Ατομική Διπλωματική Εργασία

ΑΝΑΛΥΣΗ ΤΩΝ EEMBC DIGITAL ENTERTAINMENT BENCHMARKS

Ηλιάνα Διονυσίου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2009

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ανάλυση των EEMBC Digital Entertainment Benchmarks.

Ηλιάνα Διονυσίου

Επιβλέπων Καθηγητής

Γιάννος Σαζείδης

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2009

Ευχαριστίες

Σε αυτό το σημείο θα ήθελα να ευχαριστήσω τον κύριο Γιάννο Σαζεΐδη, που ήταν ο επιβλέπων καθηγητής της ατομικής διπλωματική εργασίας καθώς επίσης και τον διδακτορικό φοιτητή κύριο Μάριο Κλεάνθους. Η στήριξη, η βοήθεια και η καθοδήγηση που μου προσέφεραν όταν το χρειαζόμουν - από την αρχή της προσπάθειας μου, μέχρι και το τέλος- ήταν πολύ σημαντική και έπαιξε καθοριστικό ρόλο στην επίτευξη των στόχων μου.

Περίληψη

Η ατομική διπλωματική μου εργασία είχε ως θέμα την ανάλυση των EEMBC Digital Entertainment Benchmarks. Τα DENbench Version 1.0 είναι μία κατηγορία των EEMBC benchmarks που επιτρέπουν στους χρήστες να δοκιμάσουν την απόδοση των υποσυστημάτων που βρίσκονται στα πολυμέσα εικόνας , video, ήχου. Σκοπός της διπλωματικής μου ήταν να αναλύσω τον κώδικα των Benchmarks ώστε να βρω το hotspot τους, το κομμάτι δηλαδή του κώδικα όπου καταναλώνεται ο περισσότερος χρόνος εκτέλεσης του προγράμματος.

Κατά την πρώτη φάση της διπλωματικής μου μελέτησα επιστημονικά άρθρα, κεφάλαια από βιβλία Αρχιτεκτονικής Υπολογιστών και δημοσιεύσεις. Επίσης διάβασα περιγραφές και κείμενα για τα benchmarks με τα οποία δούλεψα ώστε να μπορώ κατά τη δεύτερη φάση να τα αναλύσω.

Στη δεύτερη φάση της διπλωματικής μου ανέλυσα κάποια από τα EEMBC Digital Entertainment Benchmarks. Η ανάλυση μου χωρίστηκε σε 3 επίπεδα:

- Μελέτη, μεταγλώττιση και ανάλυση του κώδικα των benchmarks με τη βοήθεια του εργαλείου VTune Performance Analyzer, ώστε να βρω το σημείο του κώδικα που είναι το hotspot.

- Ανάλυση του κώδικα με την είσοδο διαφορετικών datasets και δημιουργία γραφικών για να συγκρίνω και να δω τη συμπεριφορά του κάθε benchmark με διαφορετική είσοδο.

- Μικροαρχιτεκτονική ανάλυση του benchmark στο VTune Performance Analyzer, αν δεν δικαιολογείται ο χρόνος εκτέλεσης που πήρα με βάση το πλήθος και το είδος των εντολών που εκτελούνται στο hotspot, ώστε να βρω τον πραγματικό λόγο για τον οποίο το κομμάτι του κώδικα που βρήκα καταναλώνει τον περισσότερο χρόνο εκτέλεσης.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Θέμα Ατομικής Διπλωματικής Εργασίας

Κατά τη διάρκεια της φοίτησης μου στο Τμήμα Πληροφορικής του Πανεπιστημίου Κύπρου χρειάστηκε να εκπονήσω την ακόλουθη Ατομική Διπλωματική Εργασία για την μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου μου.

Θέμα της διπλωματικής μου είναι η ανάλυση του κώδικα των EEMBC Digital Entertainment Benchmarks.

Τα EEMBC Digital Entertainment benchmarks είναι προγράμματα τα οποία ανέπτυξε ο μη κερδοσκοπικός οργανισμός EEMBC ώστε με την χρήση τους να δοκιμάζουν την επίδοση σε ενσωματωμένα συστήματα πολυμέσων.

1.2 Σκοπός Ατομικής Διπλωματικής Εργασίας

Σκοπός της διπλωματικής μου εργασίας είναι η ανάλυση του κώδικα των EEMBC Digital Entertainment Benchmarks σε ένα profiler (VTune Performance Analyzer) ώστε να βρω το hotspot, δηλαδή το κομμάτι του κώδικα όπου ο επεξεργαστής καταναλώνει τον περισσότερο χρόνο εκτέλεσης. Έπειτα να αναλύσω αυτό το κομμάτι κώδικα ώστε να βρω τον λόγο που το καθιστά το hotspot του προγράμματος.

Η ανάλυση του benchmark περιλαμβάνει και ανάλυση με διαφορετικά αρχεία ως είσοδο ώστε να παρατηρήσω την συμπεριφορά του benchmark στα διαφορετικά datasets και αν το hotspot του προγράμματος παραμένει ίδιο σε όλες τις περιπτώσεις.

Στην περίπτωση που το hotspot δεν δικαιολογείται με βάση το πλήθος και την κατηγορία των εντολών που εκτελούνται, ή στην περίπτωση που το hotspot που παίρνω είναι διαφορετικό για τα διάφορα datasets που περνώ ως είσοδο τότε πρέπει να κάνω περαιτέρω ανάλυση του κώδικα στο εργαλείο VTune Performance Analyzer για να βρω το πώς επηρεάζει η μικροαρχιτεκτονική τον κώδικα και τον καθιστά το hotspot του προγράμματος.

Τέλος θα δώσω τα συμπεράσματα που προέκυψαν από την ανάλυση και την έρευνα μου.

1.3 Περίληψη Ατομικής Διπλωματικής Εργασίας

Στην διπλωματική μου εργασία ερεύνησα το πεδίο της επίδοσης υπολογιστικών συστημάτων, έκανα χαρακτηρισμό της επίδοσης των υπολογιστικών συστημάτων γενικά αλλά και συγκεκριμένα της επίδοσης των ενσωματωμένων συστημάτων. Στην συνέχεια μελέτησα μέσα από την ιστοσελίδα της EEMBC τα benchmarks με τα οποία ασχολείται ο οργανισμός και τον τρόπο με τον οποίο τα χρησιμοποιεί για να μετρήσει την επίδοση ενσωματωμένων συστημάτων. Μεταγλώττισα τα DENbench EEMBC benchmarks και ανέλυσα κάποια από αυτά με τη χρήση στατικής και δυναμικής ανάλυσης.

Μέσα από αυτή την ανάλυση μπόρεσα να μάθω και να καταλάβω τους λόγους για τους οποίους κάποια σημεία στον κώδικα καταναλώνουν περισσότερο χρόνο εκτέλεσης από τον υπόλοιπο κώδικα. Υπάρχουν πολλοί παράγοντες που οδηγούν στην αύξηση του χρόνου εκτέλεσης. Τις πλείστες φορές το hotspot βρίσκεται στο περισσότερο φωλιασμένο Loop του κώδικα, αφού σημαίνει πως ο κώδικας αυτός εκτελείται πολλές φορές και άρα χρειάζεται περισσότερο χρόνο για να αποπερατωθεί η εκτέλεση του. Hotspot όμως μπορεί να είναι μια συνάρτηση η οποία καλείται πάρα πολλές φορές από άλλες συναρτήσεις, άρα ο κώδικας της εκτελείται τόσες φορές όσες είναι οι κλήσεις και ο χρόνος εκτέλεσης της είναι πολύ μεγάλος. Το μέγεθος του Input σχεδόν πάντα παίζει σημαντικό ρόλο στο χρόνο εκτέλεσης όταν η πολυπλοκότητα του προγράμματος εξαρτάται από αυτό το μέγεθος. Όσο μεγαλύτερο είναι το αρχείο εισόδου τόσο περισσότερος είναι ο χρόνος που χρειάζεται για να το προσπελάσει και να το επεξεργαστεί το πρόγραμμα ώστε να εκτελέσει τις αλλαγές που πρέπει. Υπάρχουν περιπτώσεις όμως που hotspot είναι ένα κομμάτι κώδικα που εκτελεί πολλές προσβάσεις στη μνήμη αφού η πρόσβαση στη μνήμη κοστίζει υπολογιστικά στον επεξεργαστή. Μπορεί να ευθύνονται ακόμα και αποτυχίες στην πρόβλεψη των branches και πολλοί άλλοι παράγοντες μικροαρχιτεκτονικής.

1.4 Περιγραφή Ατομικής Διπλωματικής Εργασίας

Στο σημείο αυτό θα δώσω μια μικρή περιγραφή του τι περιλαμβάνεται πιο κάτω. Στο Κεφάλαιο 2 δίνεται ο ορισμός της επίδοσης των υπολογιστικών συστημάτων, περιγράφονται τα μετρικά της επίδοσης υπολογιστικών συστημάτων, και γίνεται αναφορά στην επίδοση ενσωματωμένων συστημάτων.

Στο Κεφάλαιο 3 αναφέρομαι στα benchmarks και συγκεκριμένα στα DENBench EEMBC Benchmarks με τα οποία ασχολήθηκα αναλυτικότερα. Έπειτα παραθέτω τον τρόπο με τον οποίο υπολογίζονται τα αποτελέσματα μέτρησης της επίδοσης των επεξεργαστών που τρέχουν τα benchmarks αυτά. Στη συνέχεια του κεφαλαίου περιγράφω κάθε ένα από τα DENBench benchmarks.

Στο Κεφάλαιο 4 περιγράφω την μεθοδολογία την οποία ακολούθησα για να αναλύσω τον κώδικα των benchmarks. Παραθέτω τα βασικά χαρακτηριστικα του συστήματος πάνω στο οποιο εργαζόμουν, δίνω λεπτομερή βηματική περιγραφή του τρόπου με τον οποίο μεταγλώττισα τα benchmarks στο Cygwin, περιγράφω το VTune τον profiler που χρησιμοποίησα για την ανάλυση των benchmarks, και τέλος δίνω μια λεπτομερή βηματική περιγραφή του τρόπου με τον οποίο ανέλυσα τα benchmarks στο VTune.

Στο Κεφάλαιο 5 δίνω μια μικρή περίληψη των άρθρων τα οποία μελέτησα, και έπειτα παρουσιάζω την ανάλυση των benchmarks τα συμπεράσματα και τις παρατηρήσεις μου.

Τέλος στο Κεφάλαιο 6 παραθέτω τα τελικά μου συμπεράσματα. Τα τεχνικά συμπεράσματα, αλλά και τα μη τεχνικά συμπεράσματα, τις εμπειρίες και γνώσεις που αποκόμισα.

Κεφάλαιο 2

Επίδοση Υπολογιστικών συστημάτων

2.1 Τι είναι η Επίδοση Υπολογιστικών Συστημάτων [3]

Μπορούμε να ορίσουμε την έννοια της επίδοσης με πολλούς διαφορετικούς τρόπους. Δεν μπορούμε δηλαδή να δώσουμε μόνο μια απάντηση στην ερώτηση ποιό computer έχει την καλύτερη επίδοση. Πρέπει πρώτα να ορίσουμε ως ποιο παράγοντα θέλουμε να μετρήσουμε την επίδοση π.χ. ταχύτητα, μνήμη.

Αν για παράδειγμα τρέξουμε ένα πρόγραμμα σε δύο διαφορετικούς επιτραπέζιους προσωπικούς υπολογιστές θα πούμε ότι ο πιο γρήγορος είναι ο υπολογιστής ο οποίος εκτέλεσε πρώτος το πρόγραμμα. Αν χρησιμοποιήσουμε ένα κέντρο δεδομένων το οποίο έχει πολλούς εξυπηρετητές που τρέχουν τις διεργασίες που δόθηκαν από περισσότερους από ένα χρήστες, μπορούμε να πούμε ότι το πιο γρήγορο είναι το κέντρο που εκτέλεσε τις περισσότερες διεργασίες σε μια μέρα.

Ένας χρήστης ενδιαφέρεται να μειώσει το χρόνο απόκρισης. Ο χρόνος αυτός ο οποίος ονομάζεται και χρόνος εκτέλεσης είναι ο χρόνος που χρειάζεται ο υπολογιστής από την αρχή μέχρι την ολοκλήρωση μιας εργασίας. Οι διαχειριστές

των κέντρων δεδομένων ενδιαφέρονται στο να αυξήσουν το Throughput, τη συνολική ποσότητα δουλειάς που γίνεται σε ένα ορισμένο χρονικό διάστημα.

Στις περισσότερες περιπτώσεις χρειαζόμαστε διαφορετικά μετρικά καθώς και διαφορετικές εφαρμογές για να μετρήσουμε την επίδοση των επιτραπέζιων υπολογιστών, των εξυπηρετητών και των ενσωματωμένων υπολογιστών.

Για να μεγιστοποιήσουμε την επίδοση πρέπει να ελαχιστοποιήσουμε το χρόνο απόκρισης για κάποια εργασία. Όσο πιο μεγάλη είναι η επίδοση τόσο πιο μικρός είναι ο χρόνος εκτέλεσης.

Ενδιαφερόμαστε για την μέτρηση της επίδοσης γιατί με αυτό τον τρόπο προσδιορίζεται εάν ένα σύστημα πληροί τις ανάγκες των εφαρμογών που ο χρήστης θεωρεί σημαντικές. Αν δηλαδή το σύστημα που θέλει να χρησιμοποιήσει είναι κατάλληλο και αποτελεσματικό για την δουλειά που θέλει να εκτελέσει. Τον εξυπηρετεί γιατί έτσι θα πάρει το πιο κατάλληλο σύστημα το οποίο θα τον ευχαριστεί και δεν θα του σπαταλά πολύτιμο χρόνο και κόπο.

Η επίδοση ενός προγράμματος εξαρτάται από πολλούς παράγοντες, όπως τον αλγόριθμο, τη γλώσσα προγραμματισμού, τον μεταγλωττιστή, την αρχιτεκτονική και το υλικό.

Για να αξιολογήσουμε δύο υπολογιστικά συστήματα πρέπει να συγκρίνουμε το χρόνο εκτέλεσης του φόρτου εργασίας των δύο υπολογιστών.

Οι περισσότεροι χρήστες όμως δεν μπορούν να το κάνουν αυτό και έτσι πρέπει να ακολουθήσουν άλλες μεθόδους. Αυτές οι άλλες μέθοδοι περιλαμβάνουν τα Benchmarks.

2.2 Μέτρηση επίδοσης

Ο χρόνος είναι η μονάδα μέτρησης της επίδοσης ενός υπολογιστή. Ο χρόνος μπορεί να οριστεί με διαφορετικούς τρόπους ανάλογα με το τι θέλουμε να μετρήσουμε. Όταν ο επεξεργαστής δουλεύει ταυτόχρονα σε διαφορετικά προγράμματα ο χρήστης θέλει να ξεχωρίσει το χρόνο που έχει σπαταλήσει ο επεξεργαστής για το δικό του πρόγραμμα από τον συνολικό χρόνο που έχει σπαταλήσει για την εκτέλεση του

συνόλου των προγραμμάτων. Έτσι μπορούμε να παραθέσουμε την έννοια του χρόνου εκτέλεσης της κεντρικής μονάδας επεξεργασίας(CPU execution time). Ο χρόνος αυτός είναι ο ακριβής χρόνος που σπατάλησε η ΚΜΕ στον υπολογισμό μιας συγκεκριμένης διεργασίας χωρίς να συμπεριλαμβάνεται ο χρόνος εισόδου/εξόδου ή ο χρόνος που χρειάστηκε για να τρέξει τα υπόλοιπα προγράμματα. Το CPU execution time χωρίζεται στον χρόνο που σπαταλήθηκε στο πρόγραμμα (user CPU time) και στον χρόνο που σπαταλήθηκε στο λειτουργικό σύστημα (system CPU time).

Για να μπορούν οι χρήστες να κατανοούν και να έχουν μια σωστή εικόνα του πόσο αποδοτικά είναι τα συστήματα τους, οι υπολογιστές κατασκευάζονται με ένα ρολόι το οποίο μετρά με μια σταθερή τιμή χρόνου και καθορίζεται όταν συμβαίνουν κάποια γεγονότα στο hardware. Αυτά τα διαστήματα χρόνου λέγονται clock cycles.

2.3 Επίδοση στα ενσωματωμένα συστήματα

Η επίδοση στα ενσωματωμένα συστήματα συχνά χαρακτηρίζεται από περιορισμούς πραγματικού χρόνου. Αυτό σημαίνει ότι οι συγκεκριμένες τους εφαρμογές πρέπει να εκτελεστούν σε ένα περιορισμένο χρονικό διάστημα . Υπάρχουν δύο ειδών περιορισμοί πραγματικού χρόνου. Το hard real time και το soft real time.

Στο hard real time ορίζεται ένα σταθερό και αμετάβλητο χρονικό διάστημα στο οποίο το σύστημα πρέπει απαραιτήτως να ανταποκριθεί ή να επεξεργαστεί κάποιο γεγονός. Για παράδειγμα το σύστημα που επεξεργάζεται τους αερόσακους ενός αυτοκινήτου, πρέπει απαραίτητα να ανταποκριθεί την κρίσιμη ώρα αλλιώς θα επέλθει τραγωδία. Στα soft real time είναι ικανοποιητική μια μέση ανταπόκριση ή η ανταπόκριση ενός μεγάλου αριθμού γεγονότων στο καθορισμένο χρονικό διάστημα ανταπόκρισης. Για παράδειγμα ο τρόπος εκτέλεσης ενός πλαισίου εικόνας σε ένα σύστημα DVD. Μπορεί κάποιο πλαίσιο να απορριφθεί και άρα το μεγαλύτερο πλήθος πλαισίων ανταποκρίνεται στην αίτηση του χρήστη αλλά υπάρχει και ένα μικρό ποσοστό πλαισίων τα οποία δεν ανταποκρίνονται. Όταν υπάρξει περιορισμός στην επίδοση του χρόνου απόκρισης σε αυτά τα συστήματα οι σχεδιαστές τους προσπαθούν να μειώσουν το κόστος ή να βελτιώσουν το σύνολο της δουλειάς που πρέπει γίνει σε ένα περιορισμένο χρονικό διάστημα.

Κεφάλαιο 3

Εισαγωγή στα Benchmarks

3.1 Τι είναι benchmark

Στην επιστήμη των υπολογιστών, ένα benchmark (συγκριτική μέτρηση επιδόσεων) είναι η πράξη κατά την οποία τρέχουμε ένα πρόγραμμα υπολογιστών , ένα σύνολο από προγράμματα ή άλλες εφαρμογές, προκειμένου να αξιολογηθεί η σχετική επίδοση ενός συστήματος.

Τα Benchmarks είναι δηλαδή προγράμματα ειδικά επιλεγμένα για να δοκιμάζουν την επίδοση των υπολογιστικών συστημάτων. Τα benchmarks δημιουργούν ένα workload το οποίο οι χρήστες ευελπιστούν να αντικατοπτρίζει την επίδοση του πραγματικού workload της εφαρμογής.

Στις μέρες μας ως benchmarks χρησιμοποιούνται πραγματικές εφαρμογές. Οι πραγματικές εφαρμογές μπορεί να είναι εφαρμογές τις οποίες ο χρήστης εφαρμόζει καθημερινά ή απλές τετριμμένες εφαρμογές. Χρησιμοποιώντας πραγματικές εφαρμογές γίνεται δύσκολη η εξεύρεση τρόπων για αύξηση της επίδοσης του

benchmark. Όταν βρεθούν τεχνικές για να αυξήσουν την επίδοση αυτές οι τεχνικές είναι περισσότερο πιθανό να βοηθήσουν άλλα προγράμματα παρά τα ίδια τα benchmarks.

Διαφορετικές κλάσεις και διαφορετικές εφαρμογές υπολογιστών μπορεί να απαιτούν διαφορετικούς τύπους benchmarks. Γι' αυτό το λόγω γίνεται ο διαχωρισμός τους σε τρεις κύριες ομάδες.

- Benchmarks για επιτραπέζιους υπολογιστές
  Για τους επιτραπέζιους υπολογιστές περισσότερο συνηθισμένα είναι τα benchmarks που εστιάζονται σε μια συγκεκριμένη εργασία.

- Benchmarks για εξυπηρετητές
  Για τους εξυπηρετητές η απόφαση για το ποιο benchmark θα χρησιμοποιηθεί εξαρτάται από τη φύση της εφαρμογής.

- Benchmarks για ενσωματωμένα συστήματα
  Τα benchmarks για τα ενσωματωμένα συστήματα είναι πολύ πιο ανεπτυγμένα από τα benchmarks για τους επιτραπέζιους υπολογιστές ή τους εξυπηρετητές.

  Από νωρίς έγινε κατανοητό ότι τα απλά σύνολα των benchmarks δεν μπορούν να χρησιμοποιηθούν στη μέτρηση της επίδοσης των ενσωματωμένων συστημάτων. Αυτό συμβαίνει εξαιτίας της τεράστιας ποικιλίας ενσωματωμένων εφαρμογών όπως και εξαιτίας των διαφορών στις απαιτήσεις επίδοσης, αφού κάποιες ανταποκρίνονται σε soft real time και άλλες σε hard real time.

  Στα ενσωματωμένα συστήματα τα «καλά» benchmarks είναι πολύ πιο σπάνια. Συχνά κάποιοι χρήστες χρησιμοποιούν τη δική τους συγκεκριμένη ενσωματωμένη εφαρμογή ή ένα τμήμα αυτής της εφαρμογής. Τα πιο κύρια benchmarks που χρησιμοποιήθηκαν για ενσωματωμένα συστήματα είναι τα EEMBC benchmarks.

Πολλοί κατασκευαστές ενσωματωμένων συστημάτων σχεδιάζουν benchmarks τα οποία απεικονίζουν την εφαρμογή τους άλλοτε σαν πυρήνες και άλλοτε σαν μικρές αυτόνομες εκδόσεις ολόκληρης της εφαρμογής. Γι' αυτές τις συγκεκριμένες εφαρμογές χρησιμοποιήθηκαν τα EEMBC benchmarks.

Παρόλα αυτά πολλές ενσωματωμένες εφαρμογές είναι ευαίσθητες στην επίδοση των μικρών πυρήνων αφού τις περισσότερες φορές η ολική επίδοση ολόκληρης της εφαρμογής, που συχνά φτάνει τις πολλές χιλιάδες γραμμές κώδικα, είναι επίσης κρίσιμη.

Γι' αυτό το λόγο πολλά ενσωματωμένα συστήματα χρησιμοποιούν τα Benchmarks μόνο για να μετρήσουν μερικώς την επίδοση.

3.2 EEMBC benchmarks [3,7]

EEMBC είναι τα αρχικά της Ενσωματωμένης Κοινοπραξίας Συγκριτικής Μέτρησης Επιδόσεων Μικροεπεξεργαστών. Είναι μια μη κερδοσκοπική οργάνωση που δημιουργήθηκε το 1997 με σκοπό της να αναπτύξει benchmarks τα οποία δοκιμάζουν την επίδοση του υλικού και του λογισμικού που χρησιμοποιούνται στα ενσωματωμένα συστήματα.

Τα EEMBC benchmarks στοχεύουν στην απεικόνιση των εφαρμογών πραγματικού κόσμου και τις απαιτήσεις στις οποίες πρέπει να αντεπεξέλθουν αυτά τα ενσωματωμένα συστήματα (embedded systems) ώστε να είναι το μέγιστο δυνατό αποδοτικά.

Τα EEMBC benchmarks χωρίζονται σε πέντε ομάδες τα αυτόματα/μηχανικά, καταναλωτικά, δικτυακά, αυτοματοποίησης γραφείου και τηλεπικοινωνιών.

3.3 DENBench EEMBC benchmarks [7]

Τα DENbench Version 1.0 είναι τα Digital Entertainment Benchmarks και υπάγονται στα Consumer EEMBC benchmarks. Είναι μια ακολουθία από benchmarks που επιτρέπουν στους χρήστες να δοκιμάσουν την επίδοση των υποσυστημάτων που βρίσκονται στα πολυμέσα(εικόνας, video, συμπίεσης και αποσυμπίεσης audio file) άλλα benchmarks στην ακολουθία εστιάζουν στους αλγόριθμους κρυπτογράφησης και αποκρυπτογράφησης που χρησιμοποιούνται συνήθως στις ψηφιακές εφαρμογές διαχείρισης δικαιωμάτων και στις εφαρμογές ηλεκτρονικού εμπορίου. Με αυτά τα benchmarks εξετάζονται πολυμέσα όπως PDAs, κινητά τηλέφωνα, Mp3 players, DVD players/recorders, ψυχαγωγικά συστήματα αυτοκινήτου και ψηφιακές κάμερες.

Τα DENbench περιλαμβάνουν τους ακόλουθους αλγόριθμους ή μίνι ακολουθίες:

- MPEG[9]

  Περιλαμβάνουν αποκωδικοποιήσεις MP3, κωδικοποίηση και αποκωδικοποίηση MPEG2, κωδικοποίηση και αποκωδικοποίηση MPEG4, κάθε ένα από τα οποία εφαρμόζεται σε πέντε διαφορετικά σύνολα δεδομένων για συνολικά εικοσιπέντε αποτελέσματα.

- Κρυπτογραφία

  Μια συλλογή από τέσσερα benchmark tests για κοινά κρυπτογραφικά πρότυπα και αλγόριθμους. Τα προηγμένα πρότυπα κρυπτογράφησης(AES), τα ψηφιακά πρότυπα κρυπτογράφησης (DES), ο αλγόριθμος Rivest-Shamir-Adleman(RSA) για το βασικό δημόσιο σύστημα κρυπτογραφίας, και η αποκωδικοποίηση Huffman για αποσυμπίεση δεδομένων.

- Ψηφιακή επεξεργασία εικόνας

  Οι δοκιμές JPEG και μετατροπή χώρου χρώματος περιλαμβάνουν συμπίεση JPEG.

- MPEG κωδικοποίηση κινητής υποδιαστολής

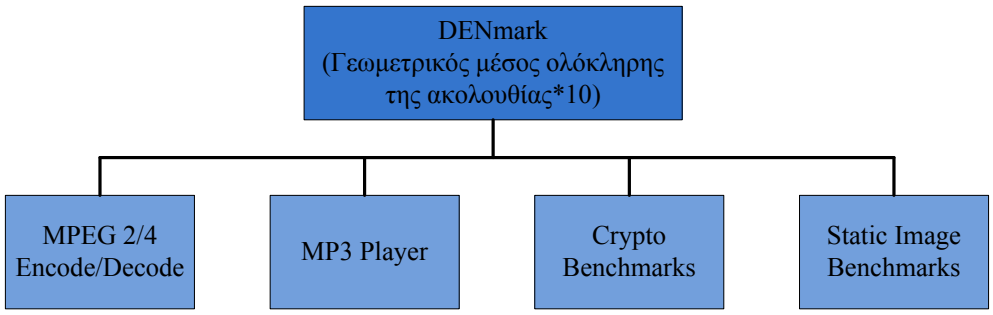
  Μια έκδοση floating point του MPEG2 encoder benchmark, χρησιμοποιεί μονής ακρίβειας κινητή υποδιαστολή και όχι ακέραιους αριθμούς.

3.4 Μέτρηση DENBench EEMBC Benchmarks [7]

Όταν τρέξουμε τα benchmarks στο σύστημα του οποίου θέλουμε να μετρήσουμε την επίδοση παίρνουμε κάποια αποτελέσματα τα επονομαζόμενα scores. Αυτά τα scores μετρούνται με iterations per second, δηλαδή πόσες φορές το σύστημα του οποίου δοκιμάζουμε την επίδοση μπορεί να εκτελέσει το benchmark πρόγραμμα μέσα σε ένα δευτερόλεπτο.

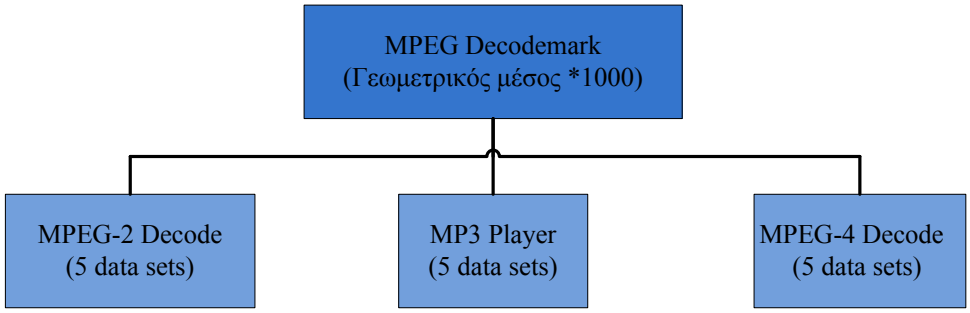
Τα DENmark, MPEG Decodemark, MPEG Encodemark, Cryptomark και Imagemark είναι αποτελέσματα μονών αριθμών τους οποίους παρέχει η EEMBC στην πρόσθεση με τους βαθμούς από τις ξεχωριστές εφαρμογές των benchmarks εσωτερικά της DENbench ακολουθίας του, για να εμπλουτίσει την εμφάνιση των συγκριτικών δεδομένων στην επίδοση του επεξεργαστή. Αυτοί οι αριθμοί προτείνεται να παρέχουν μιας πρώτης τάξεως αναπαράσταση της επίδοσης του επεξεργαστή σε εργασίες που σχετίζονται με τις εφαρμογές ψηφιακής ψυχαγωγίας. Τα λεπτομερή αποτελέσματα στα ξεχωριστά benchmarks και στα σύνολα δεδομένων θα συνεχίσουν να προσφέρουν την υψηλότερη τιμή στους σχεδιαστές συστήματος, επιτρέποντας τη σύγκριση των ξεχωριστών εφαρμογών τα οποία είναι συγκεκριμένα στους σχεδιασμούς τους.

- DENmark: Ολόκληρος ο βαθμός DENmark παρέχει ένα μονό αριθμό εκτίμησης της επίδοσης για όλη την DENbench ακολουθία. Άρα ένα μέλος πρέπει να τρέξει και τα 69 Tests της ακολουθίας ώστε να αντλήσει το συνολικό αποτέλεσμα DENmark.



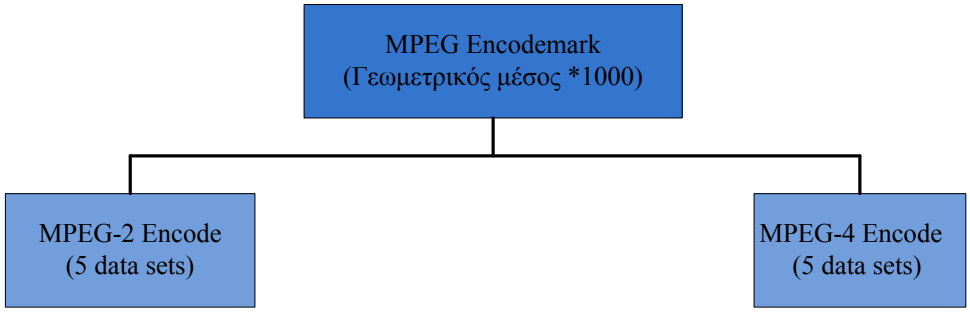
Σχήμα 3.1 Υπολογισμός DENmark

- MPEG Decodemark: Ο MPEG Decodemark συγκεντρώνει αποτελέσματα παρέχοντας ένα στιγμιότυπο της επίδοσης σε συγκεκριμένες ομάδες των tests.



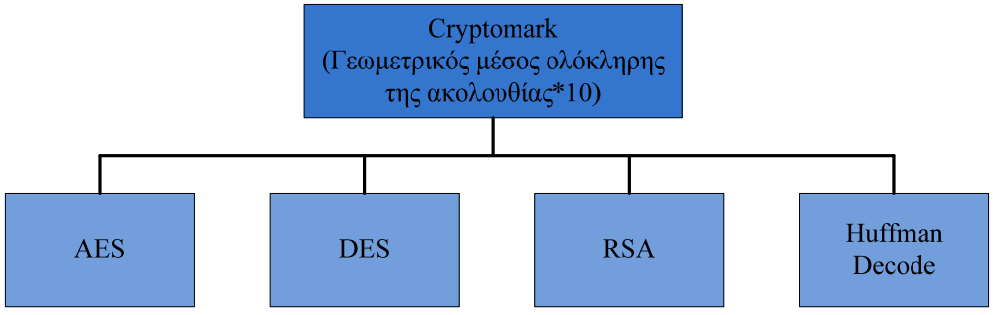
Σχήμα 3.2 Υπολογισμός MPEG Decodemark

- MPEG Encodemark: Ο MPEG Encodemark συγκεντρώνει αποτελέσματα παρέχοντας ένα στιγμιότυπο της επίδοσης σε συγκεκριμένες ομάδες των tests.



Σχήμα 3.3 Υπολογισμός MPEG Encodemark

- Cryptomark: Ο Cryptomark συγκεντρώνει αποτελέσματα παρέχοντας ένα στιγμιότυπο της επίδοσης σε συγκεκριμένες ομάδες των tests.



Σχήμα 3.4 Υπολογισμός Cryptomark

Υπολογισμός βαθμών αποτελεσμάτων

Στο πιο κάτω σχήμα φαίνεται η μέθοδος που χρησιμοποιείται στον υπολογισμό καθενός από τα αποτελέσματα. Ο γεωμετρικός μέσος κάθε αποτελέσματος πολλαπλασιάζεται με ένα σταθερό παράγοντα τέτοιο ώστε να διατηρηθούν οι περισσότεροι βαθμοί μέσα στην ίδια διάταξη ποσότητας.

Όνομα αποτελέσματος	**Εφάρμοσε γεωμετρικό μέσο στα:**	**Η ν-οστή ρίζα**	**Παράγοντες κανονικοποίησης**
MPEG Decodemark	5 αποτελέσματα των MPEG-2, MPEG-4 Decode benchmark, MP3 player benchmark.	15	1000
MPEG Encodemark	5 αποτελέσματα για καθένα από τα MPEG-2 Encode benchmark, MPEG-4 Encode benchmark.	10	1000
Cryptomark	Για αποτελέσματα από τα AES, DES, RSA και Huffman Decode benchmarks.	4	10
Imagemark	7 αποτελέσματα για RGB/YIQ, RGB/HPG, PGB/CMYK, JPEG συμπίεση, αποσυμπίεση.	35	10
DENmark	Τα 69 ανεξάρτητα αποτελέσματα δεδομένων της DENbench ακολουθίας.	69	10

3.5 Τα διάφορα είδη DENBench EEMBC Benchmarks [7]

- AES:

  Το AES benchmark παρέχει μια ένδειξη της επίδοσης ενός μικροεπεξεργαστή ή ενός υποσυστήματος επεξεργαστή ψηφιακού σήματος ο οποίος διενεργεί AES κρυπτογραφήσεις και αποκρυπτογραφήσεις.

  Η AES κρυπτογραφία χρησιμοποιείται σε πολλά κρυπτογραφικά πρωτόκολλα. Το AES προτάθηκε για να αντικαταστήσει τα ψηφιακά πρότυπα κρυπτογράφησης (DES). Χρησιμοποιεί τον αλγόριθμο Rijndael.

  Ο αλγόριθμος Rijndael είναι μια συνεχώς επαναλαμβανόμενη κρυπτογραφική ακολουθία block με μεταβλητό μήκος block και μεταβλητό μήκος κλειδιού. Το μήκος του Block και το μήκος του κλειδιού μπορούν ανεξάρτητα να καθοριστούν στα 128,192 ή 256 bits. Τα EEMBC AES benchmark υλοποιεί και τα τρία αυτά μήκη κλειδιών για κάθε επανάληψη. Όπως και στις περισσότερες λειτουργίες κρυπτογραφίας, υπάρχει ένας πίνακας (ή «block») ο οποίος υποβάλλεται σε πολλές μετατροπές. Αυτό το benchmark εκτελεί 1000 Monte Carlo δοκιμές για 16 περάσματα και κάνει μία ολοκληρωμένη κωδικοποίηση, ακολουθούμενη από αποκωδικοποίηση για να επικυρώσει την ορθότητα. Υλοποιώντας τις FIPS δοκιμές εσωτερικά του κώδικα εμπλουτίζεται επιπλέον η ορθότητα. Τελικά ο κώδικας υλοποιεί τη Wide Trail Strategy για να παρεκκλίνει από τις επιθέσεις των layer ώστε να μην διαρρέουν τα δεδομένα και υλοποιεί και τα τρία είδη layer: γραμμική ανάμειξη, μη γραμμικό layer, key addition layer.

  Το benchmark αυτό είναι υπολογιστικά ενδιαφέρον αφού σε αυτό χρησιμοποιούνται πολλοί μαθηματικοί υπολογισμοί. Υλοποιείται σε ακέραια μαθηματικά. Αυτό το benchmark είναι σχεδόν αποκλειστικά όριο της ΚΜΕ και η ποιότητα της βιβλιοθήκης μαθηματικών, όπως και αυτή της βιβλιοθήκης μνήμης έχει συνέπεια στην επίδοση.

- DES:

  Δοκιμάζει την επίδοση του Digital Encryption Standard αλγόριθμου (DES). Ο Digital Encryption Standard αλγόριθμος είναι ένας κρυπτογραφικός αλγόριθμος ο οποίος παρέχει μια ένδειξη της ενδεχόμενης της επίδοσης ενός μικροεπεξεργαστή η ενός υποσυστήματος επεξεργαστή ψηφιακού σήματος διενεργώντας DES κρυπτογραφήσεις και αποκρυπτογραφήσεις.

  Η DES κρυπτογραφία χρησιμοποιείται σε πολλά κρυπτογραφικά πρωτόκολλα. Ο αλγόριθμος DES χρησιμοποιείται σε πολλές eCommerce m-Commerce εφαρμογές αλλά για τις πιο αξιόπιστες εφαρμογές έχει αντικατασταθεί από τον AES αλγόριθμο.

  Ο αλγόριθμος DES είναι μια συνεχώς επαναλαμβανόμενη κρυπτογραφική ακολουθία block με σταθερό μήκος block των 64 bits και σταθερό μήκος κλειδιού των 64 bits. Αλλά μόνο τα πρώτα 56 bits χρησιμοποιούνται για κρυπτογραφικούς σκοπούς, τα υπόλοιπα 8 bits χρησιμοποιούνται για έλεγχο ισότητας. Η EEMBC υλοποιεί το σωστό μήκος κλειδιού. Όπως και στις περισσότερες λειτουργίες κρυπτογραφίας, υπάρχει ένας πίνακας (ή «block») ο οποίος υποβάλλεται σε πολλές μετατροπές, στην περίπτωση του DES έχουμε 16 μετατροπές. Αντίθετα με την υλοποίηση του AES benchmark η EEMBC δεν υλοποιεί τους FIPS ελέγχους στον DES. Ο έλεγχος γίνεται με τον κυκλικό έλεγχο πλεονάζοντος αθροίσματος (Cyclical Redundancy Checksum).

  Το benchmark αυτό είναι υπολογιστικά ενδιαφέρον αφού σε αυτό χρησιμοποιούνται πολλοί μαθηματικοί υπολογισμοί. Υλοποιείται σε ακέραια μαθηματικά. Αυτό το benchmark είναι σχεδόν αποκλειστικά όριο της ΚΜΕ και η ποιότητα της βιβλιοθήκης μαθηματικών, όπως και αυτή της βιβλιοθήκης μνήμης έχει συνέπεια στην επίδοση.

- High-Pass Grey-Scale Filter:

  Δοκιμάζει την επίδοση της επεξεργασίας εικόνας που χρησιμοποιείται στις ψηφιακές κάμερες και σε άλλα προϊόντα ψηφιακής εικόνας. Το High-Pass Grey-Scale Filter χρησιμοποιείται στη front end επεξεργασία της ψηφιακής κάμερας. Τα RGB δεδομένα είτε από CCD είτε από CMOS αισθητήρες είναι ήδη προεπεξεργασμένα από αυτό το φίλτρο για να τονιστεί η εικόνα, και μετά περνούν για JPEG επεξεργασία συμπίεσης εικόνων. Δηλαδή το High-Pass Grey-Scale Filter παίρνει μια σκοτεινή εικόνα και κάνει πιο εμφανή τα σημεία της.

  Αυτό το benchmark είναι ένας από τους πιο συχνά χρησιμοποιημένους αλγόριθμους στην επεξεργασία εικόνας και αναπαριστά μια καλή μονάδα μέτρησης της επίδοσης της ΚΜΕ στα προϊόντα ψηφιακής εικόνας.

  Το benchmark αυτό εξετάζει την ικανότητα της ΚΜΕ να εκτελέσει προσπέλαση δισδιάστατου πίνακα δεδομένων και να εκτελέσει υπολογισμούς πολλαπλασιασμού/αθροίσματος. Για κάθε pixel της εικόνας, το φίλτρο υπολογίζει την νέα τιμή χρησιμοποιώντας τα 9 pixels γύρω από το pixel που δουλεύουμε συμπεριλαμβάνοντας και αυτό, πολλαπλασιάζει την τιμή αυτή με κάποιους συντελεστές φίλτρου, τα προσθέτει. Το αποτέλεσμα της άθροισης είναι 16 bit οπότε το κάνουμε shifting 8 bits προς τα αριστερά ώστε να έχουμε δεδομένα ενός byte.

  Ένα for loop υπολογίζει κάθε φορά την τιμή ενός pixel. Για τον υπολογισμό ενός pixel τα δεδομένα του κεντρικού pixel και των 8 γειτονικών του πρέπει να φορτωθούν. Αυτή η διαδικασία καταναλώνει χρόνου, αν αναλογιστούμε το χρόνο που σπαταλάτε για την προσπέλαση της κύριας μνήμης ή της μνήμης cache.

- Huffman Decoding:

  Δοκιμάζει την επίδοση της ενδεχόμενης επίδοσης του επεξεργαστή σε μια ψηφιακή κάμερα και μοντελοποιείτε σε μια εικόνα δεδομένων (YUV δεδομένα). Η αποκωδικοποίηση Huffman είναι ένας αλγόριθμος κλειδιού στα JPEG, MPEG και σε άλλα πλάνα συμπίεσης που χρησιμοποιούνται στις ψηφιακές κάμερες.

  Το Huffman Decoding benchmark αρχικοποιεί τους AC και DC χρωματικούς και φωτεινούς πίνακες(4 πίνακες). Συμπληρώνει τα διάφορα buffers(ενδιάμεση μνήμη) και μετά εκτελεί την λειτουργία Huffman Decoding χρησιμοποιώντας μια αρκετά σταθερή υλοποίηση Huffman.

  Το benchmark εστιάζεται περισσότερο στη διαδικασία εύρεσης στοιχείων σε πίνακες, στον bit manipulation και το shifting παρά στα αρχεία εισόδου/εξόδου. Οι memory-to-memory τελεστές(operations) είναι σημαντικοί για την επίδοση, όταν οι ενδιάμεσες τιμές αποθηκεύονται σταθερά.

- MP3 Decode:

  Το MP3 Decoder benchmark παρέχει μια ένδειξη της ενδεχόμενης επίδοσης ενός υποσυστήματος μικροεπεξεργαστή που τρέχει σε μια εφαρμογή ενός MP3 player.

  Το benchmark αυτό είναι μια ακέραια υλοποίηση του ISO 13818-3 MPEG-2 Layer 3 Decoder με χαμηλότερες δειγματοληψίες συχνότητας. Οι κανονικές δειγματοληψίες συχνότητας είναι 32 KHz, 44.1 KHz (τυπική συχνότητα του audio CD-ROM), ή 48 KHz. Χαμηλότερες δειγματοληψίες συχνότητας είναι 16 KHz, 22.05 KHz, ή 24 KHz. Επιλέγουμε χαμηλότερη δειγματοληψία για να εκφράσουμε αυτή που είναι συχνά χρησιμοποιημένη στα PDAs, στα κινητά τηλέφωνα και στις ιστοσελίδες όπου το bandwidth είναι εμπλεκόμενο.

  Το Benchmark προσομοιώνει την αποκωδικοποίηση και αναπαράγει συμπυκνώνοντας τα στατικά δεδομένα των πιο κάτω MP3 κωδικοποιημένων αρχείων:

- JUPITER.mp3
- music128stereo.mp3
- music48_128stereo.mp3
- music64stereo.mp3
- music48mono.mp3

Επεξεργασία MP3 Decode benchmark:

- Διάβασμα του MP3 αρχείου που επιλέχθηκε.
- Διάβασμα και διερμήνευση των πληροφοριών επικεφαλίδας.
- Διάβασμα και αποκωδικοποίηση των πλαισίων δεδομένων.
- Επεξεργασία των δεδομένων που βασίζονται στις πληροφορίες επικεφαλίδας.
- Εξαγωγή μουσικής ως δύο κανάλια των 16-bit παλμών κώδικα προσαρμοσμένων δεδομένων
- Μια PSNR τιμή υπολογίζεται για κάθε PCM πλαίσιο.

Μια μονή επανάληψη του benchmark ολοκληρώνεται όταν το τέλος του εισαγόμενου αρχείου επεκταθεί και δεν υπάρχουν άλλα διαθέσιμα δεδομένα για να υποστούν επεξεργασία.

Αυτή είναι μια ακέραια υλοποίηση του MPEG-1/2 Layer 3 audio και είναι ένα Benchmark το οποίο εστιάζει περισσότερο στην υπολογιστική επεξεργασία παρά στα αρχεία εισόδου/εξόδου.

- <u>MPEG-2 Decode:</u>

Το MPEG-2 Decoder benchmark παρέχει μια ένδειξη της ενδεχόμενης επίδοσης ενός υποσυστήματος μικροεπεξεργαστή που τρέχει σε μια εφαρμογή ενός MPEG-2 Decoder, όπως αυτές που βρίσκονται σε ένα DVD player ή σε ένα ψηφιακό μετασχηματιστή.

Το benchmark περιλαμβάνει μια εφαρμογή σταθερών ακέραιών σημείων των MSSG ISO πηγών. Ο αποκωδικοποιητής mpeg-2 χρησιμοποιεί μια

τυποποιημένη εφαρμογή αναφοράς του αλγορίθμου πυρήνων, συμπεριλαμβανομένης της αποκωδικοποίησης Huffman και τις τροποποιημένες αντίστροφες ιδιαίτερες ρουτίνες μετατροπής συνημίτονου (iDCT).

Το δεδομένα εισόδου του benchmark είναι μια σειρά αρχείων .MPEG, και η παραγωγή είναι μια σειρά αρχείων .PPM, η οποία μπορεί να παρουσιαστεί χρησιμοποιώντας οποιοδήποτε κατάλληλο γραφικό παρουσιαστή αρχείων. Η ακρίβεια ελέγχεται από τον κυκλικό πλεονασμό Checksum (κέντρο ανίχνευσης και ελέγχου), η ποιότητα μετριέται χρησιμοποιώντας μέγιστη αναλογία ανάλυσης σήματος/διαταραχής (PSNR). Το κέντρο ανίχνευσης και ελέγχου (CRC) χρησιμοποιείται ως σημείο ελέγχου μόνο, όχι ως κανονική επικύρωση.

Επεξεργασία MPEG-2 Decoder benchmark:

- Διάβασμα του MPEG-2 αρχείου.
- Διάβασμα και διερμήνευση των πληροφοριών επικεφαλίδας.
- Διάβασμα και αποκωδικοποίηση των πλαισίων δεδομένων.
- Επεξεργασία των δεδομένων που βασίζονται στις πληροφορίες επικεφαλίδας.
- Εξαγωγή του .PPM αρχείου στη μνήμη.
- Μια PSNR τιμή υπολογίζεται.

Η ποιότητα παραγωγής μετριέται χρησιμοποιώντας τον κώδικα μέγιστης αναλογίας σήματος/διαταραχής (PSNR) που αναπτύχθηκε από την EEMBC.

Το Benchmark εστιάζει περισσότερο στην υπολογιστική επεξεργασία παρά στα αρχεία εισόδου/εξόδου. Οι βασικοί αλγόριθμοι είναι η αντίστροφη ιδιαίτερη μετατροπή συνημίτονου.

- MPEG-2 Encode:

  Το MPEG-2 Encoder benchmark παρέχει μια ένδειξη της ενδεχόμενης επίδοσης ενός υποσυστήματος μικροεπεξεργαστή που τρέχει σε μια εφαρμογή ενός MPEG-2 Encoder.

  Το benchmark περιέχει δύο διαφορετικές παραλλαγές: ένα προαιρετικό ενιαίας ακρίβειας floating-point αλγόριθμο και μια έκδοση σταθερών σημείων προερχόμενων από τις πηγές του ISO. Το Mpeg-2 encoder χρησιμοποιεί μια αρκετά τυποποιημένη εφαρμογή αναφοράς του αλγορίθμου πυρήνων, συμπεριλαμβανομένης της αποκωδικοποίησης Huffman και των τροποποιημένων αντίστροφων ιδιαίτερων ρουτινών μετατροπής συνημιτόνου (iDCT).

  Η εισαγωγή είναι μια σειρά πέντε σύνολα δεδομένων, τα οποία λαμβάνουν τη μορφή .PPM αρχείων μαζί με YUV. Η παραγωγή είναι ένα αρχείο αρχείων MPEG (.MPEG) που μπορεί να παιχτεί χρησιμοποιώντας το Windows Media Player ή τη Apple QuickTime για να βοηθήσει στον έλεγχο για το αν η κωδικοποίηση ήταν σωστή. Η ακρίβεια ελέγχεται επίσης από κυκλικό checksum πλεονασμού (CRC Checking), και μετράμε την ποιότητα χρησιμοποιώντας PSNR ανάλυση.

  Επεξεργασία MPEG-2 Encoder benchmark:

  - Ανάγνωση των επιλεγμένων πλαισίων YUV.
  - Ανάγνωση και ερμηνεία των πληροφοριών επικεφαλίδας.
  - Ανάγνωση και κωδικοποίηση των πλαισίων δεδομένων.
  - Επεξεργασία των δεδομένων που βασίζονται στις πληροφορίες επικεφαλίδων.
  - Έξοδος του αρχείου .mpeg στη μνήμη.
  - Υπολογισμός μιας PSNR τιμής.

  Η ποιότητα παραγωγής μετριέται χρησιμοποιώντας τον κώδικα μέγιστης αναλογίας σήματος/διαταραχής (PSNR). Η PSNR μετριέται έξω από το βρόχο συγχρονισμού του benchmark.

Υπάρχουν δύο υλοποιήσεις, μία υλοποίηση fixed point και μία υλοποίηση floating point. Η floating point έκδοση είναι προαιρετική. Το Benchmark εστιάζει περισσότερο στην υπολογιστική επεξεργασία παρά στα αρχεία εισόδου/εξόδου.Τα PSNR αποτελέσματα τα οποία παράγονται πρέπει να παρουσιαστούν σε report.

- MPEG-4 Decode:

Το MPEG-4 Decoder benchmark παρέχει μια ένδειξη της ενδεχόμενης επίδοσης ενός υποσυστήματος μικροεπεξεργαστή που τρέχει σε μια εφαρμογή ενός MPEG-4 Decoder. Τα MPEG-4 encode και decode είναι δημοφιλή στις φορητές συσκευές και στο διαδίκτυο όπως και στις ψηφιακές τηλεοράσεις καθώς και στα video πέρα από τις IP εφαρμογές. Στην EEMBC υλοποιείται ο κώδικάς βάσης XviD με τροποποιήσεις για benchmarking και για ιδιόκτητα datasets.

Η υλοποίηση του XviD από την EEMBC χρησιμοποιεί απλό profile level 3 για να κωδικοποιήσει τα αρχεία. Αποκωδικοποιούν τα αποτελέσματα του MPEG-4 Encoder. Τα datasets που εισάγονται (YUV αρχεία) είναι τα ίδια με αυτά για τα MPEG-2 benchmarks της EEMBC και το εξαγόμενο των MPEG-4 Decode benchmarks είναι mp4u αρχεία τα οποία μπορούν να συγκριθούν με τα MPEG-2 ισοδύναμα τους.

Η εισαγωγή είναι ένα mp4u αρχείο και αυτό που παράγεται είναι μια σειρά από YUV αρχεία τα οποία μπορούν να παρουσιαστούν χρησιμοποιώντας GIMP ή άλλο πρόγραμμα παρουσίασης εικόνων ώστε να βοηθήσουν να επαληθεύσουν ότι η αποκωδικοποίηση ήταν σωστή. Επαληθεύουμε την ορθότητα αυτού που παράχθηκε μετρώντας την ποιότητα αυτού χρησιμοποιώντας PSNR ανάλυση.

Επεξεργασία MPEG-4 Decoder benchmark:

- Ανάγνωση των επιλεγμένων πλαισίων YUV.
- Ανάγνωση και ερμηνεία των πληροφοριών επικεφαλίδας.
- Ανάγνωση και κωδικοποίηση των πλαισίων δεδομένων.

- Επεξεργασία των δεδομένων που βασίζονται στις πληροφορίες επικεφαλίδων.
- Έξοδος του αρχείου .mpeg στη μνήμη.
- Υπολογισμός μιας PSNR τιμής.

Η ποιότητα παραγωγής μετριέται χρησιμοποιώντας τον κώδικα μέγιστης αναλογίας σήματος/διαταραχής (PSNR). Η PSNR μετριέται έξω από το βρόχο συγχρονισμού του benchmark.

MPEG-4 Decode benchmark προσφέρεται μόνο σε έκδοση σταθερού σημείου. Είναι ένα benchmark που εστιάζει περισσότερο στην υπολογιστική επεξεργασία παρά στα αρχεία εισόδου/εξόδου. Τα PSNR αποτελέσματα πρέπει να παρουσιαστούν σε report ώστε να πιστοποιηθούν και να δημοσιοποιηθούν.

- <u>MPEG-4 Encode:</u>

Το MPEG-4 Encoder benchmark παρέχει μια ένδειξη της ενδεχόμενης επίδοσης ενός υποσυστήματος μικροεπεξεργαστή που τρέχει σε μια εφαρμογή ενός MPEG-4 Encoder. Τα MPEG-4 encode και decode είναι δημοφιλή στις φορητές συσκευές και στο διαδίκτυο όπως και στις ψηφιακές τηλεοράσεις καθώς και στα video πέρα από τις IP εφαρμογές. Στην EEMBC υλοποιείται ο κώδικάς βάσης XviD με τροποποιήσεις για benchmarking και για ιδιόκτητα datasets.

Η υλοποίηση του XviD από την EEMBC χρησιμοποιεί απλό profile level 3 για να κωδικοποιήσει τα αρχεία. Αφού παραχθούν τα κωδικοποιημένα αχρεία από τα μη επεξεργασμένα YUV images, υπάρχει η δυνατότητα να δημιουργηθούν κωδικοποιημένα MPEG-4 αρχεία με διαφορετικά προφίλ, αν αυτό χρειαστεί.

Τα σύνολα δεδομένων που εισάγονται (YUV αρχεία) είναι τα ίδια με αυτά των MPEG-2 benchmarks της EEMBC.

Το benchmark περιλαμβάνει δύο διαφορετικές παραλλαγές. Μια υλοποίηση ακέραιου σταθερού σημείου και μια προαιρετική υλοποίηση μονής ακρίβειας κινητής υποδιαστολής.

Η εισαγωγή είναι μια σειρά από .PPM αρχεία μαζί με YUV. Η παραγωγή είναι

mp4u αρχεία τα οποία μπορούν να αποκωδικοποιηθούν από XviD ώστε να επαληθεύσουν ότι η κωδικοποίηση ήταν σωστή. Επαληθεύουμε την ορθότητα αυτού που παράχθηκε μετρώντας την ποιότητα με τη χρήση της PSNR ανάλυσης.

Επεξεργασία MPEG-4 Encoder benchmark:

- Ανάγνωση των επιλεγμένων πλαισίων YUV.
- Ανάγνωση και ερμηνεία των πληροφοριών επικεφαλίδας.
- Ανάγνωση και κωδικοποίηση των πλαισίων δεδομένων.
- Επεξεργασία των δεδομένων που βασίζονται στις πληροφορίες επικεφαλίδων.
- Έξοδος του αρχείου .mpeg στη μνήμη.
- Υπολογισμός μιας PSNR τιμής.

Μια μονή επανάληψη του benchmark ολοκληρώνεται όταν το τέλος του εισαγόμενου αρχείου επεκταθεί και δεν υπάρχουν άλλα διαθέσιμα δεδομένα για να υποστούν επεξεργασία. Η ποιότητα παραγωγής μετριέται χρησιμοποιώντας τον κώδικα μέγιστης αναλογίας σήματος/διαταραχής (PSNR). Η PSNR μετριέται έξω από το βρόχο συγχρονισμού του benchmark.

Υπάρχουν δύο υλοποιήσεις, fixed point και floating point. Η floating point έκδοση είναι προαιρετική. Το Benchmark εστιάζει περισσότερο στην υπολογιστική επεξεργασία παρά στα αρχεία εισόδου/εξόδου.Τα PSNR αποτελέσματα πρέπει να παρουσιαστούν σε report.

- <u>RGB to CMYK Conversion:</u>

Η εφαρμογή αυτή χρησιμοποιείται ευρέως στα printer με χρώματα. Τα RGB δεδομένα εισόδου από δεδομένα του υπολογιστή μετατρέπονται σε CMYK σήματα χρωμάτων για εκτύπωση.

Το benchmark εξετάζει την ικανότητα της ΚΜΕ να εκτελεί βασική αριθμητική και ελάχιστη ανίχνευση τιμών.

Οι RGB τιμές όπως και οι CMYK τιμές βρίσκονται μεταξύ των ορίων [0:255]. Τα δεδομένα εισαγωγής και εξαγωγής ποικίλουν. Για παράδειγμα τα δεδομένα της εικόνας 320x240 για το RGB CYMK αποθηκεύονται σειριακά ως εξής:

R[0], G[0], B[0], R[1], G[1], B[1], R[76799], G[76799], B[76799]

C[0], M[0], Y[0], K[0], C[1], M[1], Y[1], K[1], C[76799], M[76799], Y[76799], K[76799]

Οι δείκτες αυξάνονται κατά ένα για να προσπελάσουν τα R,G,B ή τα C,M,Y,K δεδομένα με αυτή τη σειρά. Αν το αποτέλεσμα του benchmark εξάγεται για μια μεγαλύτερη εικόνα, ο χρόνος επεξεργασίας είναι σχεδόν γραμμικά ανάλογος στη μέτρηση του pixel. Για παράδειγμα για μία 640x480 εικόνα, θα πολλαπλασιαστεί επί 4 φορές.

Ένα "for loop" υπολογίζει την μετατροπή ενός συνόλου από RGB δεδομένα εισόδου και CMYK δεδομένα εξόδου κάθε φορά.

Ένα σύνολο από R,G,B δεδομένα εισόδου διαβάζεται από τη μνήμη αυξάνοντας ένα δείκτη διαβάσματος. Ένα σύνολο από C,M,Y,K δεδομένα εξόδου γράφεται πίσω στην μνήμη αυξάνοντας ένα δείκτη γραψίματος.

- <u>RGB to YIQ Conversion:</u>

  Η εφαρμογή αυτή χρησιμοποιείται στους NTSC αποκωδικοποιητές όπου τα PGB δεδομένα εισόδου από την κάμερα μετατρέπονται για να φωτίσουν(Y) και να χρωματίσουν (I, Q) την πληροφορία. Στον αποκωδικοποιητή NTSC τα σήματα I,Q προσαρμόζονται από ένα υπομεταφορέα και προσθέτονται στο Y σήμα.

  Στα πραγματικά προϊόντα, αυτός ο συνηθισμένος υπολογισμός συνήθως εκτελείται σε ειδικό για αυτό το σκοπό υλικό, ειδικά στα προϊόντα ψηφιακού video. Για λιγότερο κόστος και ευκαμψία, αυτός ο αλγόριθμος μπορεί να υλοποιηθεί στο λογισμικό εάν η ΚΜΕ είναι αρκετά ισχυρή και η ψηφιακή εικόνα είναι μια στατική φωτογραφία.

Το benchmark εξετάζει την ικανότητα της ΚΜΕ να εκτελέσει ένα ακριβή υπολογισμό πολλαπλασιασμού/αύξησης. Οι RGB τιμές βρίσκονται μεταξύ των ορίων [0:255]. Τα δεδομένα εξαγωγής είναι μεγέθους 8-bit. Οι Y τιμές βρίσκονται μεταξύ των ορίων [0:255] και οι I,Q είναι μεταξύ των ορίων [-127:127]. Το μέγεθος των δεδομένων εισαγωγής και εξαγωγής είναι 320 pixels η οριζόντια κατεύθυνση και 240 η κάθετη κατεύθυνση. Τα δεδομένα της εικόνας 320x240 για το RGB και το YIQ αποθηκεύονται σειριακά ως εξής:

R[0], G[0], B[0], R[1], G[1], B[1], R[76799], G[76799], B[76799]

Y[0], I[0], Q[0], Y[1], I[1], Q[1], Y[76799], I[76799], Q[76799]

Οι δείκτες αυξάνονται κατά ένα για να προσπελάσουν τα R,G,B ή τα Y,I,Q δεδομένα με αυτή τη σειρά. Η ποιότητα παραγωγής μετριέται χρησιμοποιώντας Non Intrusive CRC κώδικα ο οποίος αναπτύχθηκε από την EEMBC. Και δεν επηρεάζει το αποτέλεσμα του benchmark.

Ένα "for loop" υπολογίζει την μετατροπή ενός συνόλου από RGB δεδομένα εισόδου και YIQ δεδομένα εξόδου κάθε φορά. Ένα σύνολο από R,G,B δεδομένα εισόδου διαβάζεται από τη μνήμη αυξάνοντας ένα δείκτη διαβάσματος. Ένα σύνολο από Y,I,Q δεδομένα εξόδου γράφεται πίσω στην μνήμη αυξάνοντας ένα δείκτη γραψίματος.

- <u>RSA:</u>

  Ο αλγόριθμος RSA περιγράφηκε για πρώτη φορά το 1977 από τους Ron Rivest, Adi Shamir, και Len Adleman στο MIT. Τα γράμματα RSA είναι τα αρχικά από τα επίθετα τους. Είναι ένας από τους πρώτους δυνατούς κρυπτογραφικούς αλγόριθμους δημόσιου κλειδιού. Χρησιμοποιείται για ψηφιακά σήματα και για κρυπτογράφηση. Η κρυπτογραφία RSA χρησιμοποιείται σε πολυάριθμα κρυπτογραφικά πρωτόκολλα όπως τα Transport Layer Security (TLS), Secure Socket Layer, (SSL), Secure Shell (SSH), and Internet Protocol Security (IPSEC).

Ο RSA είναι πολύ πιο αργός και γι' αυτό πιο εντατικά υπολογιστικός από τον DES αλγόριθμο. Επίσης αντίθετα με τον DES δεν είναι συμμετρικός. Συνεπώς, υπάρχουν διαφορετικά κλειδιά για κρυπτογράφηση και αποκρυπτογράφηση.

Το EEMBC RSA benchmark είναι ένας κρυπτογραφικός αλγόριθμος ο οποίος παρέχει μια ένδειξη της ενδεχόμενης επίδοσης ενός υποσυστήματος μικροεπεξεργαστή η επεξεργαστή ψηφιακού σήματος που κάνει κρυπτογραφικές αποκρύψεις και αποκρυπτογραφήσεις.

Ο EEMBC RSA benchmark χειρισμός των τελεστών ιδιωτικού κλειδιού δεν εξαρτάται από τα τμήματα ιδιωτικού κλειδιού που παρουσιάζονται. Ο συνιστάμενος αριθμός επαναλήψεων είναι 30, και παίρνει περίπου ένα δευτερόλεπτο να τρέξει σε ένα επιτραπέζιο x86 προσωπικό υπολογιστή των 1.7 GHz. Ο έλεγχος γίνεται με την χρήση του CRC.

Το benchmark αυτό είναι υπολογιστικά ενδιαφέρον αφού σε αυτό χρησιμοποιούνται πολλοί μαθηματικοί υπολογισμοί. Υλοποιείται σε ακέραια μαθηματικά. Αυτό το benchmark είναι σχεδόν αποκλειστικά όριο της KME και η ποιότητα της βιβλιοθήκης μαθηματικών, όπως και αυτή της βιβλιοθήκης μνήμης έχει συνέπεια στην επίδοση.

Κεφάλαιο 4

Μεθοδολογία ανάλυσης Benchmarks

4.1 Σχετική δουλειά

4.2 Πλαίσιο Ανάλυσης

4.3 Περιγραφή συστήματος που χρησιμοποίησα

4.4 Περιληπτική περιγραφή μεθοδολογίας μεταγλώττισης

4.5 Βηματική περιγραφή μεταγλώττισης

4.6 Εργαλείο VTune™ Performance Analyzer

4.7 Βηματική περιγραφή ανάλυσης benchmark στο VTune

4.1 Σχετική δουλειά

Οι συγγραφείς στο άρθρο "FacePerf: Benchmarks for Face Recognition Algorithms" [1] αναφέρονται στο FacePerf benchmark το οποίο είναι μια συλλογή από 3 αλγόριθμους αναγνώρισης προσώπων οι οποίοι δοκιμάστηκαν για να καλύψουν τα κύρια συστατικά των αυτόματων face recognition συστημάτων. Οι 3 συγκεκριμένοι αλγόριθμοι επιλέχτηκαν γιατί είναι σημαντικοί αλγόριθμοι στην κοινωνία βιομετρήσεων. Ο κάθε ένας από τους αλγόριθμους εκτελεί πολύ διαφορετικούς τύπους υπολογισμού. Και οι τρεις έχουν υλοποιήσεις ανοικτού κώδικα. Τα test scripts που συμπεριλαμβάνονται στο FacePref παρέχουν μια σταθερή μεθοδολογία ελέγχου τρέχοντας τους αλγόριθμους στα γνωστά face recognition datasets.

Το πρώτο βήμα του face recognition είναι το Face detection, όπου προσδιορίζεται που βρίσκεται ένα πρόσωπο στην εικόνα. Το δεύτερο βήμα είναι να υπολογιστεί η

ομοιότητα μεταξύ του προσώπου που εντοπίστηκε με ένα ή περισσότερα face image chips που βρίσκονται αποθηκευμένα σε μια βάση δεδομένων. Το FacePref περιλαμβάνει ένα face detection αλγόριθμο και 2 face identification αλγόριθμους. Οι αλγόριθμοι είναι OpenCV Cascade Classifier, CSU Principal Components Analysis, CSU Elastic Bunch Graph Maching.

Ο κώδικας των αλγορίθμων είχε μεταγλωττιστεί από τον μεταγλωττιστή gcc version –o3. Σύμφωνα με τους συγγραφείς η πιο απλή μέθοδος για αύξηση της επίδοσης είναι η ενεργοποίηση των αυτομάτων βελτιστοποιήσεων που παρέχει ο μεταγλωττιστής.

Με τη χρήση του εργαλείου gprof παρατήρησαν πως ένα μεγάλο μέρος του υπολογιστικού χρόνου ξοδευόταν για τον υπολογισμό ενός συνημίτονου, πράγμα το οποίο βελτίωσαν εν μέρει χρησιμοποιώντας είτε τις τιμές του συνημίτονου από -1 μέχρι 0, είτε τις τιμές του από 0 μέχρι 1.

Οι συγγραφείς στο άρθρο “Comparing Benchmarks Using Key Micro architecture Independent Characteristics” [2] κάνουν λόγο για τις αυξανόμενες απαιτήσεις και επιθυμίες των χρηστών υπολογιστών που οδηγούν τις εταιρείες να σχεδιάζουν νέες εφαρμογές. Αυτές οι εφαρμογές για να είναι αποδοτικές πρέπει να ελέγχονται με τη χρήση benchmarks. Συνήθως οι δημιουργοί χρησιμοποιούν benchmarks τα οποία αναπαριστούν το φόρτο εργασίας των πραγματικών εφαρμογών.

Έτσι συγκρίνουν benchmarks για να βρουν την αποδοτικότερη λύση για τις εφαρμογές τους. Μια μεθοδολογία χαρακτηρισμού των benchmarks είναι ο χαρακτηρισμός που βασίζεται στην μικροαρχιτεκτονική. Τα χαρακτηριστικά τα οποία χρησιμοποιούν στις μετρήσεις τους είναι τα Instruction mix, Instruction Level Parallelism, Register traffic characteristics, Working Set, Data stream strides και Branch predictability.

Μια μεθοδολογία χαρακτηρισμού των benchmarks είναι ο χαρακτηρισμός που βασίζεται στην μικροαρχιτεκτονική είναι το hardware performance counter characterization. Συλλέγουν το hardware performance counter τιμές για τα IPC,

Branch miss prediction rate, L1 D-Cache miss rate, L1 I-Cache miss rate, L2 cache miss rate, D-TLB miss rate.

4.2 Πλαίσιο ανάλυσης

Η μεθοδολογία που ακολούθησα για την ανάλυση κάθε ενός από τα benchmarks χωρίστηκε σε δύο μέρη. Την στατική και την δυναμική ανάλυση.

Στατική Ανάλυση:

- Μελέτη και περιγραφή του benchmark χρησιμοποιώντας το documentation της EEMBC.
- Μελέτη συναρτήσεων του benchmark και δημιουργία διαγράμματος ροής δεδομένων όπου παρουσιάζονται οι κλήσεις συναρτήσεων.
- Εύρεση των κύριων δομών δεδομένων του αλγόριθμου.
- Εύρεση πολυπλοκότητας αλγόριθμου.
- Με βάση τα πιο πάνω υπολόγιζα ποιο κομμάτι κώδικα μπορεί να είναι το hotspot.

Δυναμική ανάλυση:

- Μεταγλώττιση benchmark στο εργαλείο Cygwin.
- Εκτέλεση του benchmark στο Cygwin.
- Δημιουργία πίνακα με τα αποτελέσματα που πήρα για το σύνολο της εφαρμογής.
- Δημιουργία γραφικών παραστάσεων χρησιμοποιώντας τον προηγούμενο πίνακα.
- Ανάλυση του benchmark στο VTune. Έδινα ως είσοδο 1000 επαναλήψεις.

- Δημιουργία πίνακα με τα αποτελέσματα που πήρα από το VTune.
- Δημιουργία γραφικών παραστάσεων χρησιμοποιώντας τον προηγούμενο πίνακα.
- Εύρεση της συνάρτησης με το hotspot. Ήταν η συνάρτηση που κατανάλωνε το περισσότερο χρόνο εκτέλεσης.
- Άνοιγμα της συνάρτησης μέσω του VTune.
- Εύρεση των γραμμών κώδικα της συνάρτησης που κατανάλωναν το πιο μεγάλο ποσοστό χρόνου εκτέλεσης.
- Έλεγχος και στις άλλες συναρτήσεις για επιβεβαίωση ότι δεν υπάρχει άλλο σημείο κώδικα σε άλλες συναρτήσεις που να καταναλώνει μεγαλύτερο ποσοστό χρόνου εκτέλεσης.
- Μετά την επιβεβαίωση για το πιο είναι το hotspot κοίταζα τον κώδικα.
- Έλεγχος Hotspot κώδικα:
    - μέσα σε πόσα φωλιασμένα loops βρισκεται.
    - Τα είδη των πράξεων/υπολογισμών που εκτελούσε, προσβάσεις στη μνήμη, shiftings, andings κλπ.
    - Αν ήταν μια ολόκληρη συνάρτηση κοίταζα το διάγραμμα που πήρα από την στατική ανάλυση για να δω πόσες φορές γινόταν κλήση της συγκεκριμένης συνάρτησης.
- Με βάση τα πιο πάνω αιτιολογούσα το γιατί το συγκεκριμένο κομμάτι κώδικα κατανάλωνε τον περισσότερο χρόνο εκτέλεσης.
- Τέλος παρέθετα τις παρατηρήσεις/συμπεράσματα μου.

4.3 Περιγραφή συστήματος που χρησιμοποίησα

Για να μεταγλωττίσω, να τρέξω και να αναλύσω τα benchmarks χρησιμοποίησα τον προσωπικό μου υπολογιστή. Τα χαρακτηριστικά του συστήματος μου ήταν:

Επεξεργαστής: Intel(R) Core(TM)2 Duo CPU T8100 @ 2.10 GHz

Memory (Ram): 2046 MB

System type: 32-bit Operating System

4.4 Περιληπτική περιγραφή μεθοδολογίας

Για να επιτευχθεί η ανάλυση και η εύρεση του hotspot του κώδικα των benchmarks χρειάζεται η εγκατάσταση των προγραμμάτων Cygwin και Intel VTune Performance Analyzer. Το Cygwin χρειάζεται για την μεταγλώττιση των benchmarks ώστε να παραχθεί το εκτελέσιμο αρχείο (.exe) το οποίο στη συνέχεια θα περαστεί ως είσοδος στον Intel VTune Performance Analyzer για να γίνει η ανάλυση του benchmark. Χρησιμοποιούμε το Cygwin γιατί παράγει .exe αρχεία τα οποία παίρνει σαν είσοδο το VTune και ο gcc είναι ο προεπιλεγμένος μεταγλωττιστής των benchmarks.

4.5 Βηματική περιγραφή μεταγλώττισης

1. Κατέβασμα και εγκατάσταση του τελευταίου version του Cygwin [6] που να είναι συμβατό με το λειτουργικό μας σύστημα. Επιλέγουμε την εγκατάσταση των gcc, gdb, perl, gawk, make, zip, unzip.
2. Κατέβασμα του τελευταίου version του cygwin1.dll και εγκατάταση του στον φακελο ~/cygwin/bin(όπου ~ είναι το αρχείο στο οποίο έχουμε αποθηκεύσει το cygwin), επίσης εγκαθιστούμε το ίδιο cygwin1.dll στον φάκελο C/Windows/System32.
3. Αποσυμπιέζουμε τα benchmarks και όλα τα zip files τα οποία υπάρχουν στους νέους φακέλους.
4. Αλλάζουμε τα makefiles του benchmark ώστε η μεταγλώττιση να γίνει με debugging και το εκτελέσιμο να έχει τις πληροφορίες αρίθμησης γραμμών του

κώδικα που χρειάζεται το VTune. Στα makefiles τα οποία έχουν επιλογή compiler CCC (Compiler for Benchmarking) βάζουμε CCD (Compiler for Debug). Δηλαδή:

Select Compile for Benchmarking, or Compile for Debug

COM = $(CCC)

Το αλλάζουμε σε: COM = $(CCD).

5. Παίρνουμε το zip file eembc-2.0R2 από το αρχείο EEMBC/Test Harness το βάζουμε στο αρχείο EEMBC/Consumer/ DENBench 1.0 και το αποσυμπιέζουμε.
6. Πάμε στο Cygwin και μεταφερόμαστε στο αρχείο EEMBC/Consumer/ DENBench 1.0 και μεταγλωττίζουμε με τη χρήση της εντολής make all.
7. Αν όλα πήγαν σωστά στο αρχείο EEMBC/Consumer/ DENBench 1.0 δημιουργήθηκε φάκελος με το όνομα gcc ο οποίος περιλαμβάνει τα αποτελέσματα της μεταγλώττισης. Στον φάκελο EEMBC/Consumer/ DENBench 1.0/gcc/bin υπάρχουν τα εκτελέσιμα αρχεία τα οποία θα χρησιμοποιήσουμε στο εργαλείο Intel VTune Performance Analyzer.

4.6 Εργαλείο VTune™ Performance Analyzer [8]

Το εργαλείο VTune παρέχει πληροφορίες για την επίδοση του κώδικα μας και μας βοηθά στο να αναλύσουμε και να χαρακτηρίσουμε την επίδοση ενός προγράμματος. Μέσω του εργαλείου αυτού συλλέγονται δεδομένα επίδοσης από το σύστημα που τρέχει το πρόγραμμα. Μετρούνται τα clock cycles ενός προγράμματος, τα branches, τα branch mispredictions, το CPI κλπ. Οργανώνει και παρουσιάζει τα δεδομένα με ποικίλες αλληλεπιδρούσες όψεις, προσδιορίζει την ενδεχόμενη επίδοση και εισηγείται βελτιώσεις.

Χρησιμοποιώντας το VTune(TM) Performance Analyzer βρίσκουμε τις περιοχές του ενεργού κώδικα, ή το που σπαταλά ο επεξεργαστής ένα σημαντικό ποσοστό χρόνου. Η λειτουργία με το υψηλότερο ποσοστό χρόνου εκτέλεσης περιλαμβάνει την περιοχή κώδικα στην οποία ο επεξεργαστής σπαταλά τον περισσότερο χρόνο.

Παραδείγματα από hotspots είναι: ένα μεγάλο loop μέσα στο οποίο η λειτουργία εκτελεί πολλές πράξεις, ή μια λειτουργία η οποία καλείται πολλές φορές από μία άλλη λειτουργία υψηλότερου επιπέδου(level).

Παραδείγματα από πιθανές βελτιστοποιήσεις είναι: μείωση των πράξεων που χρειάζονται πολλούς κύκλους για να εκτελεστούν, όπως διάβασμα δεδομένων από την κύρια μνήμη, ή μείωση των branch mispredictions.

Αν το hotspot έχει πρόσβαση σε δομές δεδομένων, πρέπει να μαζέψουμε δείγματα δεδομένων από ποικίλα γεγονότα μνήμης (memory events) για να διευκρινίσουμε εάν ο επεξεργαστής χρησιμοποιεί αποτελεσματικά τη μνήμη. Αυτό περιλαμβάνει πότε ο κώδικας προσπελάζει τη μνήμη με τέτοιο τρόπο ώστε το hardware prefetcher μπορεί αποτελεσματικά να προβλέψει τις ανάγκες του κώδικα μας σε δεδομένα.

4.7 Βηματική περιγραφή ανάλυσης benchmark στο VTune

1. Στο εργαλείο Intel VTune Performance Analyzer επιλέγουμε New Project, από το παράθυρο που θα εμφανιστεί επιλέγουμε Quick Performance Analysis Wizard και βάζουμε το όνομα του project μας στο Project Name. Επιλέγουμε OK.
2. Στο νέο παράθυρο που εμφανίζεται επιλέγουμε το browse(…) στο Application to Launch και βρίσκουμε το εκτελέσιμο αρχείο του προγράμματος που θέλουμε να αναλύσουμε, πατούμε Open και μετά Go.
3. Το πρόγραμμα μας ξεκινά την εκτέλεσή του. Δίνουμε τις εισόδους που επιθυμούμε. Δίνοντας ως είσοδο την εντολή help μας δίνει όλες τις εντολές τις οποίες μπορούμε να δώσουμε ως είσοδο. Δύο σημαντικές εντολές είναι το n (iterations) όπου το iterations είναι το πλήθος των επαναλήψεων που θέλουμε να εκτελεστεί το πρόγραμμα μας. Και η εντολή g η οποία δίνει το έναυσμα στο πρόγραμμα να ξεκινήσει την εκτέλεση του.
4. Αφού εκτελεστεί το πρόγραμμα, παίρνουμε τα αποτελέσματα μας. Για να δούμε πόσο ποσοστό χρόνου σπαταλήθηκε ανά συνάρτηση, επιλέγουμε το

CPU_CLK_UNHALTED.CORE στο Tuning Browser που βρίσκεται στα αριστερά την οθόνης.

5. Για να δούμε το hotspot και σε ποιες ακριβώς γραμμές κώδικα καταναλώνεται ο χρόνος εκτέλεσης στο Tuning Browser που βρίσκεται στα αριστερά την οθόνης επιλέγουμε το Run1 και επιλέγουμε στο Process View που εμφανίζεται το όνομα του εκτελέσιμου αρχείου που τρέξαμε. Το ίδιο κάνουμε και στο Threads View, Modules View και φτάνουμε στο Hotspot View.
6. Στο Hotspot View η συνάρτηση με το περισσότερο ποσοστό κατανάλωσης χρόνου εκτέλεσης, και συνήθως αυτή που περιέχει το hotspot, βρίσκεται πρώτη στη σειρά. Επιλέγουμε την συνάρτηση αυτή.
7. Στο Map Source File παράθυρο που εμφανίζεται πατούμε το Browse(…) και επιλέγουμε το πρόγραμμα που αναλύουμε και ακολούθως το OK.
8. Τώρα στην οθόνη μας εμφανίζεται ο κώδικας αριθμημένος και στα δεξιά και κάτω τα μετρικά που θέλουμε να μετρήσουμε και να αναλύσουμε.
9. Αν θέλουμε να αλλάξουμε ή να προσθέσουμε νέα μετρικά τότε στο Tuning Browser που βρίσκεται στα αριστερά την οθόνης επιλέγουμε το Activity(Sampling) και ανάλογα με την επιθυμία μας επιλέγουμε αν θέλουμε να αλλάξουμε το ήδη υπάρχον αρχείο ή αν θέλουμε να κάνουμε ένα copy του και να αλλάξουμε αυτό.
10. Στο παράθυρο που εμφανίζεται επιλέγουμε Configure και μετά Event Ratios και επιλέγουμε τα μετρικά που θέλουμε.

Κεφάλαιο 5

Ανάλυση κώδικα Benchmarks

5.1 Εισαγωγή στην ανάλυση των EEMBC DENBench benchmarks
5.2 Ανάλυση RGB to CMYK Conversion benchmark
5.3 Ανάλυση Huffman Decoding Benchmark
5.4 Ανάλυση RGB to YIQ Conversion benchmark
5.5 Ανάλυση AES Benchmark

5.1 Εισαγωγή στην ανάλυση των EEMBC DENBench benchmarks

Στην συνολική ανάλυση μου ανέλυσα τέσσερα από τα benchmark της ακολουθίας των Digital Entertainment benchmarks.

Ανέλυσα δύο από τα benchmarks που υπάγονται στην κατηγορία της επεξεργασίας εικόνας και δύο benchmarks που υπάγονται στην κατηγορία κρυπτογράφησης/αποκρυπτογράφησης.

Πρόκειται για τα RGB to CMYK conversion benchmark, RGB to YIQ conversion benchmark Huffman Decoding Benchmark, AES benchmark.

Τα Huffman Decoding Benchmark και AES benchmark τα ανέλυσα ως προς τις συναρτήσεις των εφαρμογών χρησιμοποιώντας ένα input αρχείο.

Το CMYK conversion benchmark και το RGB to YIQ conversion benchmark τα ανέλυσα την κύρια συνάρτηση η οποία εκτελούσε όλη την δουλειά του benchmark και κατανάλωνε σχεδόν όλο το ποσοστό του χρόνου εκτέλεσης.

Τα δύο benchmarks της επεξεργασίας εικόνας τα ανέλυσα χρησιμοποιώντας περισσότερα από ένα αρχεία εισαγωγής, αφού η πολυπλοκότητα του προγράμματος όπως προέκυψε από την στατική μου ανάλυση εξαρτιόταν από το μέγεθος του αρχείου.

Τα αρχεία που χρησιμοποίησα σαν εισόδους στα CMYK conversion benchmark και το RGB to YIQ conversion benchmark είναι:

Σχήμα 5.1 Mandrake

Είναι μια κοντινή φωτογραφία ενός ζώου το οποίο ονομάζεται Mandrill Baboon. Ως εικόνα έχει πολλές λεπτομέρειες και χρώματα. Είναι το προκαθορισμένο image για τα filter benchmarks και με χρώματα και ασπρόμαυρη. Είναι ένα απλό image το οποίο περιλαμβάνεται και στα BMP, PPM, PGM, και
JPEG formats. Οι διαστάσεις του είναι 320x240. Το image έχει 71,482 μοναδικά χρώματα.

Σχήμα 5.2 Goose

Είναι ένα απλό image το οποίο περιλαμβάνεται και σε BMP και σε JPEG formats. Οι διαστάσεις του είναι 320x240. Το image έχει 22,921 μοναδικά χρώματα.

Σχήμα 5.3 EEMBC Group Shot

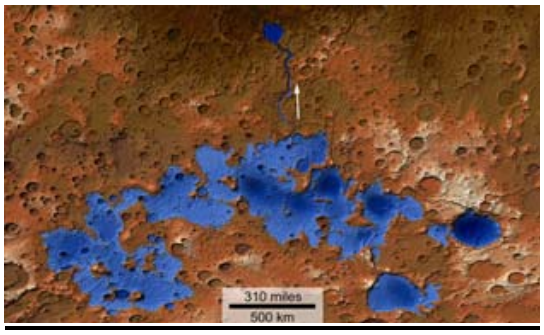
EEMBC Group Shot είναι ένα φωτογραφικό στιγμιότυπο από τα EEMBC Board of Directors μέλη σε μια σύσκεψη του 2003. Έχει ένα μεγάλο αριθμό από τόνους που κυμαίνονται στο χρώμα της σάρκας(flesh tones) και τον μεγαλύτερο αριθμό μοναδικών χρωμάτων στη βιβλιοθήκη. Είναι ένα απλό image το οποίο περιλαμβάνεται σε BMP, PPM, , PGM, και JPEG formats. Οι διαστάσεις του είναι 640x480. Το image έχει 181,872 μοναδικά χρώματα.

Σχήμα 5.4 David and Dogs

Είναι ένα φωτογραφικό στιγμιότυπο του David Weiss και των σκύλων του Sandy, Toga, και Trudy κατά τη διάρκεια μιας χιονοθύελλας στο Austin. Χρησιμοποιείται σαν ένα ασπρόμαυρο (grayscale) image, με πολλές λεπτομέρειες contrast στο λιωμένο χιόνι. Είναι ένα απλό image το οποίο περιλαμβάνεται στα BMP, PPM, PGM, και JPEG formats. Οι διαστάσεις του είναι 564x230. Το image έχει 215 μοναδικά χρώματα.

Σχήμα 5.5 Dragon Fly

Είναι ένα image που περιλαμβάνει highlights, και μία ευρεία έκταση από contrast. Είναι ένα απλό image το οποίο περιλαμβάνεται σε
BMP, PPM, PGM, και JPEG formats. Οι διαστάσεις είναι 606x896. Το image έχει 162,331 μοναδικά χρώματα.



Σχήμα 6.6 Mars Former Lakes

Είναι μια εικόνα γραφικών της NASA. Είναι ένα απλό image το οποίο περιλαμβάνεται σε BMP, PPM, PGM, και JPEG formats. Οι διαστάσεις του είναι 800x482, των 16 εκατομμυρίων χρωμάτων. Το image έχει 91,152 μοναδικά χρώματα.

Σχήμα 5.7 Galileo

Είναι ένα σύνθετο image της NASA το οποίο βασίζεταιcomposite σε πραγματικά images του πλανήτη Δία και αρκετών από τα φεγγάρια του. Είναι ένα απλό image το οποίο περιλαμβάνεται στα BMP, PPM, PGM, και

JPEG formats. Οι διαστάσεις του είναι 290x415, των 16 εκατομμυρίων χρωμάτων. Το image έχει 36,557 μοναδικά χρώματα, και επίσης περιλαμβάνει “real black” σε περισσότερο από το 30% της εικόνας.

5.2 Ανάλυση RGB to CMYK Conversion benchmark

Η εφαρμογή RGB to CMYK conversion, χρησιμοποιείται ευρέως στα printer με χρώματα. Τα RGB δεδομένα εισόδου από δεδομένα του υπολογιστή μετατρέπονται σε CMYK σήματα χρωμάτων για εκτύπωση.

Το benchmark εξετάζει την ικανότητα της KME να εκτελεί βασική αριθμητική και ελάχιστη ανίχνευση τιμών.

Οι RGB τιμές όπως και οι CMYK τιμές βρίσκονται μεταξύ των ορίων [0:255]. Τα δεδομένα εισαγωγής και εξαγωγής ποικίλουν. Για παράδειγμα τα δεδομένα της εικόνας 320x240 για το RGB CYMK αποθηκεύονται σειριακά ως εξής:

R[0], G[0], B[0], R[1], G[1], B[1], R[76799], G[76799], B[76799]

C[0], M[0], Y[0], K[0], C[1], M[1], Y[1], K[1], C[76799], M[76799], Y[76799], K[76799]

Οι δείκτες αυξάνονται κατά ένα για να προσπελάσουν τα R,G,B ή τα C,M,Y,K δεδομένα με αυτή τη σειρά. Ένα "for loop" υπολογίζει την μετατροπή ενός συνόλου από RGB δεδομένα εισόδου και CMYK δεδομένα εξόδου κάθε φορά.

Ένα σύνολο από R,G,B δεδομένα εισόδου διαβάζεται από τη μνήμη αυξάνοντας ένα δείκτη διαβάσματος. Ένα σύνολο από C,M,Y,K δεδομένα εξόδου γράφεται πίσω στην μνήμη αυξάνοντας ένα δείκτη γραψίματος.

Τα αρχεία εισόδου και εξόδου του benchmark:
Τα αρχεία εισόδου του benchmark έχουν περιγραφεί στο υποκεφάλαιο 6.1. Αφού γίνει η μετατροπή, παίρνουμε τα CMYK images τα οποία μπορούν να εισαχθούν στο printer ώστε να εκτυπωθούν.

Πολυπλοκότητα Benchmark:
Η πολυπλοκότητα του RGB to CMYK conversion Benchmark εξαρτάται από τον αριθμό των επαναλήψεων που επιλέγουμε να τρέξει το benchmark μέσω του command

line και από την τιμή της μεταβλητής Pels. Η μεταβλητή Pels είναι το γινόμενο του πλατους επί το ύψος του RGB image που προσπαθουμε να μετατρέψουμε σε CMYK. Στην κύρια συνάρτηση του benchmark υπάρχουν 2 κύρια Loops ένα που εκτελείται με βάση τα iterations και ένα που εκτελείται με βάση τα Pels. Τελικά η πολυπλοκότητα του benchmark είναι Θ(iterations*width*height).

Ανάλυση χρόνου εκτέλεσης και των εντολών του συνολικού προγράμματος ανά dataset:
Μετά από την εκτέλεση των 7 εκτελέσιμων αρχείων που πήρα από την μεταγλώττιση του RGB to CMYK conversion benchmark προέκυψαν κάποια αποτελέσματα που αφορούν την εκτέλεση του benchmark.

Στον Πίνακα 5.1 μπορούμε να δούμε τον χρόνο που πήρε στο benchmark για να εκτελέσει τη λειτουργία του, πόσες επαναλήψεις του benchmark έτρεξαν ανά δευτερόλεπτο και πόσο χρόνο κατανάλωσε κάθε iteration για να ολοκληρωθεί, ανά dataset.

	Total Run Time (sec)	Iterations/second	Time/iterations (sec)
Mandrake.ppm	8.361	119.6	0.008361
Goose.ppm	8.361	119.6	0.008361
EEMBCGroupShotMiami.ppm	30.623	32.6	0.030623
DavidAndDogs.ppm	17.691	56.5	0.017691
DragonFly.ppm	57.252	17.4	0.057252
MarsFormerLakes.ppm	35.411	28.2	0.035411
Galileo.ppm	11.202	89.2	0.011202

Πίνακας 5.1 Αποτελέσματα εκτέλεσης στο Cygwin ανά dataset.

Στο εργαλείο VTune έκανα Quick Performance Analysis των εκτελέσιμων αρχείων. Όταν έτρεξαν τα εκτελέσιμα έδωσα ως είσοδο την επιλογή να εκτελεστούν 1000 iterations του benchmark ώστε τα αποτελέσματα να είναι περισσότερο αντιπροσωπευτικά. Μετά το πέρας της εκτέλεσης των αρχείων προέκυψαν τα αποτελέσματα των μετρικών τα οποία επέλεξα να μετρήσει το VTune.

Από το VTune προέκυψε ότι όποιο dataset και αν δώσουμε ως είσοδο στο Benchmark RGB to CMYK conversion, η συνάρτηση t_run_test() είναι αυτή που καταναλώνει το περισσότερο χρόνο για να εκτελεστεί, η διαφορά από τις υπόλοιπες συναρτήσεις του benchmark είναι πάρα πολύ μεγάλη. Το ποσοστό του χρόνου εκτέλεσης της συνάρτησης κυμαίνεται γύρω από το 94% και οι υπόλοιπες συναρτήσεις καταναλώνουν όλες μαζί μόλις το 6%. Ως εκ τούτου αναμένω ότι το hotspot του benchmark θα βρίσκεται στη συνάρτηση t_run_test().

Μπαίνοντας στον κώδικα του Benchmark και ελέγχοντας τα ποσοστά χρόνου που σπαταλήθηκαν στα διάφορα σημεία του κώδικα όντως το hotspot βρισκόταν στην συνάρτηση t_run_test(). Είναι ένα κομμάτι κώδικα σε ένα τριπλά φωλιασμένο loop.Στον Πίνακα 6.2 μπορούμε να δούμε κάποια μετρικά του VTune για τη συνάρτηση t_run_test() του benchmark για κάθε ένα από τα datasets που δόθηκαν σαν είσοδος. Πρόκειται για το ποσοστό του χρόνου της συνολικής εφαρμογής που κατανάλωσε η συνάρτηση, το πραγματικό αριθμό των clock cycles που χρειάστηκαν για να τρέξει η κάθε εφαρμογή για 1000 επαναλήψεις, το ποσοστό των εντολών που ολοκληρώθηκαν κατά τη διάρκεια εκτέλεσης του benchmark, τον πραγματικό αριθμό των εντολών που ολοκληρώθηκαν κατά τη διάρκεια εκτέλεσης του benchmark και τέλος το CPI της συνάρτησης για κάθε dataset.

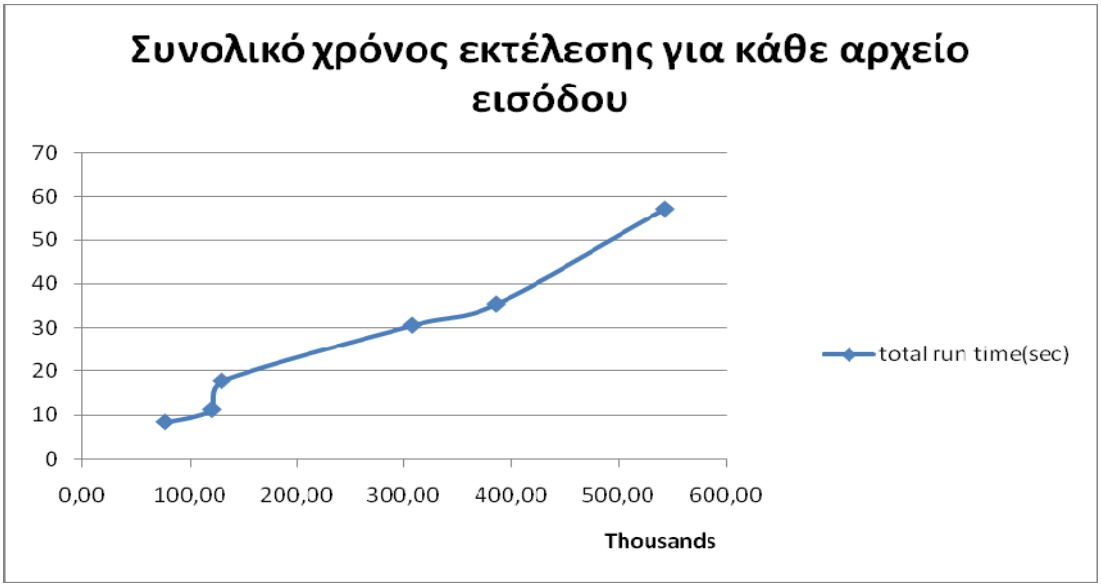
	Μετρικά				
	Ποσοστό χρόνου εκτέλεσης	Clock cycles	Ποσοστό εντολών	Instructions Retired	CPI
Mandrake.ppm	94,76%	5.981 $x10^6$	97,59%	6.119$x10^6$	0,977
Goose.ppm	95,10%	5.991 $x10^6$	97,73%	6.161 $x10^6$	0,972
EEMBCGroupShotMiami.ppm	94,87%	22.308 $x10^6$	97,64%	24.551 $x10^6$	0,907
DavidAndDogs.ppm	94,64%	12.671 $x10^6$	97,70%	14.549 $x10^6$	0,871
DragonFly.ppm	95,19%	42.187 $x10^6$	97,69%	43.455 $x10^6$	0,971
MarsFormerLakes.ppm	94,45%	26.260 $x10^6$	97,67%	31.118 $x10^6$	0,844
Galileo.ppm	94,83%	8.435 $x10^6$	97,58%	9.578 $x10^6$	0,881

Πίνακας 5.2 Αποτελέσματα εκτέλεσης των datasets στο VTune Performance Analyzer.

Από τις μετρήσεις των δύο πινάκων πήρα κάποιες γραφικές παραστάσεις ώστε να συγκρίνω τα αποτελέσματα που πήρα λαμβάνοντας υπόψη και τα χαρακτηριστικά κάθε dataset. Οι γραφικές παραστάσεις που ακολουθουν περιλαμβάνουν στον άξονα των Y τα μετρικά και στον άξονα των X τα μεγέθη των εικόνων. Στον πινακα πιο κάτω γίνεται η αντιστοιχηση αρχείου εισόδου και μεγεθους εικόνας σε pixels.

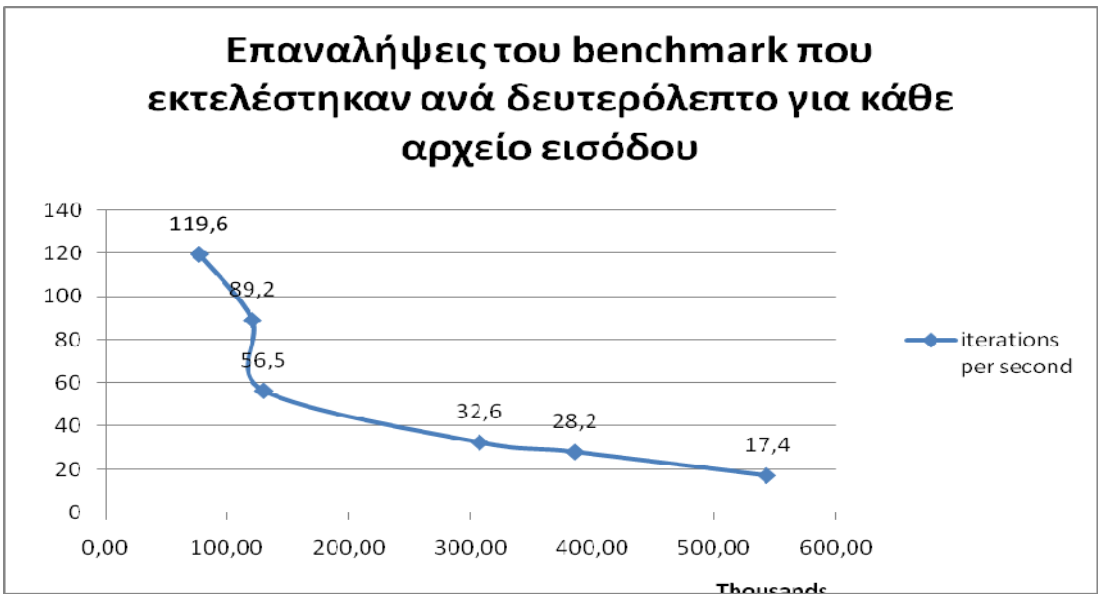
	Διαστάσεις	**Pixels**
Mandrake.ppm	320x240	76800
Goose.ppm	320x240	76800
EEMBCGroupShotMiami.ppm	640x480	307200
DavidAndDogs.ppm	564x230	129720
DragonFly.ppm	606x896	542976
MarsFormerLakes.ppm	800x482	385600
Galileo.ppm	290x415	120350

Πίνακας 5.3 Αντιστοίχηση αρχείου εισόδου-μεγεθους σε pixels



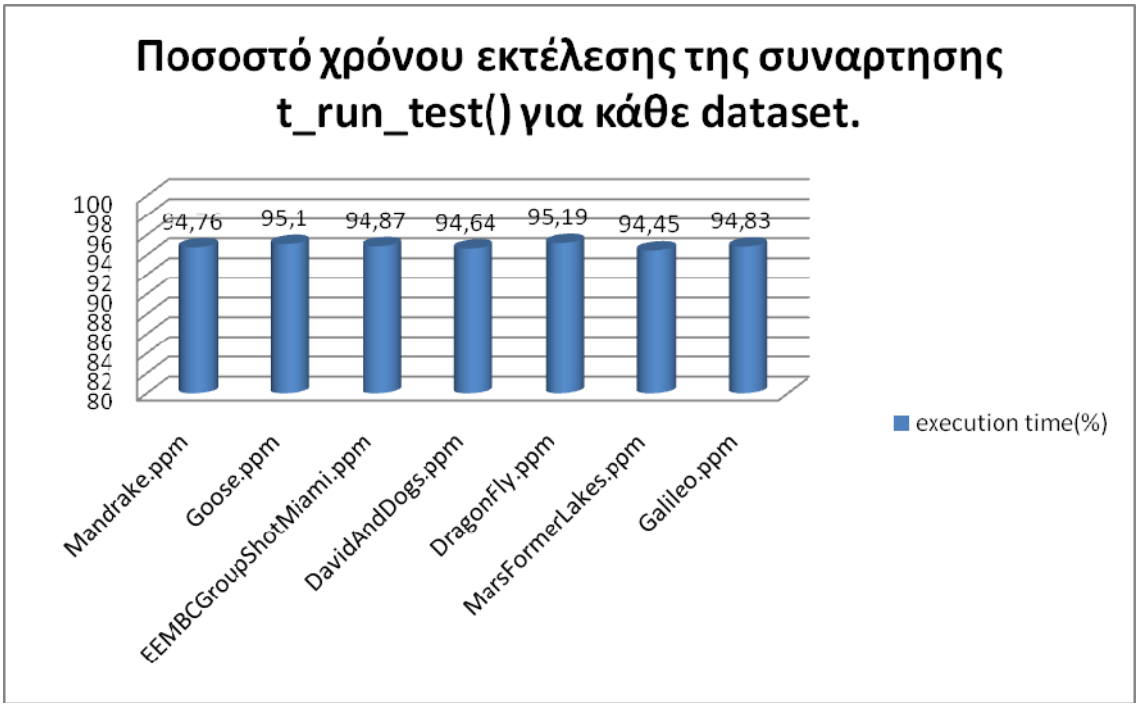
Σχήμα 5.8

Παρατηρούμε ότι στην γραφική παράσταση του συνολικού χρόνου εκτέλεσης του benchmark, για κάθε διαφορετικό dataset που δίναμε ως είσοδο ο χρόνος εκτέλεσης είναι διαφορετικός. Το λιγότερο χρόνο εκτέλεσης έχουν τα datasets Mandrake.ppm και Goose.ppm που συμπίπτουν, ακολουθεί το Galileo.ppm, το DavidAndDogs.ppm, το EEMBCGroupShotMiami.ppm, το MarsFormerLakes.ppm και τον περισσότερο χρόνο εκτέλεσης έχει το DragonFly.ppm.



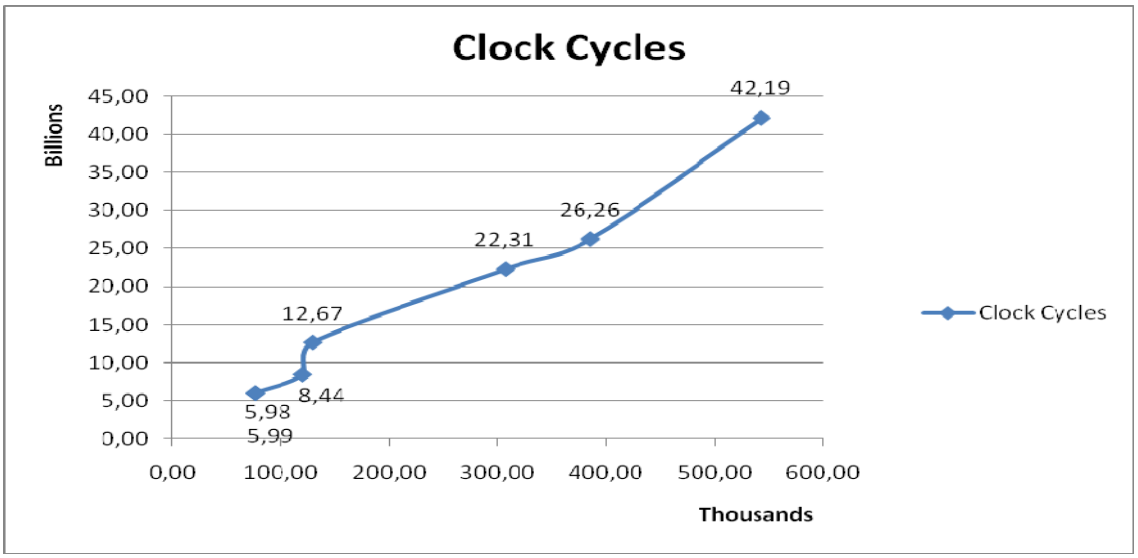
Σχήμα 5.9

Παρατηρούμε ότι στην γραφική παράσταση των επαναλήψεων του benchmark που εκτελέστηκαν ανά δευτερόλεπτο, για κάθε διαφορετικό dataset που δίναμε ως είσοδο ο αριθμός των iterations ήταν διαφορετικός. Τα benchmark όταν έτρεχε με είσοδο τα datasets Mandrake.ppm και Goose.ppm εκτελούσε 119.6 επαναλήψεις ανά δευτερόλεπτο, ακολουθεί το Galileo.ppm με 89.2 , το DavidAndDogs.ppm με 56.5, το EEMBCGroupShotMiami.ppm με 32.6, το MarsFormerLakes.ppm με 28.2 και τέλος το DragonFly.ppm το οποίο εκτελεί τις λιγότερες επαναλήψεις ανά δευτερόλεπτο, μόλις 17.4.



Σχήμα 5.10

Παρατηρούμε ότι το ποσοστό χρόνου εκτέλεσης που σπαταλήθηκε για την εκτέλεση της συνάρτησης t_run_test κυμαίνεται στα ίδια επίπεδα για όλα τα datasets. Κατά συνέπεια μπορούμε να σκεφτούμε ότι το hotspot του benchmark βρίσκεται στο ίδιο σημείο για κάθε dataset.

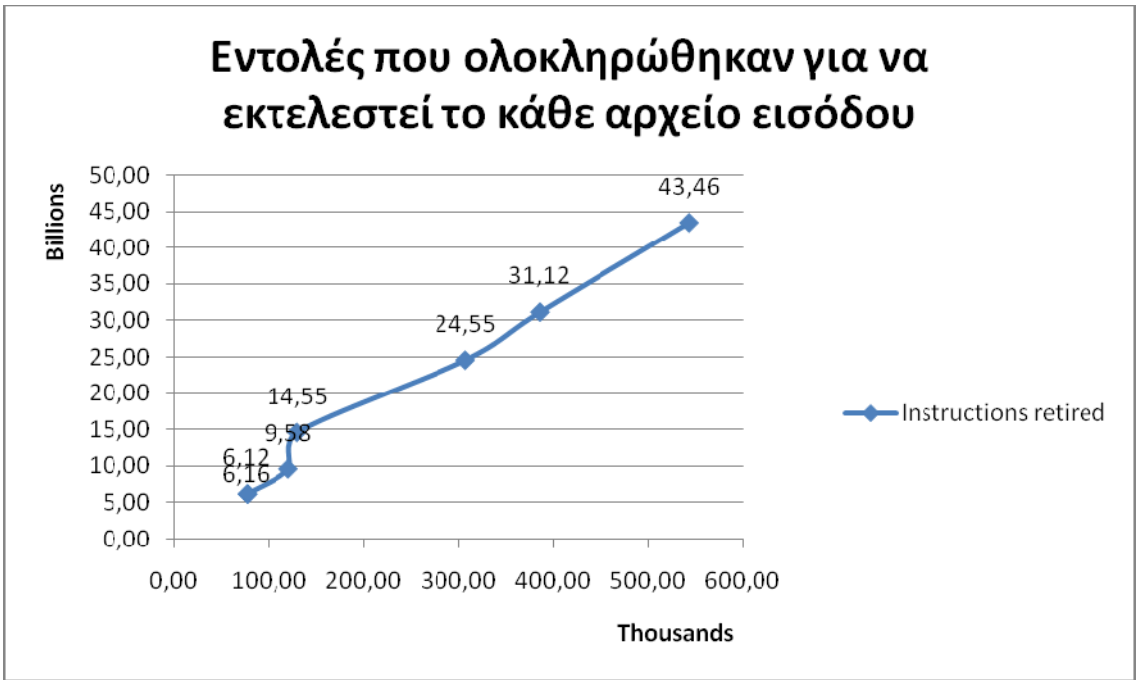


Σχήματα 5.11

Παρατηρούμε ότι στην γραφική παράσταση των clock cycles που καταναλώθηκαν ώστε να εκτελεστεί η συνάρτηση t_run_test() του benchmark, για κάθε διαφορετικό dataset που δίναμε ως είσοδο ο αριθμός των clock cycles ήταν διαφορετικός. Το benchmark όταν έτρεχε με είσοδο τα datasets Mandrake.ppm και Goose.ppm κατανάλωνε τα λιγότερα clock cycles στην γραφική παράσταση βλέπουμε ότι συμπίπτουν τα αποτελέσματα των δύο, ακολουθεί το Galileo.ppm, το DavidAndDogs.ppm, το EEMBCGroupShotMiami.ppm, το MarsFormerLakes.ppm και τέλος το DragonFly.ppm το οποίο χρειάστηκε τα περισσότερα clock cycles.

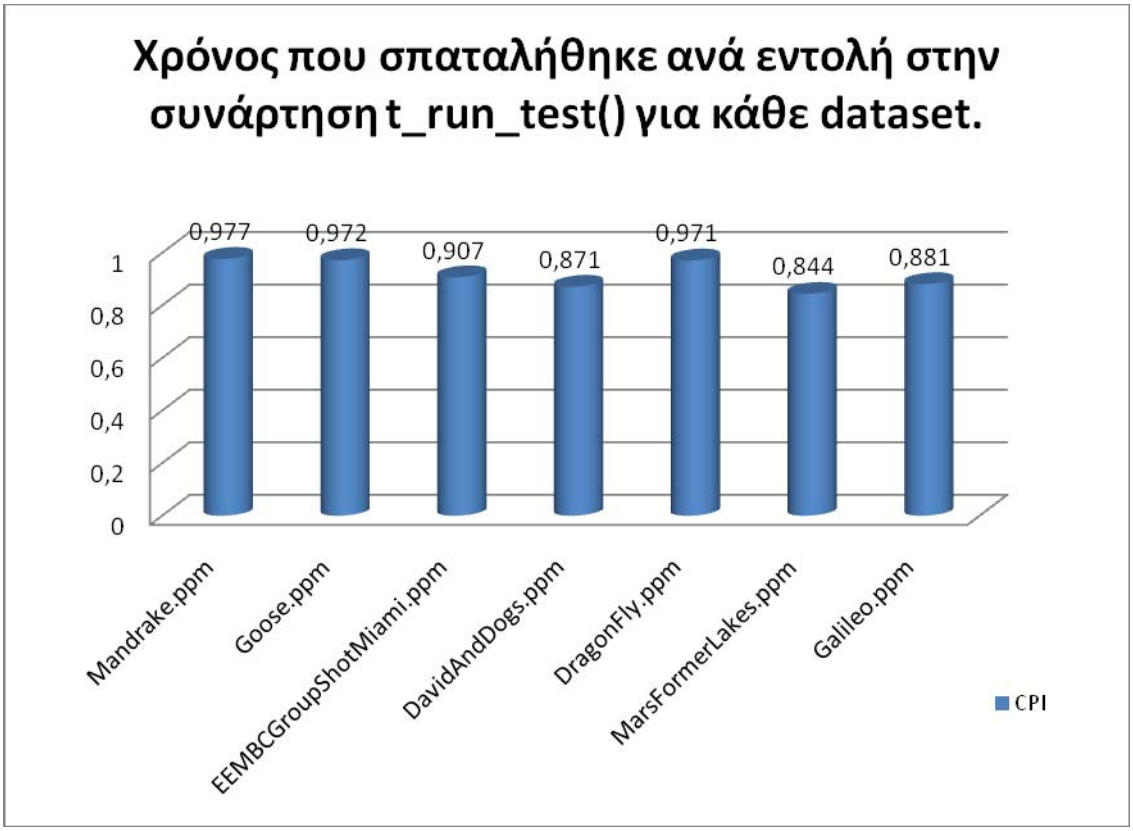
Σχήμα 5.12

Παρατηρούμε ότι το ποσοστό των εντολών που ολοκληρώθηκαν κατά την εκτέλεση της συνάρτησης t_run_test κυμαίνεται στα ίδια επίπεδα για όλα τα datasets.



Σχήμα 5.13

Παρατηρούμε ότι ο αριθμός των εντολών που ολοκληρώθηκαν κατά την εκτέλεση της συνάρτησης t_run_test(), για κάθε διαφορετικό dataset που δίναμε ως είσοδο ο αριθμός των instructions retired ήταν διαφορετικός. Το benchmark όταν έτρεχε με είσοδο τα datasets Mandrake.ppm και Goose.ppm χρειάστηκε να ολοκληρώσει λιγότερες εντολές, ακολουθεί το Galileo.ppm, το DavidAndDogs.ppm, το EEMBCGroupShotMiami.ppm, το MarsFormerLakes.ppm και τέλος το DragonFly.ppm το οποίο χρειάστηκε να ολοκληρώσει τις περισσότερες εντολές.



Σχήμα 5.14

Παρατηρούμε ότι στα EEMBCGroupShotMiami.ppm, DavidAndDogs.ppm, MarsFormerLakes.ppm Galileo.ppm το CPI έχει διαφορά από τα CPIs των υπολοίπων τριών benchmarks. Εκτέλεσα μικροαρχιτεκτονική ανάλυση στο VTune Performance Analyzer και παρατήρησα ότι στον κώδικα που διαβάζουμε από την μνήμη τα RGB δεδομένα και στον κώδικα που γράφουμε πίσω στην μνήμη τα CMYK δεδομένα που παίρνουμε το ποσοστό προσβάσεων στη Level 2 Cache είναι μεγαλύτερος στα 3 benchmarks Που έχουν ψηλότερο CPI. Ως αποτέλεσμα οι εντολές πρόσβασης στην μνήμη χρειάζονται περισσότερο χρόνο για να εκτελεστούν και το CPI είναι μεγαλύτερο.

Βλέποντας τα αποτελέσματα που πήραμε από την εκτέλεση του benchmark και από τα μετρικά που προέκυψαν από την ανάλυση του στο VTune συμπεραίνουμε ότι ανάλογα με το dataset αλλάζουν και οι αριθμοί που παίρνουμε στα αποτελέσματα.

Χρόνος εκτέλεσης: Το λιγότερο χρόνο εκτέλεσης τον έχουν τα datasets Mandrake.ppm και Goose.ppm, ακολουθεί το Galileo.ppm, το DavidAndDogs.ppm, το EEMBCGroupShotMiami.ppm, το MarsFormerLakes.ppm και τον περισσότερο χρόνο εκτέλεσης τον έχει το DragonFly.ppm.

Επαναλήψεις ανά δευτερόλεπτο: Τα datasets Mandrake.ppm και Goose.ppm εκτελούν 119.6 επαναλήψεις ανά δευτερόλεπτο, ακολουθεί το Galileo.ppm με 89.2 , το DavidAndDogs.ppm με 56.5, το EEMBCGroupShotMiami.ppm με 32.6, το MarsFormerLakes.ppm με 28.2 και τέλος το DragonFly.ppm το οποίο εκτελεί τις λιγότερες επαναλήψεις ανά δευτερόλεπτο, μόλις 17.4.

Clock Cycles: Τα datasets Mandrake.ppm και Goose.ppm κατανάλωνουν τα λιγότερα clock cycles, ακολουθεί το Galileo.ppm, το DavidAndDogs.ppm, το EEMBCGroupShotMiami.ppm, το MarsFormerLakes.ppm και τέλος το DragonFly.ppm το οποίο χρειάστηκε τα περισσότερα clock cycles.

Εντολές που ολοκληρώθηκαν: Τα datasets Mandrake.ppm και Goose.ppm χρειάστηκαν να ολοκληρώσει λιγότερες εντολές για να εκτελεστεί το πόγραμμα, ακολουθεί το Galileo.ppm, το DavidAndDogs.ppm, το EEMBCGroupShotMiami.ppm, το MarsFormerLakes.ppm και τέλος το DragonFly.ppm το οποίο χρειάστηκε να ολοκληρώσει τις περισσότερες εντολές.

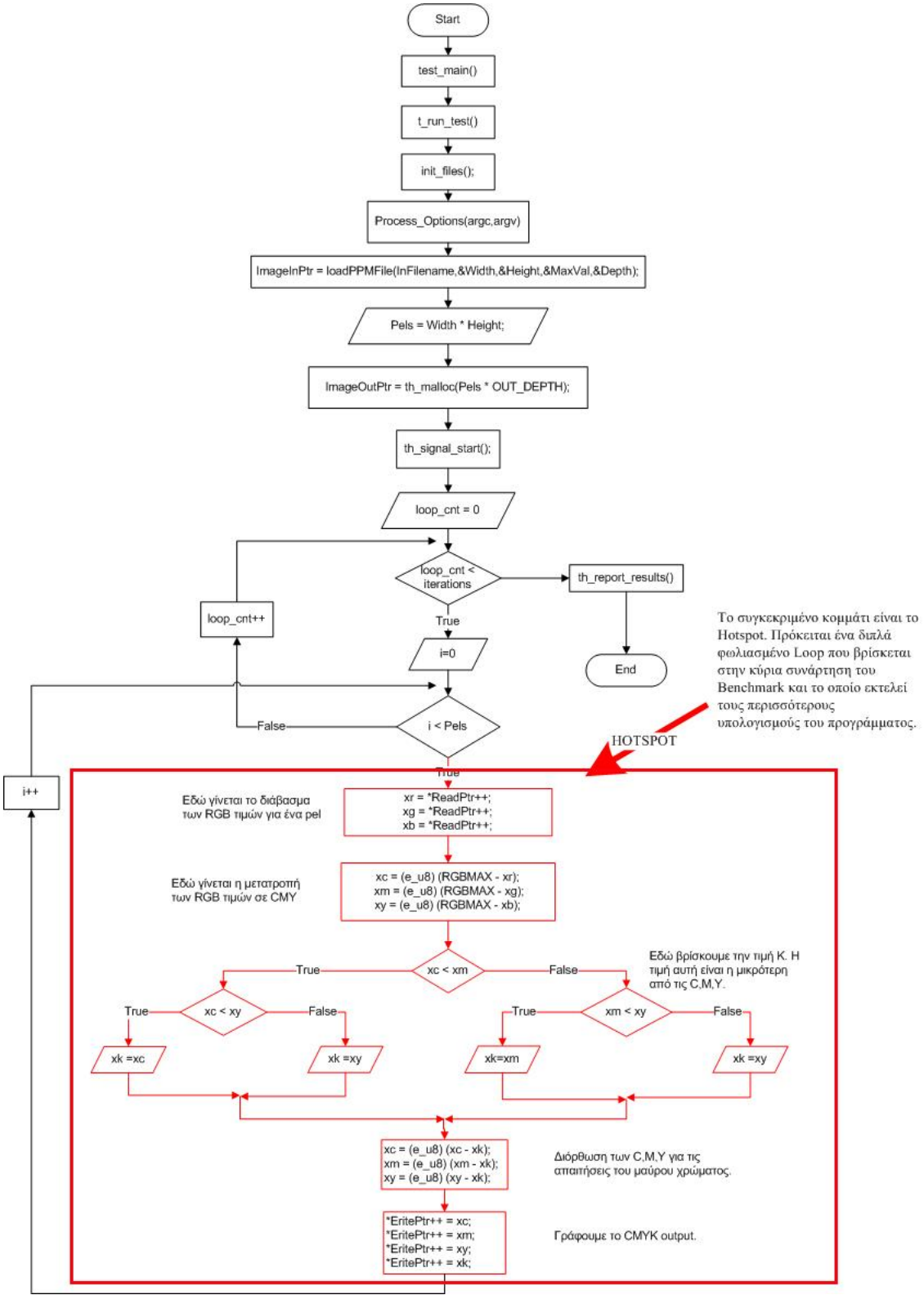
Τελικά μπορούμε να εξάγουμε το συμπέρασμα ότι για τις διαφορές στους αριθμούς ευθύνονται οι διαστάσεις των εικόνων. Αφού τα datasets Mandrake.ppm και Goose.ppm έχουν τις ίδιες διαστάσεις και έχουν το λιγότερο χρόνο εκτέλεσης, τις περισσότερες επαναλήψεις ανά δευτερόλεπτο, τα λιγότερα clock cycles, και χρειάστηκαν να ολοκληρώσουν τις λιγότερες εντολές. Έπειτα από αυτά ανάλογα με τις διαστάσεις της κάθε εικόνας μειώνονται οι επαναλήψεις ανά δευτερόλεπτο, και αυξάνονται ο χρόνο εκτέλεσης, τα clock cycles και οι εντολές που ολοκληρώθηκαν.

Παρατηρώντας τις γραφικές βλέπουμε ότι το αποτέλεσμα του benchmark που εξάγεται για μια μεγαλύτερη εικόνα, ο χρόνος επεξεργασίας είναι σχεδόν γραμμικά ανάλογος

στη μέτρηση του pixel. Πήρα το αρχείο εισόδου Mandrake.ppm ως το κύριο προκαθορισμένο αρχείο εισόδου μεγέθους 76800 pixels, το αρχείο EEMBCGroupShotMiami.ppm είναι 307200 pixels. Αν διαιρέσουμε το 307200 με το 76800 βλέπουμε ότι το EEMBCGroupShotMiami.ppm είναι 4 φορές μεγαλύτερο από το Mandrake.ppm. Τώρα αν πολλαπλασιάσουμε τον συνολικό χρόνο εκτέλεσης του Mandrake.ppm, τα clock cycles, τα instructions retired επί τέσσερα βλέπουμε ότι παίρνουμε σχεδον τα ίδια αποτελέσματα που έχουμε πάρει από το VTune για το EEMBCGroupShotMiami.ppm. Δηλαδη έχουμε 8.361sec *4≈ 30.623sec, 5.981 $x10^6$ *4≈ 22.308 $x10^6$ και 6.119$x10^6$ *4≈ 24.551 $x10^6$.

Αυτό ισχύει και για τα υπόλοιπα 5 αρχεία. Όσο μεγαλύτερα είναι τα αρχεία από το Mandrake.ppm περίπου τόσο μεγαλύτερος είναι και ο συνολικός χρόνος εκτέλεσης του τα clock cycles και τα instructions retired των αρχείων.

Έλεγχος ροής κώδικα και Hotspot



Σχήμα 5.15

Το hotspot του κώδικα βρίσκεται στην συνάρτηση t_run_test() η οποία είναι η κύρια συνάρτηση του benchmark, δηλαδή στη συνάρτηση αυτήγίνονται οι περισσότεροι και

οι πιο σημαντικοί υπολογισμοί του προγράμματος. Το κομμάτι κώδικα που βρήκα ως Hotspot είναι ένα διπλά φωλιασμένο Loop που βρίσκεται στην συνάρτηση αυτή. Στο κομμάτι αυτό του κώδικα γίνονται όλοι οι υπολογισμοί και οι πράξεις ώστε τα RGB δεδομένα εισόδου να γίνουν CMYK.

Παρατήρησα ότι σε όλες τις συναρτήσεις, καθώς επίσης και συγκεκριμένα στην συνάρτηση t_run_test, η οποία αναμένεται να περιλαμβάνει το hotspot, το CPI(Cycles per Instructions) κυμαίνεται πολύ κοντά στο 1 που είναι το ιδανικό CPI. Άρα συμπεραίνουμε ότι για την μεγάλη κατανάλωση χρόνου δεν ευθύνεται η μικροαρχιτεκτονική αλλά οι πολλές εντολές που πρέπει να εκτελέσει ο υπολογιστής για να εκτελέσει τον κώδικα.

Σωστά το VTune έδωσε αυτό το σημείο για hotspot αφού ο υπόλοιπος κώδικας της συνάρτησης t_run_test() είναι απλός κώδικας με αρχικοποιήσεις και απλές κλήσεις συναρτήσεων, πουθενά στον κώδικα δεν υπάρχει άλλος βρόγχος άρα οι εντολές που περιλαμβάνονται στο hotspot αφού βρίσκονται σε ένα διπλά φωλιασμένο Loop πολλαπλασιάζονται σε πλήθος και έτσι ο επεξεργαστής χρειάζεται περισσότερο χρόνο ώστε να εκτελέσει το σημείο αυτό του κώδικα.

Με όποιο dataset και αν τρέξουμε το πρόγραμμα παίρνουμε το ίδιο Hotspot κώδικα και αυτό είναι λογικό αφού όπως προανέφερα στο σημείο αυτό του κώδικα γίνονται όλες οι πράξεις για μετατροπή του dataset εισόδου από RGB σε CMYK. Αυτό επίσης δικαιολογεί και το γεγονός ότι όσο πιο μεγάλο είναι το dataset τόσο πιο μεγάλος είναι και ο χρόνος εκτέλεσης του προγράμματος. Αφού περισσότερα RGB δεδομένα απαιτούν περισσότερες πράξεις για τη μετατροπή τους σε CMYK δεδομένα.

5.3 Ανάλυση Huffman Decoding Benchmark

Το Huffman Decoding benchmark αρχικοποιεί τους AC και DC χρωματικούς και φωτεινούς πίνακες(4 πίνακες). Συμπληρώνει τα διάφορα buffers(ενδιάμεση μνήμη) και μετά εκτελεί την λειτουργία Huffman Decoding χρησιμοποιώντας μια αρκετά σταθερή υλοποίηση Huffman.

Η υλοποίηση είναι ακέραιη και υλοποιεί το CRC για να ελέγξει την ποιότητα των στοιχείων εξόδου.

Αρχείο εισόδου και εξόδου του benchmark:

Το input του benchmark είναι μία κωδικοποιημένη εικόνα η οποία βρίσκεται στο αρχείο data.h. Το output του benchmark είναι μια συμπιεσμένη αποκωδικοποιημένη εικόνα.

Δομές δεδομένων

Η κύρια δομή δεδομένων στο benchmark είναι ένας δισδιάστατος πίνακας τύπου Huffman table. Η δήλωση στο πρόγραμμα είναι huffman_table=huffman_tables[2][2];.

Το huffman_tables[0][0] είναι ο huffman πίνακας για AC LUMINANCE, το huffman_tables[0][1] είναι ο huffman πίνακας για AC CHROMINANCE, το huffman_tables[1][0] είναι ο huffman πίνακας για DC LUMINANCE,το huffman_tables[1][1] είναι ο huffman πίνακας για DC CHROMINANCE.

Κάθε κελί του πίνακα αντιπροσωπεύει δηλαδή ένα Huffman Table με τα εξής χαρακτηριστικά:

```
typedef struct huffman_table_type {
n_int *value_offset;
                                    n_int *max_code;
                                    n_int *look_nbits;
                                    n_int *look_sym;
                                    const e_u8 *values;
                                    const e_u8 *bits;
} huffman_table;
```

Πολυπλοκότητα Benchmark:

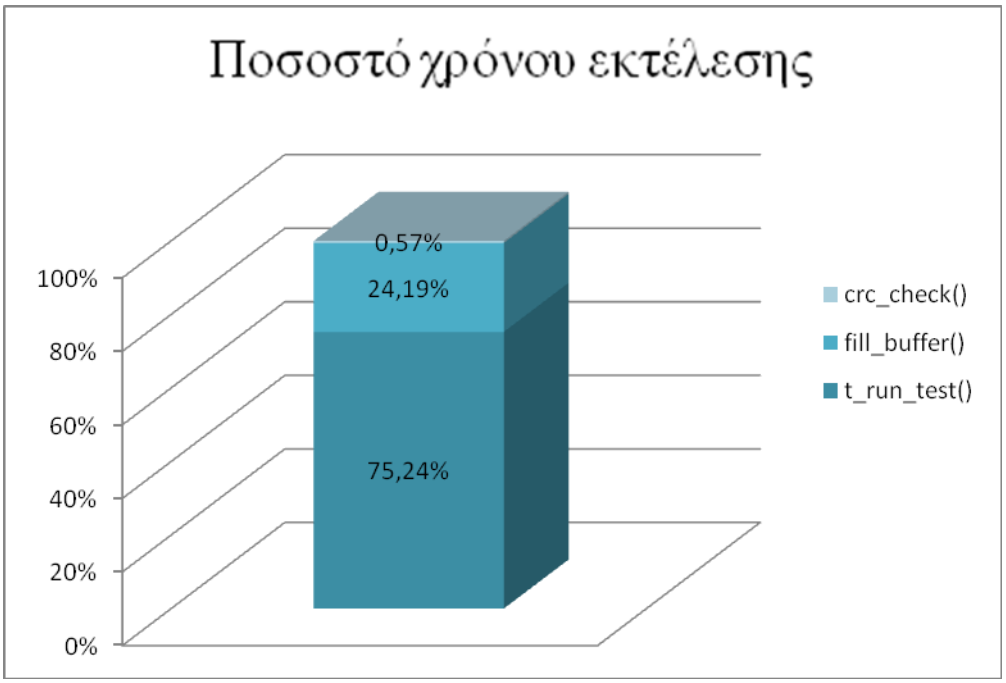
Η πολυπλοκότητα του Huffman Decoding Benchmark εξαρτάται από τον αριθμό των επαναλήψεων που επιλέγουμε να τρέξει το benchmark μέσω του command line και από το αν υπάρχουν Bytes στο ενδιάμεσο Buffer που γεμίζει σε κάθε επανάληψη. Στην κύρια συνάρτηση του benchmark υπάρχουν 2 κύρια Loops ένα που εκτελείται με βάση τα iterations και ένα που εκτελείται με βάση το αν υπάρχουν Bits στο buffer.

Ανάλυση χρόνου εκτέλεσης και των εντολών του συνολικού προγράμματος:

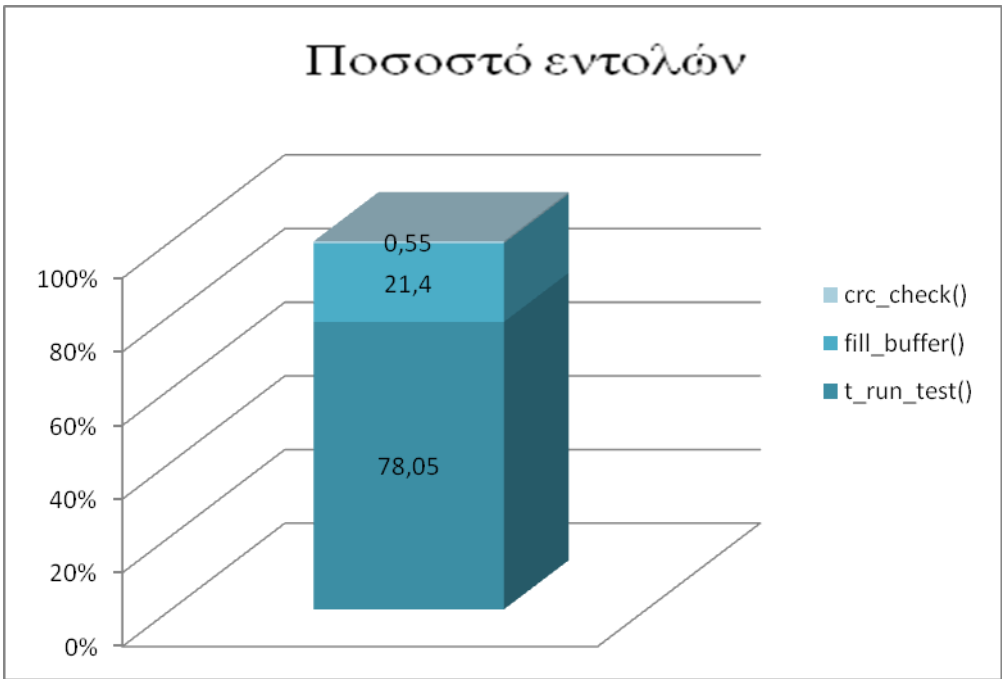
Μετρικά	**Συναρτήσεις**		
	t_run_test	fill_buffer	crc_check
Ποσοστό χρόνου εκτέλεσης	75,24	24,19	0,57
Ποσοστό εντολών	78,05	21,40	0,55
CPI	0,926	1	1,085

Πίνακας 5.4

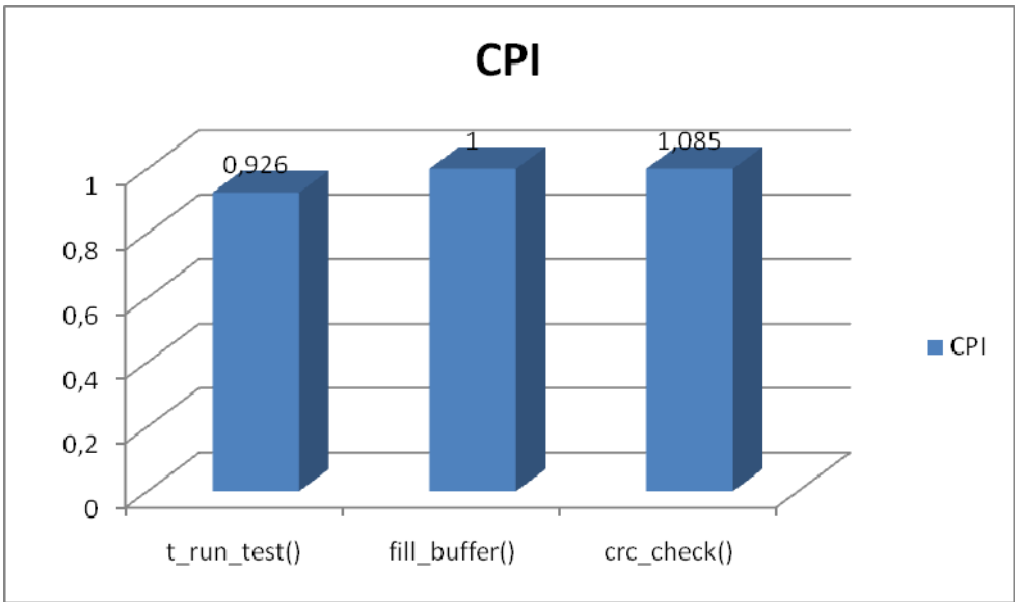
Οι περισσότερο ενεργές functions του Huffman Decoding Benchmark είναι η t_run_test η οποία σπαταλά το 75,24% του συνολικού χρόνου εκτέλεσης του Huffman Decoding Benchmark. Ακολουθεί με μεγάλη διαφορά η fill_buffer, η οποία σπαταλά το 24,19% και η crc_check η οποία σπαταλά μόλις το 0,57% του χρόνου εκτέλεσης. Από τον πιο πάνω πίνακα δημιούργησα τις πιο κάτω γραφικές παραστάσεις:



Σχήμα 5.16



Σχήμα 5.17

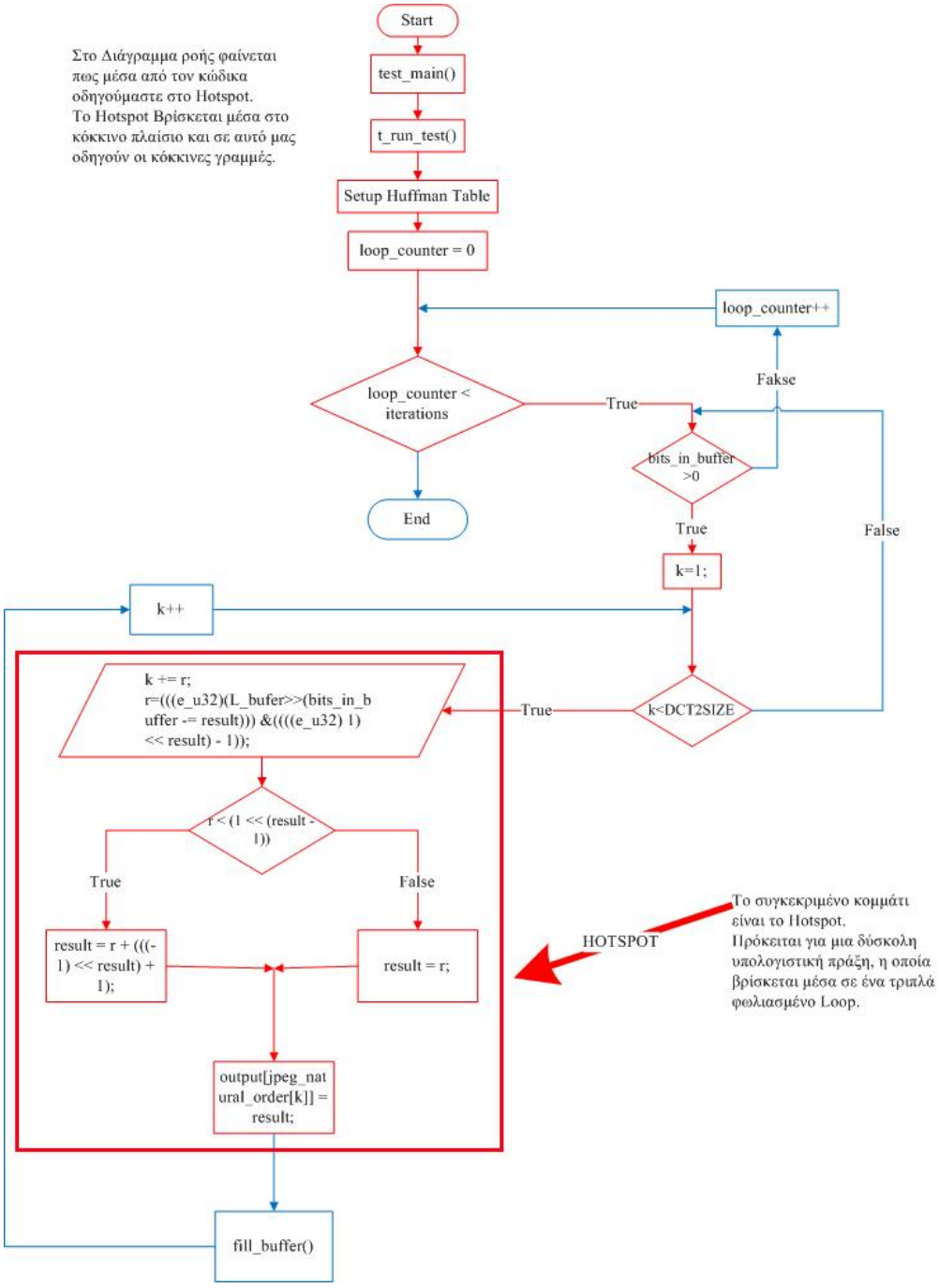


Σχήμα 5.18

Με βάση αυτά τα στοιχεία αναμένω ότι το hotspot θα βρίσκεται στην συνάρτηση t_run_test η οποία είναι και η κύρια συνάρτηση του benchmark και στην οποία εκτελούνται όλες οι σημαντικές λειτουργίες του benchmark με την κλήση και των υπόλοιπων συναρτήσεων. Επίσης παρατήρησα ότι σε όλες τις συναρτήσεις, καθώς επίσης και συγκεκριμένα στην συνάρτηση t_run_test, η οποία αναμένεται να περιλαμβάνει το hotspot, το CPI(Cycles per Instructions) κυμαίνεται πολύ κοντά στο 1 που είναι το ιδανικό CPI. Άρα συμπεραίνουμε ότι για την μεγάλη κατανάλωση χρόνου δεν ευθύνεται η μικροαρχιτεκτονική αλλά οι πολλές εντολές που πρέπει να εκτελέσει ο υπολογιστής για να εκτελέσει τον κώδικα.

Μπαίνοντας στον κώδικα του Benchmark και ελέγχοντας τα ποσοστά χρόνου που σπαταλήθηκαν στα διάφορα σημεία του κώδικα όντως το hotspot βρισκόταν στην συνάρτηση t_run_test(). Είναι ένα κομμάτι κώδικα σε ένα τριπλά φωλιασμένο loop.

Έλεγχος ροής κώδικα και Hotspot



Σχήμα 5.19

Το hotspot βρήκα ότι είναι ένα τριπλά φωλιασμένο loop που βρίσκεται στη κύρια συνάρτηση t_run_test. Το loop αυτό, βρίσκεται μέσα στο Loop το οποίο εκτελείται όσο υπάρχουν Bits στο buffer και το οποίο με τη σειρά του βρίσκεται μέσα στο κύριο Loop της συνάρτησης που μετρά τα iterations. Είναι λογικό να είναι αυτό το σημείο του προγράμματος στο οποίο ο επεξεργαστής καταναλώνει τον περισσότερο του χρόνο γιατί με βάση το documentation του vtune hotspot μπορεί να θεωρηθεί και ένα μεγάλο loop το οποίο εκτελεί πολλές πράξεις. Στην περίπτωση αυτή έχουμε ένα πολύ μεγάλο τριπλά φωλιασμένο Loop το οποίο εκτελεί πολλές πράξεις όπως bit manipulation(shifting) , anding κλπ.

```
int t_run_test( size_t iterations, int argc, const char* argv[] )
{
for(loop_counter = 0; loop_counter < (LoopCount) iterations; loop_counter++) {

/* The DC coefficient is first. We collect the bits from the stream */
while(bits_in_buffer) {

        /* We loop around here until we have got 63 AC values */
        for (k = 1; k < DCT2SIZE; k++) {

        /* This is where AC differs from DC decoding */
        r = result >> 4;
        result &= 15;

        if(result) {
                k += r;
                r = (((e_u32) (L_buffer >> (bits_in_buffer -= result))) & ((((e_u32) 1) <<
                result) - 1));
                if(r < (1 << (result - 1))) { result = r + (((-1) << result) + 1);}
                 else { result = r;}
                }
        }
  }     /* This is the end of the while(bits_in_buffer) */
```

}/* This is the end of the iterations loop */

Το σημείο του κώδικα που βρίσκεται μέσα στο for (k = 1; k < DCT2SIZE; }k++){ είναι το σημείο που παρατηρήθηκε η κατανάλωση περισσότερων κύκλων του επεξεργαστή.

Όταν έκανα την ανάλυση μου ως hotspot θεώρησα ένα while loop το οποίο βρισκόταν φωλιασμένο στο συγκεκριμένο for loop το οποίο τελικά είναι και το hotspot του κώδικα, δηλαδή ήταν τετραπλά φωλιασμένο. Περισσότερους κύκλους επεξεργαστή όμως στο for loop αυτό καταναλώνει η πράξη:

r = (((e_u32) (L_buffer >> (bits_in_buffer -= result))) & ((((e_u32) 1) << result) - 1))

και όχι το while (code > max_code[nb])

```
{
        code <<= 1;
                        code |= ((e_u32) (L_buffer >>(bits_in_buffer -= 1))) & 0x01;
                        nb++;
}
```

Αυτό είναι λογικό αν λάβουμε υπόψη μας ότι το while αυτό βρίσκεται μέσα στο else ενός if statement το οποίο τις περισσότερες φορές είναι true. Άρα μπορεί το while αυτό να καταναλώνει πολλούς κύκλους αλλά η πολυπλοκότητα της γραμμής κώδικα r = (((e_u32) (L_buffer >> (bits_in_buffer -= result))) & ((((e_u32) 1) << result) - 1)) εξαρτάται από τα iterations, το DCT2SIZE και το αν υπάρχουν bits στο buffer, επίσης για να εκτελεστεί χρειάζεται μεγάλη υπολογιστική δύναμη από μέρους του επεξεργαστή. Ενώ η πολυπλοκότητα του while loop είναι O(n)<Πολυπλοκότητα της γραμμής hotspot αφού συγκαταλέγεται μέσα στο ίδιο Loop αλλά δεν εκτελείται τις περισσότερες φορές.

Συμπερασματικά hotspot του Huffman Decoding benchmark είναι ένα τριπλά φωλιασμένο Loop που βρίσκεται στην κύρια συνάρτηση t_run_test και η γραμμή κώδικα με την μεγαλύτερη κατανάλωση κύκλων επεξεργαστή είναι μια πράξη που περιλαμβάνει bit manipulation και anding που χρειάζονται μεγάλη υπολογιστική δύναμη εκ μέρους του υπολογιστή και ως εκ τούτου πολλούς κύκλους ζωής για να εκτελεστούν.

5.4 Ανάλυση RGB to YIQ Conversion benchmark

Η εφαρμογή αυτή χρησιμοποιείται στους NTSC αποκωδικοποιητές όπου τα RGB δεδομένα εισόδου από την κάμερα μετατρέπονται για να φωτίσουν(Y) και να χρωματίσουν (I, Q) την πληροφορία. Στον αποκωδικοποιητή NTSC τα σήματα I,Q διαμορφώνονται από ένα υπομεταφορέα και προσθέτονται στο Y σήμα.

Στα πραγματικά προϊόντα, αυτός ο συνηθισμένος υπολογισμός συνήθως εκτελείται σε ειδικό για αυτό το σκοπό υλικό, ειδικά στα προϊόντα ψηφιακού video. Για λιγότερο κόστος και ευκαμψία, αυτός ο αλγόριθμος μπορεί να υλοποιηθεί στο λογισμικό εάν η ΚΜΕ είναι αρκετά ισχυρή και η ψηφιακή εικόνα είναι μια στατική φωτογραφία.

Το benchmark εξετάζει την ικανότητα της ΚΜΕ να εκτελέσει ένα ακριβή υπολογισμό πολλαπλασιασμού και συγκέντρωσης αποτελεσμάτων.

Τα R, G, B inputs των 8-bit pixel της έγχρωμης φωτογραφίας επεξεργάζονται ως εξής:
Y = 0.299*R + 0.587*G + 0.114*B

I = 0.596*R – 0.275*G – 0.321*B

Q = 0.212*R – 0.523*G + 0.311*B

Οι RGB τιμές βρίσκονται μεταξύ των ορίων [0:255]. Οι συντελεστές μετατροπής είναι 16 bits. Τα αποτελέσματα πολλαπλασιασμού/πρόσθεσης γίνονται shifted δεξιά για 16 bits. Πριν το shift, προστίθεται 1 στη θέση του bit που βρίσκεται δεξιά του LSB του shifted αποτελέσματος για να στρογγυλοποιήσουμε το αποτέλεσμα στον πιο κοντινό ακέραιο αριθμό. Τα δεδομένα εξαγωγής είναι 8-bit. Οι Y τιμές βρίσκονται μεταξύ των ορίων [0:255] και οι I,Q είναι μεταξύ των ορίων [-127:127]. Το μέγεθος των δεδομένων εισαγωγής και εξαγωγής είναι 320 pixels η οριζόντια κατεύθυνση και 240 η κάθετη κατεύθυνση. Τα δεδομένα της εικόνας 320x240 για το RGB και το YIQ αποθηκεύονται σειριακά ως εξής:

R[0], G[0], B[0], R[1], G[1], B[1], R[76799], G[76799], B[76799]

Y[0], I[0], Q[0], Y[1], I[1], Q[1], Y[76799], I[76799], Q[76799]

Οι δείκτες αυξάνονται κατά ένα για να προσπελάσουν τα R,G,B ή τα Y,I,Q δεδομένα με αυτή τη σειρά.

Η ποιότητα παραγωγής μετριέται χρησιμοποιώντας Non Intrusive CRC κώδικα ο οποίος αναπτύχθηκε από την EEMBC. Και δεν επηρεάζει το αποτέλεσμα του benchmark.

Ένα "for loop" υπολογίζει την μετατροπή ενός συνόλου από RGB δεδομένα εισόδου και YIQ δεδομένα εξόδου κάθε φορά. Ένα σύνολο από R,G,B δεδομένα εισόδου διαβάζεται από τη μνήμη αυξάνοντας ένα δείκτη διαβάσματος. Ένα σύνολο από Y,I,Q δεδομένα εξόδου γράφεται πίσω στην μνήμη αυξάνοντας ένα δείκτη γραψίματος.

Ο υπολογισμός είναι ένας απλός πολλαπλασιασμός και άθροισμα. Το μέγεθος του κώδικα είναι μικρό και εύκολα προσαρμόζεται σε μια μικρή L1 Instruction Cache.

Τα αρχεία εισόδου και εξόδου του benchmark:
Τα αρχεία εισόδου του benchmark έχουν περιγραφεί στο υποκεφάλαιο 6.1. Αφού γίνει η μετατροπή παίρνουμε τα χρωματισμένα και φωτισμένα YIQ images.

Πολυπλοκότητα Benchmark:
Η πολυπλοκότητα του RGB to YIQ conversion Benchmark εξαρτάται από τον αριθμό των επαναλήψεων που επιλέγουμε να τρέξει το benchmark μέσω του command line και από την τιμή της μεταβλητής Pels. Η μεταβλητή Pels είναι το γινόμενο του πλατους επί το ύψος του RGB image που προσπαθουμε να μετατρέψουμε σε YIQ. Στην κύρια συνάρτηση του benchmark υπάρχουν 2 κύρια Loops ένα που εκτελείται με βάση τα iterations και ένα που εκτελείται με βάση τα Pels. Τελικά η πολυπλοκότητα του benchmark είναι Θ(iterations*width*height).

Ανάλυση χρόνου εκτέλεσης και των εντολών του συνολικού προγράμματος ανά dataset:
Μετά από την εκτέλεση των 7 εκτελέσιμων αρχείων που πήρα από την μεταγλώττιση του RGB to CMYK conversion benchmark προέκυψαν κάποια αποτελέσματα που αφορούν την εκτέλεση του benchmark.

Στον Πίνακα 5.1 μπορούμε να δούμε τον χρόνο που πήρε στο benchmark για να εκτελέσει τη λειτουργία του, πόσες επαναλήψεις του benchmark έτρεξαν ανά δευτερόλεπτο και πόσο χρόνο κατανάλωσε κάθε iteration για να ολοκληρωθεί, ανά dataset.

	Total Run Time (sec)	Iterations/second	Time/iterations (sec)
Mandrake.ppm	2.34	427.35	0.00234
Goose.ppm	2.37	421.76	0.00237
EEMBCGroupShotMiami.ppm	9.26	107.92	0.00926
DavidAndDogs.ppm	5.60	178.57	0.00560
DragonFly.ppm	16.19	61.75	0.01619
MarsFormerLakes.ppm	12.07	82.81	0.01207
Galileo.ppm	3.85	259.53	0.00385

Πίνακας 5.5 Αποτελέσματα εκτέλεσης στο Cygwin ανά dataset.

Στο εργαλείο VTune έκανα Quick Performance Analysis των εκτελέσιμων αρχείων. Όταν έτρεξαν τα εκτελέσιμα έδωσα ως είσοδο την επιλογή να εκτελεστούν 1000 iterations του benchmark ώστε τα αποτελέσματα να είναι περισσότερο αντιπροσωπευτικά. Μετά το πέρας της εκτέλεσης των αρχείων προέκυψαν τα αποτελέσματα των μετρικών τα οποία επέλεξα να μετρήσει το VTune.

Από το VTune προέκυψε ότι όποιο dataset και αν δώσουμε ως είσοδο στο Benchmark RGB to YIQ conversion, η συνάρτηση t_run_test() είναι αυτή που καταναλώνει το περισσότερο χρόνο για να εκτελεστεί, και η διαφορά από τις υπόλοιπες συναρτήσεις του benchmark είναι πάρα πολύ μεγάλη. Το ποσοστό του χρόνου εκτέλεσης της συνάρτησης κυμαίνεται γύρω από το 97,5% και οι υπόλοιπες συναρτήσεις

καταναλώνουν όλες μαζί μόλις το 2,5%. Ως εκ τούτου αναμένω ότι το hotspot του benchmark θα βρίσκεται στη συνάρτηση t_run_test().

	Μετρικά				
	Ποσοστό χρόνου εκτέλεσης	Clock cycles	Ποσοστό εντολών	Instructions Retired	CPI
Mandrake.ppm	97,45%	$4.410x10^6$	99.09%	$6.151\ x10^6$	0,717
Goose.ppm	97,54%	$4.406\ x10^6$	99.09%	$6.161\ x10^6$	0,715
EEMBCGroupShotMiami.ppm	97,49%	$17.690\ x10^6$	99.12%	$24.599\ x10^6$	0,719
DavidAndDogs.ppm	97,50%	$10.664\ x10^6$	99.19%	$14.908\ x10^6$	0,715
DragonFly.ppm	97,45%	$31.304\ x10^6$	99.13%	$43.468\ x10^6$	0,720
MarsFormerLakes.ppm	97,49%	$22.237\ x10^6$	99.12%	$30.847\ x10^6$	0,721
Galileo.ppm	97,48%	$6.909\ x10^6$	99.14%	$9.647\ x10^6$	0,716

Πίνακας 5.6. Αποτελέσματα εκτέλεσης των datasets στο VTune Performance Analyzer.

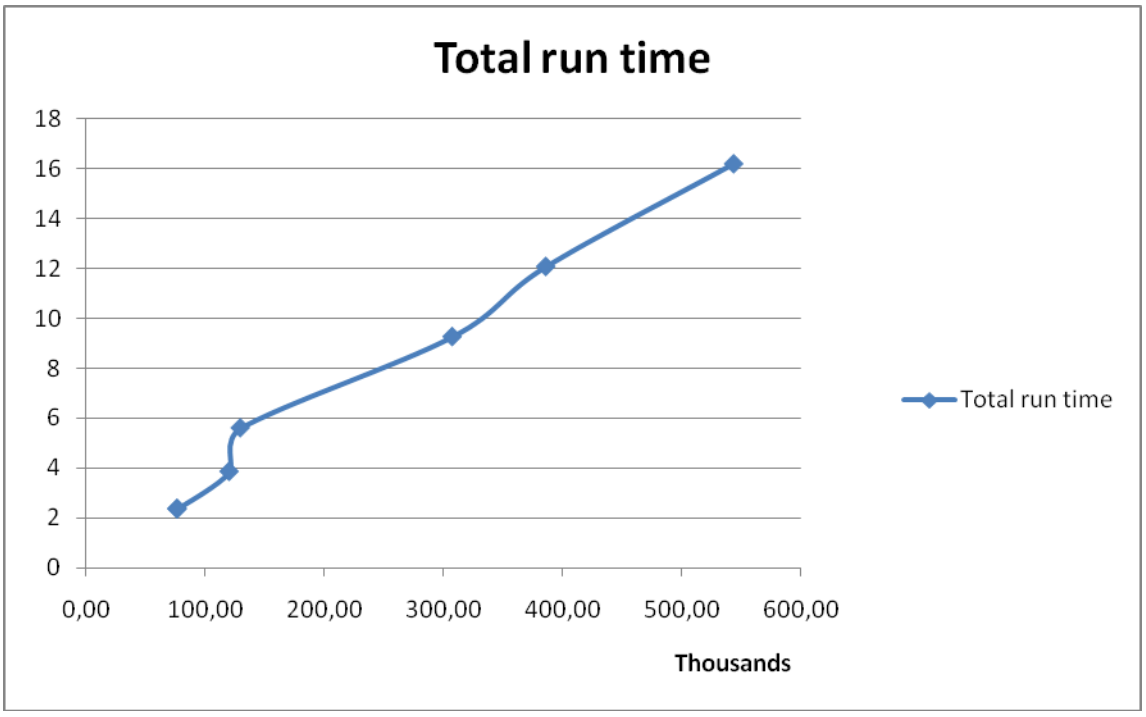
Μπαίνοντας στον κώδικα του Benchmark και ελέγχοντας τα ποσοστά χρόνου που σπαταλήθηκαν στα διάφορα σημεία του κώδικα όντως το hotspot βρισκόταν στην συνάρτηση t_run_test(). Είναι ένα κομμάτι κώδικα σε ένα διπλά φωλιασμένο loop.Στον Πίνακα 6.6 μπορούμε να δούμε κάποια μετρικά του VTune για τη συνάρτηση t_run_test() του benchmark για κάθε ένα από τα datasets που δόθηκαν σαν είσοδος. Πρόκειται για το ποσοστό του χρόνου της συνολικής εφαρμογής που κατανάλωσε η συνάρτηση, το πραγματικό αριθμό των clock cycles που χρειάστηκαν για να τρέξει η κάθε εφαρμογή για 1000 επαναλήψεις, το ποσοστό των εντολών που ολοκληρώθηκαν

κατά τη διάρκεια εκτέλεσης του benchmark, τον πραγματικό αριθμό των εντολών που ολοκληρώθηκαν κατά τη διάρκεια εκτέλεσης του benchmark και τέλος το CPI της συνάρτησης για κάθε dataset.

Από τις μετρήσεις των δύο πινάκων πήρα κάποιες γραφικές παραστάσεις ώστε να συγκρίνω τα αποτελέσματα που πήρα λαμβάνοντας υπόψη και τα χαρακτηριστικά κάθε dataset. Οι γραφικές παραστάσεις που ακολουθουν περιλαμβάνουν στον άξονα των Y τα μετρικά και στον άξονα των X τα μεγέθη των εικόνων. Στον πινακα πιο κάτω γίνεται η αντιστοίχιση αρχείου εισόδου και μεγεθους εικόνας σε pixels.

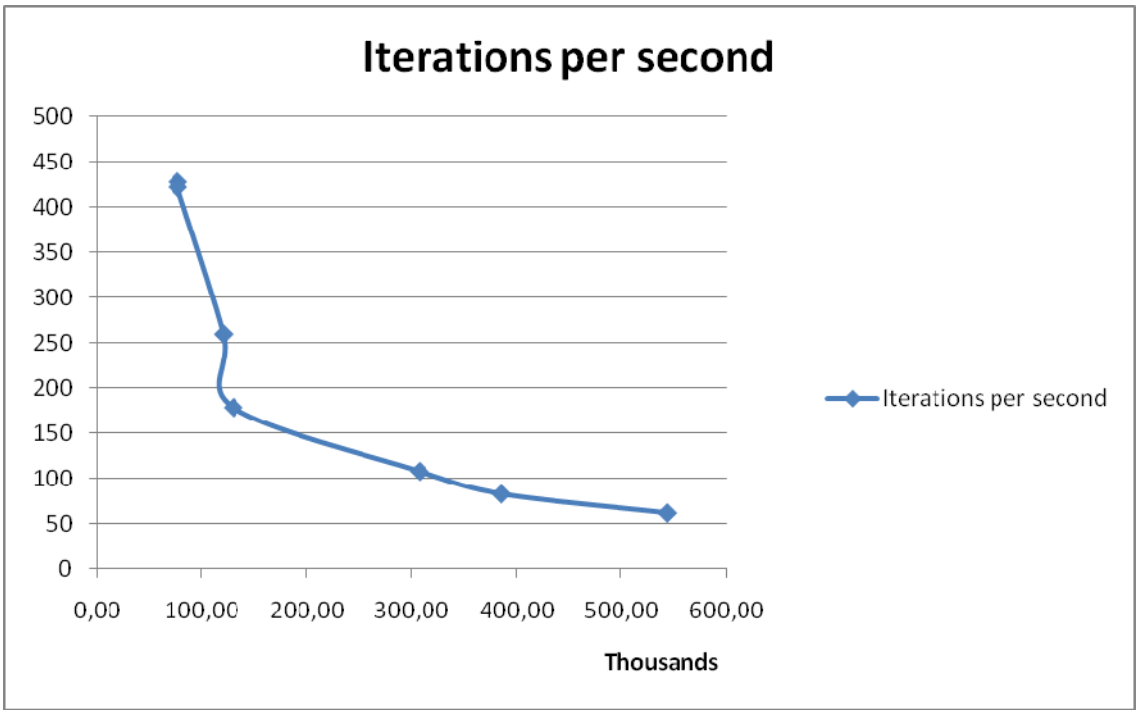
	Διαστάσεις	**Pixels**
Mandrake.ppm	320x240	76800
Goose.ppm	320x240	76800
EEMBCGroupShotMiami.ppm	640x480	307200
DavidAndDogs.ppm	564x230	129720
DragonFly.ppm	606x896	542976
MarsFormerLakes.ppm	800x482	385600
Galileo.ppm	290x415	120350

Πίνακας 5.7 Αντιστοίχηση αρχείου εισόδου-μεγεθους σε pixels



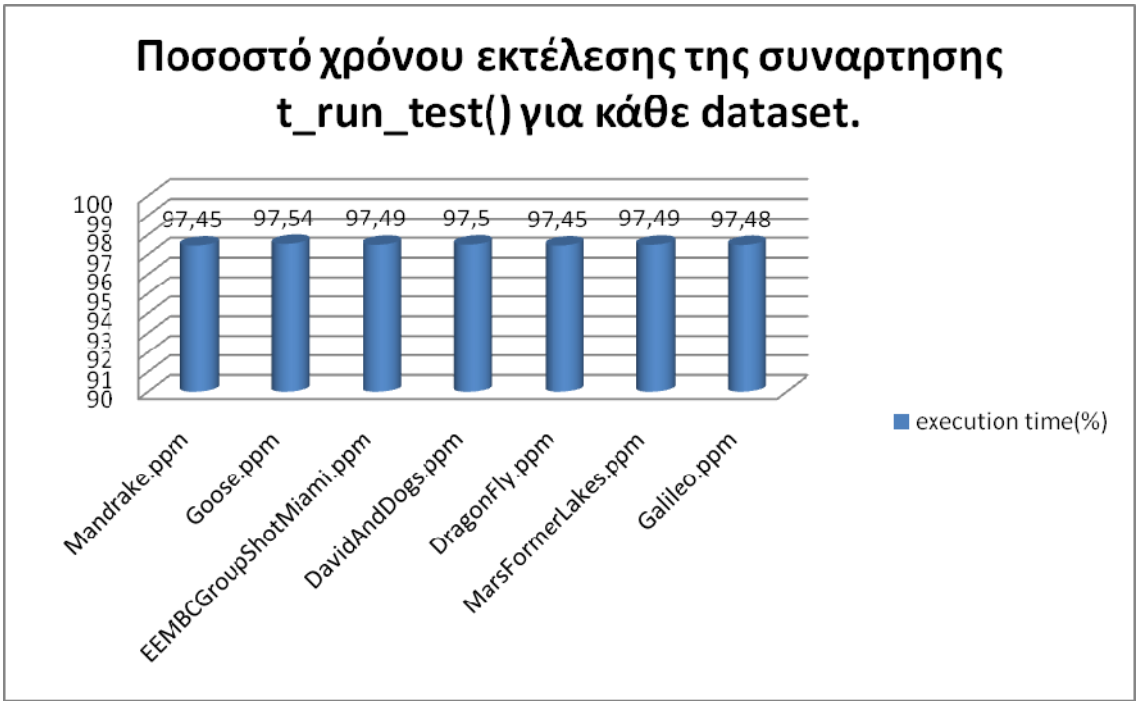
Σχήμα 5.20

Παρατηρούμε ότι στην γραφική παράσταση του συνολικού χρόνου εκτέλεσης του benchmark, για κάθε διαφορετικό dataset που δίναμε ως είσοδο ο χρόνος εκτέλεσης είναι διαφορετικός. Το λιγότερο χρόνο εκτέλεσης έχουν τα datasets Mandrake.ppm και Goose.ppm, ακολουθεί το Galileo.ppm, το DavidAndDogs.ppm, το EEMBCGroupShotMiami.ppm, το MarsFormerLakes.ppm και τον περισσότερο χρόνο εκτέλεσης έχει το DragonFly.ppm.



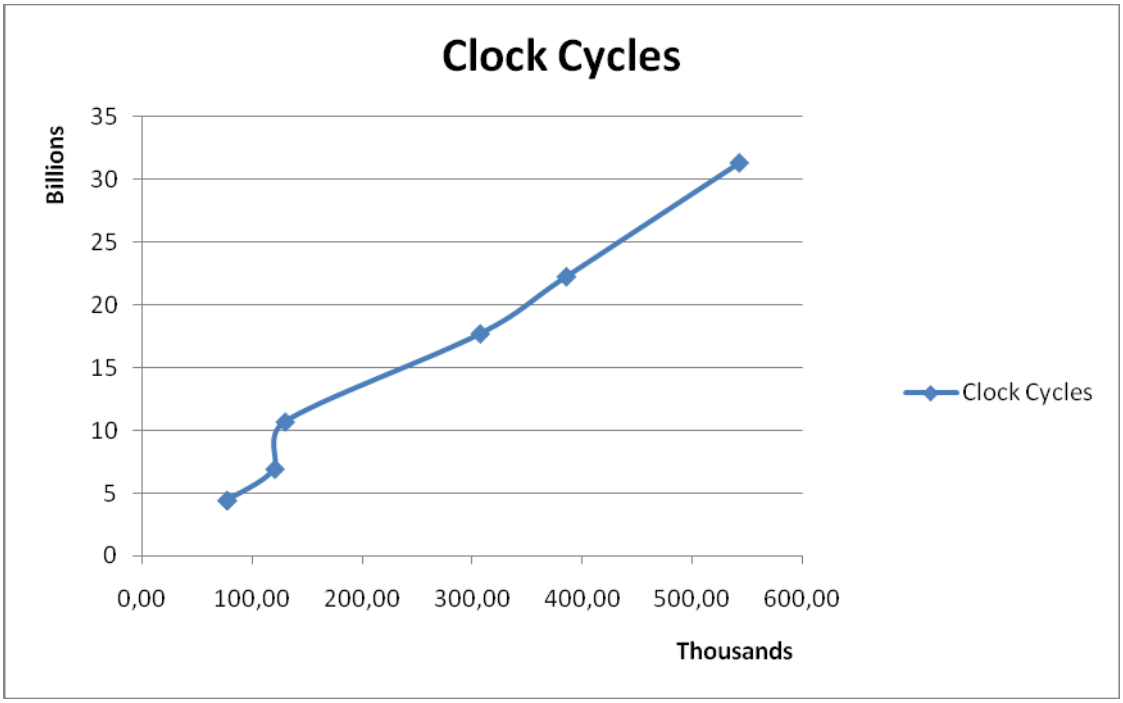
Σχήμα 5.21

Παρατηρούμε ότι στην γραφική παράσταση των επαναλήψεων του benchmark που εκτελέστηκαν ανά δευτερόλεπτο, για κάθε διαφορετικό dataset που δίναμε ως είσοδο ο αριθμός των iterations ήταν διαφορετικός. Τα benchmark όταν έτρεχε με είσοδο τα datasets Mandrake.ppm και Goose.ppm εκτελούσε 427,35 και 421,76 επαναλήψεις ανά δευτερόλεπτο αντίστοιχα, ακολουθεί το Galileo.ppm με 259,53 , το DavidAndDogs.ppm με 178,57, το EEMBCGroupShotMiami.ppm με 107,92, το MarsFormerLakes.ppm με 82,81 και τέλος το DragonFly.ppm το οποίο εκτελεί τις λιγότερες επαναλήψεις ανά δευτερόλεπτο, μόλις 61,75.



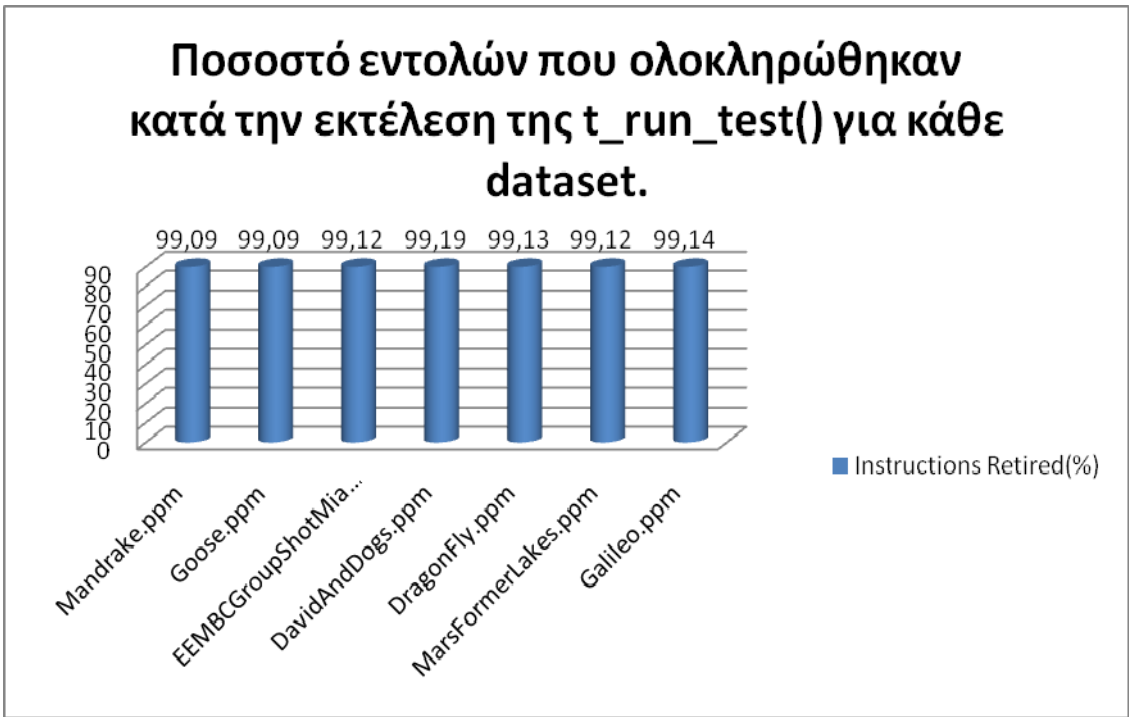
Σχήμα 5.22

Παρατηρούμε ότι το ποσοστό χρόνου εκτέλεσης που σπαταλήθηκε για την εκτέλεση της συνάρτησης t_run_test κυμαίνεται στα ίδια επίπεδα για όλα τα datasets. Κατά συνέπεια μπορούμε να σκεφτούμε ότι το hotspot του benchmark βρίσκεται στο ίδιο σημείο για κάθε dataset.



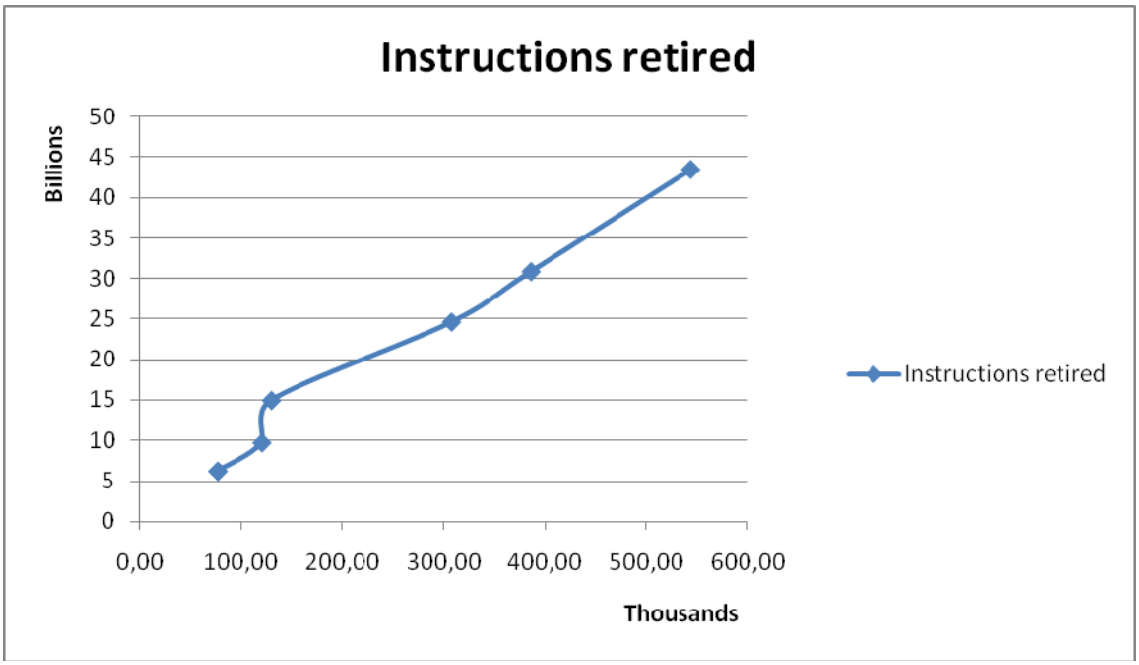
Σχήματα 5.23

Παρατηρούμε ότι στην γραφική παράσταση των clock cycles που καταναλώθηκαν ώστε να εκτελεστεί η συνάρτηση t_run_test() του benchmark, για κάθε διαφορετικό dataset που δίναμε ως είσοδο ο αριθμός των clock cycles ήταν διαφορετικός. Το benchmark όταν έτρεχε με είσοδο τα datasets Mandrake.ppm και Goose.ppm κατανάλωσαν τα λιγότερα clock cycles, ακολουθεί το Galileo.ppm, το DavidAndDogs.ppm, το EEMBCGroupShotMiami.ppm, το MarsFormerLakes.ppm και τέλος το DragonFly.ppm το οποίο χρειάστηκε τα περισσότερα clock cycles για να εκτελεστεί.



Σχήμα 5.24

Παρατηρούμε ότι το ποσοστό των εντολών που ολοκληρώθηκαν κατά την εκτέλεση της συνάρτησης t_run_test κυμαίνεται στα ίδια επίπεδα για όλα τα datasets.



Σχήμα 5.25

Παρατηρούμε ότι ο αριθμός των εντολών που ολοκληρώθηκαν κατά την εκτέλεση της συνάρτησης t_run_test(), για κάθε διαφορετικό dataset που δίναμε ως είσοδο ο αριθμός των clock cycles ήταν διαφορετικός. Το benchmark όταν έτρεχε με είσοδο τα datasets Mandrake.ppm και Goose.ppm χρειάστηκε να ολοκληρώσει λιγότερες εντολές, ακολουθεί το Galileo.ppm, το DavidAndDogs.ppm, το EEMBCGroupShotMiami.ppm, το MarsFormerLakes.ppm και τέλος το DragonFly.ppm το οποίο χρειάστηκε να ολοκληρώσει τις περισσότερες εντολές.

Σχήμα 5.26

Παρατηρούμε ότι το CPI του benchmark δεν εξαρτιόταν από τη διαφορετικότητα του dataset εισόδου, ούτε και από προβλήματα που προκαλούσε η μικροαρχιτεκτονική αφού κυμαινόταν στα ίδια επίπεδα για όλα τα datasets.

Βλέποντας τα αποτελέσματα που πήραμε από την εκτέλεση του benchmark και από τα μετρικά που προέκυψαν από την ανάλυση του στο VTune συμπεραίνουμε ότι ανάλογα

με το dataset αλλάζουν τα πλήθη που παίρνουμε στα αποτελέσματα, αλλά όχι τα ποσοστά.

Χρόνος εκτέλεσης: Το λιγότερο χρόνο εκτέλεσης τον έχουν τα datasets Mandrake.ppm και Goose.ppm, ακολουθεί το Galileo.ppm, το DavidAndDogs.ppm, το EEMBCGroupShotMiami.ppm, το MarsFormerLakes.ppm και τον περισσότερο χρόνο εκτέλεσης τον έχει το DragonFly.ppm.

Επαναλήψεις ανά δευτερόλεπτο: Τα datasets Mandrake.ppm και Goose.ppm εκτελούν 119.6 επαναλήψεις ανά δευτερόλεπτο, ακολουθεί το Galileo.ppm με 89.2 , το DavidAndDogs.ppm με 56.5, το EEMBCGroupShotMiami.ppm με 32.6, το MarsFormerLakes.ppm με 28.2 και τέλος το DragonFly.ppm το οποίο εκτελεί τις λιγότερες επαναλήψεις ανά δευτερόλεπτο, μόλις 17.4.

Clock Cycles: Τα datasets Mandrake.ppm και Goose.ppm κατανάλωνουν τα λιγότερα clock cycles, ακολουθεί το Galileo.ppm, το DavidAndDogs.ppm, το EEMBCGroupShotMiami.ppm, το MarsFormerLakes.ppm και τέλος το DragonFly.ppm το οποίο χρειάστηκε τα περισσότερα clock cycles.

Εντολές που ολοκληρώθηκαν: Τα datasets Mandrake.ppm και Goose.ppm χρειάστηκαν να ολοκληρώσει λιγότερες εντολές για να εκτελεστεί το πόγραμμα, ακολουθεί το Galileo.ppm, το DavidAndDogs.ppm, το EEMBCGroupShotMiami.ppm, το MarsFormerLakes.ppm και τέλος το DragonFly.ppm το οποίο χρειάστηκε να ολοκληρώσει τις περισσότερες εντολές.

Τελικά μπορούμε να εξάγουμε το συμπέρασμα ότι για τις διαφορές στους αριθμούς ευθύνονται οι διαστάσεις των εικόνων. Αφού τα datasets Mandrake.ppm και Goose.ppm έχουν τις ίδιες διαστάσεις και έχουν το λιγότερο χρόνο εκτέλεσης, τις περισσότερες επαναλήψεις ανά δευτερόλεπτο, τα λιγότερα clock cycles, και χρειάστηκαν να ολοκληρώσουν τις λιγότερες εντολές. Έπειτα από αυτά ανάλογα με τις διαστάσεις της κάθε εικόνας μειώνονται οι επαναλήψεις ανά δευτερόλεπτο, και αυξάνονται ο χρόνο εκτέλεσης, τα clock cycles και οι εντολές που ολοκληρώθηκαν.

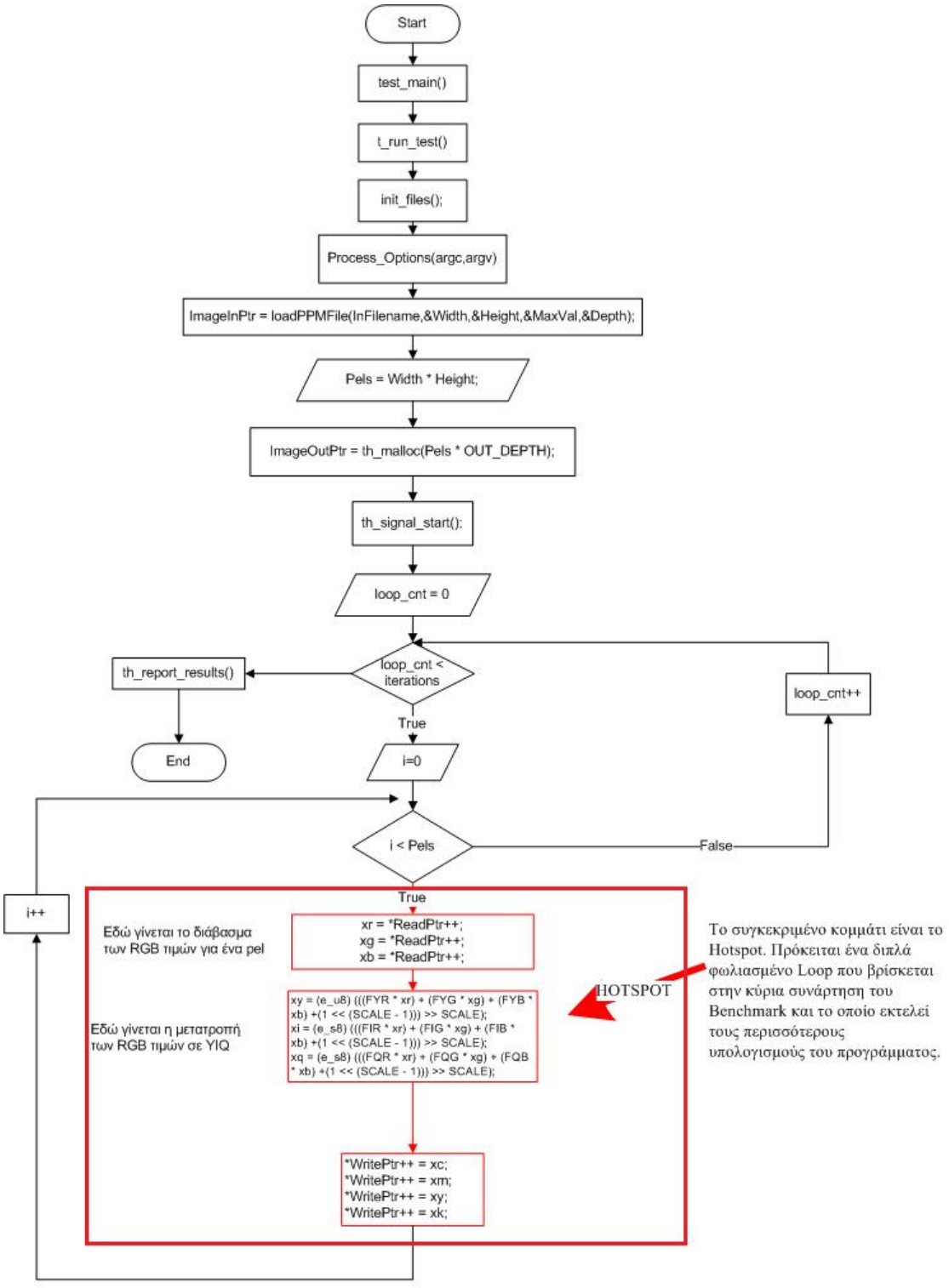
Παρατηρώντας τις γραφικές βλέπουμε ότι το αποτέλεσμα του benchmark που εξάγεται για μια μεγαλύτερη εικόνα, ο χρόνος επεξεργασίας είναι σχεδόν γραμμικά ανάλογος στη μέτρηση του pixel. Πήρα το αρχείο εισόδου Mandrake.ppm ως το κύριο

προκαθορισμένο αρχείο εισόδου μεγέθους 76800 pixels, το αρχείο EEMBCGroupShotMiami.ppm είναι 307200 pixels. Αν διαιρέσουμε το 307200 με το 76800 βλέπουμε ότι το EEMBCGroupShotMiami.ppm είναι 4 φορές μεγαλύτερο από το Mandrake.ppm. Τώρα αν πολλαπλασιάσουμε τον συνολικό χρόνο εκτέλεσης του Mandrake.ppm, τα clock cycles, τα instructions retired επί τέσσερα βλέπουμε ότι παίρνουμε σχεδον τα ίδια αποτελέσματα που έχουμε πάρει από το VTune για το EEMBCGroupShotMiami.ppm. Δηλαδη έχουμε 8.361sec *4≈ 30.623sec, 5.981 $x10^6$ *4≈ 22.308 $x10^6$ και 6.119$x10^6$ *4≈ 24.551 $x10^6$.

Αυτό ισχύει και για τα υπόλοιπα 5 αρχεία. Όσο μεγαλύτερα είναι τα αρχεία από το Mandrake.ppm περίπου τόσο μεγαλύτερος είναι και ο συνολικός χρόνος εκτέλεσης του τα clock cycles και τα instructions retired των αρχείων.

Έλεγχος ροής κώδικα και

Hotspot:



Σχήμα 5.27 Διαγραμμα ροής δεδομένων του Benchmark.

Το hotspot του κώδικα βρίσκεται στην συνάρτηση t_run_test() η οποία είναι η κύρια συνάρτηση του benchmark. Στη συνάρτηση αυτή γίνονται οι περισσότεροι και οι πιο σημαντικοί υπολογισμοί του προγράμματος. Το κομμάτι κώδικα που βρήκα ως Hotspot είναι ένα διπλά φωλιασμένο Loop που βρίσκεται στην συνάρτηση αυτή. Στο κομμάτι αυτό του κώδικα γίνονται όλοι οι υπολογισμοί και οι πράξεις ώστε τα RGB δεδομένα εισόδου να γίνουν YIQ.

Παρατήρησα ότι σε όλες τις συναρτήσεις, καθώς επίσης και συγκεκριμένα στην συνάρτηση t_run_test, η οποία αναμένεται να περιλαμβάνει το hotspot, το CPI(Cycles per Instructions) είναι μικρότερο του 1 που είναι το ιδανικό CPI. Άρα συμπεραίνουμε ότι για την μεγάλη κατανάλωση χρόνου δεν ευθύνεται η μικροαρχιτεκτονική αλλά οι πολλές εντολές που πρέπει να εκτελέσει ο υπολογιστής για να εκτελέσει τον κώδικα.

Σωστά το VTune έδωσε αυτό το σημείο για hotspot αφού ο υπόλοιπος κώδικας της συνάρτησης t_run_test() είναι απλός κώδικας με αρχικοποιήσεις και απλές κλήσεις συναρτήσεων, πουθενά στον κώδικα δεν υπάρχει άλλος βρόγχος άρα οι εντολές που περιλαμβάνονται στο hotspot αφού βρίσκονται σε ένα διπλά φωλιασμένο Loop πολλαπλασιάζονται σε πλήθος και έτσι ο επεξεργαστής χρειάζεται περισσότερο χρόνο ώστε να εκτελέσει το σημείο αυτό του κώδικα.

Με όποιο dataset και αν τρέξουμε το πρόγραμμα παίρνουμε το ίδιο Hotspot κώδικα και αυτό είναι λογικό αφού όπως προανέφερα στο σημείο αυτό του κώδικα γίνονται όλες οι πράξεις για μετατροπή του dataset εισόδου από RGB σε YIQ. Αυτό επίσης δικαιολογεί και το γεγονός ότι όσο πιο μεγάλο είναι το dataset τόσο πιο μεγάλος είναι και ο χρόνος εκτέλεσης του προγράμματος. Αφού περισσότερα RGB δεδομένα απαιτούν περισσότερες πράξεις για τη μετατροπή τους σε YIQ δεδομένα.

5.5 Ανάλυση AES Benchmark

Το AES benchmark παρέχει μια ένδειξη της επίδοσης ενός μικροεπεξεργαστή ή ενός υποσυστήματος επεξεργαστή ψηφιακού σήματος ο οποίος διενεργεί AES κρυπτογραφήσεις και αποκρυπτογραφήσεις.

Η AES κρυπτογραφία χρησιμοποιείται σε πολλά κρυπτογραφικά πρωτόκολλα. Το AES προτάθηκε για να αντικαταστήσει τα ψηφιακά πρότυπα κρυπτογράφησης (DES). Χρησιμοποιεί τον αλγόριθμο Rijndael [4].

Το benchmark αυτό είναι υπολογιστικά ενδιαφέρον αφού σε αυτό χρησιμοποιούνται πολλοί μαθηματικοί υπολογισμοί.

Ο αλγόριθμος Advanced Encryption Standard υλοποιεί τον αλγόριθμο Rijndael. Rijndael είναι μια block cipher η οποία κρυπτογραφεί και αποκρυπτογραφεί blocks των 128,192, και 256 bits, χρησιμοποιώντας κλειδιά των 128,192 και 256 byte σε οποιοδήποτε συνδυασμό. Το block θεωρείται ότι είναι δομημένο σε 4, 6 ή 8 στήλες των 4 bytes, ανάλογα με το block size.

Ο αριθμός των κύκλων που πρέπει να ολοκληρωθούν για να πάρουμε το τελικό κρυπτογραφημένο input εξαρτάται από το μέγεθος του block και από το μέγεθος του κλειδιού.
Αν το μέγεθος του block και το μέγεθος του κλειδιού είναι και τα δύο 128 bits, τότε έχουμε 10 rounds, 9 κανονικούς και ένα μικρό round στο τέλος χωρίς εκτέλεση της MixColumn.

Αν ένα από τα Block ή Key sizes είναι 192, αλλά όχι 256 Bits τότε υπάρχουν 12 rounds. Αν ένα από τα block ή key sizes είναι 256 bits τότε έχουμε 14 rounds.
Ο Πίνακας 5.8 δίνει τον αριθμό των rounds που εκτελούνται για όλες τις περιπτώσεις μεγεθών κλειδιού και block.

Block size(bits)	Key size(bits)	Rounds
128	128	10
192	128	12
192	192	12
128	192	12
256	128	14
256	192	14
256	256	14
128	256	14
192	256	14

Πίνακας 5.8 Rounds που εκτελούνται ανά block και μέγεθος κλειδιου

Τα αρχεία εισόδου και εξόδου του benchmark:

Αρχείο εισόδου είναι ένα αρχείο δεδομένων με το κλειδί κρυπτογράφησης και αρχείο εξόδου είναι αυτό το αρχείο κρυπτογραφημένο με το κλειδί αποκρυπτογράφησης.

Πολυπλοκότητα Benchmark:

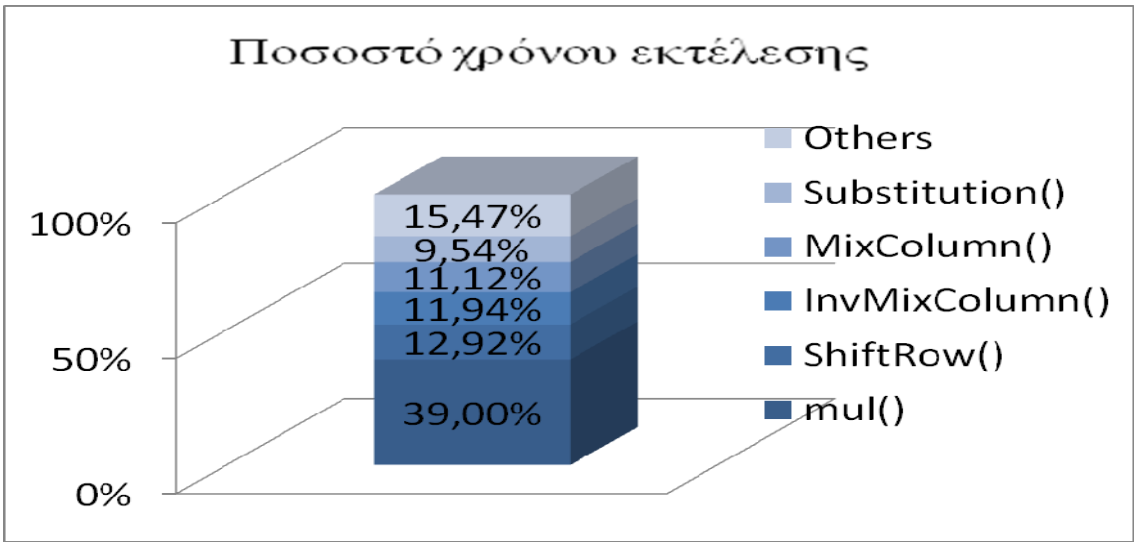
Η πολυπλοκότητα του RGB to CMYK conversion Benchmark εξαρτάται από τον αριθμό των επαναλήψεων που επιλέγουμε να τρέξει το benchmark μέσω του command line και από τον αριθμό των rounds που όπως προανέφερα εξαρτάται από το μέγεθος του κλειδιού και του block.

Ανάλυση χρόνου εκτέλεσης και των εντολών του συνολικού προγράμματος:

Μετρικά	**Συναρτήσεις**					
	Mul()	ShiftRow()	InvMixColumn()	MixColumn()	Substitution()	Others()
Ποσοστό χρόνου εκτέλεσης	39.00	12.92	11.94	11.12	9.54	15.47
Ποσοστό εντολών	41.41	12.59	11.98	12.03	7.70	14.29
CPI	0,703	0,766	0,720	0,741	0,920	------

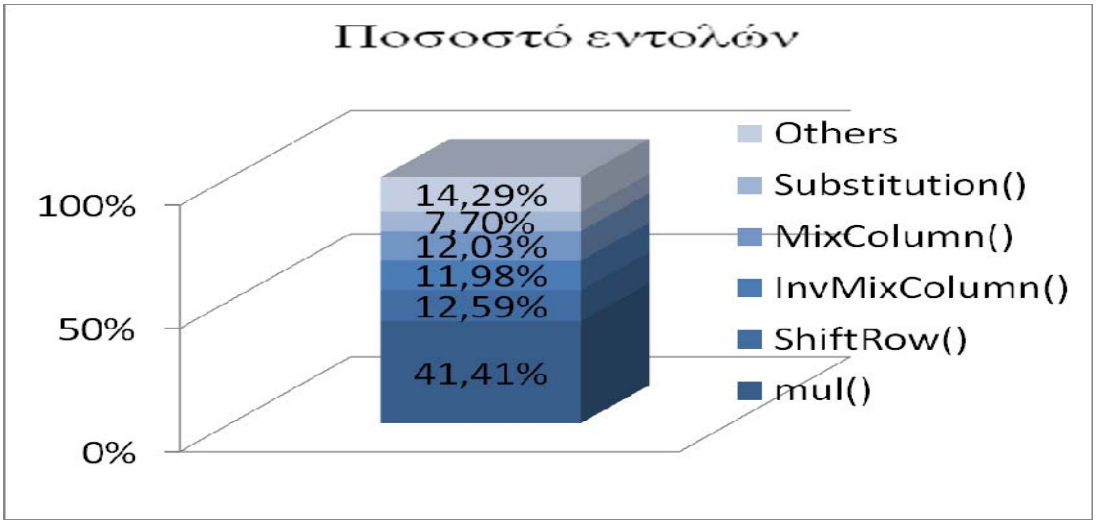
Πίνακας 5.9

Οι περισσότερο ενεργές functions του AES Benchmark είναι η mul() η οποία σπαταλά το 39% του συνολικού χρόνου που σπαταλείται στην εκτέλεση του AES Benchmark. Ακολουθεί με μεγάλη διαφορά η ShiftRow(), η οποία σπαταλά το 12,92%, η InvMixColumn() και η MixColumn() με 11,94% και 11,12% αντίστοιχα, οι δύο αυτές συναρτήσεις είναι λογικό να σπαταλούν περίπου τον ίδιο χρόνο εκτέλεσης αφού εκτελούν την ίδια δουλειά η μία από πάνω προς τα κάτω και η άλλη από κάτω προς τα πάνω. Τέλος έχουμε την Substitution() η οποία σπαταλά το 9,54% του συνολικού χρόνου εκτέλεσης. Όλες οι υπόλοιπες functions συνολικά καταναλώνουν το 15,47%. Από τον πιο πάνω πίνακα δημιούργησα τις εξής γραφικές παραστάσεις:



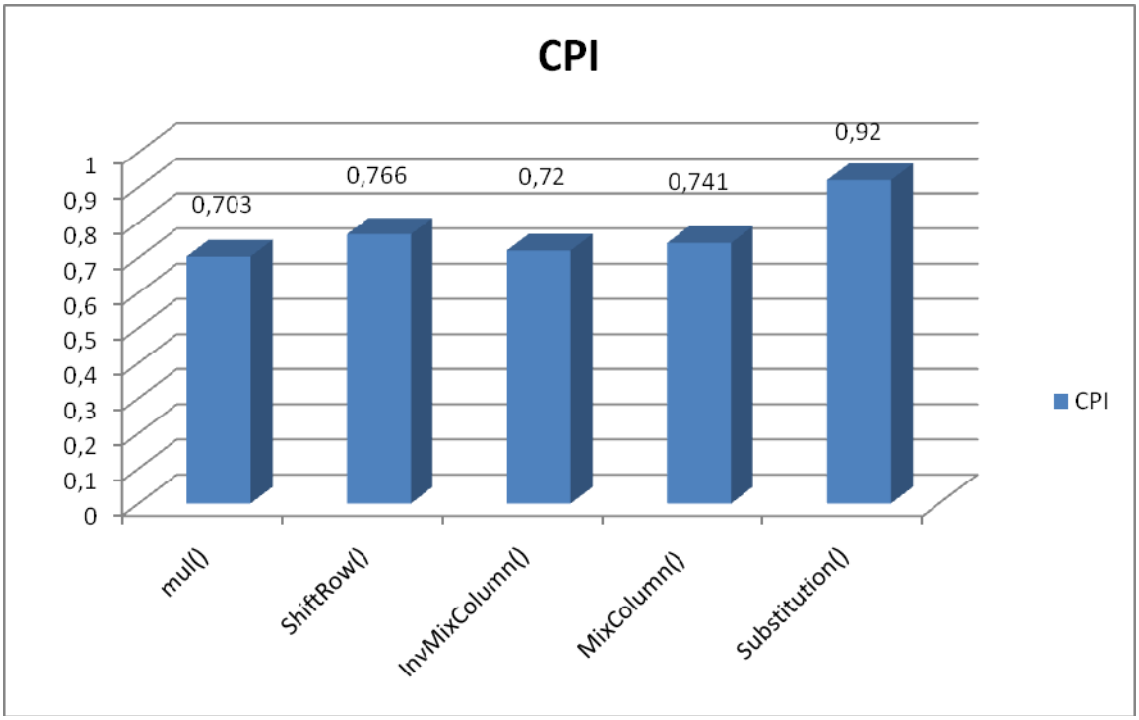
Σχήμα 5.28

Το περισσότερο ποσοστό χρόνου εκτέλεσης το καταναλώνει η συνάρτηση mul(). Η συνάρτηση αυτή είναι μια πολύ μικρή συνάρτηση η οποία καλείται από τις MixColumn() και InvMixColumn() πάρα πολλές φορές. Οι δύο αυτές συναρτήσεις έχουν μικρότερο ποσοστό χρόνου εκτέλεσης.



Σχήμα 5.29

Τις μεγαλύτερο ποσοστό εντολών το εκτελεί η συνάρτηση mul(). Η συνάρτηση mul() αφού καλείται πολλές φορές από άλλες συναρτήσεις πρέπει να εκτελέσει τις εντολές που περιλαμβάνει πάρα πολλές φορές.

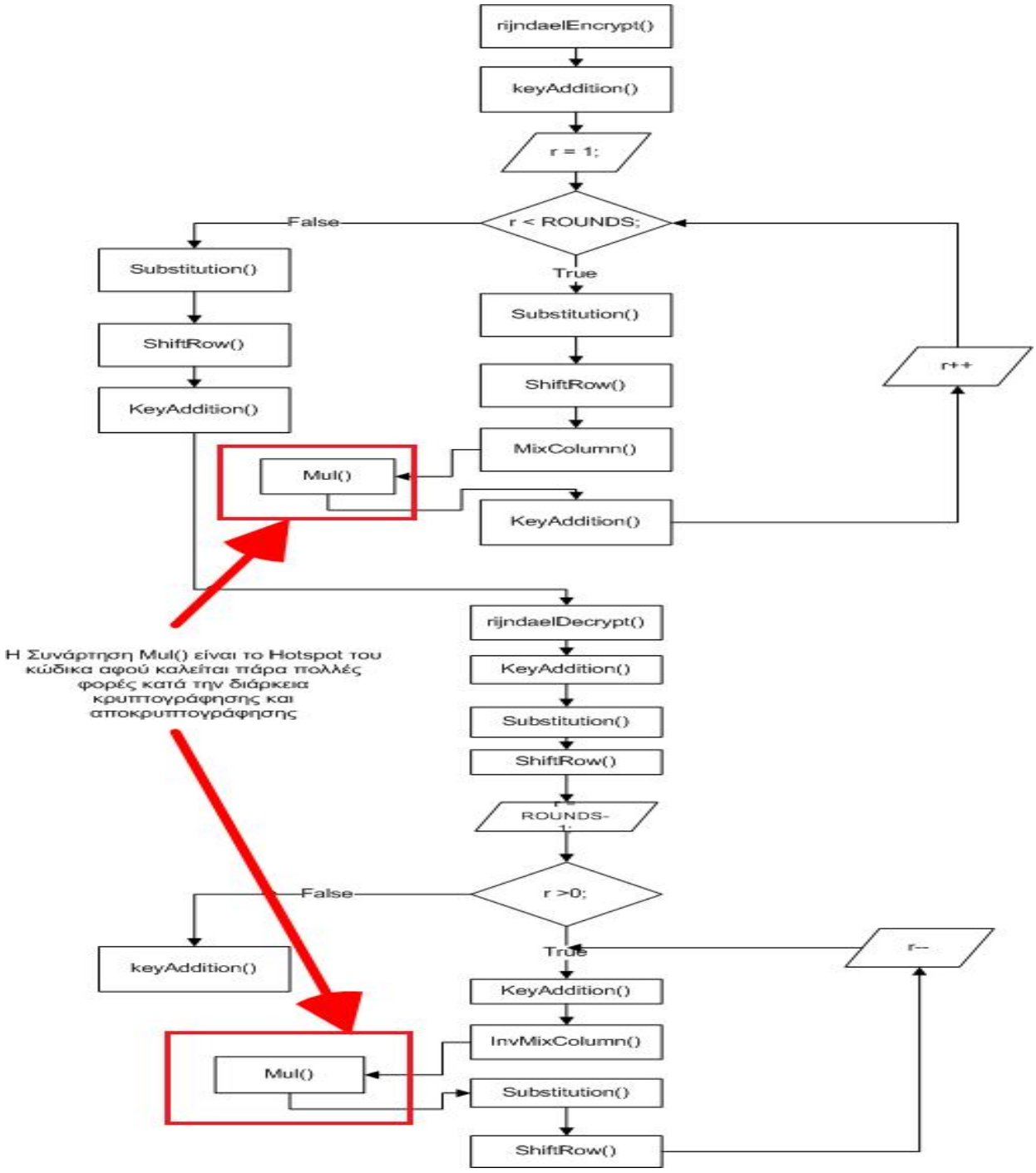


Σχήμα 5.30

Το CPI είναι διαφορετικό από συνάρτηση σε συνάρτηση. Αυτό συμβαίνει γιατί κάθε συνάρτηση εκτελεί διαφορετικό πλήθος εντολών καθώς και διαφορετικό είδος εντολών που μπορεί να χρειάζονται διαφορετικό χρόνο για να εκτελεστούν.

Βλέποντας τα αποτελέσματα του VTune και τις γραφικές παραστάσεις που προέκυψαν από αυτά αναμένω ότι το hotspot θα βρίσκεται στην συνάρτηση mul().

Έλεγχος ροής κώδικα και hotspot:



Σχήμα 5.31

Η συνάρτηση mul() καταναλώνει το μεγαλύτερο ποσοστό του συνολικού χρόνου εκτέλεσης της εφαρμογής του AES Benchmark γιατί είναι μια συνάρτηση η οποία καλείται πάρα πολλές φορές. Αν λάβουμε υπόψη ότι καλείται 2 φορές στη MixColumn και 4 φορές στην InvMixColumn και αυτές οι δύο καλούνται το λιγότερο από 10 φορές η κάθε μια σε κάθε επανάληψη τότε καταλαβαίνουμε ότι είναι μια function η οποία εκτελείται πάρα πολλές φορές. Επίσης εκτελεί δύσκολους μαθηματικούς υπολογισμούς αφού πολλαπλασιάζει δύο στοιχεία σε GF(2^m), οι υπολογισμοί αυτοί χρειάζονται μεγάλη υπολογιστική δύναμη εκ μέρους του επεξεργαστή.

Γι' αυτούς τους λόγους η συγκεκριμένη συνάρτηση δικαίως καταναλώνει ένα πολύ σημαντικό ποσοστό του χρόνου εκτέλεσης του προγράμματος.

Κεφάλαιο 6

Συμπεράσματα

6.1 Τεχνικά Συμπεράσματα

Μέσα από την έρευνα, μελέτη και ανάλυση των Digital Entertainment Benchmarks έχω διαπιστώσει ότι υπάρχουν περισσότεροι από ένα παράγοντες που καθορίζουν τη συμπεριφορά ενός προγράμματος και το χρόνο εκτέλεσής του.

Το hotspot, το κομμάτι του κώδικα δηλαδή με τον μεγαλύτερο χρόνο εκτέλεσης τις περισσότερες φορές είναι το πιο φωλιασμένο Loop του προγράμματος. Αυτό εξηγείται με το γεγονός ότι οι εντολές που πρέπει να ολοκληρωθούν για να εκτελεστεί το κομμάτι αυτό του κώδικα πολλαπλασιάζονται σε σχέση με τις υπόλοιπες και άρα ο κώδικας αυτός χρειάζεται περισσότερο χρόνο για να εκτελεστεί.

Το μέγεθος και το είδος του input παίζουν σημαντικό ρόλο ως προς το χρόνο εκτέλεσης του προγράμματος. Μεγαλύτερο input τις περισσότερες φορές απαιτεί και περισσότερο χρόνο εκτέλεσης. Αυτό είναι λογικό γιατί τα προγράμματα δουλεύουν πάνω σε αυτό το input και τις περισσότερες φορές με χρήση loop για να καλύψουν όλο το input. Άρα με μεγαλύτερο Input το loop τρέχει περισσότερες φορές, οι εντολές που πρέπει να ολοκληρωθούν είναι περισσότερες και ως συνέπεια αυξάνετε και ο χρόνος εκτέλεσης.

Υπάρχουν βέβαια και περιπτώσεις κατά τις οποίες περισσότερο χρόνο εκτέλεσης καταναλώνει ένα κομμάτι κώδικα που δεν απαιτεί πολλές εντολές για να ολοκληρωθεί, αλλά οι εντολές που πρέπει να εκτελεστούν απαιτούν περισσότερο χρόνο από άλλες.

Τότε πρέπει να ψάξουμε το λόγο που το καθιστά hotspot σε παράγοντες της μικροαρχιτεκτονικής. Για παράδειγμα όταν έχουμε προσβάσεις στη μνήμη, αυξάνεται το CPI αφού η πρόσβαση στη μνήμη κοστίζει αρκετό χρόνο στον επεξεργαστή και έτσι έχουμε ως αποτέλεσμα ο κώδικας αυτός να χρειάζεται πολύ χρόνο εκτέλεσης για να ολοκληρώσει τη λειτουργία του.

6.2 Μη τεχνικά συμπεράσματα

Κατά τη διάρκεια εκπόνησης της διπλωματικής μου εργασίας απέκτησα πολλές γνώσεις και ανέπτυξα διάφορες ικανότητες μου.

Μέσα από την τριβή μου με την διπλωματική κατάφερα να αναπτύξω τις ικανότητες μελέτης και έρευνας μου. Έμαθα να μελετώ και να ερευνώ σωστά ώστε να μπορώ να βρίσκω και να αποκτώ γνώσεις που χρειάζομαι χωρίς να χάνω χρόνο.

Μπόρεσα να αναπτύξω τις επαγγελματικές μου ικανότητες και σκέψεις. Για να μπορέσω να πάρω τα αποτελέσματα που ήθελα χρειάστηκα πολλές ώρες δουλειάς και σωστός χρονοπρογραμματισμός ώστε να μπορώ να αντεπεξέρχομαι στα deadlines που απαιτούσε ο επιβλέπων καθηγητής. Αυτό θα με βοηθήσει και στην μετέπειτα επαγγελματική μου καριέρα.

Επίσης μέσα από τις εμπειρίες μου απέκτησα υπομονή ώστε να μπορώ να ξεπερνώ τα διάφορα προβλήματα και εμπόδια που μου παρουσιάζονταν και να προχωρώ στον στόχο μου.

Το μεγαλύτερο πρόβλημα που συνάντησα κατά τη διάρκεια της εκπόνησης της διπλωματικής μου ήταν η μεταγλώττιση των benchmarks. Έπρεπε να τα μεταγλωττίσω έτσι ώστε να πάρω εκτελέσιμα με πληροφορίες που χρειάζεται το VTune για να μπορεί να τα αναλύσει σωστά. Χρειάστηκε πολλή μελέτη ώστε να μπορέσω να αλλάξω τα makefiles και να μπορέσω να πάρω τα σωστά εκτελέσιμα αρχεία.

Βιβλιογραφία

[1] David S. Bolme and Michele Strout and J. Ross Beveridge, “FacePerf: Benchmarks for Face Recognition Algorithms”

[2] Kenneth Hoste and Lieven Eechkout, “Comparing Benchmarks Using Key Micro architecture – Independent Characteristics”

[3] David A. Patterson and John L. Hennesssy, “Computer organization and design”

[4] http://en.wikipedia.org/wiki/Main_Page

[5] http://www.bdti.com/articles/dspdictionary.htm

[6] http://www.cygwin.com

[7] http://www.eembc.org/home.phpl

[8] http://www.intel.com

[9] http://www.mpeg.org

Παράρτημα Α[5]

ΓΛΩΣΣΑΡΙΟ:

Επιτραπέζιος Υπολογιστής- Desktop computer

Ένας Επιτραπέζιος Υπολογιστής είναι ένα προσωπικός Η/Υ (PC) σε μια μορφή προοριζόμενη για χρήση σε μια σταθερή θέση, σε αντίθεση με έναν φορητό υπολογιστή. Περιλαμβάνει σύνολο διαφορετικών συσκευών –οθόνη, ΚΜΕ, πληκτρολόγιο, ποντίκι κτλ-και όχι μια ενιαία όπως οι φορητοί υπολογιστές.

Εξυπηρετητής ή Κεντρικός Υπολογιστής- Server

Ένας υπολογιστής που χειρίζεται τα αιτήματα για τα δεδομένα, το ηλεκτρονικό ταχυδρομείο, τις μεταφορές αρχείων και άλλες υπηρεσίες δικτύου από άλλους υπολογιστές.

Ενσωματωμένο σύστημα- Embedded System

Ένα ηλεκτρονικό υπολογιστικό σύστημα που μπορεί να είναι τμήμα μιας μεγαλύτερης μηχανής ή ενός συστήματος. Τα ενσωματωμένα συστήματα μπορούν να αποκριθούν στα γεγονότα πραγματικού χρόνου. Οι περισσότερες ψηφιακές συσκευές, όπως τα ρολόγια ή τα αυτοκίνητα, χρησιμοποιούν ένα ενσωματωμένο σύστημα.

Μετασχηματιστής - Set-top box

Ένας μετασχηματιστής (STB) είναι μια συσκευή που συνδέει μια τηλεόραση και μια εξωτερική πηγή σήματος, και μετατρέπει το σήμα σε περιεχόμενο που επιδεικνύεται έπειτα στην τηλεοπτική οθόνη.

Τιμή PSNR

Είναι ένας όρος εφαρμοσμένης μηχανικής για την αναλογία μεταξύ της μέγιστης πιθανής δύναμης ενός σήματος και της δύναμης της αλλοίωσης του θορύβου που έχει επιπτώσεις στην ορθή αντιπροσώπευσή του. Επειδή πολλά σήματα έχουν μια πολύ ευρεία δυναμική περιοχή, το PSNR εκφράζεται συνήθως από την άποψη της λογαριθμικής decibel κλίμακας. Το PSNR συνηθέστερα χρησιμοποιείται ως μέτρο της ποιότητας της αναδημιουργίας στη συμπίεση εικόνας.

MPEG [9]

Η **MPEG** (Moving Pictures Experts Group) είναι μια ομάδα εργασίας ISO/IEC υπεύθυνη για την ανάπτυξη των τηλεοπτικών και ακουστικών προτύπων κωδικοποίησης.

MPEG -2: Πρότυπα τηλεοπτικά, ακουστικά και μεταφορικά, για την ποιοτική τηλεοπτική μετάδοση.
Χρησιμοποιείται για την πέρα από τον αέρα ψηφιακή τηλεόραση ATSC, DVB και ISDB, για τις ψηφιακές υπηρεσίες δορυφορικής τηλεόρασης, για τα ψηφιακά σήματα καλωδιακών τηλεοράσεων, SVCD, και με τις μικρές τροποποιήσεις, όπως τα .VOB (τηλεοπτικό αντικείμενο) αρχεία που φέρνουν τις εικόνες στα DVDs.

MPEG -4: Είναι μια συλλογή μεθόδων που καθορίζουν τη συμπίεση των ακουστικών και οπτικών (AV) ψηφιακών στοιχείων Υποστηρίζει τα τηλεοπτικά/ακουστικά αντικείμενα, τρισδιάστατα, χαμηλού ποσοστού κωδικοποιήσεις και υποστηρίζει επίσης την ψηφιακή διαχείριση δικαιωμάτων. Συμπεριλαμβάνονται διάφορα νέα τηλεοπτικά πρότυπα υψηλότερης αποδοτικότητας (νεώτερα από το βίντεο mpeg-2).

Hotspot ή Bottleneck

Είναι ένα τμήμα κώδικα στο οποίο εκτελείται μια σημαντική εργασία, η οποία εργασία ορίζεται ως ο χρόνος εκτέλεσης ή γεγονότα του επεξεργαστή τα οποία επηρεάζουν την επίδοση. Είναι δηλαδή τμήματα του κώδικα στα οποία σπαταλείται ο περισσότερος χρόνος εκτέλεσης και τα οποία αν βελτιστοποιηθούν τότε θα βελτιωθεί η επίδοση του προγράμματος. Πολλές φορές είναι αποτυχίες της μνήμης cache(cache misses) ή branch mispredictions.

Latency

Είναι ο αριθμός των κύκλων που παίρνει σε μια εντολή για να εκτελέσει το αποτέλεσμα της.

Thread

Το thread είναι μέρος ενός προγράμματος το οποίο μπορεί να τρέξει ανεξάρτητα χωρίς να βασίζεται σε άλλα μέρη του προγράμματος.