις έγιναν με την ακόλουθη σειρά:

- LOOP INTERCHANGE

  Στην convolution2D εξαλείφεται τελείως βγάζοντας τους υπολογισμούς εκτός του loop λόγω των λίγων επαναλήψεων. Αυτό έχει ως αποτέλεσμα την τεράστια μείωση στον χρόνο εκτέλεσης καθώς εκτελούνται όλες οι πραξεις η μία μετά την άλλη, ενώ στο for loop θα χρειαζόταν και οι πράξεις για το index. Επίσης μια ιδέα για τον λόγο που τα for loops καταναλώνουν πολύ χρόνο είναι λόγω του πιθανού false prediction για το αν ή όχι θα γίνει επανάληψη ή θα συνεχιστεί η εκτέλεση του προγράμματος με την εντολή που ακολουθεί το for loop.

- LOOP UNROLLING

  Στην convolution2D εξαλείφεται τελείως βγάζοντας τους υπολογισμούς εκτός του loop λόγω των λίγων επαναλήψεων.

- FUNCTION INLINING

  Η συνάρτηση convolution2D αντικαταστείται μέσα στη συνάρτηση sobel. Εκτελείται το περιεχόμενο της convolution2D δύο φορές και το αποτέλεσμα αποθηκεύεται σε δύο τοπικές μεταβλητές temp1 και temp2. , Στη συνέχεια, αντικαθιστούμε την κλήση της convolution2D με τις μεταβλητές temp1 και temp2 αντίστοιχα.

- LOOP INVARIANT CODE MOTION

  Ορίζω τις τομικές μεταβλητές temp3, temp4 και temp5. Για τις temp4 & temp5 αναθέτω τιμές temp4 = SIZE + 1 και temp5 = SIZE - 1. Αυτές οι δύο μεταβλητές θα μπορούσαν να ενταχθούν και στην κατηγορία Common Subexpression Elimination, όμως θεώρησα ότι καλύτερα ταιριάζουν σε αυτήν την κατηγορία. Για την temp3 στην αρχή του κάθε loop με index i της αναθέτω τιμή temp3 = i*SIZE, το οποίο είναι ανεξάρτητο του j. Κατ' αυτόν τον τρόπο η temp3 παίρνει τιμή μία φορά σε επαύξηση του i. Κοινώς η πράξη i*SIZE θα γίνει SIZE - 1 φορές, ενώ πριν γινόταν (SIZE-1)^2. (σημαντική διαφορά). Επίσης, τώρα οι πράξεις SIZE+1 & SIZE-1 εκτελούνται μία φορά κατά τη δήλωση των μεταβλητών temp4 και temp5, ενώ πριν η κάθε μία εκτελούνταν 2*(SIZE-1)^2 (επίσης πολύ σημαντική διαφορά). Παρ' όλα αυτά παρατηρώ ελάχιστη διαφορά στο χρόνο εκτέλεσης. Πιθανόν αυτό να οφείλεται στο χαμηλό βάρος αυτών των πράξεων σε σχέση με άλλα κομμάτια κώδικα πιο βαριά, π.χ. pow, sqrt, /.

- COMMON SUBEXPRESSION ELIMINATION

Σε αυτό το κομμάτι για τις temp1 & temp2 αντί να υπολογίζουμε ένα γινόμενο τη φορά και να το προσθέτουμε στην προηγούμενη τιμή τους αναθέτουμε στις temp1 & temp2 κατευθείαν όλο το άθροισμα, περιορίζοντας έτσι τις χρονικές απαιτήσεις για την αποθήκευση του αποτελέσματος της κάθε πράξης στις temp1 και temp2.

• STRENGTH REDUCTION

1. Αντικατάσταση των horiz_operator & vert_operator με τα αντίστοιχα νούμερά τους. Έτσι επιτυγχάνεται καλύτερος χρόνος καθώς γίνεται κατευθείαν πολλαπλασιασμός με την καρφωτή τιμή, ενώ πριν έπρεπε ο επεξεργαστής να πάει να βρει το νούμερο στην αντίστοιχη θέση του πίνακα και μετά να γίνει η πράξη. Επίσης, οι πίνακες αυτοί έχουν μία στήλη (horiz_operator) και μία στήλη (vert_operator) μηδενική. Έτσι, αντικαθιστώντας τις τιμές των αντίστοιχων θέσεων των πινάκων κάποιοι όροι του αθροίσματος εξαλίφονται. Επίσης, οι πολλαπλασιασμοί με (+1) και (-1) αλλάζουν απλά το πρόσημο με αποτέλεσμα να μειώνονται κι αυτοί οι υπολογισμοί.

2. Μετατροπή των πολλαπλασιασμών σε αθροίσματα που είναι πιο φθηνή πράξη ή αν ο πολλαπλασιασμός είναι με δύναμη του δύο είναι πιο αποδοτικό να αντικατασταθεί με << που είναι πιο απλή και γρήγορη διαδικασία για τον υπολογιστή.

3. Αντικατάσταση του pow() με πολλαπλασιασμό είναι πιο αποδοτικό καθώς αποφεύγουμε την κλήση συνάρτησης βιβλιοθήκης είναι επίπονη διαδικασία.

4. Το for loop for (i = 1; i < temp5; i++) μπορούμε να αλλάξουμε τα όρια και το βήμα διατηρώντας όμως τον ίδιο αριθμό επαναλήψεων. Δηλαδή, μιας και μέσα στο σώμα του for το i δεν εμφανίζεται ποτέ μόνο του αλλά πάντα ως γινόμενο με το SIZE μπορούμε να κάνουμε το for loop: for (i = SIZE; i < temp6; i += SIZE), όπου temp6 = temp5 * SIZE.

5. Η κλήση συνάρτησης βιβλιόθηκης sqrt() αντικαθίσταται από την έυρεσης της τιμής της σε ένα lookup table. Αυτό είναι πιο γρήγορο σε σχέση με την κλήση της sqrt().

6. Σπάσιμο του αθροίσματος στις temp1 & temp2 se μικρότερα μιας και υπάρχουν κοινά τμήματα (sum1 & sum2) . Αυτό οδηγεί σε καλύτερη τοπικότητα, δηλαδή λιγότερα cache misses, καθώς και λιγότερα access σε πίνακα.

7. Ο υπολογισμός του PSNR = 10*log10(65536/PSNR) μπορεί να γίνει λίγο πιο αποδοτικός γραμμένος ως PSNR ≃ 48 - 10*log10(PSNR), το οποίο όμως παρά το ≃ οδηγεί σε PSNR inf καθώς όσο το PSNR τείνει στο inf το σφάλμα

υπολογισμού αυτού (κλίμακα δεκαδικών ψηφίων) γίνεται μηδαμινό. Επίσης μπορούμε να παραλείψουμε την προηγούμενη πράξη και να την ενσωματώσουμε στο log το οποίο έπειτα να το αναθέσουμε σε μία μεταβλητή t. Δηλαδή: $t = -\log10(PSNR) = \log10(PSNR)^{-1} = \log10(PSNR/POWSIZE)^{-1} = \log10(POWSIZE/PSNR)$. Έτσι $PSNR \simeq 48 + 10*\log10(POWSIZE/PSNR) = 48 +$

$10*t = 48 + t + t + t + t + t + t + t + t + t + t$, λόγω του ότι η πρόσθεση είναι φθηνότερη του πολλαπλασιασμού.

8. Στο δεύτερο διπλό for loop που υπολογίζει το PSNR μπορούμε να υπολογίζουμε πρώτα την διαφορά output - golden, να την αναθέτουμε σε μία μεταβλητή temp3 και έπειτα να υπολογίζουμε το temp3*temp3 και να το προσθέτουμε στο PSNR. Αυτό έχει ως αποτέλεσμα λίγο λιγότερους υπολογισμούς καθώς και accesses στους πίνακες output και golden.

9. Κάτι ακόμη που μπορούμε να κάνουμε είναι να περιορίσουμε τον χρόνο που χρειάζεται για την sqrt(). Αρχικά θα μπορούσε η sqrt να μπει μέσα στην else έτσι ώστε να υπολογίζεται η ρίζα μόνο όταν χρειάζεται και ο έλεγχος της συνθήκης να γίνει p > 65025. Έτσι πετυχαίνουμε καλύτερο χρόνο γιατί εκτελείται λιγότερες φορές. Επιπλέον βάζουμε σε έναν lookup table όλα τα αποτελέσματα της sqrt για κάθε αριθμό μέχρι το 65024. Έτσι από την κλίση της συνάρτησης βιβλιοθήκης κάνουμε access σε πίνακα που είναι πιο γρήγορο.

• ΥΠΟΒΟΗΘΙΣΗ ΜΕΤΑΓΛΩΤΤΙΣΤΗ

1. Οι μεταβλητές i, j & p μπορούν να δηλωθούν ως register unsigned καθώς δεν θα πάρουν ποτέ αρνητική τιμή και μιας και οι μεταβλητές αυτές χρησιμοποιούνται συχνά είναι καλή τακτική να μπουν σε registers ώστε να είναι εύκολα και γρήγορα προσβάσιμες.

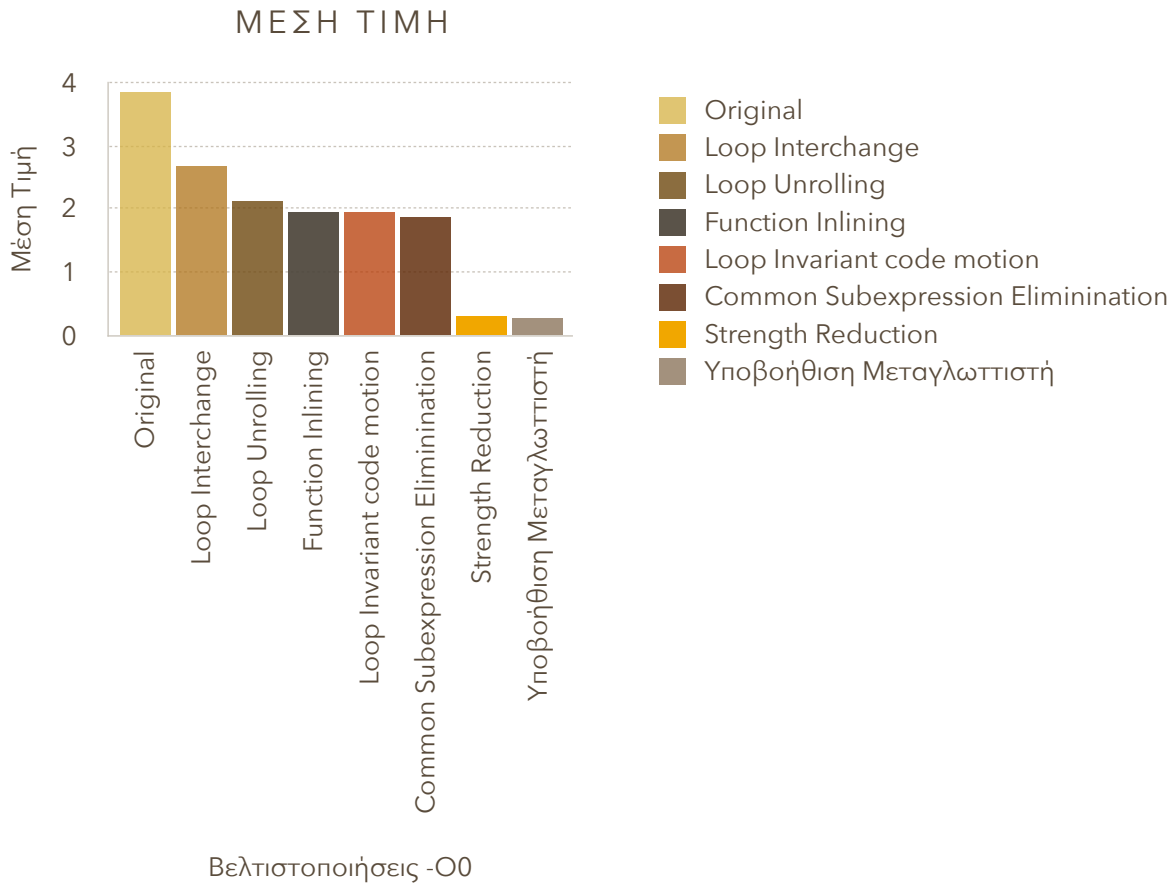
2. Επίσης, οι μεταβλητές PSNR, t, temp1, temp2, sum1, sum2, temp3 δηλώνονται επίσης ως registers για τον ίδιο ακριβώς λόγο.

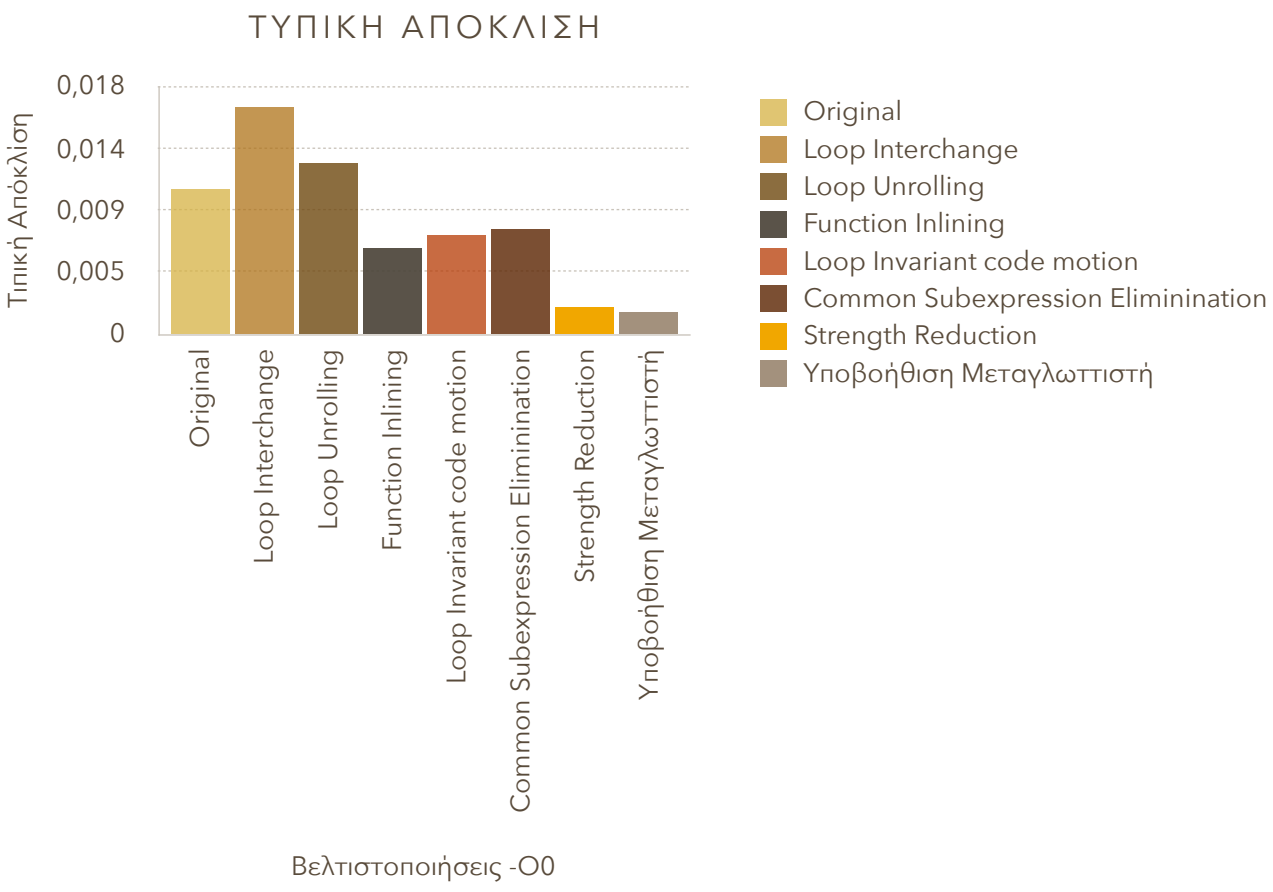
3. Τέλος, οι μεταβλητές temp4, temp5 και loops μιας και δεν αλλάζουν τιμή κατά την εκτέλεση του προγράμματος και είναι πάντα θετικές, δηλώνονται ως const unsigned.

• LOOP FUSION

1. Το μόνο που θα μπορούσε να γίνει είναι τα δύο διπλά for loops να συγχωνευθούν σε 1. Αυτό όμως οδηγεί σε μεγαλύτερο χρόνο καθώς μεγαλώνει το instruction block με αποτέλεσμα να έχουμε misses.

2. Ίσως να μπορούσε να υλοποιηθεί αν μπορούσα να αυξήσω το stack size, όμως λόγω του ότι δούλευα σε mac δεν μου επιτρεπτοταν.


ΜΕΣΗ ΤΙΜΗ
Μέση Τιμή
4
3
2
1
0
Original
Loop Interchange
Loop Unrolling
Function Inlining
Loop Invariant code motion
Common Subexpression Elimininination
Strength Reduction
Υποβοήθιση Μεταγλωττιστή
Βελτιστοποιήσεις -O0


ΤΥΠΙΚΗ ΑΠΟΚΛΙΣΗ
Τιπική Απόκλίση
0,018
0,014
0,009
0,005
0
Original
Loop Interchange
Loop Unrolling
Function Inlining
Loop Invariant code motion
Common Subexpression Elimininination
Strength Reduction
Υποβοήθιση Μεταγλωττιστή
Βελτιστοποιήσεις -O0

-FAST OPTIMISATIONS

Οι βελτιστοποιήσεις έγιναν με την ακόλουθη σειρά:

•LOOP UNROLLING
•LOOP INTERCHANGE

Στην βελτιστοποίηση με -fast το loop interchange αύξανε τον χρόνο ολοκλήρωσης. Για την ακρίβεια αν έκανα πρώτα loop interchange και μετά loop unrolling τότε θα αύξανε τον χρόνο το loop unrolling και αντίστροφα. Έτσι κρίθηκε πιο αποδοτικό να επιλεγεί ως πρώτη βελτιστοποίηση η loop unrolling καθώς έτσι θα δινόταν περισσότερες δυνατότητες βελτιστοποιήσεων σε επόμενα στάδια, λόγου χάρη στο στάδιο του strength reduction.

•FUNCTION INLINING
•LOOP INVARIANT CODE MOTION

Σε αυτό το κομμάτι δεν όρισα τη μεταβλητή temp3 = i*SIZE καθώς αύξανε το χρόνο λόγω του ότι εκτελούνταν πολύ περισσότερες φορές σε σχέση με την -O0.

•COMMON SUBEXPRESSION ELIMINATION & STRENGTH REDUCTION

Σε αυτό το κομμάτι έγιναν ακριβώς ότι βελτιστοποιήσεις έγιναν και στο -O0.

•ΥΠΟΒΟΗΘΗΣΗ ΜΕΤΑΓΛΩΤΤΙΣΤΗ
•LOOP FUSION

Όπως και στην -O0 ούτε και εδώ βοήθησε στη μείωση του χρόνου.

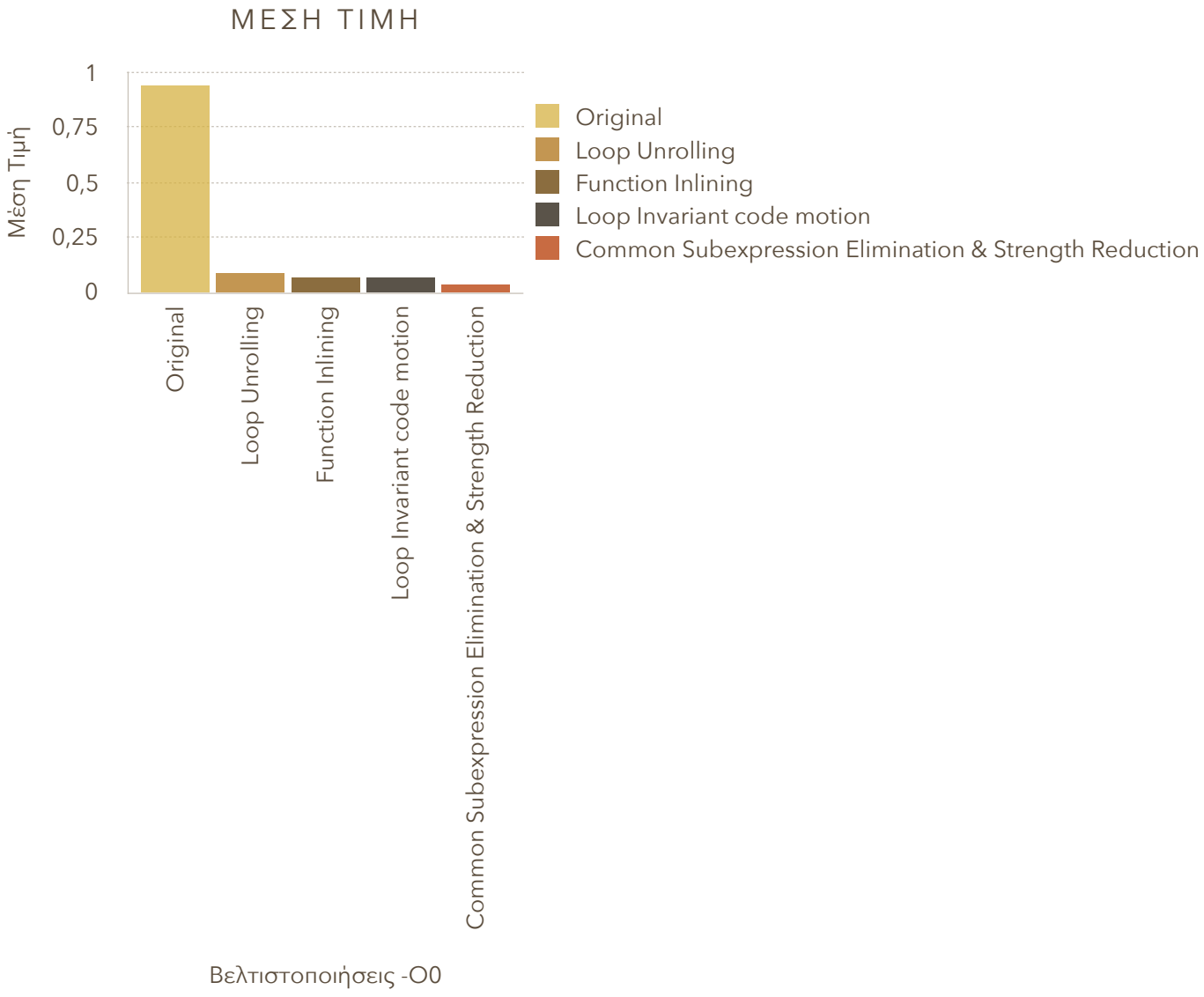
Το -fast ενεργοποιεί τα -xHOST -O3 -ipo -no-prec-div -mdynamic-no-pic -fp-model fast=2

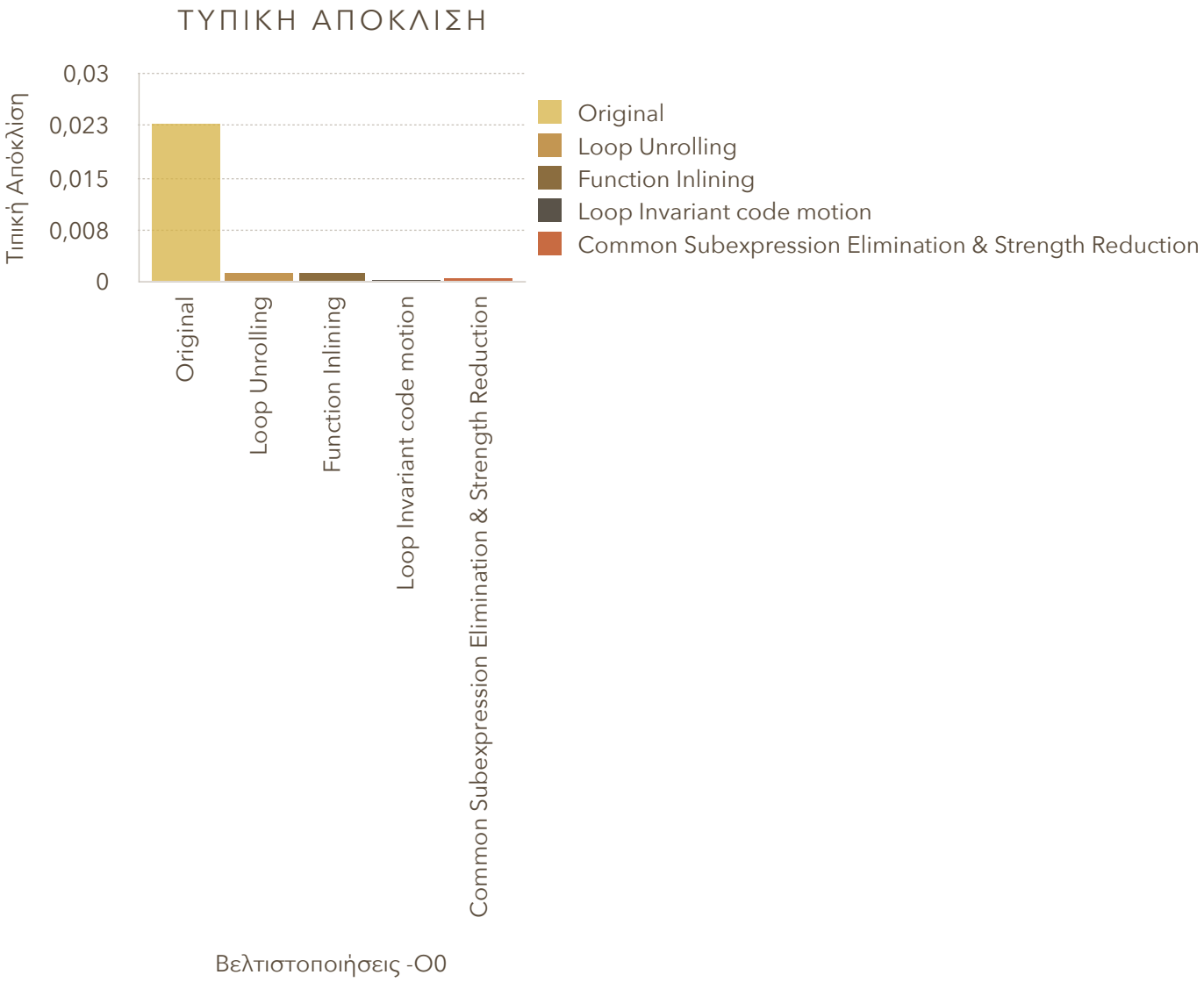
όπου το καθένα από αυτά έχει την ακόλουθη επίδραση:

-xHOST	generate instructions for the highest instruction set and processor available on the compilation host machine
-O3	optimise for maximum speed and enable more aggressive optimisations that may not improve performance on some programs
-ipo	enable multi-file IP optimisation between files.
-no-prec-div	improve precision of FP divides (some speed impact

-mdynamic-no-pic	disables the generation of position independent code. Use when compiling code that will be linked into an executable and not shared library
-fp-model <name>	enable <name> floating point model variation

Στοιχεία παρμένα από το help του icc


ΜΕΣΗ ΤΙΜΗ
Μέση Τιμή
1
0,75
0,5
0,25
0
Original
Loop Unrolling
Function Inlining
Loop Invariant code motion
Common Subexpression Elimination & Strength Reduction
Βελτιστοποιήσεις -O0


ΤΥΠΙΚΗ ΑΠΟΚΛΙΣΗ
Τιπική Απόκλιση
0,03
0,023
0,015
0,008
0
Original
Loop Unrolling
Function Inlining
Loop Invariant code motion
Common Subexpression Elimination & Strength Reduction
Βελτιστοποιήσεις -O0