Ατομική Διπλωματική Εργασία

ΑΝΑΛΥΣΗ ΤΩΝ EEMBC NETWORKING VERSION 2.0 BENCHMARKS

Μαρία Ολυμπίου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ιούνιος 2009

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ανάλυση των EEMBC Networking version 2.0 benchmarks

Μαρία Ολυμπίου

Επιβλέπων Καθηγητής

Γιάννος Σαζεΐδης

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

Ιούνιος 2009

Ευχαριστίες

Σε αυτό το σημείο θα ήθελα να ευχαριστήσω τον κύριο Γιάννο Σαζεΐδη, που ήταν ο επιβλέπων καθηγητής μου σε αυτή τη διπλωματική εργασία. Η στήριξη και η βοήθεια που μου παρείχε από την αρχή της προσπάθειας μου, μέχρι και το τέλος ήταν πολύ σημαντική και καθοριστικής σημασίας για μένα.

Επίσης θα ήθελα να ευχαριστήσω τον διδακτορικό φοιτητή Μάριο Κλεάνθους, ο οποίος με βοήθησε στην εκμάθηση των εργαλείων που χρησιμοποίησα για τη διπλωματική μου εργασία.

Περίληψη

Το θέμα της διπλωματικής μου εργασίας αφορά την ανάλυση των EEMBC Networking Version 2.0 benchmarks.Τα benchmarks είναι προγράμματα που χρησιμοποιούνται για να αξιολογήσουν την επίδοση επεξεργαστών. Για αυτή τη διπλωματική εργασία προσπάθησα να εντοπίσω τα σημεία εκείνα στον κώδικα των προγραμμάτων, τα οποία καταναλώνουν τον περισσότερο από το χρόνο εκτέλεσης και να βρω τους λόγους για τους οποίους καταναλώνουν μεγάλο ποσοστό του χρόνου εκτέλεσης. Δηλαδή τα σημεία εκείνα του κώδικα που απαιτούν μεγάλη ποσότητα υπολογιστικών πόρων, από τον επεξεργαστή. Έπρεπε επίσης να εξηγήσω τη συμπεριφορά των προγραμμάτων για διάφορα σενάρια εκτέλεσης.

Η πρώτη φάση της διπλωματικής εργασίας ήταν το αφιερωμένη στην προετοιμασία έτσι ώστε να μπορέσω στη συνέχεια να αναλύσω τα προγράμματα. Περιείχε το διάβασμα επιστημονικών άρθρων, κεφαλαίων από διάφορα βιβλία Αρχιτεκτονικής Υπολογιστών όπως επίσης και την έρευνα για τα benchmarks με τα οποία θα ασχολούμουν στην συνέχεια.

Στη δεύτερη φάση, προχώρησα στην ανάλυση των προγραμμάτων που παρέλαβα.

Η ανάλυση των προγραμμάτων που αναφέρεται στη συνέχεια αυτού του δοκιμίου, επεκτείνεται σε 3 άξονες.

- Στην ανάλυση του κώδικα των προγραμμάτων όπου παρουσιάζονται οι δομές που χρησιμοποιούνται, οι αλγόριθμοι που τρέχει το κάθε πρόγραμμα και κάποια γενικά χαρακτηριστικά των προγραμμάτων.
- Στην ανάλυση των χρόνων εκτέλεσης τους με διάφορες τροποποιήσεις στα δεδομένα εισόδου
- Και στη μικροαρχιτεκτονική ανάλυση που έγινε με σκοπό την εύρεση των σημείων του κώδικα που σπαταλούν το μεγαλύτερο ποσοστό του χρόνου εκτέλεσης.

Για τα σημεία όπου ήταν εφικτό, έγινε και σύγκριση μεταξύ των διαφόρων μεγεθών που αφορούσαν τα δεδομένα εισόδου, με τη χρήση γραφικών παραστάσεων, πινάκων και άλλων διαγραμμάτων.

Στη συνέχεια, με βάση την ανάλυση που έκανα προέκυψαν κάποια συμπεράσματα, ως προς το πού αναλώνουν το χρόνο εκτέλεσης τους τα προγράμματα, τα οποία παραθέτονται σε αυτή την αναφορά.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Σκοπός της διπλωματικής εργασίας

Στα πλαίσια απόκτησης πτυχίου στο τμήμα Πληροφορικής, απαιτείται η εκπόνηση μιας διπλωματικής εργασίας η οποία αποτελεί μέρος της ευρύτερης έρευνας του τμήματος. Σκοπός δεν είναι μόνο η απόκτηση του πτυχίου αλλά και η απόκτηση καινούριων γνώσεων καθώς και η εμπειρία διεξαγωγής προσωπικής έρευνας.

1.2 Θέματα που καλύπτονται

Το κυριότερο θέμα της εργασίας αυτής είναι η εύρεση των σημείων στον κώδικα των EEMBC Networking V2.0 benchmarks όπου σπαταλείται το μεγαλύτερο ποσοστό του χρόνου εκτέλεσης και η ανάλυση των λόγων για τους οποίους σπαταλείται τόσος χρόνος σε αυτά τα σημεία. Τα σημεία αυτά, αναφέρονται στη συνέχεια ως hot spots. Χρησιμοποιήθηκαν δύο μέθοδοι για την εύρεση αυτών των σημείων. Η στατική ανάλυση και η δυναμική ανάλυση. Κατά τη στατική ανάλυση έγινε χρήση διαγραμμάτων ροής που αναπαριστούν τη σειρά κλήσης των συναρτήσεων στα προγράμματα.

Κατά τη δυναμική ανάλυση έγινε η χρήση του εργαλείου Intel VTune Performance Analyzer. Το εργαλείο αυτό κάνει δυναμικά ανάλυση των υπολογιστικών πόρων που χρησιμοποιούνται κατά την εκτέλεση μιας εφαρμογής. Επιπλέον επιτρέπει την εύρεση

των hot spots της εφαρμογής, εμφανίζοντας τον κώδικα των σημείων αυτών. Για το συγκεκριμένο εργαλείο υπάρχει εκτενέστερη περιγραφή στη συνέχεια.

Άλλα θέματα που καλύπτονται σε αυτή τη διπλωματική εργασία, αφορούν την ευρύτερη περιοχή της Αρχιτεκτονικής Υπολογιστών και ήταν απαραίτητο να συμπεριληφθούν για να είναι εφικτή η ανάλυση της συμπεριφοράς των προγραμμάτων, βάσει χαρακτηριστικών μικροαρχιτεκτονικής.

Σε αυτό το σημείο, παραθέτονται περιληπτικά τα θέματα που καλύπτει το κάθε κεφάλαιο του εγγράφου αυτού.

Στο κεφάλαιο 2 παρουσιάζονται κάποιοι γενικοί ορισμοί οι οποίοι χρησιμοποιούνται σε αυτή την εργασία. Επίσης παρουσιάζονται οι περιλήψεις κάποιων επιστημονικών άρθρων που καλύπτουν θέματα σχετικά με το θέμα αυτής της διπλωματικής εργασίας.

Στο κεφάλαιο 3 παρουσιάζεται ένας γενικός χαρακτηρισμός των EEMBC Networking Version 2.0 benchmarks τα οποία ανέλυσα. Σε αυτό το χαρακτηρισμό περιλαμβάνεται ο αλγόριθμος που υλοποιεί το κάθε benchmark, η περιγραφή των δομών που χρησιμοποιούνται και οι υπολογιστικοί πόροι τους οποίους εξετάζει το benchmark πάνω στον επεξεργαστή στον οποίο τρέχει.

Στο κεφάλαιο 4 παρουσιάζεται η μεθοδολογία ανάλυσης την οποία χρησιμοποίησα για να αναλύσω τα προγράμματα. Επίσης παρουσιάζονται κάποια χαρακτηριστικά του εργαλείου VTune, το οποίο ήταν το εργαλείο που χρησιμοποίησα για την εύρεση των hot spots.

Στο κεφάλαιο 5 παρουσιάζεται η ανάλυση των benchmarks. Για το κάθε benchmark παρουσιάζονται τα πιο κάτω:

- Εφαρμογή που υλοποιεί το benchmark
- Περιγραφή του benchmark
- Ανάλυση των υπολογιστικών πόρων που εξετάζει το benchmark
- Input και output του benchmark

- Δομές δεδομένων που χρησιμοποιούνται
- Λειτουργίες που εκτελεί ο κώδικας του benchmark
- Ενδεικτικό διάγραμμα ροής του benchmark (στατική ανάλυση)
- Περιγραφή των συναρτήσεων που εμφανίζονται στο διάγραμμα ροής
- Συνοπτικός πίνακας που παρουσιάζει τα αποτελέσματα από το χρόνο εκτέλεσης του προγράμματος όπως προκύπτει από το Cygwin και τα αποτελέσματα της ανάλυσης στο VTune (δυναμική ανάλυση).
- Γραφικές παραστάσεις που δείχνουν διαγραμματικά την κατανομή των τιμών στο συνοπτικό πίνακα
- Γραφικές παραστάσεις που δείχνουν διαγραμματικά την κατανομή του χρόνου εκτέλεσης στις συναρτήσεις του προγράμματος
- Για τις συναρτήσεις που καταναλώνουν τον περισσότερο από το χρόνο εκτέλεσης, παρουσιάζεται ο κώδικας των σημείων που προκαλούν καθυστέρηση και επεξηγείται η αιτία της καθυστέρησης.
- Στο τέλος της ανάλυσης του κάθε προγράμματος παρουσιάζονται κάποια συμπεράσματα που προκύπτουν από την ανάλυση.

Στο κεφάλαιο 6, παρουσιάζονται τα γενικά τεχνικά συμπεράσματα που προέκυψαν από αυτή τη διπλωματική εργασία καθώς και τα μη τεχνικά συμπεράσματα που αφορούν γενικά τις εμπειρίες που αποκόμισα από αυτή την εργασία. Τα συμπεράσματα που έχουν προκύψει από την ανάλυση που έκανα ήταν ότι το hot spot των πλείστων από τα προγράμματα που ανέλυσα βρίσκεται συνήθως σε βρόγχους ή σε σημεία που γίνονται συχνά προσβάσεις στη μνήμη. Επιπλέον είχα παρατηρήσει σε πολλές περιπτώσεις πως πράξεις όπως shifting που χρειάζονται πολλούς κύκλους μηχανής για να ολοκληρωθούν, αυξάνουν το συνολικό CPI των προγραμμάτων.

Στο παράρτημα Α παρουσιάζεται ένας οδηγός εγκατάστασης, μεταγλώττισης και ανάλυσης των benchmarks.

Κεφάλαιο 2

Γενικοί Ορισμοί και σχετική δουλειά

2.1 Τι είναι τα benchmarks

Το benchmark είναι ένα πρόγραμμα – δοκιμή το οποίο τρέχει πάνω σε επεξεργαστές για να παραχθούν διάφορα συμπεράσματα τα οποία αφορούν κυρίως την επίδοση του εν λόγω επεξεργαστή βάσει κάποιων μετρικών. Υπάρχουν και σύνολα από benchmarks τα οποία ανήκουν στην ίδια κατηγορία και τα οποία όταν χρησιμοποιηθούν σε συνδυασμό παράγουν αποτελέσματα για τη συνολική επίδοση των επεξεργαστών [3,4]

Υπάρχουν διάφορα είδη από benchmarks τα οποία έχουν να κάνουν με εφαρμογές διαφόρων τύπων όπως για παράδειγμα Networking, Real Program, Kernel κτλ
Στην εργασία αυτή θα αναλυθούν τα EEMBC Networking V2.0 benchmarks. Αυτά τα benchmarks αναπαριστούν λειτουργίες που τρέχουν στα διάφορα επίπεδα (layers) του δικτύου, και χρησιμοποιούνται για να ελέγχουν την επίδοση ενσωματωμένων επεξεργαστών σε δρομολογητές (network processors).

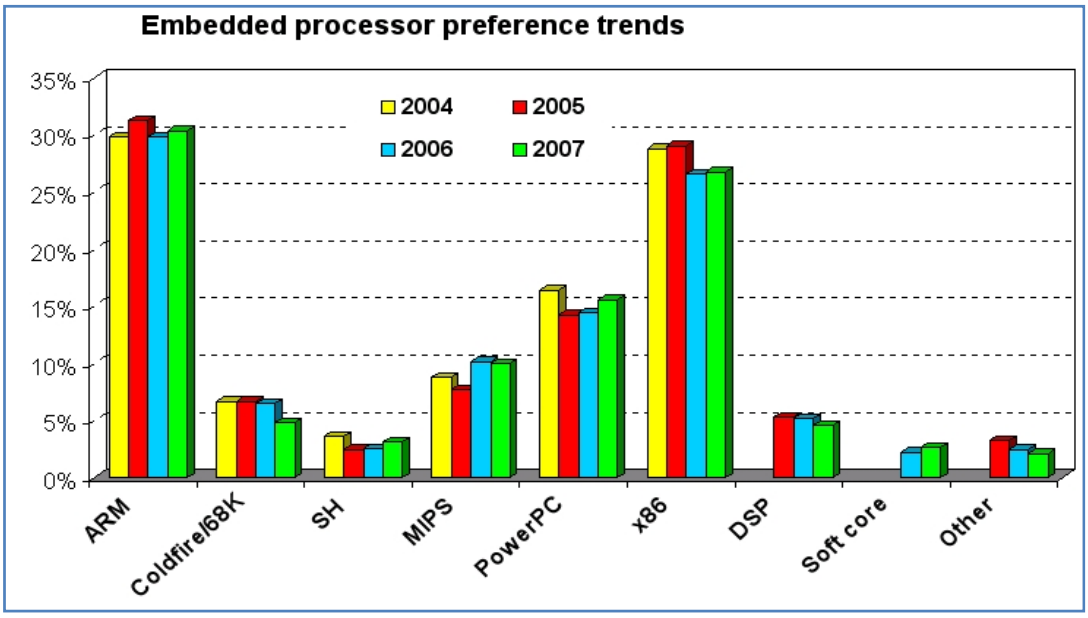
2.2 Ενσωματωμένοι επεξεργαστές

Περιλαμβάνουν ενσωματωμένους μικροεπεξεργαστές που τρέχουν σε διάφορες μηχανές, για παράδειγμα σε πλυντήρια, αυτοκίνητα, κινητά τηλέφωνα κτλ

Τα ενσωματωμένα υπολογιστικά συστήματα (embedded computing systems) είναι σχεδιασμένα για να τρέχουν μια εφαρμογή ή ένα σύνολο από εφαρμογές, οι οποίες σχετίζονται μεταξύ τους και μαζί με το υλικό (hardware) του υπολογιστή αποτελούν ένα απλό υπολογιστικό σύστημα. [3,4]

Οι απαιτήσεις των ενσωματωμένων επεξεργαστών όσον αφορά τους υπολογιστικούς πόρους είναι πολύ μικρότερες σε σχέση με εκείνες των επεξεργαστών γενικής χρήσης. Αυτό συμβαίνει γιατί οι εφαρμογές που τρέχουν είναι πολύ συγκεκριμένες και επιτελούν ένα πολύ μικρό αριθμό λειτουργιών και έτσι δε χρειάζεται παροχή μεγάλης ποσότητας υπολογιστικών πόρων. Επίσης λόγω του ότι το κόστος ενός ενσωματωμένου επεξεργαστή είναι πολύ περιορισμένο εάν προστεθούν υπολογιστικοί πόροι, αυτό θα έχει ως αποτέλεσμα την αύξηση του κόστους του επεξεργαστή.

Πιο κάτω φαίνεται ένα διάγραμμα στο οποίο παρουσιάζονται στοιχεία για την αγορά επεξεργαστών για τα έτη 2004 - 2007.



Εικόνα 2.1: Διάγραμμα για την αγορά επεξεργαστών για τα έτη 2004 - 2007 [8]

Στο διάγραμμα ο άξονας Χ αναπαριστά εταιρίες κατασκευής επεξεργαστών, ενώ ο άξονας Υ αναπαριστά το ποσοστό της αγοράς που κατείχε η κάθε εταιρία για τα έτη 2004 με 2007. Όπως παρατηρείται, η εταιρία ARM η οποία κατασκευάζει ενσωματωμένους επεξεργαστές, κατείχε το μεγαλύτερο ποσοστό της αγοράς

επεξεργαστών για αυτό το χρονικό διάστημα, γεγονός που αποδεικνύει το εύρος της χρήσης των ενσωματωμένων επεξεργαστών, αφού έχουν την πρώτη θέση στις πωλήσεις επεξεργαστών.

2.3 Χαρακτηρισμός της επίδοσης (Performance Characterization)

Τα χαρακτηριστικά επίδοσης ή performance characteristics είναι κάποια χαρακτηριστικά τα οποία κανείς πρέπει να λαμβάνει συνολικά υπόψη όταν πρόκειται να αξιολογήσει την επίδοση κάποιου ενσωματωμένου επεξεργαστή.

- **Τι είναι:** Ο χαρακτηρισμός της επίδοσης κάποιου προγράμματος είναι ο χαρακτηρισμός της ταχύτητας κάποιου επεξεργαστή βάσει κάποιων μετρικών ή χαρακτηριστικών οι οποίες σε συνδυασμό μας δίνουν την ταχύτητα στην οποία ο επεξεργαστής αυτός τρέχει κάποια προγράμματα. Για να χαρακτηρίσουμε την επίδοση κάποιου επεξεργαστή πρέπει να λαμβάνονται υπόψη όλες οι μετρικές γιατί εάν αγνοήσουμε κάποια από αυτές τότε τα αποτελέσματα που θα εξάγουμε για την επίδοση του, σε αυτή την περίπτωση δεν θα είναι αντιπροσωπευτικά. Παλιότερα το μόνο που ενδιέφερε τους σχεδιαστές ήταν ο χρόνος στο οποίο ένας επεξεργαστής έτρεχε ένα πρόγραμμα. Τώρα όμως τους ενδιαφέρουν και άλλες μετρικές οι οποίες καταγράφονται στη συνέχεια.

- **Γιατί το κάνουμε:** Ο λόγος για τον οποίο χαρακτηρίζεται η επίδοση διαφόρων συστημάτων είναι είτε για να συγκριθούν με άλλα συστήματα του ίδιου είδους που τρέχουν τις ίδιες εφαρμογές, είτε για σκοπούς προώθησης στην αγορά, είτε για βελτίωση των συστημάτων αυτών από τους σχεδιαστές τους.

- **Πώς το χρησιμοποιούμε:** Το χαρακτηρισμό της επίδοσης τον χρησιμοποιούμε είτε σαν αγοραστές για να επιλέξουμε κάποιο σύστημα για αγορά, είτε ως σχεδιαστές για να βελτιώσουμε κάποιο σύστημα το οποίο εμείς σχεδιάσαμε είτε ως ερευνητές για να συγκρίνουμε διάφορα συστήματα μεταξύ τους.

- **Ποια είναι τα χαρακτηριστικά του:** Τα διάφορα χαρακτηριστικά τα οποία λαμβάνουμε υπόψη για να χαρακτηρίσουμε την απόδοση κάποιου επεξεργαστή είναι τα εξής:

    - Ενέργεια που καταναλώνει
    - Κύκλοι μηχανής ανά μονάδα χρόνου
    - Αριθμός εντολών που εκτελούνται για κάποιο πρόγραμμα
    - Κύκλοι μηχανής που χρειάζονται για να εκτελεστεί μια εντολή
    - Συνολικός χρόνος CPU που χρειάζεται για να τρέξει το πρόγραμμα

Βέβαια αναλόγως του επεξεργαστή που εξετάζουμε υπάρχουν και άλλα χαρακτηριστικά τα οποία πρέπει να λάβουμε υπόψη όπως τα πιο κάτω:

- Χρόνος απόκρισης όταν έχουμε πρόσβαση στη μνήμη
- Χρόνος απόκρισης όταν χρησιμοποιούνται συσκευές Ε/Ε (Εισόδου Εξόδου)

- **Γιατί μας ενδιαφέρει:** Μας ενδιαφέρει αφού μπορεί να είναι μέτρο σύγκρισης διαφόρων επεξεργαστών, για τους οποίους αν πάρουμε τα αποτελέσματα για την επίδοση τους μεμονωμένα, δε θα μπορούμε να τους συγκρίνουμε. Επίσης είναι πολύ σημαντικό για τους σχεδιαστές να γίνεται χαρακτηρισμός της επίδοσης των συστημάτων που σχεδιάζουν γιατί μπορεί να λαμβάνουν υπόψη κάποιους συγκεκριμένους παράγοντες για βελτίωση της επίδοσης και να αγνοούν άλλους με αποτέλεσμα να μην βελτιώνονται στο σύνολο τα συστήματα τους, αλλά μόνο σε ορισμένες πτυχές. [3,4]

2.4 Επίδοση για ενσωματωμένους επεξεργαστές

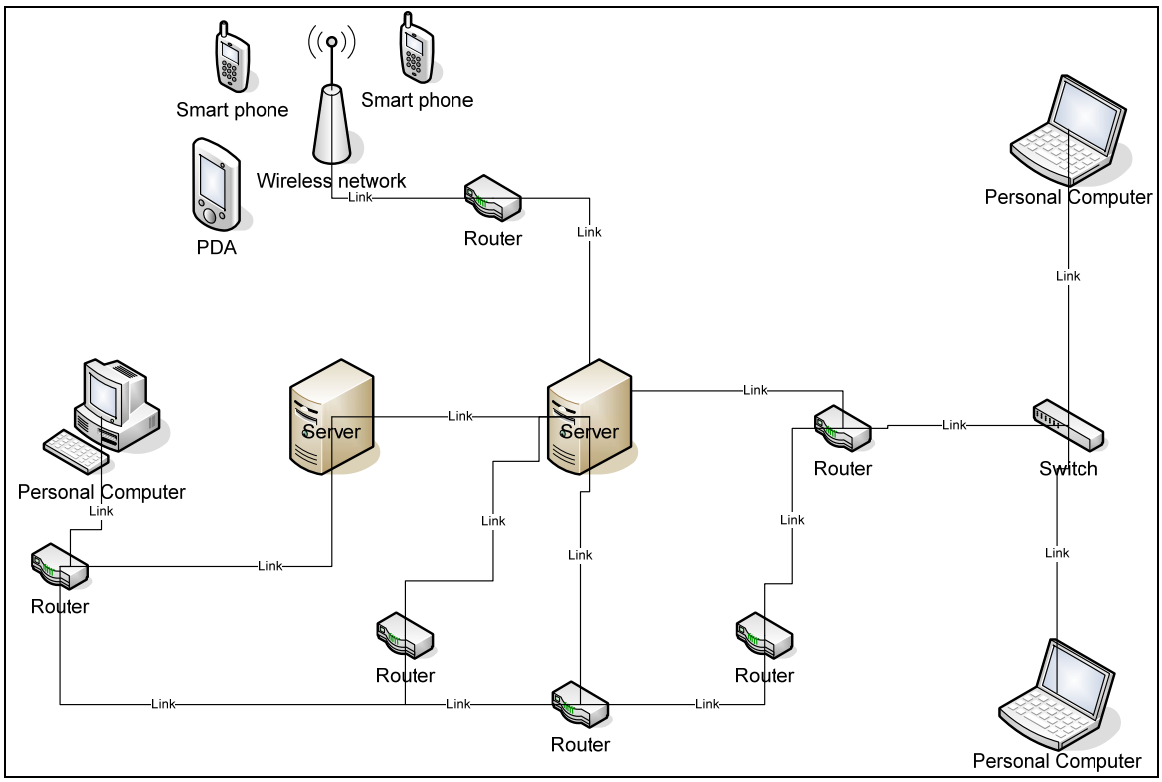
Η επίδοση για τους ενσωματωμένους επεξεργαστές χαρακτηρίζεται συνήθως από κάποιους περιορισμούς πραγματικού χρόνου. Οι περιορισμοί πραγματικού χρόνου χωρίζονται σε 2 κατηγορίες. Τους Hard Real Time και τους Soft Real Time. Οι hard

real time περιορισμοί ορίζουν πως μια ενέργεια από το σύστημα πρέπει να γίνει σε πολύ συγκεκριμένο και προκαθορισμένο χρόνο (για παράδειγμα οι αερόσακοι κάποιου αυτοκινήτου πρέπει να ανοίξουν αμέσως μετά τη σύγκρουση του). Οι soft real time περιορισμοί ορίζουν πως κάποια ενέργεια στο σύστημα μπορεί να γίνει μέσα σε κάποιο χρονικό πλαίσιο αλλά με πιο χαλαρά πλαίσια χρόνου, και όχι τόσο αυστηρά προκαθορισμένα όπως συμβαίνει με τους hard real time περιορισμούς. [3,4]

2.5 Αρχιτεκτονική δικτύου υπολογιστών

Στη συγκεκριμένη διπλωματική εργασία θα αναλυθούν πολλές από τις λειτουργίες των δικτύων υπολογιστών αφού αυτές εξετάζει η κατηγορία από benchmarks την οποία επιλέξαμε για ανάλυση.

Η πιο κάτω εικόνα δείχνει ένα μικρό δείγμα από την τοπολογία ή αρχιτεκτονική κάποιου δικτύου



Εικόνα 2.2: Αρχιτεκτονική δικτύου υπολογιστών και κινητών συσκευών

Όπως δείχνει και η εικόνα, στο διαδίκτυο (internet) υπάρχουν πολλά υποδίκτυα που αποτελούνται από δρομολογητές. Όλα τα επιμέρους υποδίκτυα, συνθέτουν μαζί το διαδίκτυο. Στα δίκτυα αυτά μπορούμε να συναντήσουμε servers αλλά και προσωπικούς υπολογιστές οι οποίοι και αποτελούν τους end users, δηλαδή τα άκρα του δικτύου. Όπως παρατηρούμε εκτός από ενσύρματες συνδέσεις, υπάρχουν και ασύρματες συνδέσεις οι οποίες επιτρέπουν σε διαφόρων ειδών ασύρματες συσκευές να ενώνονται με το δίκτυο.

Οι δρομολογητές και τα switches εκτελούν την ίδια εργασία, δηλαδή την προώθηση των πακέτων στο δίκτυο. Παρόλα αυτά στο network core χρησιμοποιούνται δρομολογητές και όχι switches, γιατί τα switches μπορούν να προωθήσουν πακέτα μόνο σε απόσταση 100 μέτρων ενώ οι δρομολογητές μπορούν να προωθήσουν πακέτα σε πολύ μεγαλύτερες αποστάσεις, ακόμα και μεταξύ 2 διαφορετικών χωρών.
Επίσης είναι σημαντικό να αναφέρουμε ότι οι δρομολογητές και τα switches δεν διαμοιράζουν την ταχύτητα στους χρήστες που είναι ενωμένοι πάνω τους αλλά στέλνουν πακέτα προς όλους τους χρήστες με την ίδια ταχύτητα, δηλαδή με την ταχύτητα με την οποία παραλαμβάνουν πακέτα. Αυτή η ιδιότητα τους είναι σημαντική στο να γίνεται αποδοτική αποστολή και παραλαβή πακέτων. Επίσης είναι σημαντική στο να διατηρούνται υψηλές ταχύτητες μέσα στο δίκτυο. Αντίστοιχο των switches είναι το hub, μια συσκευή η οποία όμως διαμοιράζει την ταχύτητα στους υπολογιστές που είναι ενωμένοι με αυτή.

2.6 Σχετική δουλειά

Στο άρθρο «**Characterizing the effect of Micro architecture Design Parameters on Workload Dynamic Behaviour [1]**», οι συγγραφείς αναφέρουν πως η δυναμική συμπεριφορά των προγραμμάτων είναι το πρώτο βήμα για αποδοτική εναλλαγή πόρων. Στη μεθοδολογία τους χρησιμοποιούν τα wavelets. Τα wavelets είναι μαθηματικά εργαλεία τα οποία χρησιμοποιούν μια συνάρτηση πρωτοτύπου για να μετατρέπουν τα δεδομένα που τους ενδιαφέρουν σε διαφορετικούς συντελεστές συχνοτήτων και έπειτα για να αναλύσουν τον κάθε συντελεστή με ανάλυση που ταιριάζει στην κλίμακά του. Η ανάλυση με τα wavelets επιτρέπει την επιλογή του συνδυασμού της κλίμακας και φίλτρου από τα wavelets, μέσα από ένα αριθμό συναρτήσεων. Επίσης χρησιμοποιούν

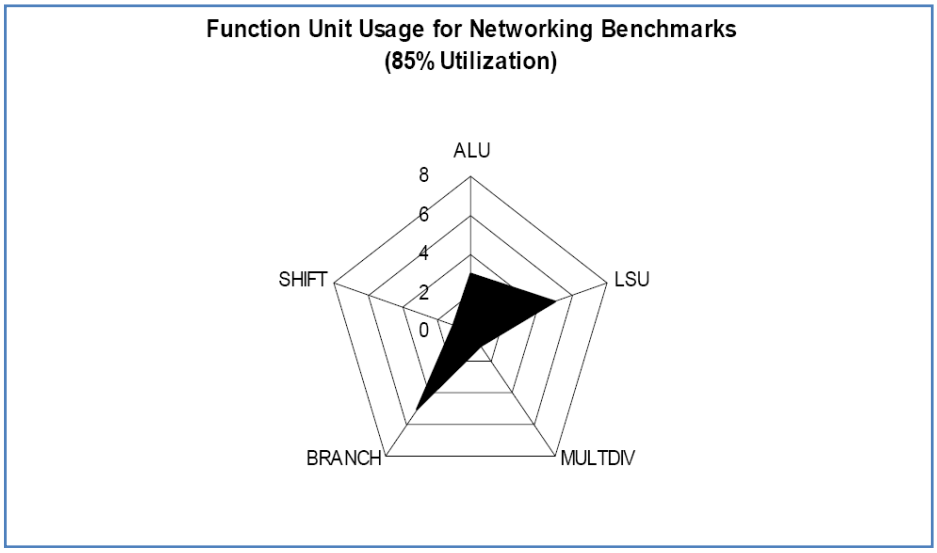
τη μέθοδο του linear regression είναι μια μέθοδος η οποία καθορίζει τη σχέση μεταξύ των αναφερόμενων μεταβλητών και των παραμέτρων εισόδου.

Οι συγγραφείς του άρθρου **«Comparing Benchmarks Using Key Micro architecture – Independent Characteristics [5]»** χρησιμοποιούν διάφορα χαρακτηριστικά μικροαρχιτεκτονικής για να συγκρίνουν κάποια benchmarks μεταξύ τους. Για τη μελέτη τους θεωρούν ότι τα πάντα στον επεξεργαστή είναι τέλεια και απεριόριστα εκτός από το window size το οποίο δίνεται. Θεωρούν δηλαδή ότι έχουν τέλειες μνήμες cache, τέλεια πρόβλεψη διακλάδωσης και απεριόριστο αριθμό από λειτουργικά τμήματα. Χρησιμοποίησαν για την ανάλυση τους κάποια χαρακτηριστικά τα οποία ήταν σχετικά με τους καταχωρητές, άρα και σχετικά με τη μικροαρχιτεκτονική και κάποια χαρακτηριστικά που είναι ανεξάρτητα της μικροαρχιτεκτονικής. Το συμπέρασμα που έβγαλαν από αυτή τη μελέτη έδειξε ότι οι συγκρίσεις που έγιναν με χαρακτηριστικά που είναι ανεξάρτητα τις μικροαρχιτεκτονικής είναι καλύτερες, γιατί παράγουν αποτελέσματα που μπορούν να επεκταθούν και για άλλες μικροαρχιτεκτονικές. Αντίθετα οι συγκρίσεις που έγιναν με χαρακτηριστικά που εξαρτώνται από τη μικροαρχιτεκτονική δεν είναι τόσο καλές γιατί δεν είναι επεκτάσιμες.

Στο άρθρο **«FacePerf: Benchmarks for Face Recognition Algorithms [2]»,** περιγράφονται 3 αλγόριθμοι για face recognition. Οι αλγόριθμοι που χρησιμοποιούν είναι πολύ γρήγοροι και εκτός από face recognition μπορούν να χρησιμοποιηθούν και για real time συστήματα. Σύμφωνα με τους συγγραφείς, πιο αποδοτικές υλοποιήσεις των αλγορίθμων για αναγνώριση προσώπων θα μπορούσαν να οδηγήσουν στη δημιουργία καλύτερων συστημάτων εικόνας. Οι συγγραφείς χρησιμοποίησαν 3 αλγόριθμους οι οποίοι εκτελούν διαφορετικά ήδη υπολογισμών μεταξύ τους και έχουν όλοι open source υλοποιήσεις. Ο κώδικας αυτών των 3 αλγορίθμων, τους οποίους χρησιμοποίησαν για τη μελέτη τους προτείνεται για performance benchmarking και είχε τροποποιηθεί αποκλειστικά για τους σκοπούς της μελέτης τους. Μετά από ανάλυση των 3 αλγορίθμων, καθόρισαν το bottleneck της επίδοσης για την υλοποίηση του κάθε αλγορίθμου. Για να το βρουν έκαναν profiling των 3 αλγορίθμων χρησιμοποιώντας το εργαλείο gprof. 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.
Για να βελτιώσουν αυτή την κατανάλωση χρόνου, μείωσαν το εύρος των τιμών που μπορεί να πάρει το συνημίτονο, από (-1, 1) σε (-1, 0) ή σε (0, 1). Με αυτό τον τρόπο, περιόρισαν το πεδίο τιμών που μπορούσε να πάρει το συνημίτονο και ταυτόχρονα το χρόνο υπολογισμού του.

Στο άρθρο «**Understanding the EEMBC Benchmark Suite [6]**» ο συγγραφέας του άρθρου συνέλεξε τα χαρακτηριστικά διαφόρων αρχιτεκτονικών και ο συνδυασμός των μετρικών που προέκυψε από την εκτέλεση του κάθε benchmark πάνω σε αυτές, έδωσε μια ακριβή αναπαράσταση για την δραστηριότητα του workload. Έτσι οι πελάτες που ενδιαφέρονται για κάποιο benchmark μπορούν να συγκρίνουν το workload με το δικό τους και να δουν εάν ταιριάζει για τους σκοπούς που το θέλουν.
Για να μετρήσουν την επίδοση χρησιμοποίησαν διάφορα χαρακτηριστικά. Μερικά από αυτά εξαρτούνται από την αρχιτεκτονική ενώ άλλα είναι ανεξάρτητα της αρχιτεκτονικής. Επίσης γίνεται μια σύγκριση των διαφόρων ομάδων από benchmarks μεταξύ τους. Χρησιμοποίησαν διάφορες παραμέτρους για να καθορίσουν τα χαρακτηριστικά της cache είναι το block size, το set associativity και το total size. Η επίδοση της μνήμης cache είχε κάποιες διαφοροποιήσεις που ήταν σχετικές με τις αλλαγές που γίνονταν στις παραμέτρους αυτές. Επίσης προσομοίωσαν μια ιδεατή μηχανή η οποία έτρεχε τις εντολές out-of-order. Οι εντολές μπορούσαν να έχουν απεριόριστο μήκος και επίσης θεωρούσαν ότι είχαν ένα τέλειο branch predictor. Έτσι το μόνο bottleneck, ήταν οι εξαρτήσεις που είχαν μεταξύ τους οι εντολές και καθόριζαν τον ελάχιστο χρόνο εκτέλεσης του προγράμματος. Κατά την εμπειρική τους ανάλυση έκαναν διάφορες προσομοιώσεις. Για να αναπαραστήσουν τις μετρικές των προσομοιώσεών τους χρησιμοποίησαν τα kiviat graphs που μπορούν να αναπαραστήσουν διαφορετικές μεταβλητές με τις τιμές τους σε ένα μόνο γράφο.
Το πιο κάτω Kiviat graph, δείχνει τα χαρακτηριστικά της προσομοίωσης που έγινε για τα Networking Benchmarks και τον αριθμό των λειτουργικών μονάδων από το κάθε είδος που χρειάστηκαν για την εκτέλεση των προγραμμάτων.



Εικόνα 2.3: Kiviat Graph που περιγράφει γενικά τις απαιτήσεις λειτουργικών μονάδων που έχουν τα EEMBC Networking benchmarks

Όπως φαίνεται από το kiviat graph τα networking benchmarks έχουν το μεγαλύτερο ποσοστό από branches και απαιτούν το μεγαλύτερο ποσοστό από λειτουργικές μονάδες (functional units) από όλες τις κατηγορίες των benchmarks.Λόγω του ότι τα networking benchmarks έχουν παρόμοιο workload τα kiviat graphs τους έχουν παρόμοια μορφή.

Ο branch predictor έχει μεγάλο ποσοστό ακρίβειας για τα περισσότερα benchmarks. Το συμπέρασμα που προκύπτει από το άρθρο είναι ότι υπάρχει ποικιλομορφία ανάμεσα στα benchmarks και έτσι οι σχεδιαστές με τη χρήση διαφόρων ειδών από benchmarks μπορούν να ελέγξουν διάφορες πτυχές της επίδοσης των συστημάτων που σχεδιάζουν.

Κεφάλαιο 3

Χαρακτηρισμός των EEMBC Networking V2.0 Benchmarks

3.1 Γενικές πληροφορίες για το EEMBC

Το EEMBC είναι ένας μη κερδοσκοπικός οργανισμός ο οποίος ιδρύθηκε το 1997 με σκοπό τη δημιουργία benchmarks για την αξιολόγηση της επίδοσης του υλικού και του λογισμικού ενσωματωμένων συστημάτων.

Το ακρωνύμιο EEMBC αντιστοιχεί στο Embedded Microprocessor Benchmark Consortium.

Τα benchmarks του EEMBC αναπαριστούν πραγματικές εφαρμογές που χρησιμοποιούνται για να ελέγξουν την επίδοση ενσωματωμένων συστημάτων και έτσι βοηθούν τους σχεδιαστές να επιλέξουν ποιος είναι ο πιο κατάλληλος ενσωματωμένος επεξεργαστής για το σύστημα τους.

Η μεθοδολογία αξιολόγησης είναι συγκεκριμένη για όλους τους επεξεργαστές και αυτό εγγυάται πως από τα αποτελέσματα που προκύπτουν θα μπορούν να γίνουν σωστές συγκρίσεις των επεξεργαστών που ελέγχονται.

Επίσης αξίζει να αναφερθεί πως το EEMBC παρέχει benchmarks για διάφορες κατηγορίες εφαρμογών όπως για παράδειγμα εφαρμογές δικτύων, Java, Digital Entertainment κτλ και έτσι οι σχεδιαστές μπορούν να συγκρίνουν την επίδοση των επεξεργαστών τους για διάφορες πραγματικές εφαρμογές.

3.2 Γενικές πληροφορίες για τα EEMBC Networking V2.0 benchmarks

Πρόκειται για μια ομάδα από benchmarks, τα οποία έχουν ως σκοπό την αξιολόγηση της επίδοσης διαφόρων διεργασιών που τρέχουν στα δίκτυα υπολογιστών.

Πολλά από τα benchmarks της συγκεκριμένης ομάδας έχουν να κάνουν με πρωτόκολλα που τρέχουν στα διάφορα layers του διαδικτύου και ελέγχουν την επίδοση των επεξεργαστών τους οποίους ελέγχουν βάσει αυτών. Σε αρκετές από τις περιπτώσεις έχουμε προσομοιώσεις λειτουργιών που τρέχουν στους δρομολογητές (routers) και οι οποίοι είναι υπεύθυνοι για την προώθηση των πακέτων μέσα σε ένα δίκτυο και από δίκτυο σε δίκτυο.

Πιο κάτω γίνεται μια σύντομη περιγραφή των λειτουργιών που εκτελεί το κάθε benchmark στην οποία αργότερα θα προστεθούν και λεπτομέρειες για τα αποτελέσματα που προκύπτουν με βάση τον κώδικα των benchmarks. Η πιο περιγραφή γίνεται με βάση τη βιβλιογραφία [7].

3.3 Περιγραφή του benchmark IP Packet Check

Το benchmark αυτό προσομοιώνει ένα δρομολογητή (router) με 4 network interfaces διαφορετικών ταχυτήτων μεταξύ τους. Ασχολείται με πράξεις υπολογισμού του checksum όπως και με λογικές πράξεις σύγκρισης.

Εφαρμογή

Το benchmark αυτό υλοποιεί ένα υποσύνολο των λειτουργιών που εκτελεί η συνάρτηση προώθησης (forwarding function) των πακέτων στο network layer του Internet Protocol suite όπως ορίζεται από το RFC1812. Η κανονική λειτουργία ενός TCP/IP router ορίζει πως ο router αυτός εξετάζει το IP header των πακέτων που παραλαμβάνει, τα αποθηκεύει σε ένα προσωρινό χώρο αποθήκευσης και βάσει της ετικέτας (header) που έχουν, τα στέλνει παρακάτω σε κάποιο άλλο δρομολογητή στο δίκτυο μέχρι να φτάσουν στον προορισμό τους. Συγκεκριμένα όταν ένας router παραλάβει ένα πακέτο αφαιρεί το header του και βάζει ένα καινούριο link layer header (που είναι το επόμενο layer του διαδικτύου στο οποίο θα προωθηθεί). Η πιο πάνω διαδικασία στο συγκεκριμένο benchmark υποθέτουμε ότι έχει ήδη γίνει.

Περιγραφή του benchmark

Για το κάθε πακέτο (datagram) υπολογίζεται ένα checksum μόνο για την IP ετικέτα (header) του και αποθηκεύεται σε εκείνο. Το checksum του IP header του πακέτου υπολογίζεται, γιατί σε περίπτωση λάθους το πακέτο διαγράφεται από την προσωρινή μνήμη του δρομολογητή (router) και δεν στέλνεται παρακάτω. Το μέγεθος των πακέτων επιλέγεται τυχαία έτσι ώστε να αναπαριστάται με τον καλύτερο δυνατό τρόπο η κατάσταση που υπάρχει στα πραγματικά δίκτυα υπολογιστών. Επίσης έχουμε υλοποίηση 2 ουρών. Η μια ουρά είναι η ουρά **receive_queue,** η οποία κρατά τα πακέτα που καταφθάνουν για εξυπηρέτηση στο δρομολογητή και η άλλη είναι η **holding_queue,** η οποία κρατά τα πακέτα τα οποία θα σταλούν παρακάτω σε άλλους δρομολογητές.

Πιο κάτω φαίνονται οι έλεγχοι που κάνει το benchmark για το κάθε πακέτο που παραλαμβάνει:

- Το μέγεθος του πακέτου να είναι μεγαλύτερο από 20 bytes
- Το checksum είναι ορθό
- Το version του IP protocol είναι το 4

- Το μήκος του IP header είναι αρκετά μεγάλο για να κρατά το ελάχιστο μέγεθος κάποιου IP πακέτου.
- Το συνολικό μέγεθος του πεδίου IP είναι αρκετά μεγάλο για να κρατά το header του πακέτου του οποίου το μέγεθος καθορίζεται από το μήκος του IP header.

Για να ολοκληρωθεί μια επανάληψη (ένα iteration) του κώδικα πρέπει να αδειάσει η ουρά **receive_queue.**

Ανάλυση των υπολογιστικών πόρων

Οι πράξεις που εκτελούνται στο συγκεκριμένο benchmark είναι integer math με 16 bit unsigned, πράξεις shift και πράξεις λογικών συγκρίσεων.
Το benchmark αυτό ελέγχει και την ανάκτηση δεδομένων από τη μνήμη αφού οι πράξεις που εκτελούνται από το συγκεκριμένο benchmark ανακτούν δεδομένα από τη μνήμη.

3.4 Περιγραφή του benchmark IP Packet Network Address Translator (NAT)

Αυτό το benchmark είναι βασισμένο στο NetBSD Kernel Code και ελέγχει την αποδοτικότητα της μνήμης cache και τον χρόνο απόκρισής της .

Εφαρμογή

Το NAT είναι μια τεχνολογία η οποία επιτρέπει στους δρομολογητές να ομαδοποιούν διευθύνσεις από ένα δίκτυο σε κάποιο άλλο και να στέλνουν τα πακέτα σε μια μόνο IP διεύθυνση και αυτά να πηγαίνουν στο σωστό παραλήπτη που ανήκει στο δίκτυο στο οποίο στέλνονται τα πακέτα. Τα IP header checksums είναι ανανεωμένα ώστε να δίνουν τη διεύθυνση που προκύπτει από τη μετάφραση.

Περιγραφή του benchmark

Το συγκεκριμένο benchmark επικεντρώνεται στη διαχείριση των πακέτων τα οποία φεύγουν από ένα δίκτυο και οδηγούνται σε ένα άλλο δίκτυο. Όταν ένα πακέτο παραληφθεί από το δίκτυο, το benchmark αυτό εξακριβώνει εάν πρέπει να γίνει κάποια διεργασία. Υπάρχει ένας hash table 128 εισόδων, ο οποίος κρατά πληροφορίες για τις υφιστάμενες συνδέσεις. Το πρόγραμμα λαμβάνοντας υπόψη κάποιους παράγοντες όπως διεύθυνση αποστολέα, διεύθυνση παραλήπτη και πρωτόκολλο του πακέτου υπολογίζει μια διεύθυνση στο πίνακα στην οποία και θα τοποθετήσει το πακέτο. Αν το πακέτο που παραλήφθηκε ανήκει σε μια σύνδεση η οποία ήδη υπάρχει, τότε το πακέτο αυτό τοποθετείται στη συγκεκριμένη θέση του πίνακα και καταχωρείται ως πακέτο αυτής της σύνδεσης. Αν η σύνδεση δεν υπάρχει καταχωρημένη στον πίνακα, τότε δημιουργείται μια νέα καταχώρηση στον πίνακα για τη συγκεκριμένη σύνδεση έτσι ώστε να διαχειρίζεται μελλοντικά τα πακέτα που καταφθάνουν για αυτή τη σύνδεση.

Ανάλυση των υπολογιστικών πόρων

Από αυτό το benchmark γίνεται έλεγχος του χρόνου αντίδρασης της μνήμης cache. Η ύπαρξη του hash table και οι αναζητήσεις που γίνονται σε αυτό ελέγχουν την ικανότητα των επεξεργαστών να ανακτούν δεδομένα από τη μνήμη. Επίσης συγκρίνει την ικανότητα τους να επιλέγουν τη σωστή διακλάδωση στον κώδικα βάσει προβλέψεων και επιπλέον την ικανότητα επαναφοράς τους μετά από κάποιο σφάλμα.

3.5 Περιγραφή του benchmark OSPF

Αυτό το benchmark ελέγχει τη σχετική απόδοση των δρομολογητών

Εφαρμογή

Αυτό το benchmark υλοποιεί τον αλγόριθμο του Dijkstra ο οποίος επιλέγει το συντομότερο ή το πιο φθηνό σε κόστος μονοπάτι από ένα σύνολο μονοπατιών από τον αναφερόμενο δρομολογητή, μια μέθοδος η οποία χρησιμοποιείται ευρέως στα δίκτυα υπολογιστών.

Περιγραφή του benchmark

Ο αλγόριθμος του Dijkstra επιλέγει το συντομότερο ή το πιο φθηνό σε κόστος μονοπάτι από ένα σύνολο μονοπατιών τα οποία γνωρίζει ο αναφερόμενος δρομολογητής. Δημιουργείται ένας πίνακας στον οποίο αποθηκεύεται ο κάθε κόμβος για τον οποίο ο υφιστάμενος κόμβος γνωρίζει. Οι κόμβοι είναι οι υπόλοιποι δρομολογητές για τους οποίους γνωρίζει πληροφορίες ο δρομολογητής που μας ενδιαφέρει. Ο κάθε κόμβος έχει ένα σύνολο από κατευθυνόμενες ακμές οι οποίες αναπαριστούν τη σύνδεση του με κάποιο άλλο δρομολογητή. Η κάθε ακμή έχει κάποια τιμή που αναπαριστά το κόστος της που είναι και το κόστος της ένωσης των 2 δρομολογητών, με λίγα λόγια το κόστος αποστολής δεδομένων από τον ένα δρομολογητή στον άλλο.

Ο αλγόριθμος του Dijkstra ξεκινά από τον αναφερόμενο ως βάση κόμβο και υπολογίζοντας κάθε φορά το μικρότερο κόστος βρίσκει την πιο φθηνή διαδρομή μέσω της οποίας θα κινηθούν τα πακέτα. Οι πίνακες με τους δρομολογητές σε αυτό το benchmark κτίζονται δυναμικά.

Ανάλυση των υπολογιστικών πόρων

Σημαντικοί παράγοντες για αυτό το benchmark είναι η ικανότητα του επεξεργαστή να ανακτά δεδομένα από τη μνήμη και επίσης η ικανότητα του να εκτελεί συχνά CTI (Control Transfer Instructions).

3.6 Περιγραφή του benchmark QoS (Quality of Service)

Εφαρμογή

Το benchmark αυτό είναι βασισμένο στον NetBSD kernel code. Προσομοιώνει τη διαδικασία που εκτελείται από το σύστημα διαχείρισης του bandwidth το οποίο χρησιμοποιείται ώστε οι ροές των πακέτων να ικανοποιούν τις απαιτήσεις του Quality of Service (QoS).

Περιγραφή του benchmark

Στο συγκεκριμένο benchmark υπάρχουν 2 δομές το pipe και το queue. Τα πακέτα που εξέρχονται από τον router καταλήγουν σε μια από τις 2 πιο πάνω δομές. Το queue είναι μια ουρά με σταθερό μέγεθος και πολιτική FIFO, όπως ορίζεται για μια ουρά. Το pipe είναι μια προσομοίωση ενός link με σταθερό bandwidth. Το pipe σχετίζεται με ένα ή περισσότερα queues όπου όλα τα πακέτα για αυτό το pipe αποθηκεύονται σε εκείνο το queue. Το bandwidth που σχετίζεται με αυτό το pipe διαμοιράζεται στα queues με τα οποία σχετίζεται. Τα πακέτα που εξέρχονται, αποθηκεύονται σε ένα από τα heaps που υπάρχουν έτσι ώστε ο scheduler να ξέρει πότε να προωθήσει το πακέτο.

Ανάλυση των υπολογιστικών πόρων

Το συγκεκριμένο benchmark είναι «πιεστικό» με θέματα μνήμης. Υπάρχουν μεγάλες ανάγκες σε μνήμη αφού υπάρχουν test data πολύ μεγάλου μεγέθους. Εξετάζεται όμως και ο χρόνος αντίδρασης της μνήμης αφού γίνονται συχνές προσβάσεις στη μνήμη.

3.7 Περιγραφή του benchmark Route Lookup

Εφαρμογή

Αυτό το benchmark είναι υλοποίηση μιας βασικής λειτουργίας την οποία εκτελούν οι δρομολογητές, της παραλαβής και αποστολής πακέτων στο δίκτυο.

Περιγραφή του benchmark

Όλοι οι δρομολογητές κρατούν ένα πίνακα με πληροφορίες ο οποίος τους λέει σε ποια πύλη (port) να στείλουν ένα πακέτο που έχουν παραλάβει έτσι ώστε το πακέτο αυτό να προωθηθεί παρακάτω στο δίκτυο. Ο μηχανισμός για το IP lookup υλοποιείται βάσει του Patricia tree. Το δέντρο αυτό είναι ένα συμπαγές δυαδικό δέντρο το οποίο επιτρέπει γρήγορες και αποδοτικές αναζητήσεις βάσει συμβολοσειρών απεριόριστου μήκους.

Ανάλυση των υπολογιστικών πόρων

Ο κώδικας αυτού του benchmark επαναληπτικά διατρέχει το δέντρο. Συνεπώς εξετάζεται ο χρόνος απόκρισης του επεξεργαστή για ανάκτηση δεδομένων και η ικανότητα του να διαχειρίζεται αποδοτικά συχνές Control Transfer Instructions πράξεις.

3.8 Περιγραφή του benchmark Transmission Control Protocol (TCP)

Το benchmark αυτό ασχολείται με το πιο συχνά χρησιμοποιούμενο και το πιο δύσκολο κομμάτι από πλευράς επεξεργαστή του πρωτοκόλλου TCP. Προσομοιώνει τα χαρακτηριστικά του εν λόγω πρωτοκόλλου σε συνθήκες πραγματικών δικτύων. Επίσης κάνει χρήση 3 διαφορετικών συνόλων δεδομένων για να προσομοιώνει μια ποικιλία από workloads. (traffics)

Εφαρμογή

Η ικανότητα ενός ενσωματωμένου επεξεργαστή να διαχειρίζεται το TCP πρωτόκολλο είναι σημαντική για να αποφεύγονται τα bottlenecks στα δίκτυα. Σε αντίθεση με άλλα πρωτόκολλα, το TCP πρωτόκολλο μπορεί να εξυπηρετηθεί από επεξεργαστές γενικού σκοπού. Το TCP είναι εύκολα προσαρμόσιμο, τόσο που χρησιμοποιείται και σε ασύρματες και σε ενσύρματες εφαρμογές. Το μοντέλο αναφοράς ISO, το οποίο χρησιμοποιείται σαν σημείο αναφοράς όταν πρόκειται για διαχωρισμό των πρωτοκόλλων σε επίπεδα, θεωρεί πως το επίπεδο του πρωτοκόλλου TCP βρίσκεται στην κορυφή του IP (Internet Protocol) και κάτω από το επίπεδο εφαρμογών (Application layer).

Ο σκοπός του IP είναι να παρέχει τα μέσα για τη διακίνηση των TCP πακέτων ανάμεσα σε δίκτυα που επικοινωνούν μεταξύ τους.

Μερικά βασικά χαρακτηριστικά του πρωτοκόλλου TCP είναι τα πιο κάτω:

- Βασική μεταφορά δεδομένων
- Σίγουρη μεταφορά των δεδομένων από σημείο σε σημείο (Αξιοπιστία)

- Έλεγχος ροής
- Πολυπλεκτικότητα (Multiplexing)
- Συνδέσεις μεταξύ δικτύων
- Προτεραιότητα και ασφάλεια

Η βασική δουλειά που κάνει το TCP πρωτόκολλο είναι να μεταφέρει δεδομένα μεταξύ 2 σημείων που βρίσκονται στα άκρα του δικτύου. Περιλαμβάνονται πράξεις ελέγχου και δεδομένων. Το πρωτόκολλο αυτό απαιτεί αυστηρό έλεγχο και σηματοδότηση των πακέτων για να μπορεί να επιτευχθεί η αξιοπιστία, η αποδοτικότητα και η διαχείριση των συνδέσεων την οποία προσφέρει.

Περιγραφή του benchmark

Το benchmark αυτό μετρά την απόδοση της διαχείρισης των δεδομένων και των προσωρινών χώρων αποθήκευσης. Οι πράξεις αυτές είναι συνήθεις, αλλά και πολύ ακριβές στο TCP πρωτόκολλο.

Το benchmark για το TCP το οποίο ανήκει στην EEMBC μπορεί να ικανοποιήσει τις πιο κάτω απαιτήσεις:

- **Ακριβές:** Το benchmark ελέγχει όλες τις βασικές λειτουργίες που εκτελούνται στο TCP με τους όρους του κόστους επεξεργασίας
- **Ρεαλιστικό:** Προσομοιώνονται τα χαρακτηριστικά του TCP traffic σε πραγματικά δίκτυα
- **Ντετερμινιστικό:** Δεν συνδέει την ταχύτητα του επεξεργαστή με οποιαδήποτε τυχαία επιλογή τιμής γίνεται στο πρωτόκολλο
- **Απλοϊκό:** Επιτρέπει τη δοκιμή για μια απλή εφαρμογή TCP με λογικές υποθέσεις

Οι μετρικές απόδοσης του benchmark αυτού περιλαμβάνουν

- Πλήρης event-driven TCP state machine και διαχειριστή συνδέσεων
- Διαχείριση μνήμης προσωρινής αποθήκευσης

- Διαχειριστή ουρών
- Ξεχωριστή διαδικασία επαναληπτικής εισδοχής client – server με εναλλαγή περιβάλλοντος
- Βασικός έλεγχος ροής
- Κατανομή πακέτων διαφόρων μεγεθών για διαφορετικά είδη traffic
- Για να αντεπεξέλθει με διάφορα σενάρια το benchmark αυτό είναι διαμορφώσιμο

Ανάλυση των υπολογιστικών πόρων

Τα δεδομένα που στέλνονται μαζικά ελέγχουν το μέγιστο μέγεθος πακέτου που μπορεί να σταλεί. Ο αριθμός των πακέτων που στέλνονται μπορεί να εφαρμοστεί σε στη βάση του Gigabit Ethernet. Επίσης το μείγμα διαφόρων μεγεθών των πακέτων που στέλνονται είναι μια μέση περίπτωση για το Standard Ethernet.

Το κάθε workload προσομοιώνει το traffic σε ένα δίκτυο χρησιμοποιώντας τα πιο κάτω βήματα:

- Αρχικοποίηση του server (εξυπηρετητή)
- Είσοδος ενός network channel
- Αρχικοποίηση του client (πελάτη)
- Είσοδος των λειτουργιών του client στο network channel

Αυτό το workload επαναλαμβάνεται μέχρι να κλείσουν οι συνδέσεις όλων των clients.

3.9 Συνοπτικός πίνακας με τα χαρακτηριστικά των benchmarks

Benchmark name	Εφαρμογή που υλοποιείται	Περιγραφή του benchmark	Υπολογιστικοί πόροι που εξετάζει το benchmark
IP Packet Check	Υποσύνολο των λειτουργιών που εκτελεί η συνάρτηση προώθησης (forwarding function) των πακέτων στο network layer του Internet Protocol.	Για κάθε πακέτο που παραλαμβάνεται γίνονται διάφοροι έλεγχοι ορθότητας για να μπορεί να προωθηθεί παρακάτω στο δίκτυο.	Integer math με 16 bit unsigned, πράξεις shift, πράξεις λογικών συγκρίσεων και ανάκτηση δεδομένων από τη μνήμη
IP Network Address Translator	Αποστολή πακέτων από ένα κόμβο κάποιου υποδικτύου σε κάποιο κόμβο κάποιου άλλου υποδικτύου.	Χρήση ενός hash table 128 εισόδων ο οποίος κρατά τις συνδέσεις από τις οποίες καταφθάνουν τα πακέτα.	Χρόνος απόκρισης της μνήμης cache, χρόνος ανάκτησης δεδομένων από τη μνήμη, branch predictability, γρήγορη επαναφορά μετά από λάθος πρόβλεψη.
OSPF	Υλοποίηση του αλγόριθμου του Dijkstra.	Γράφος που αναπαριστά δρομολογητές και συνδέσεις μεταξύ τους.	Ανάκτηση δεδομένων από τη μνήμη και ικανότητα εκτέλεσης συχνών CTI (Control Transfer Instructions).
QoS	Σύστημα διαχείρισης του bandwidth. Οι ροές των πακέτων πρέπει να ανταποκρίνονται στο Quality of Service	Δομές pipe και queue. Συσχετισμός του pipe ένα ή περισσότερα queues. Διαμοιρασμός του bandwidth στο queue.	Έλεγχος για μεγάλες ανάγκες σε μνήμη, χρόνος απόκρισης της μνήμης.
Route Lookup	Διαδικασία αποστολής και παραλαβής πακέτων στο δίκτυο.	Μηχανισμός αναζήτησης του σωστού Port. Χρήση του Patricia tree.	Χρόνος για ανάκτηση δεδομένων από τη μνήμη, ικανότητα αποδοτικής διαχείρισης συχνών Control Transfer Instructions.
Transmission Control Protocol (TCP)	Υλοποίηση των λειτουργιών που εκτελούνται από το πρωτόκολλο TCP. Αξιόπιστη μεταφορά πακέτων στο δίκτυο.	Προσομοίωση του traffic σε ένα δίκτυο με μια σειρά από βήματα.	Διαχείρισης δεδομένων και προσωρινών χώρων αποθήκευσης. Διαχείριση μνήμης προσωρινής αποθήκευσης.

Πίνακας 3.1: Συνοπτικός πίνακας για όλα τα EEMBC Networking V2.0 Benchmarks

3.10 Υπολογισμός των βαθμολογιών των επεξεργαστών για τα benchmarks

Τα scores που προκύπτουν για τον κάθε επεξεργαστή από το Networking Version 2.0, χωρίζονται σε 2 κατηγορίες, ανάλογα με το benchmark το οποίο εξετάζει τον επεξεργαστή. Τα scores αυτά ονομάζονται **IPmark και TCPmark**. Το IPmark απευθύνεται στους κατασκευαστές του εξοπλισμού υποδομής των δικτύων, δηλαδή στους κατασκευαστές δρομολογητών, switches κτλ. Το TCPmark επικεντρώνεται σε client-server based υλικό (hardware), δηλαδή σε υπολογιστές που βρίσκονται στα άκρα των δικτύων ή σε serves.

Πιο κάτω περιγράφεται το πώς υπολογίζεται το κάθε score ανάλογα με τα benchmarks που συμμετέχουν σε αυτό.

Υπολογισμός του IP mark

Είναι ο γεωμετρικός μέσος των scores που προκύπτουν για τα QoS, Route lookup, OSPF, IP reassembly και Network Address Translation μαζί με το γεωμετρικό μέσο των scores του IP packet check benchmarks, όλα διαιρούμενα με το 10.
Πιο κάτω φαίνεται η μαθηματική εξίσωση για υπολογισμό του IP mark.

GEOMEAN_IP_PCK_CH = Geomean (IP Packet Check [0.5MB], IP Packet Check [1MB], IP Packet Check [2MB], IP Packet Check [4MB])

IPmark = Geomean (GEOMEAN_IP_PCK_CH, QoS, Route Lookup, OSPF, IP Reassembly, NAT) / 10

Υπολογισμός του TCP mark

Είναι ο γεωμετρικός μέσος του TCPjumbo, του TCPbulk και του TCPmixed, όλα μαζί διαιρούμενα με το 100

TCPmark = Geomean (TCP jumbo, TCP bulk, TCP Mixed) / 100

Ο γεωμετρικός μέσος υπολογίζεται όπως δείχνει η πιο κάτω εξίσωση

$$\sqrt[nth]{a * b * c * d \ldots}$$

Τα scores που παρουσιάζονται αντιπροσωπεύουν iterations per second που ολοκληρώνονται στον επεξεργαστή όταν τρέχει κάποιο benchmark.

Silicon Scores

Τα silicon scores αναφέρονται για ορισμένες κατηγορίες προϊόντων όπως είναι οι πιο κάτω:

a) Παραγωγή επεξεργαστών σιλικόνης,
b) Προ-παραγωγή επεξεργαστών σιλικόνης και
c) Chips για έλεγχο πρωτοτύπων για FPGA.

Simulation scores

Τα simulation scores αναφέρονται για κατηγορίες προϊόντων όπως είναι οι πιο κάτω:

a) Προσομοιωτές, εξομοιωτές,
b) Πραγματικό υλικό (hardware), με σκοπό την πρόβλεψη της απόδοσης μελλοντικών εφαρμογών σιλικόνης.

Κεφάλαιο 4

Μεθοδολογία ανάλυσης

4.1 Γενική περιγραφή της μεθοδολογίας ανάλυσης

Σε αυτή την ενότητα περιγράφεται η μεθοδολογία που χρησιμοποιήθηκε για την ανάλυση των προγραμμάτων. Σε γενικές γραμμές για να αναλύσω τα προγράμματα χρειάστηκε να εγκαταστήσω στον υπολογιστή μου τα προγράμματα Cygwin και VTune. Επίσης ακολούθησα κάποιες ειδικές διαδικασίες κατά τη μεταγλώττιση, για να διατηρούνται κάποιες επιπρόσθετες πληροφορίες για τα προγράμματα έτσι ώστε αυτά να μπορούν να αναλυθούν στο VTune. Πιο κάτω αναφέρονται αναλυτικά τα βήματα της μεθοδολογίας που ακολούθησα.

4.2 Εισαγωγή στο θέμα της διπλωματικής εργασίας

Αρχικά διάβασα το data book [7] που συνοδεύει τα benchmarks. Σε αυτό αναφέρονται γενικές πληροφορίες για το κάθε benchmark οι οποίες στη συνέχεια είναι σημαντικές για να μπορούμε να αναλύσουμε τα προγράμματα. Οι πληροφορίες αυτές αφορούν τη διαδικασία από τα πραγματικά δίκτυα που προσομοιώνει το κάθε πρόγραμμα, τον

αλγόριθμο που τρέχει για την προσομοίωση αυτής της διαδικασίας και τους υπολογιστικούς πόρους που ελέγχει το πρόγραμμα πάνω στον επεξεργαστή στον οποίο τρέχει. Κατέγραψα συνοπτικά αυτές τις πληροφορίες τις οποίες χρησιμοποίησα στη συνέχεια για να αναλύσω τα προγράμματα στο VTune.

4.3 Εγκατάσταση των προγραμμάτων στον υπολογιστή

Στο δεύτερο βήμα παρέλαβα το CD με τα προγράμματα. Για να μπορέσω να μεταγλωττίσω και να τρέξω τα προγράμματα ακολούθησα κάποια συγκεκριμένη διαδικασία. Αρχικά, αντέγραψα τα περιεχόμενα του φακέλου EEMBC στο δίσκο C: του υπολογιστή μου. Έπειτα με τη χρήση του προγράμματος WinZip αποσυμπίεσα τα αρχεία τα οποία θα χρησιμοποιούσα στη συνέχεια. Συγκεκριμένα αποσυμπίεσα το φακέλου Networking ο οποίος περιέχει τα benchmarks πάνω στα οποία εργάστηκα και το φάκελο Test Harness ο οποίος περιέχει κάποιες ειδικές συναρτήσεις στον κώδικά του μέσω των οποίων μπορούμε να τρέξουμε τα benchmarks, είτε πάνω σε επεξεργαστή γενικής χρήσης είτε πάνω σε ενσωματωμένο επεξεργαστή. Στη συγκεκριμένη περίπτωση τρέξαμε τα benchmarks μόνο πάνω σε επεξεργαστή γενικής χρήσης, του οποίου τα χαρακτηριστικά αναφέρονται στη συνέχεια.

4.4 Μεταγλώττιση των προγραμμάτων

Για τη μεταγλώττιση του προγράμματος χρειάστηκε να κατεβάσω το πρόγραμμα Cygwin, από την ιστοσελίδα www.cygwin.com. Το Cygwin είναι ένα πρόγραμμα το οποίο τρέχει σε λειτουργικό σύστημα Windows σαν Linux shell, εκτελώντας όλες τις λειτουργίες τις οποίες μπορεί κανείς να εκτελέσει σε ένα Linux shell. Κατεβάζοντας το πρόγραμμα αυτό από τη συγκεκριμένη ιστοσελίδα, για να μπορέσουμε να μεταγλωττίσουμε και να τρέξουμε τα benchmarks πρέπει οπωσδήποτε να συμπεριλάβουμε τις προδιαγραφές για τα πιο κάτω:

- gcc compiler
- gawk
- perl
- make

- gdb debugger

Μόλις εγκατασταθεί το πρόγραμμα μπορούμε να μεταγλωττίσουμε και να τρέξουμε τα προγράμματα.

Χρειάστηκε κάποια ειδική διαδικασία έτσι ώστε να μπορέσει να γίνει εφικτή η μεταγλώττιση των προγραμμάτων. Πρώτα αντέγραψα τα περιεχόμενα του φάκελου C:\EEMBC\Test Harness\eembc-2.0R2\, στο φάκελο των benchmarks τα οποία είχα αναλύσει, και συγκεκριμένα στο φάκελο C:\EEMBC\Networking\Networking 2.0\networking-2.0R1. Είναι σημαντικό η επικόλληση των περιεχομένων του φακέλου eembc-2.0R2 να γίνει μέσα στο συγκεκριμένο φάκελο, γιατί υπό άλλες συνθήκες δε θα μπορεί να γίνει η μεταγλώττιση των προγραμμάτων σωστά. Σημειώνεται πως ανοίγεται ο φάκελος eembc-2.0R2 και αντιγράφουμε και επικολλούμε τα περιεχόμενά του και όχι το φάκελο ως έχει.

Στη συνέχεια χρειάστηκε να κάνω κάποιες αλλαγές στα makefiles του φακέλου όπως είχε διαμορφωθεί μετά την επικόλληση των περιεχομένων του φακέλου util.
Συγκεκριμένα στην εντολή μέσω της οποίας καθορίζονται τα compiler options για τη μεταγλώττιση έκανα μια αλλαγή και από

COM = $(CCC)

Μετέτρεψα την εντολή σε

COM = $(CCD)

Η αλλαγή αυτή, προσθέτει την επιλογή –g για debugging στην εντολή μεταγλώττισης των προγραμμάτων και έγινε για να μπορώ να ανακτώ τις πληροφορίες για τον κώδικα των hot spots μέσω του εργαλείου VTune στη συνέχεια. Εάν δεν γίνει αυτή η αλλαγή, τότε στη συνέχεια ο κώδικας του προγράμματος στο VTune εμφανίζεται μόνο σε assembly μορφή.

Επίσης για να μπορεί να γίνει η σωστή μεταγλώττιση και εκτέλεση των προγραμμάτων έπρεπε να αντιγράψω τη βιβλιοθήκη cygwin1.dll από το φάκελο C:\cygwin\bin στο φάκελο C:\Windows\System32.
Αφού ολοκλήρωσα τις πιο πάνω διαδικασίες άνοιξα το πρόγραμμα Cygwin. Για να μπορέσω να μεταγλωττίσω τα προγράμματα έπρεπε πρώτα να μεταβώ στο φάκελο που τα περιέχει.

Πληκτρολόγησα τις πιο κάτω εντολές:

$ cd C:
$ cd EEMBC/Networking/Networking\ 2.0/networking-2.0R1

Αφού έγινε η μεταβίβαση στο συγκεκριμένο φάκελο πληκτρολόγησα την εντολή

$ make all

Με αυτή την εντολή γίνεται η μεταγλώττιση των προγραμμάτων, δημιουργούνται τα εκτελέσιμα αρχεία όπως και άλλα αρχεία τα οποία περιέχουν πληροφορίες για τους χρόνους εκτέλεσης και τα μεγέθη του κώδικα του κάθε προγράμματος. Η συγκεκριμένη διαδικασία μπορεί να πάρει αρκετό χρόνο μέχρι να ολοκληρωθεί.
Μετά την ολοκλήρωση της διαδικασίας αυτής ήταν πλέον εφικτή η εκτέλεση των προγραμμάτων.

4.5 Εκτέλεση των εκτελέσιμων αρχείων και ανάλυση των αποτελεσμάτων

Τα εκτελέσιμα αρχεία τα οποία χρησιμοποίησα για να κάνω την ανάλυσή μου, βρίσκονταν μέσα στο φάκελο C:\EEMBC\Networking\Networking 2.0\networking-2.0R1\networking\gcc\bin. Μπορούμε να τα εκτελέσουμε είτε από το shell του Cygwin, είτε κάνοντας πάνω τους κλικ 2 φορές. Ανοίγει ένα παράθυρο το οποίο περιέχει τις διαδικασίες που μπορούμε να εκτελέσουμε μέσω του Test Harness όπως έχει αναφερθεί και προηγουμένως. Πατώντας **h** στο παράθυρο αυτό, εμφανίζεται μια λίστα με όλες τις εντολές που μπορούμε να εκτελέσουμε μέσω του Test Harness πάνω στα προγράμματα.

Για να καθορίσω τον αριθμό των iterations για τα οποία τρέχει το πρόγραμμα βάζω την εντολή **n <αριθμός iterations>** , όπου για τον αριθμό των iterations αν βάλουμε 0, το benchmark θα τρέξει τόσες φορές όσες ορίζεται από το σταθερό αριθμό iterations που ορίζεται από το benchmark. Αν βάλουμε οποιοδήποτε άλλο αριθμό, τότε το benchmark θα τρέξει τόσες φορές όσες ορίσαμε με αυτό τον αριθμό. Στη συνέχεια εκτέλεσα την εντολή **g** με την οποία τρέχει το πρόγραμμα για τον αριθμό των iterations τα οποία όρισα. Μόλις ολοκληρωθεί η εκτέλεση του προγράμματος τυπώνονται στην οθόνη κάποιες πληροφορίες για το μέγεθος του προβλήματος, το χρόνο εκτέλεσης του προγράμματος, τον αριθμό των iterations που ολοκληρώθηκαν ανά δευτερόλεπτο και το χρόνο εκτέλεσης του κάθε iteration. Τα αποτελέσματα από την εκτέλεση του κάθε προγράμματος, είναι αυτά που χρησιμοποίησα για την ανάλυση που έκανα, η οποία περιγράφεται στη συνέχεια.

Για την ανάλυση που έκανα στο VTune χρησιμοποίησα και πάλι τα εκτελέσιμα προγράμματα που προέκυψαν από τη μεταγλώττιση που περιγράφεται πιο πάνω.
Η διαδικασία ανάλυσης στο VTune περιλαμβάνει τα πιο κάτω βήματα:

1) Ανοίγουμε το πρόγραμμα και επιλέγουμε το “Quick performance analysis”
2) Στο δεύτερο παράθυρο που εμφανίζεται επιλέγουμε το εκτελέσιμο πρόγραμμα το οποίο θέλουμε να αναλύσουμε.
3) Στη συνέχεια επιλέγουμε “Go”

Αμέσως αρχίζει η διαδικασία εκτέλεσης του προγράμματος και παράλληλα αρχίζει και η διαδικασία δειγματοληψίας μέσω της οποίας εξάγονται τα αποτελέσματα για το χρόνους εκτέλεσης και το ποσοστό των εντολών που ολοκληρώθηκαν και αφορούν την εφαρμογή που επιλέξαμε να αναλύσουμε. Μόλις ολοκληρωθεί η διαδικασία αυτή, εμφανίζεται ένα παράθυρο που εμφανίζει όλες τις διεργασίες που έλαβαν χώρα ενόσω γινόταν η δειγματοληψία για το πρόγραμμα μας. Ανάμεσα σ’ αυτές εμφανίζεται και η διαδικασία που αφορά το πρόγραμμα που επιλέξαμε να αναλύσουμε. Πατώντας 2 φορές πάνω του εμφανίζονται τα threads μέσα στα οποία έτρεξε το πρόγραμμα. Επιλέγουμε το thread μέσα από το οποίο πάρθηκαν τα πιο πολλά δείγματα και προχωρούμε στο επόμενο στάδιο όπου μπορούμε να δούμε τα modules που έτρεξαν στο συγκεκριμένο thread. Επιλέγουμε και πάλι το module που αφορά το πρόγραμμα που αναλύουμε. Στη

συνέχεια εμφανίζονται τα ονόματα των συναρτήσεων που αποτελούν τα hot spots του προγράμματος. Καταλαβαίνουμε ότι είναι hot spots γιατί δίπλα ακριβώς από το όνομα της κάθε συνάρτησης, εμφανίζεται το ποσοστό του χρόνου που κατανάλωσε, από το συνολικό χρόνο εκτέλεσης του προγράμματος. Επιλέγουμε κάποια από αυτές και στο επόμενο παράθυρο που ανοίγει, εμφανίζεται ο κώδικας της. Στα σημεία όπου κάνουν τη συγκεκριμένη συνάρτηση hot spot εμφανίζεται ο αριθμός των εντολών που εκτελέστηκαν και το ποσοστό του χρόνου που καταναλώθηκε για την εκτέλεσή τους.

Για την ανάλυση της συμπεριφοράς των προγραμμάτων χρησιμοποίησα 2 βασικά μετρικά του προγράμματος VTune. Το χρόνο που αναλωνόταν για την εκτέλεση του κάθε προγράμματος και τον αριθμό των εντολών που εκτελούνταν για κάθε hot spot.

Ο χρόνος που κατανάλωσε η κάθε συνάρτηση εμφανίζεται σαν εκατομμύρια κύκλοι μηχανής. Ο αριθμός των εντολών εμφανίζεται σαν εκατομμύρια εντολές που ολοκληρώθηκαν.

Για καλύτερη ανάλυση των προγραμμάτων και για όσα ήταν εφικτό, τροποποίησα τα δεδομένα εισόδου για να κάνω συγκρίσεις ανάμεσα στις διάφορες εκτελέσεις για τα διαφορετικά inputs που όρισα.

Ανάλυσα τα δεδομένα που πρόεκυψαν με τη χρήση γραφικών παραστάσεων που δείχνουν ποσοστιαία την κατανομή του χρόνου και των εντολών για το κάθε input. Επίσης παράλληλα με τα δεδομένα αυτά χρησιμοποίησα και τα αποτελέσματα που προέκυψαν από την απλή εκτέλεση στο Cygwin. Αυτή η ανάλυση αποτελούσε τη δυναμική ανάλυση των προγραμμάτων.

Επίσης έχω σχεδιάσει ενδεικτικά διαγράμματα ροής που δείχνουν τη σειρά εκτέλεσης του προγράμματος και τις συναρτήσεις που αποτελούν τα hotspots του προγράμματος. Με βάση αυτά τα διαγράμματα έχω επίσης βρει την πολυπλοκότητα των προγραμμάτων. Στην πολυπλοκότητα των προγραμμάτων συμπεριλαμβάνεται και ο αριθμός των iterations για τα οποία τρέχουν τα προγράμματα, αφού αποτελεί βασικό κομμάτι των διαγραμμάτων ροής, γιατί καθορίζει τον αριθμό των φορών που θα εκτελεστεί η κάθε συνάρτηση.

4.6 Χαρακτηριστικά του υπολογιστή πάνω στον οποίο έγιναν οι δοκιμές

Υπολογιστής: SONY Vaio
Λειτουργικό σύστημα: Windows Vista Home Premium, Service Pack 1
Επεξεργαστής: Intel(R) Core(TM)2 Duo CPU T8100 @ 2.10GHz 2.10GHz
Μνήμη RAM: 3,00 GB
Τύπος συστήματος: 32-bit Operating System

4.7 Εργαλείο VTune Performance Analyzer

Το εργαλείο αυτό, που είναι κατασκευή της εταιρίας INTEL χρησιμεύει στο να γίνεται εύκολη καταμέτρηση της επίδοσης των εφαρμογών. Γίνεται χρήση μιας γραφικής διεπιφάνειας και δεν χρειάζεται να γίνονται επαναλαμβανόμενες μεταγλωττίσεις του κώδικα. Είναι ανεξάρτητο γλώσσας και μεταγλωττιστή και γι' αυτό μπορεί να δουλέψει με ένα ευρύ φάσμα από γλώσσες. Το VTune παρέχει τόσο τη γραφική ανάλυση, όσο και ένα μεγάλο φάσμα από δείγματα, αντίθετα με άλλα παρόμοια προϊόντα που παρέχουν είτε το ένα είτε το άλλο. Επιπλέον προσφέρει ένα μεγάλο σύνολο γεγονότων συντονισμού για όλους τους πρόσφατους επεξεργαστές που είναι κατασκευή της Intel.

Το εργαλείο αυτό μπορεί να χρησιμοποιηθεί τόσο στο λειτουργικό σύστημα Linux, όσο και στο λειτουργικό σύστημα Windows. Ο μόνος περιορισμός για τη χρήση του συγκεκριμένου εργαλείου είναι ότι ο επεξεργαστής της μηχανής πάνω στην οποία θα το εγκαταστήσουμε πρέπει υποχρεωτικά να είναι κατασκευής της Intel.

Πιο κάτω περιγράφονται περιληπτικά μερικές από τις λειτουργίες που μπορούμε να κάνουμε χρησιμοποιώντας το εργαλείο VTune.

- Το εργαλείο αυτό μας βοηθά να χαρακτηρίσουμε της επίδοση κάποιων μερών του συστήματος, βάσει κάποιων παραμέτρων ή να χαρακτηρίσουμε την επίδοση για ολόκληρο το σύστημα ως σύνολο. Συλλέγει δεδομένα της επίδοσης του συστήματος που τρέχει την εφαρμογή μας, οργανώνει και εμφανίζει τα δεδομένα αυτά με παραστατικό τρόπο, π.χ. με γραφικές παραστάσεις και

προσδιορίζει τα σημεία όπου η επίδοση είναι μειωμένη και δίνει διάφορες εισηγήσεις για βελτίωσή της.

- Το VTune μπορεί να πάρει μετρήσεις για 4 μετρικά το πολύ για κάθε εκτέλεση. Αν σε κάποια περίπτωση χρειάζεται να γίνει δειγματοληψία για περισσότερα μετρικά τότε η εφαρμογή η οποία εξετάζεται, εκτελείται περισσότερες από μια φορές.
- Πιο λεπτομερής περιγραφή της χρήσης του εργαλείου, εμφανίζεται στο παράρτημα Α.

Κεφάλαιο 5

Ανάλυση των προγραμμάτων

5.1 Ανάλυση του benchmark OSPF

Αυτό το benchmark ελέγχει τη σχετική απόδοση των δρομολογητών

Εφαρμογή

Αυτό το benchmark υλοποιεί τον αλγόριθμο του Dijkstra ο οποίος επιλέγει το συντομότερο ή το πιο φθηνό σε κόστος μονοπάτι από ένα σύνολο μονοπατιών από τον αναφερόμενο δρομολογητή, προς κάποιο άλλο δρομολογητή. Αυτή είναι μια μέθοδος η οποία χρησιμοποιείται ευρέως στα δίκτυα υπολογιστών.

Περιγραφή του benchmark

Ο αλγόριθμος του Dijkstra επιλέγει το συντομότερο ή το πιο φθηνό σε κόστος μονοπάτι από ένα σύνολο μονοπατιών τα οποία γνωρίζει ο αναφερόμενος δρομολογητής. Δημιουργείται ένας πίνακας στον οποίο αποθηκεύεται ο κάθε κόμβος του δικτύου, του οποίου ο υφιστάμενος κόμβος γνωρίζει τη ύπαρξη και έχει αποθηκευμένα τα χαρακτηριστικά του, όπως το κόστος μεταβίβασης από τον ένα στον άλλο. Οι κόμβοι είναι οι υπόλοιποι δρομολογητές για τους οποίους γνωρίζει πληροφορίες ο δρομολογητής που μας ενδιαφέρει. Ο κάθε κόμβος έχει ένα σύνολο από κατευθυνόμενες ακμές οι οποίες αναπαριστούν τη σύνδεση του με κάποιο άλλο δρομολογητή. Η κάθε ακμή έχει κάποια τιμή που αναπαριστά το κόστος της που είναι

και το κόστος της ένωσης των 2 δρομολογητών, με λίγα λόγια το κόστος αποστολής δεδομένων από τον ένα δρομολογητή στον άλλο.

Ο αλγόριθμος του Dijkstra ξεκινά από τον αναφερόμενο ως βάση κόμβο και υπολογίζοντας κάθε φορά το μικρότερο κόστος βρίσκει την πιο φθηνή διαδρομή μέσω της οποίας θα κινηθούν τα πακέτα, προς τον δρομολογητή στον οποίο θέλει να τα στείλει. Οι πίνακες με τους δρομολογητές σε αυτό το benchmark κτίζονται δυναμικά.

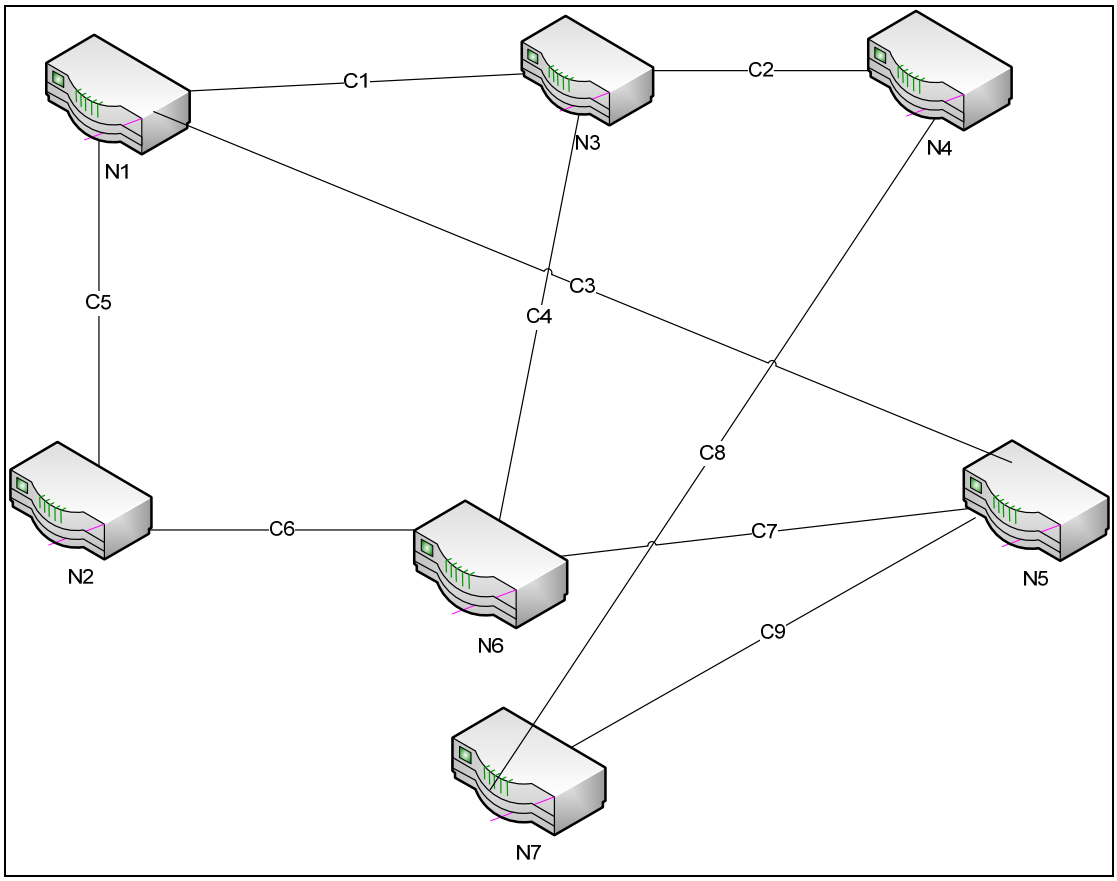
Ανάλυση των υπολογιστικών πόρων που εξετάζει το benchmark

Σημαντικοί παράγοντες για αυτό το benchmark είναι η ικανότητα του επεξεργαστή να ανακτά δεδομένα από τη μνήμη και επίσης η ικανότητα του να εκτελεί συχνά CTI (Control Transfer Instructions).

Input / Output του benchmark

Η είσοδος του προγράμματος είναι ένας γράφος με κόμβους που αναπαριστούν τους δρομολογητές του δικτύου και με κατευθυνόμενες ακμές, οι οποίες αναπαριστούν τις συνδέσεις μεταξύ των κόμβων. Ο αριθμός των κόμβων στο γράφο δεν μπορεί να ξεπερνά τους 400 και ο αριθμός των ακμών ανά κόμβο δεν μπορεί να ξεπερνά τις 399 ακμές ανά κόμβο.

Πιο κάτω φαίνεται ένα χαρακτηριστικό παράδειγμα εισόδου.



Εικόνα 5.1: Τοπολογία δικτύου πάνω στο οποίο εφαρμόζεται ο αλγόριθμος του Dijkstra

Η έξοδος του προγράμματος είναι η δημιουργία πινάκων στον κάθε δρομολογητή που να έχουν καταγραμμένο το ελάχιστο κόστος μεταφοράς πακέτων από εκείνον τον δρομολογητή προς όλους τους δρομολογητές του δικτύου, και τη διαδρομή που πρέπει να ακολουθήσουν τα πακέτα για να έχει η μεταφορά τους αυτό το ελάχιστο κόστος.

Δομές δεδομένων που χρησιμοποιούνται

Στο συγκεκριμένο benchmark η βασική δομή δεδομένων που χρησιμοποιείται είναι η λίστα από κόμβους. Οι κόμβοι της λίστας είναι τύπου node_s. Ο τύπος αυτός θα αναλυθεί στη συνέχεια.

Για αυτό το benchmark χρησιμοποιούνται 2 βασικοί τύποι δεδομένων. Ο τύπος **arc_s** και ο τύπος **node_s.** Ο arc_s χρησιμοποιείται για να περιγράψει μια ακμή μεταξύ 2 κόμβων και ο node_s χρησιμοποιείται για να περιγράψει ένα δρομολογητή.

Structure arc_s

Ένας τύπος δεδομένων ο οποίος δημιουργήθηκε για να περιγράφει μια ακμή μεταξύ 2 κόμβων στο δίκτυο. Η κάθε ακμή arc_s έχει μια τιμή που σχετίζεται με αυτή. Αυτή η τιμή είναι είτε το κόστος της μετάβασης μεταξύ των 2 κόμβων που ενώνει, είτε η απόσταση που έχουν μεταξύ τους οι 2 κόμβοι.

Ο τύπος αυτός περιγράφεται από τον κώδικα που φαίνεται πιο κάτω:

```
typedef struct arc_s {
   struct node_s*  head_node;      /* the head of the arc */
   struct arc_s*   next_arc;       /* Next arc in this node's arc list */
   e_u32           cost;           /* the arc's cost or length */
} arc;
```

Structure node_s

Ένας τύπος δεδομένων ο οποίος δημιουργήθηκε για να περιγράφει έναν δρομολογητή στο δίκτυο.

Ο τύπος αυτός περιγράφεται από τον κώδικα που φαίνεται πιο κάτω:

```
typedef struct node_s {
   e_u32           id;             /* Just a handy ID string */
   state_T         state;          /* Updated by dijkstra calc */
   e_u32           dist;           /* Filled in by dijkstra calc */
   struct node_s*  child_node;     /* Filled in by dijkstra calc */
   arc*            arclist_head;   /* Head of this node's list of arcs */
   struct node_s*  next;           /* next node in linked list */
```

} node;

Γενική περιγραφή του των λειτουργιών που εκτελεί ο κώδικας του benchmark

Υπάρχουν 2 πίνακες. Ο arc_base ο οποίος κρατά για κάθε κόμβο τις ακμές που φεύγουν από αυτόν και ο node_base ο οποίος κρατά όλους τους κόμβους του γράφου που αναπαριστά το δίκτυο πάνω στο οποίο εφαρμόζεται ο αλγόριθμος του Dijkstra τον οποίο υλοποιεί το συγκεκριμένο benchmark.

Ο κάθε πίνακας αρχικοποιείται πριν ξεκινήσει η εκτέλεση του benchmark και αρχικοποιείται ξανά στην αρχή του κάθε iteration έτσι ώστε το κάθε iteration να κάνει εκτελεί ακριβώς τις ίδιες διεργασίες.

Τα routing tables του κάθε κόμβου (δρομολογητή), δηλαδή οι πίνακες που δείχνουν ποια διαδρομή πρέπει να ακολουθήσουν τα πακέτα για να φτάσουν σε κάποιο άλλο κόμβο, κτίζονται δυναμικά. Δεν υπάρχει από πριν κάποια προκαθορισμένη διαδρομή.

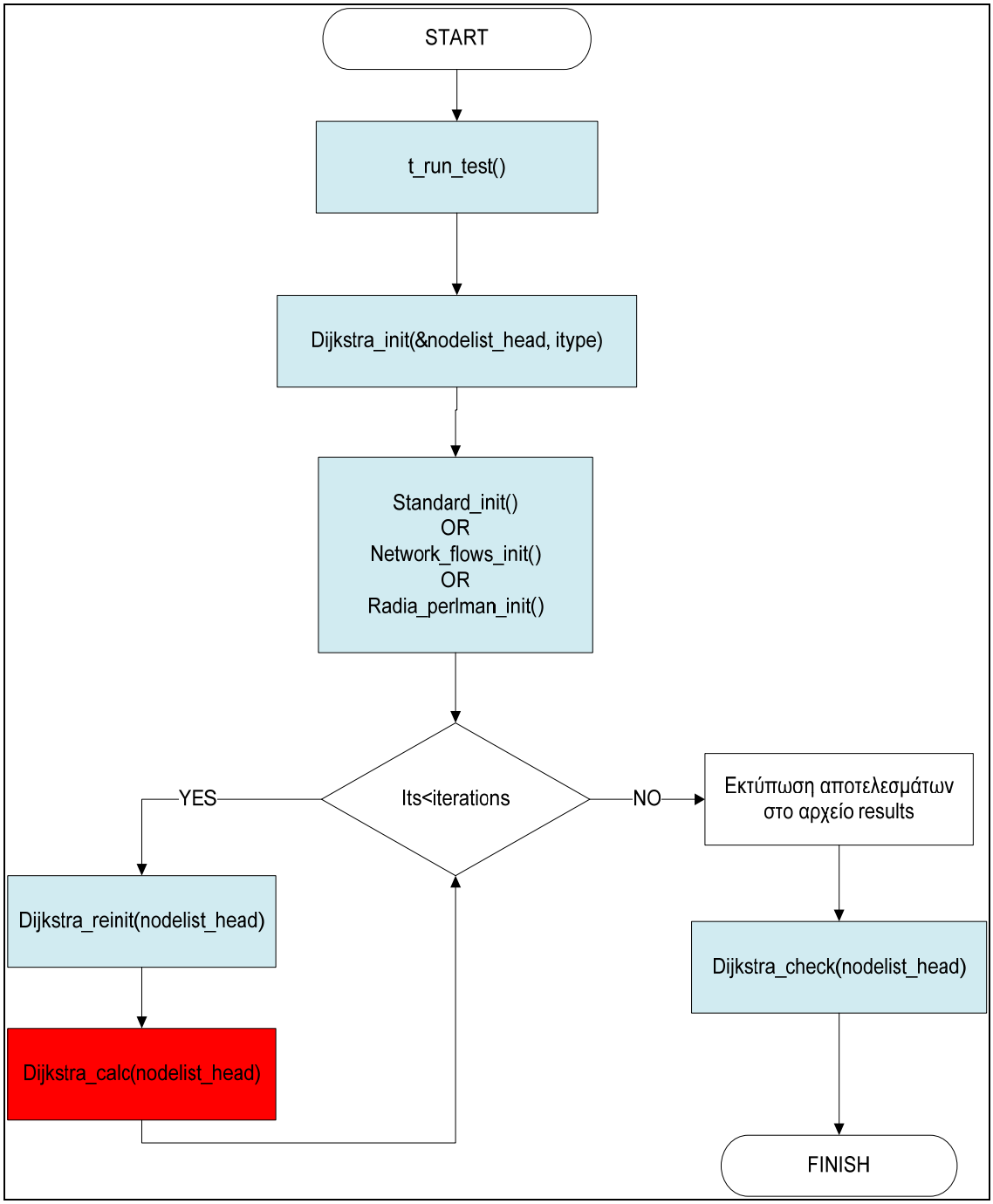
Το benchmark διατρέχει συνέχεια τη λίστα των κόμβων και ελέγχει το load use latency του επεξεργαστή και την ικανότητα του να εκτελεί συχνά Control Transfer Instructions.

Όπως αναφέρεται και πιο πάνω ο αριθμός των κόμβων στο γράφο δεν μπορεί να ξεπερνά τους 400 και ο αριθμός των ακμών ανά κόμβο, δε μπορεί να ξεπερνά τις 399.

Στη συνέχεια για σκοπούς της ανάλυσης του benchmarks οι αριθμοί αυτοί τροποποιούνται.

Πιο κάτω φαίνεται ένα ενδεικτικό διάγραμμα ροής που δείχνει τη σειρά με την οποία καλούνται οι βασικές συναρτήσεις του προγράμματος.

Ενδεικτικό διάγραμμα ροής για το benchmark ospf



Εικόνα 5.2: Διάγραμμα ροής για το benchmark ospf

Με κόκκινο χρώμα αναπαρίσταται η κλήση της συνάρτησης η οποία αναλώνει τον πιο πολύ χρόνο εκτέλεσης βάσει του κώδικα και των πράξεων που φαίνονται να γίνονται σ' αυτήν.

Με γαλάζιο χρώμα αναπαρίστανται οι κλήσεις των υπόλοιπων συναρτήσεων.

Η μεταβλητή its είναι μετρητής και μετρά τον αριθμό των iterations του προγράμματος που έχουν ολοκληρωθεί. Μπορούμε να καθορίσουμε τον αριθμό των iterations για τα οποία θέλουμε να τρέξει το πρόγραμμά μας.

Περιγραφή των συναρτήσεων

Η αρχικοποίηση του προγράμματος γίνεται με τη χρήση μιας από τις 3 πιο κάτω συναρτήσεις, ανάλογα με το είδος εκτέλεσης που επιλέγουμε.

1. **radia_perlman_init:** Έχει σταθερό αριθμό κόμβων και ακμών ανά κόμβο, 7 και 6 αντίστοιχα.
2. **network_flows_init:** Έχει σταθερό αριθμό κόμβων και ακμών ανά κόμβο, 6 και 3 αντίστοιχα.
3. **standard_init:** Για αυτή τη συνάρτηση ο αριθμός των κόμβων και των ακμών ορίζονται σαν σταθερές στον κώδικα του προγράμματος.

Σε αυτό το σημείο γίνεται η αρχικοποίηση των κόμβων και των ακμών ανά κόμβο. Για να γίνει αυτό, υπάρχει ένας διπλός φωλιασμένος βρόγχος, ο οποίος διατρέχει τη λίστα των κόμβων και τη λίστα των ακμών ανά κόμβο και αρχικοποιεί τα δεδομένα.
Αυτός ο βρόγχος έχει πολυπλοκότητα που δίνεται από την πιο κάτω σχέση:

Complexity = O (n x m)

Όπου **n**: ο αριθμός των κόμβων και
m: ο αριθμός των ακμών

Dijksrta_calc: Διατρέχει τη λίστα των κόμβων και για τον κάθε κόμβο διατρέχει τη λίστα των ακμών του. Ο source node είναι διαφορετικός κάθε φορά.

Η πολυπλοκότητα της συνάρτησης αυτής δίνεται από την πιο κάτω σχέση:

Complexity = O (n x m)

Όπου **n**: ο αριθμός των κόμβων και

m: ο αριθμός των ακμών

Για τον κάθε κόμβο, δημιουργείται μια ταξινομημένη λίστα στην οποία οι κόμβοι τοποθετούνται σε αύξουσα σειρά, ανάλογα με την απόσταση που έχουν από το συγκεκριμένο κόμβο.

Από τα πιο πάνω μπορούμε να πούμε ότι **η συνολική πολυπλοκότητα για το benchmark**, που αφορά των αριθμό των κόμβων, τον αριθμό των ακμών και τον αριθμό των iterations για τα οποία τρέχει το πρόγραμμα δίνονται από την πιο κάτω σχέση:

Complexity = O (n x m x y)

Όπου **n**: ο αριθμός των κόμβων

m: ο αριθμός των ακμών ανά κόμβο και

y: ο αριθμός των iterations για τα οποία τρέχει το benchmark

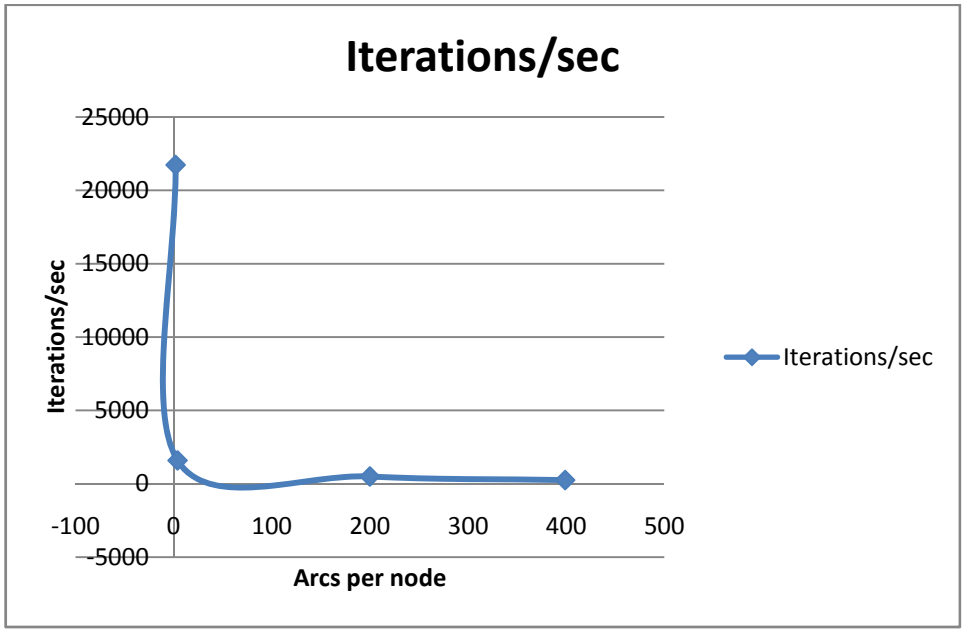
Στον πιο κάτω πίνακα φαίνονται τα αποτελέσματα από τους χρόνους εκτέλεσης του προγράμματος με διάφορες αλλαγές στο input.

Συνοπτικός πίνακας αποτελεσμάτων

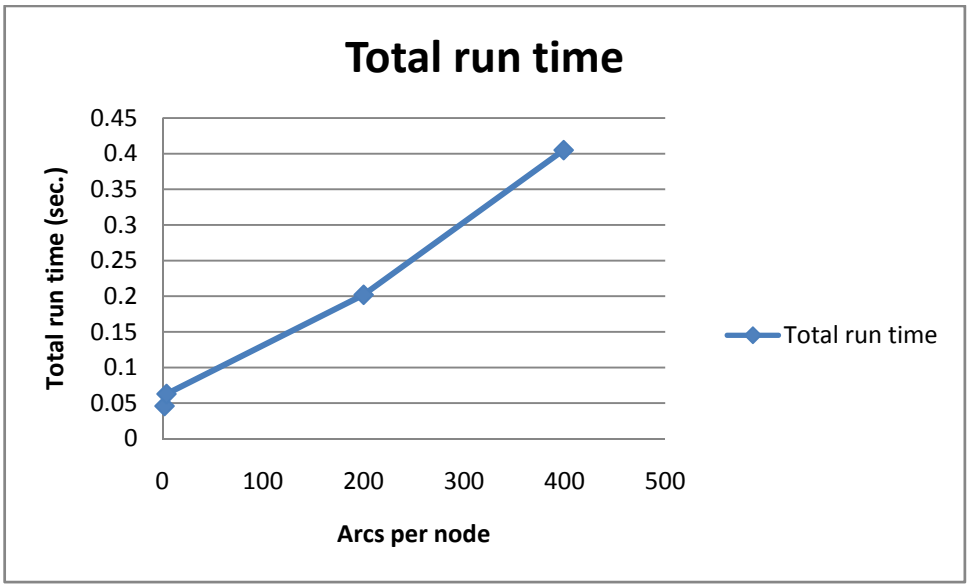
	OSPF 400 κόμβοι – 2 ακμές ανά κόμβο	**OSPF 400 κόμβοι – 4 ακμές ανά κόμβο**	**OSPF 400 κόμβοι – 200 ακμές ανά κόμβο**	**OSPF 400 κόμβοι – 399 ακμές ανά κόμβο**
Iterations/sec	21739,13	1587,3	495,05	246,91
Total run time	0,046	0,063	0,202	0,405
Time/Iteration	0,00046	0,00063	0,002	0,004
CPI	1,058	1,289	1,417	1,353
Συνολικός χρόνος (Εκατ. κύκλοι)	115,5	121,8	627,9	1335,6
Συνολικός αριθμός εντολών (Εκατ.)	109,2	94,5	443,1	987

Πίνακας 5.1: Συνοπτικός πίνακας που παρουσιάζει τη συμπεριφορά του προγράμματος για κάθε διαφορετικό input

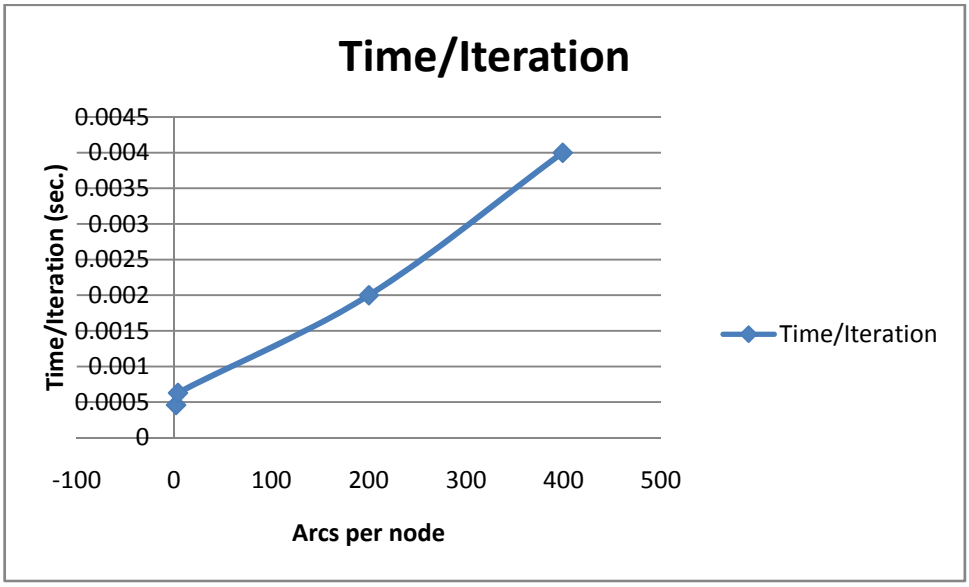
Στα πιο κάτω διαγράμματα φαίνεται η σύγκριση των ποσοστών χρόνου και εντολών για τον πιο πάνω πίνακα.



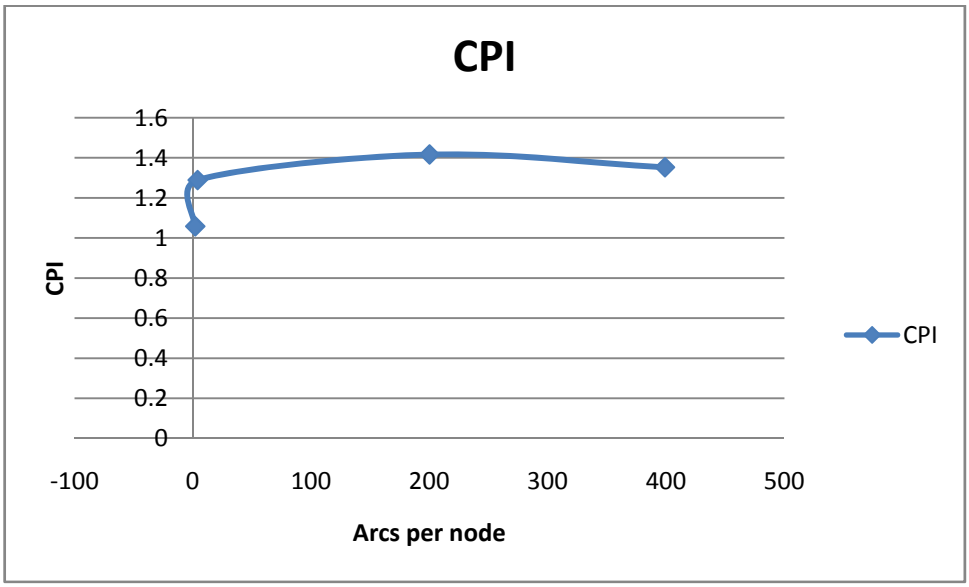
Όπως βλέπουμε όσο αυξάνεται ο αριθμός των ακμών ανά κόμβο, μειώνεται ο αριθμός των iterations που εκτελούνται ανά δευτερόλεπτο. Άρα το πρόγραμμα γίνεται πιο αργό. Αυτό συμβαίνει γιατί λόγω του αυξημένου input αυξάνεται και ο αριθμός των εντολών που εκτελούνται.



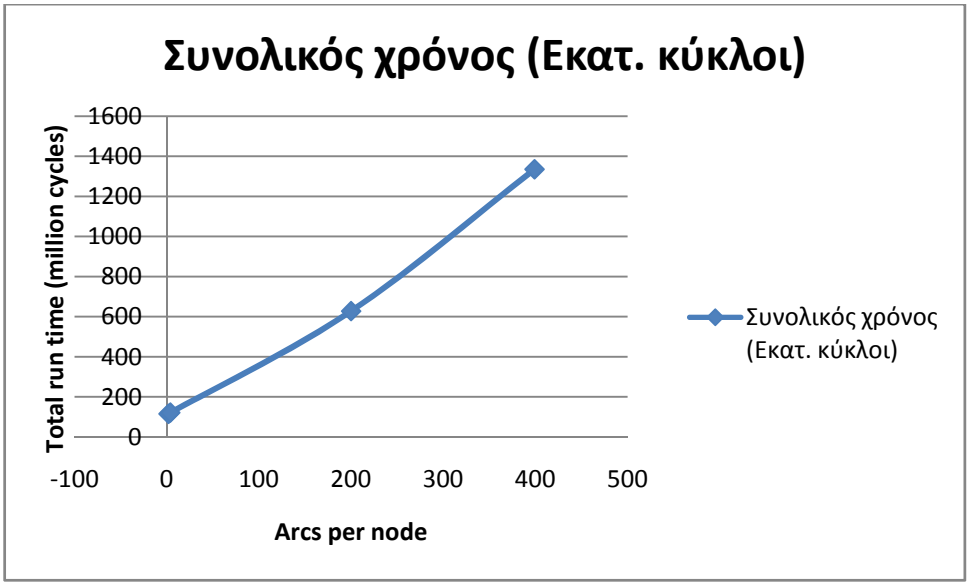
Με την αύξηση του αριθμού των κόμβων, παρατηρούμε επίσης ότι αυξάνεται ο συνολικός χρόνος εκτέλεσης του προγράμματος.



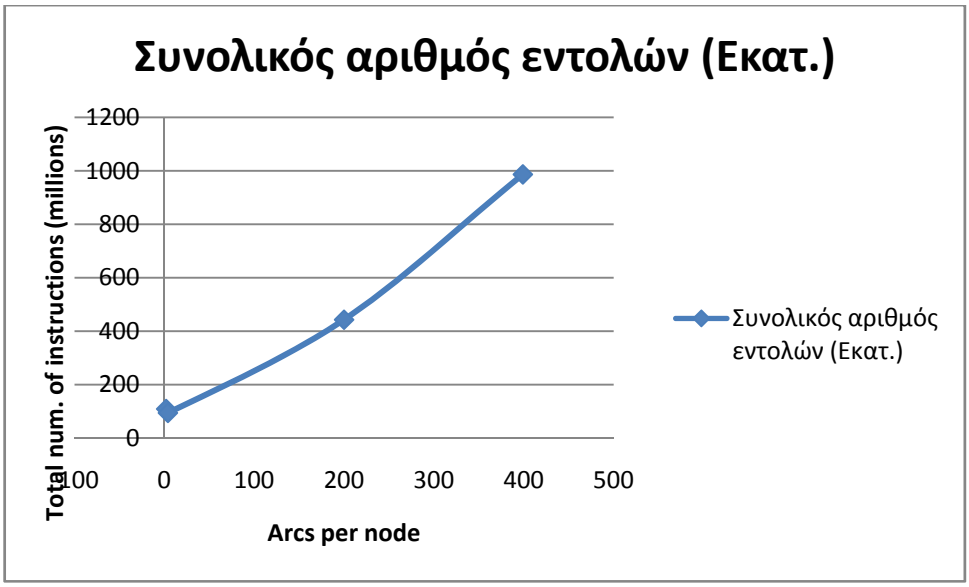
Όπως παρατηρούμε ο χρόνος που αναλώνεται για την εκτέλεση του κάθε iteration αυξάνεται ανάλογα με το συνολικό χρόνο εκτέλεσης του προγράμματος. Αυτό συμβαίνει γιατί μέσα στο κάθε iteration εκτελούνται περισσότερες εντολές και επεξεργάζεται μεγαλύτερος αριθμός δεδομένων.



Παρατηρούμε πως υπάρχουν κάποιες διαφοροποιήσεις στο CPI του προγράμματος σε σχέση με το input, αλλά δεν μπορούμε να εξάγουμε κάποιο γενικό συμπέρασμα με αυτά τα δεδομένα. Αυτό συμβαίνει γιατί μπορεί για συγκεκριμένα inputs να εκτελείται σημαντικός αριθμός εντολών που να απαιτούν πιο πολλούς κύκλους μηχανής σε σχέση με άλλες. Όπως φαίνεται στο πιο πάνω σχήμα το πρόγραμμα OSPF 400-200 έχει ελαφρώς μεγαλύτερο CPI από το πρόγραμμα OSPF 400-399. Με βάση τη μικροαρχιτεκτονική ανάλυση που έκανα σχετικά με την πιο πάνω παρατήρηση, διαπίστωσα ότι ο λόγος που το CPI είναι πιο μεγάλο, παρόλο που το input είναι πιο μικρό, είναι επειδή γίνονται περισσότερες προσβάσεις στην L2 cache για μεταφορά δεδομένων προς την L1 cache, σε σχέση με το πρόγραμμα OSPF 400-399. Αυτό εξηγεί το γεγονός ότι το CPI είναι μεγαλύτερο, γιατί η πρόσβαση στην L2 cache κοστίζει περισσότερο σε χρόνο. Άρα αφού γίνονται περισσότερες προσβάσεις στην L2 cache για το συγκεκριμένο πρόγραμμα αυτό εξηγεί το γεγονός ότι το CPI είναι μεγαλύτερο.



Παρατηρούμε επίσης μια αισθητή αύξηση του χρόνου εκτέλεσης του προγράμματος, όσο αυξάνεται ο αριθμός των δεδομένων που επεξεργάζεται.

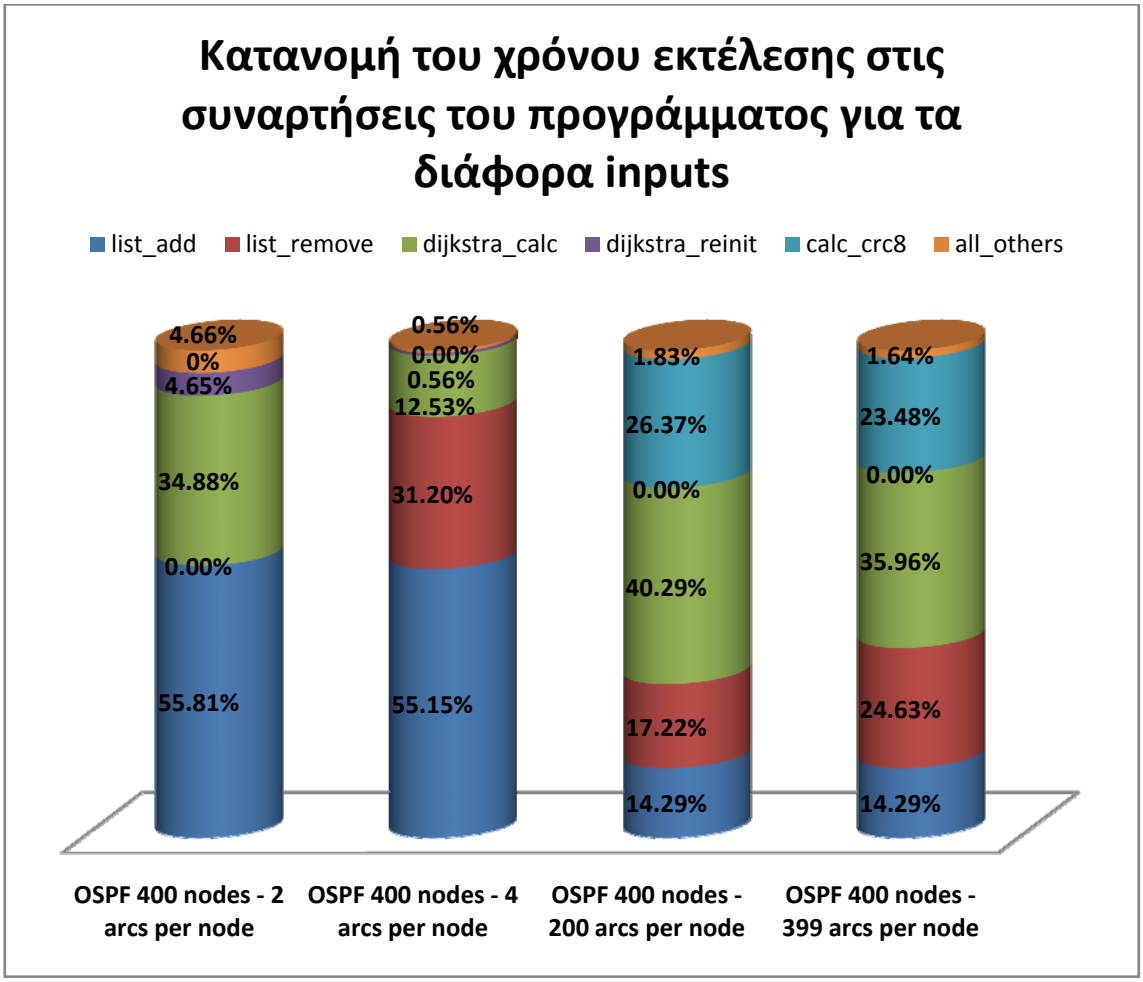


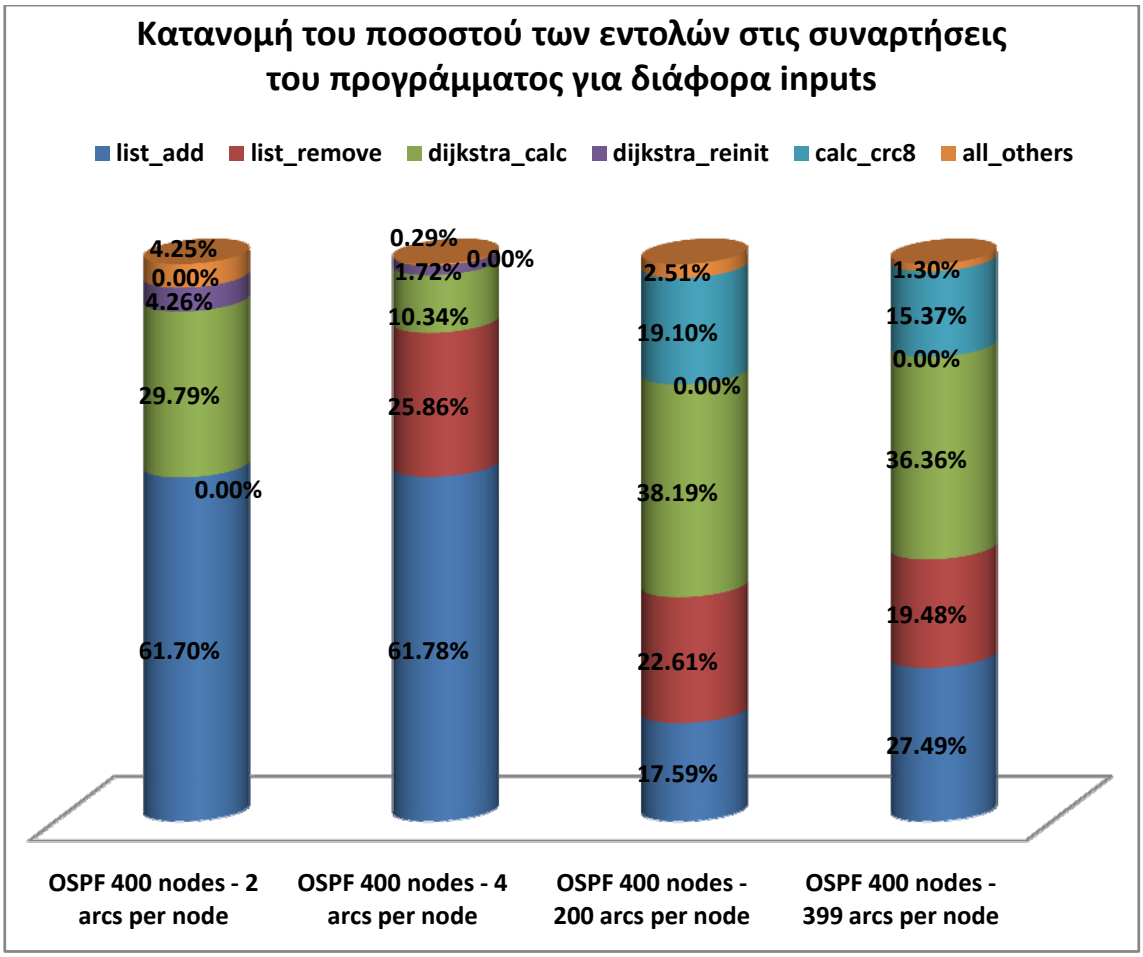
Όπως ήταν επόμενο, ο αριθμός των εντολών που εκτελούνται αυξάνεται όσο αυξάνεται το μέγεθος του input του προγράμματος. Αυτό συμβαίνει γιατί αυξάνεται ο συνολικός αριθμός των δεδομένων τα οποία πρέπει να επεξεργαστεί το πρόγραμμα.

Συμπέρασμα:

Η παρατήρηση που έχω κάνει με βάση τα πιο πάνω στοιχεία είναι ότι όσο αυξάνεται ο αριθμός των ακμών ανά κόμβο τόσο αυξάνεται ο χρόνος εκτέλεσης του προγράμματος και ο αριθμός των εντολών που εκτελούνται. Ο λόγος είναι επειδή αυξάνεται το μέγεθος του input και πρέπει να εκτελεστούν περισσότερες εντολές, έτσι ώστε να προκύψουν οι τελικοί πίνακες δρομολόγησης στον κάθε δρομολογητή. Επίσης αυξάνονται οι προσβάσεις στη μνήμη, γεγονός που αυξάνει το συνολικό CPI του προγράμματος. Για τα προγράμματα τα οποία, βάσει της μικροαρχιτεκτονικής ανάλυσης, κάνουν περισσότερες προσβάσεις στην L2 cache το συνολικό CPI αυξάνεται ακόμη περισσότερο.

Στα πιο κάτω διαγράμματα φαίνεται το ποσοστό χρόνου και εντολών που ανάλωσε η κάθε συνάρτηση του προγράμματος, ανάλογα με τις διάφορες αλλαγές που έγιναν στο input για να προκύψουν κάποια συμπεράσματα ως προς το πού αναλώνει το χρόνο εκτέλεσης το πρόγραμμα.


Κατανομή του χρόνου εκτέλεσης στις συναρτήσεις του προγράμματος για τα διάφορα inputs
list_add
list_remove
dijkstra_calc
dijkstra_reinit
calc_crc8
all_others
4.66%
0%
4.65%
34.88%
0.00%
55.81%
0.56%
0.00%
0.56%
12.53%
31.20%
55.15%
1.83%
26.37%
0.00%
40.29%
17.22%
14.29%
1.64%
23.48%
0.00%
35.96%
24.63%
14.29%
OSPF 400 nodes - 2 arcs per node
OSPF 400 nodes - 4 arcs per node
OSPF 400 nodes - 200 arcs per node
OSPF 400 nodes - 399 arcs per node



Με βάση τα πιο πάνω διαγράμματα κατανομής χρόνου και εντολών παρατηρούμε ότι για μικρότερους αριθμούς ακμών ανά κόμβο το μεγαλύτερο ποσοστό χρόνου και εντολών έχει η συνάρτηση list_add η οποία εισάγει έναν κόμβο στη λίστα κόμβων. Αντίθετα για μεγαλύτερους αριθμούς ακμών ανά κόμβο, παρατηρούμε ότι το μεγαλύτερο ποσοστό χρόνου και εντολών έχει η συνάρτηση dijkstra_calc. Ο λόγος για τον οποίο συμβαίνει το πιο πάνω οφείλεται στον τρόπο λειτουργίας του προγράμματος. Συγκεκριμένα καθώς διατρέχεται η λίστα ακμών του κάθε κόμβου, εάν κάποια ακμή οδηγεί σε ένα κόμβο που δεν υπάρχει ήδη στη λίστα των κόμβων, τότε καλείται η συνάρτηση list_add και ο κόμβος αυτός εισάγεται στη λίστα. Η λίστα αυτή, διατρέχεται περισσότερες φορές στην περίπτωση που έχουμε μικρό αριθμό ακμών ανά κόμβο, αφού είναι πιο μικρός ο αριθμός των νέων κόμβων που εντοπίζονται κάθε φορά.

Παράλληλα με το πιο πάνω προκύπτει το συμπέρασμα ότι για τα προγράμματα όπου ο αριθμός των ακμών ανά κόμβο είναι μεγαλύτερος, η συνάρτηση dijkstra_calc έχει το

μεγαλύτερο ποσοστό χρόνου και εντολών. Αυτό συμβαίνει γιατί υπάρχουν περισσότερες πιθανές διαδρομές για να σταλεί ένα πακέτο και έτσι διατρέχεται η λίστα των ακμών του κάθε κόμβου πιο πολλές φορές για να βρεθεί το πιο φθηνό σε κόστος μονοπάτι.

Συνάρτηση dijkstra_calc

Hot spots

1. Το πρώτο hot spot της συγκεκριμένης συνάρτησης που είναι και το hot spot ολόκληρου του προγράμματος είναι το σημείο στο οποίο γίνεται έλεγχος αν το state του κόμβου στον οποίο βρίσκεται το πρόγραμμα είναι unknown, δηλαδή εάν ο συγκεκριμένος κόμβος υπάρχει στη λίστα των γειτονικών κόμβων, του υφιστάμενου κόμβου. Είναι hot spot γιατί βρίσκεται μέσα σε βρόγχο ο οποίος διατρέχει τη λίστα ακμών του υφιστάμενου κόμβου. Επιπλέον γίνεται πρόσβαση στη μνήμη λόγω του δείκτη και εκτέλεση λογικής πράξης σύγκρισης. Το σημείο αυτό φαίνεται στον κώδικα πιο κάτω:

```
while ( this_arc_P != NULL )
        {
        head_node_P = this_arc_P->head_node;
        PF3( "--- Node %c dist=%d  state=%d\n",(char)head_node_P->id,
(int)head_node_P->dist, head_node_P->state );
```

2. Το δεύτερο hot spot της συνάρτησης αυτής είναι το σημείο στο οποίο γίνεται έλεγχος εάν φτάσαμε στο τέλος της λίστας, γιατί γίνεται πρόσβαση στη μνήμη και λογική πράξη σύγκρισης. Το σημείο αυτό φαίνεται στον κώδικα πιο κάτω:

```
        /* by definition, an arc must have a head node pointer */
        assert( head_node_P != NULL );
        if ( head_node_P->state == unknown )
```

Σύμφωνα με τη μικροαρχιτεκτονική ανάλυση που έκανα, σε αυτό το σημείο είχε γίνει το 51% των προσβάσεων στη L1 cache για τα benchmarks OSPF 400 nodes 200 arcs και 400 nodes 399 arcs.

Συνάρτηση list_remove

Hot spots

1. Το πρώτο hot spot της συνάρτησης αυτής βρίσκεται μέσα στο βρόγχο ο οποίος διατρέχει τη λίστα των κόμβων. Είναι το σημείο στο οποίο γίνεται αλλαγή στην τιμή του δείκτη ώστε να δείχνει στον επόμενο κόμβο της λίστας των κόμβων. Είναι hot spot γιατί έχουμε πρόσβαση στη μνήμη και πράξη άθροισης για να βρεθεί η νέα διεύθυνση στην οποία πρέπει να δείχνει ο δείκτης. Το σημείο αυτό φαίνεται στον κώδικα πιο κάτω:

```
while ( curr_tnode != NULL )
   {
   if ( curr_tnode != tnode )
      {
      /* have not found it yet
       * so  go to the next node in the list */
      prev_tnode = curr_tnode;
      curr_tnode = curr_tnode->next
}
```

2. Το δεύτερο hot spot είναι ένας βρόγχος ο οποίος διατρέχει τη λίστα των κόμβων. Η συνθήκη του βρόγχου κάνει έλεγχο εάν ο δείκτης έχει φτάσει στο τέλος της λίστας, δηλαδή στο Null. Το σημείο αυτό φαίνεται στον πιο πάνω κώδικα.

Συνάρτηση list_add

Hot spots

1. Το πρώτο hot spot της συνάρτησης αυτής βρίσκεται μέσα σε ένα for loop το οποίο ουσιαστικά διατρέχει τη λίστα τον κόμβων συγκρίνοντας την τιμή της απόστασης του υφιστάμενου κόμβου ο οποίος ελέγχεται τη δεδομένη στιγμή με την τιμή της απόστασης του κάθε κόμβου στη λίστα. Ο λόγος για τον οποίο το σημείο αυτό είναι hot spot, είναι επειδή βρισκόμαστε μέσα σε βρόγχο

επανάληψης και επίσης επειδή γίνεται πρόσβαση στη μνήμη για την ανάκτηση της τιμής της απόστασης του κάθε κόμβου και επίσης επειδή γίνεται μια πράξη σύγκρισης ανάμεσα στις 2 τιμές. Το σημείο αυτό φαίνεται στον κώδικα πιο κάτω:

```
for ( ;; )
        {
        if ( tnode->dist > curr_node->dist )
           {
           if ( curr_node->next != NULL )
              {
              prev_node = curr_node;     /* remember previous node */
              curr_node = curr_node->next; /* go to the next node */
              }
```

2. Το δεύτερο hot spot της συνάρτησης, βρίσκεται επίσης μέσα σε αυτή τη συνάρτηση και είναι το σημείο στο οποίο γίνεται έλεγχος κατά πόσο ο δείκτης του υφιστάμενου κόμβου δείχνει στο Null, δηλαδή εάν έχει φτάσει στο τέλος της λίστας. Είναι hot spot γιατί βρίσκεται μέσα σε βρόγχο, και γιατί γίνεται πρόσβαση στη μνήμη και πράξη σύγκρισης. Το σημείο αυτό φαίνεται στον πιο πάνω κώδικα.

Σύμφωνα με τη μικροαρχιτεκτονική ανάλυση που έκανα, σε αυτό το σημείο είχε γίνει το 44% των προσβάσεων στη L1 cache για τα benchmarks OSPF 400 nodes 2 arcs και 400 nodes 4 arcs.

5.2 Ανάλυση του benchmark IP Packet Check

Εφαρμογή

Το benchmark αυτό υλοποιεί ένα υποσύνολο των λειτουργιών που εκτελεί η συνάρτηση προώθησης (forwarding function) των πακέτων στο network layer του Internet Protocol suite όπως ορίζεται από το RFC1812. Η κανονική λειτουργία ενός TCP/IP router ορίζει πως ο router αυτός, εξετάζει το IP header των πακέτων που παραλαμβάνει, τα αποθηκεύει σε ένα προσωρινό χώρο αποθήκευσης και βάσει της ετικέτας (header) που έχουν τα στέλνει παρακάτω σε κάποιο άλλο δρομολογητή στο δίκτυο μέχρι να φτάσουν στον προορισμό τους. Συγκεκριμένα όταν ένας router παραλάβει ένα πακέτο αφαιρεί το header του και βάζει ένα καινούριο link layer header (που είναι το επόμενο layer του διαδικτύου στο οποίο θα προωθηθεί). Η πιο πάνω διαδικασία στο συγκεκριμένο benchmark υποθέτουμε ότι έχει ήδη γίνει.

Περιγραφή του benchmark

Για το κάθε πακέτο (datagram) υπολογίζεται ένα checksum για την IP ετικέτα (header) του και αποθηκεύεται σε εκείνο. Το checksum υπολογίζεται, γιατί σε περίπτωση λάθους το πακέτο διαγράφεται από την προσωρινή μνήμη του δρομολογητή (router) και δεν στέλνεται παρακάτω. Το μέγεθος των πακέτων επιλέγεται τυχαία έτσι ώστε να αναπαριστάται με τον καλύτερο δυνατό τρόπο η κατάσταση που υπάρχει στα πραγματικά δίκτυα υπολογιστών.

Γίνεται προσομοίωση ενός router με 4 network interfaces. Συγκεκριμένα, ενός router που έχει 4 διαφορετικά μεγέθη από buffers πακέτων (ή network interfaces). Αυτά είναι:

- 512 KB
- 1 MB
- 2 MB
- 4 MB

Λόγω του ότι τα μεγέθη των buffers διαφέρουν, διαφέρει και ο αριθμός των πακέτων που μπορούν να αποθηκευτούν στο καθένα, άρα και ο αριθμός των δεδομένων που έχει να επεξεργαστεί το πρόγραμμα κάθε φορά.

Το benchmark αυτό, υλοποιεί 2 ουρές οι οποίες περιγράφονται στη συνέχεια. Η βασική δουλεία που επιτελείται στον κώδικα του προγράμματος είναι να στέλνονται πακέτα από τη μία ουρά, η οποία είναι η ουρά που τα παραλαμβάνει, προς την άλλη ουρά, η οποία είναι η ουρά αναμονής των πακέτων μέχρι αυτά να σταλούν παρακάτω στο δίκτυο.

Ανάλυση των υπολογιστικών πόρων που εξετάζει το benchmark

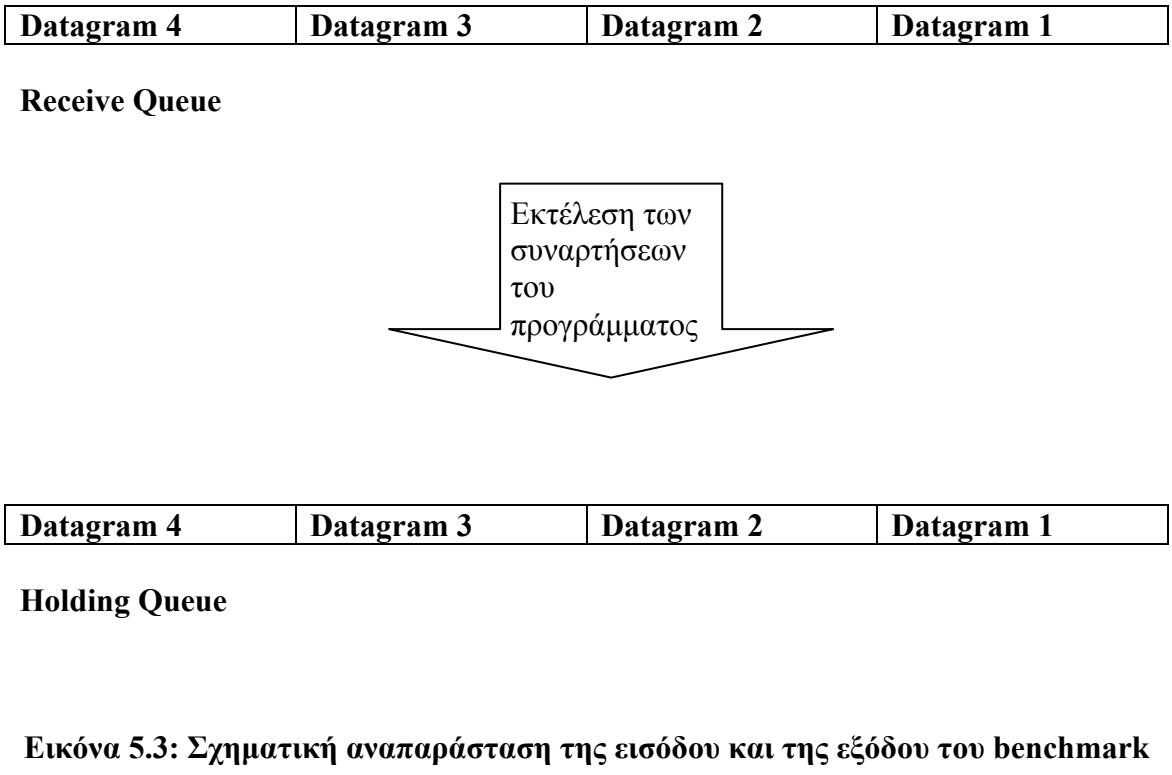
Οι πράξεις που εκτελούνται στο συγκεκριμένο benchmark και για τις οποίες ελέγχεται ο επεξεργαστής, είναι integer math με 16 bit unsigned, πράξεις shift και πράξεις λογικών συγκρίσεων. Οι πράξεις όμως στις οποίες επικεντρώνεται περισσότερο μεταξύ αυτών, είναι οι υπολογισμοί των checksum και οι πράξεις λογικών συγκρίσεων.

Το benchmark αυτό ελέγχει και την ανάκτηση δεδομένων από τη μνήμη αφού οι πράξεις που εκτελούνται από το συγκεκριμένο benchmark ανακτούν δεδομένα από τη μνήμη.

Input και output του benchmark

Η είσοδος του προγράμματος δημιουργείται δυναμικά στον υπολογιστή κατά την εκτέλεση του προγράμματος. Ουσιαστικά δημιουργούνται τα πακέτα στη μνήμη του υπολογιστή και απευθείας τοποθετούνται στην ουρά στη οποία τοποθετούνται τα πακέτα που παραλαμβάνονται.

Η έξοδος του προγράμματος είναι η ουρά στην οποία μεταφέρονται τα πακέτα και μέσω της οποίας θα σταλούν παρακάτω στο δίκτυο.
Στο πιο κάτω σχήμα φαίνεται η σχηματική αναπαράσταση της εισόδου και της εξόδου του προγράμματος:



Εικόνα 5.3: Σχηματική αναπαράσταση της εισόδου και της εξόδου του benchmark IP Packet Check

Όταν εκτελεστούν οι συναρτήσεις του προγράμματος παράγεται η holding queue η οποία κρατά τα πακέτα μέχρι αυτά να σταλούν παρακάτω στο δίκτυο.

Τύποι και δομές δεδομένων που χρησιμοποιούνται

Ο βασικός τύπος δεδομένων που χρησιμοποιείται στο πρόγραμμα είναι ένα structure το οποίο αναπαριστά τα πακέτα (datagrams) τα οποία χρησιμοποιεί το benchmark για να εκτελέσει τις λειτουργίες που είναι προγραμματισμένο να κάνει. Η δομή αυτή είναι η πιο κάτω [7]:

```
#define HDR_SIZE (20)       /* Size of IP header; shouldn't change */

static e_u8 ip_hdr_ex[] = { /* IP header */
    (0x4 << 4) | 5,             /* IPv4 (4bits); hdr len in 32b wds
(4bits) */
    0x08,                   /* TOS = bulk data (8 bits) */
```

```
    ((20 >> 8) & 0xff),      /* Total len in octets (MS 8 bits of 16)
*/
    (20 & 0xff),             /* Total len in octets (LS 8 bits of 16)
*/
    0x34,                   /* ID field (MS 8 bits of 16) */
    0x7f,                   /* ID field (LS 8 bits of 16) */
    (0<<7)|(0<<6)|(0<<5)|(0&0x1f),
                            /* Flags: rsvd/don'tfrag/morefrags;  */
                            /* fragment offset (MS 5 bits of 13) */
    (0 & 0xff),             /* Fragment offset (LS 8 bits of 13) */
    15,                     /* Time To Live (8 bits) */
    6,                      /* Protocol: 6 = TCP (8 bits) */
    0,                      /* Header chksum (MS 8 bits of 16) */
    0,                      /* Header chksum (LS 8 bits of 16) */
    192,                    /* IP src addr (MMS 8 bits of 32) */
    160,                    /* IP src addr (MLS 8 bits of 32) */
    157,                    /* IP src addr (LMS 8 bits of 32) */
    133,                    /* IP src addr (LLS 8 bits of 32) */
    192,                    /* IP dst addr (MMS 8 bits of 32) */
    160,                    /* IP dst addr (MLS 8 bits of 32) */
    157,                    /* IP dst addr (LMS 8 bits of 32) */
    134                     /* IP dst addr (LLS 8 bits of 32) */
    /* IP options would go here */
    /* Data would go here */
    /* End of datagram */
};
```

Ο συγκεκριμένος τύπος δεδομένων περιλαμβάνει όλα τα χαρακτηριστικά ενός πακέτου το οποίο αναπαρίσταται με τον τρόπο που φαίνεται πιο κάτω [7]:

```
*    0                   1                   2                   3
*    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
*   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
*   |Version|  IHL  |Type of Service|          Total Length         |
*   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
*   |         Identification        |Flags|      Fragment Offset    |
*   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
*   |  Time to Live |    Protocol   |         Header Checksum       |
*   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
*   |                       Source Address                          |
*   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
*   |                     Destination Address                       |
*   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
*   |                     Options                       |    Padding    |
*   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
*
*                   Internet Datagram Header
```

Οι βασικές δομές δεδομένων που χρησιμοποιούνται στο πρόγραμμα είναι 2 ουρές. Οι ουρές αυτές είναι τύπου datagram. Η **receive queue** στην οποία αποθηκεύονται τα πακέτα που καταφθάνουν στον δρομολογητή και η **holding queue** η οποία κρατά τα πακέτα που ο δρομολογητής θα στείλει παρακάτω στο δίκτυο.

Γενική περιγραφή των λειτουργιών που εκτελεί ο κώδικας του benchmark

Στον κώδικα του προγράμματος, βασικά υλοποιείται η μεταφορά των πακέτων από την receive queue προς τη holding queue. Μαζί με τη μεταφορά αυτή όμως γίνονται και όλοι οι απαραίτητοι έλεγχοι για την ορθότητα των πακέτων. Τα πακέτα που αποτυγχάνουν έστω και σε ένα έλεγχο, διαγράφονται.

Οι έλεγχοι που γίνονται για το κάθε πακέτο που παραλαμβάνεται είναι οι πιο κάτω:

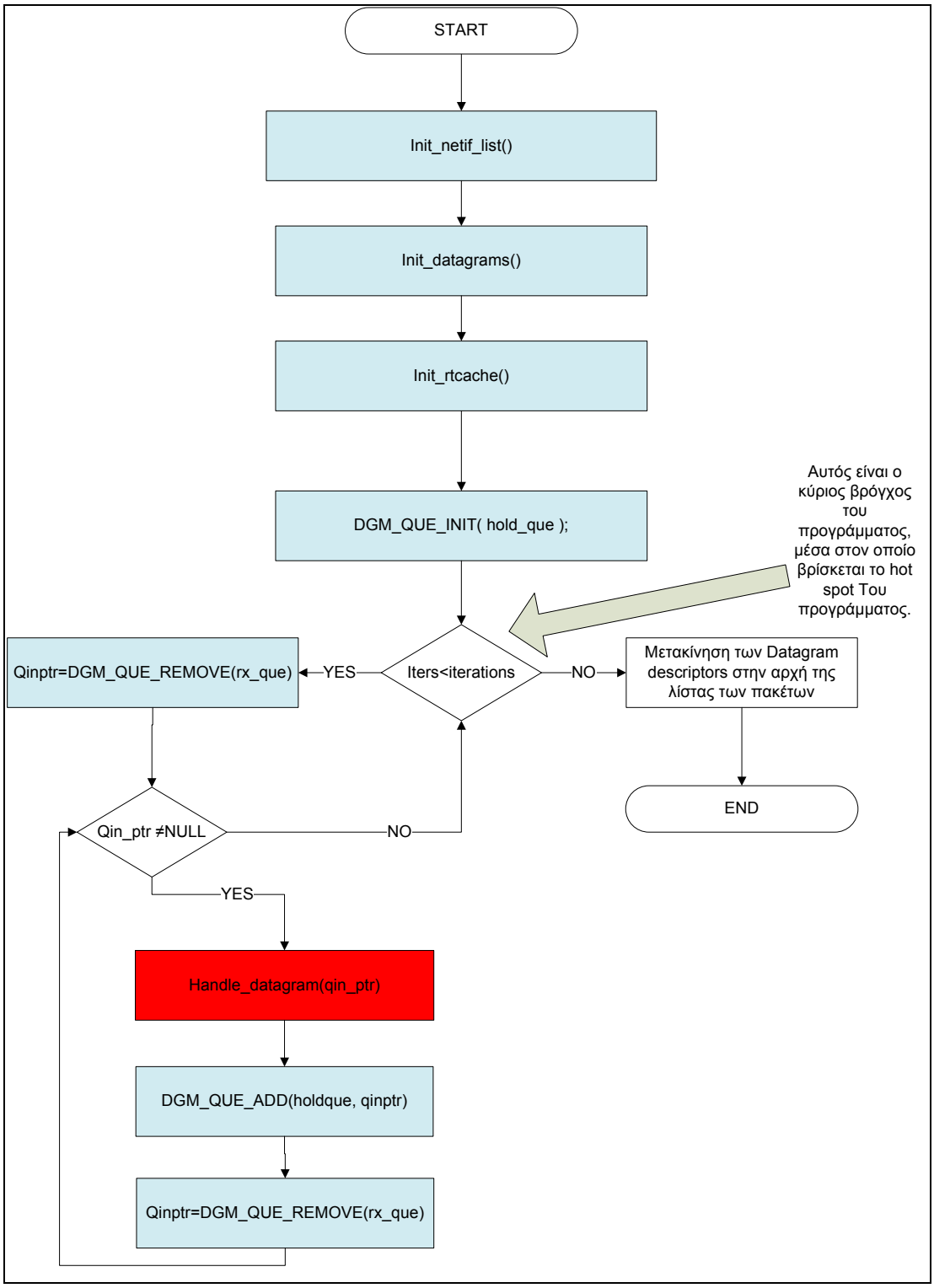
- Το μέγεθος του πακέτου να είναι μεγαλύτερο από 20 bytes
- Το checksum του IP header είναι ορθό
- Το version του IP protocol είναι το 4

- Το μήκος του IP header είναι αρκετά μεγάλο για να κρατά το ελάχιστο μέγεθος κάποιου IP πακέτου.
- Το συνολικό μέγεθος του πεδίου IP είναι αρκετά μεγάλο για να κρατά το header του πακέτου του οποίου το μέγεθος καθορίζεται από το μήκος του IP header.

Για να ολοκληρωθεί μια επανάληψη (ένα iteration) του κώδικα πρέπει να αδειάσει η ουρά **receive_queue.**

Πιο κάτω παρουσιάζεται ένα ενδεικτικό διάγραμμα ροής που παρουσιάζει τη σειρά κλήσης των συναρτήσεων του προγράμματος.

Ενδεικτικό διάγραμμα ροής για το benchmark



Εικόνα 5.4: Ενδεικτικό διάγραμμα ροής για το benchmark IP Packet Check

Με κόκκινο χρώμα αναπαρίστανται οι κλήσεις των συναρτήσεων οι οποίες αναλώνουν τον πιο πολύ χρόνο εκτέλεσης βάσει του κώδικα και των πράξεων που φαίνονται να γίνονται σ' αυτές.

Με γαλάζιο χρώμα αναπαρίστανται οι κλήσεις των υπόλοιπων συναρτήσεων.

Από το πιο πάνω διάγραμμα ροής του προγράμματος, μπορούμε να πούμε πως η πολυπλοκότητά του είναι η πιο κάτω:

Complexity = O (y x n)

Όπου:

y: είναι ο αριθμός των iterations για τα οποία τρέχει το πρόγραμμα και
n: είναι ο αριθμός των πακέτων που διαχειρίζεται το benchmark, ανάλογα με το μέγεθος του buffer των πακέτων που έχει ο δρομολογητής.

Αυτό συμβαίνει γιατί ανάλογα με τον αριθμό των πακέτων που είναι αποθηκευμένα στο buffer, τόσες φορές θα κληθεί η συνάρτηση που μετακινεί τα πακέτα από τη receive queue προς τη holding queue (handle_datagram).

Περιγραφή των συναρτήσεων που παρουσιάζονται στο ενδεικτικό διάγραμμα ροής

Συνάρτηση init_netif_list( ): Δημιουργεί μια λίστα με όλα τα network interfaces που υπάρχουν στο συγκεκριμένο δρομολογητή

Συνάρτηση init_datagrams(): Δημιουργεί τα πακέτα τα οποία διαχειρίζεται το πρόγραμμα στη μνήμη του υπολογιστή. Μόνο το IP header χρειάζετεαι να αρχικοποιηθεί

Συνάρτηση init_rtcache( ): Δημιουργεί τη routing cache, η οποία εδώ ορίζεται ως η μνήμη του δρομολογητή και δε σχετίζεται με τα πακέτα που έχουν δημιουργηθεί στη μνήμη του υπολογιστή μας.

Συνάρτηση DGM_QUE_INIT: Η συνάρτηση αυτή αρχικοποιεί μια δομή ουράς.

Συνάρτηση DGM_QUE_REMOVE: Αφαιρεί ένα πακέτο από την ουρά αναμονής των στοιχείων.

Συνάρτηση DGM_QUE_ADD: Προσθέτει ένα πακέτο στην ουρά αναμονής των πακέτων. Χρησιμοποιεί τον κώδικα πρόσθεσης στοιχείου σε ουρά.

Συνάρτηση Handle_datagram: Η συνάρτηση αυτή, είναι η συνάρτηση μέσω της οποίας γίνεται επιτρεπτή η μετακίνηση των πακέτων από τη receive queue στη holding queue. Ουσιαστικά σε αυτή τη συνάρτηση γίνονται όλοι οι απαραίτητοι έλεγχοι ορθότητας των πακέτων για να μπορούν να μετακινηθούν από τη μια ουρά στην άλλη.
Η συνάρτηση αυτή, καλεί στον κώδικα της και τη συνάρτηση check_ip_hdr_checksum η οποία κάνει τον έλεγχο του checksum για την ορθότητα του header των πακέτων, και η οποία είναι το hot spot του προγράμματος, αναλώνοντας το πιο μεγάλο ποσοστό χρόνου εκτέλεσης και ολοκληρώνοντας το μεγαλύτερο ποσοστό των εντολών.

Για το συγκεκριμένο benchmark υπάρχουν 4 εκτελέσιμα αρχεία. Το κάθε ένα από αυτά αναπαριστά ένα μέγεθος του buffer του δρομολογητή. Υπάρχουν 4 μεγέθη. Αυτά είναι:

- 512 KB
- 1 MB
- 2 MB
- 4 MB

Όσο πιο μεγάλο είναι το μέγεθος του buffer, τόσο πιο μεγάλος είναι ο αριθμός των πακέτων που έχει να διαχειριστεί το πρόγραμμα.

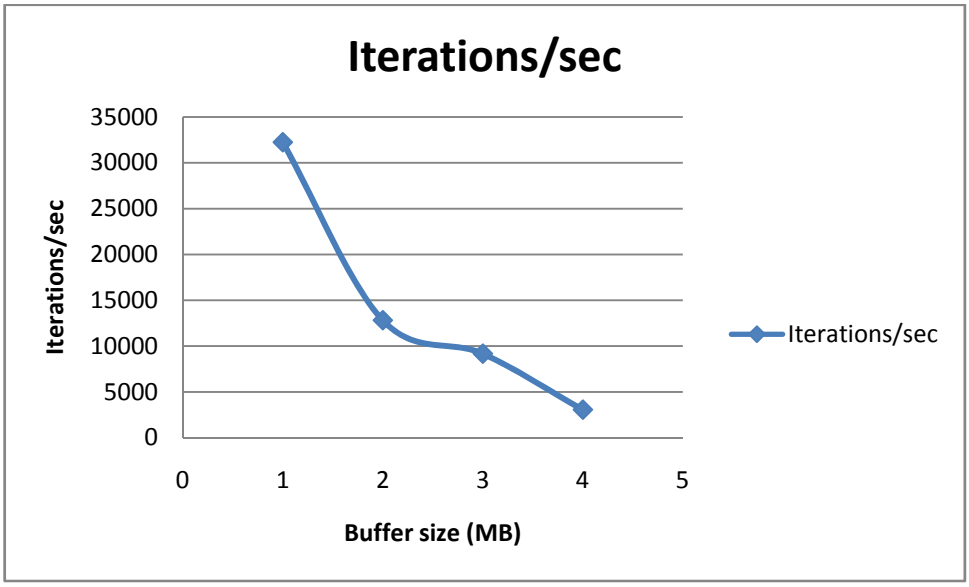
Πιο κάτω παρουσιάζεται ένας συνοπτικός πίνακας αποτελεσμάτων για τους χρόνους εκτέλεσης και τους αριθμούς των εντολών που εκτελέστηκαν για τα τέσσερα διαφορετικά προγράμματα.

Συνοπτικός πίνακας αποτελεσμάτων

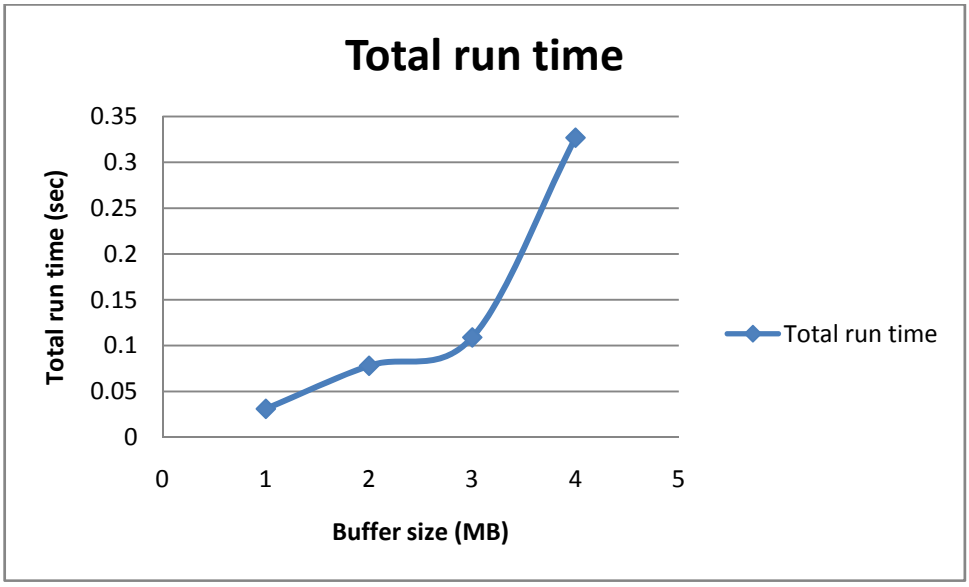
	IP packet check 512 KB	IP packet check 1 MB	IP packet check 2 MB	IP packet check 4 MB
Iterations/sec	32258,06	12820,51	9174,31	3058,1
Total run time	0,031	0,078	0,109	0,327
Time/Iteration	0,000031	0,000078	0,000109	0,000327
CPI	1,172	0,96	0,983	0,904
Συνολικός χρόνος (Εκατ. κύκλοι)	142,8	203,7	373,8	676,2
Συνολικός αριθμός εντολών (Εκατ.)	121,8	212,1	380,1	747,6

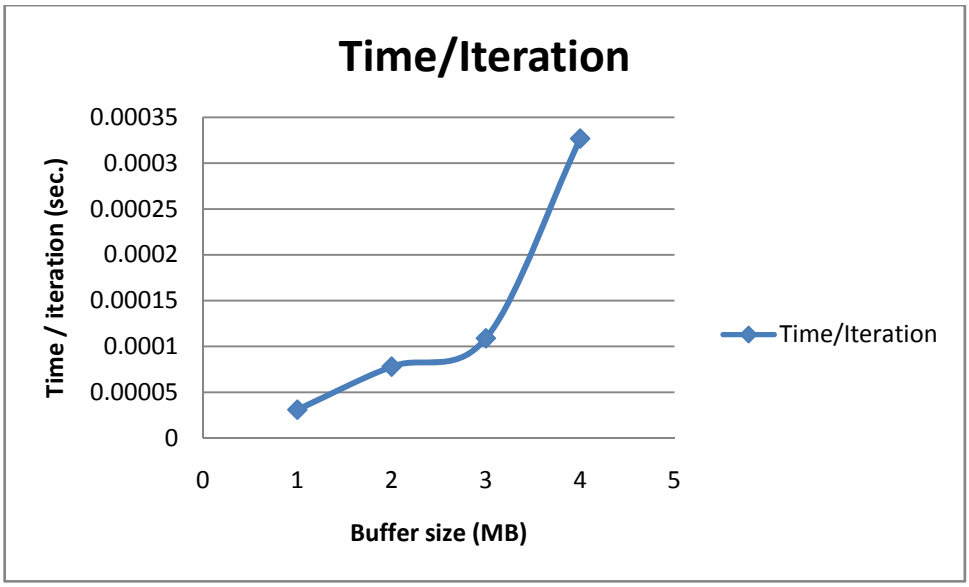
Πίνακας 5.2: Συνοπτικός πίνακας που παρουσιάζει τη συμπεριφορά του προγράμματος για κάθε διαφορετικό input

Στα πιο κάτω διαγράμματα φαίνεται η σύγκριση των ποσοστών χρόνου και εντολών για τον πιο πάνω πίνακα.

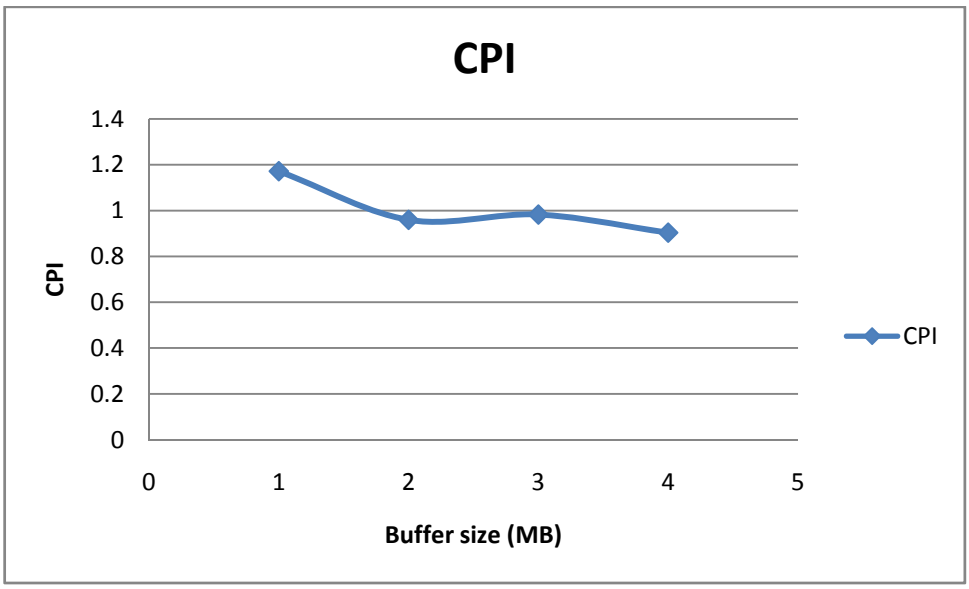


Όπως παρατηρούμε, όσο αυξάνεται το μέγεθος του buffer, τόσο μειώνεται ο αριθμός των iterations που ολοκληρώνονται ανά δευτερόλεπτο. Άρα το πρόγραμμα γίνεται πιο αργό, επειδή μεγαλώνει ο αριθμός των πακέτων που επεξεργάζεται.

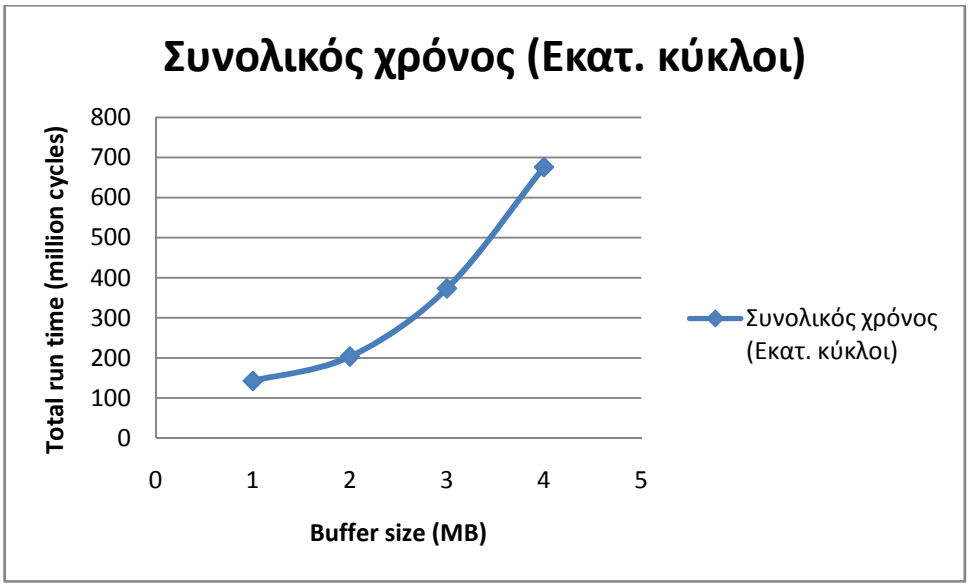




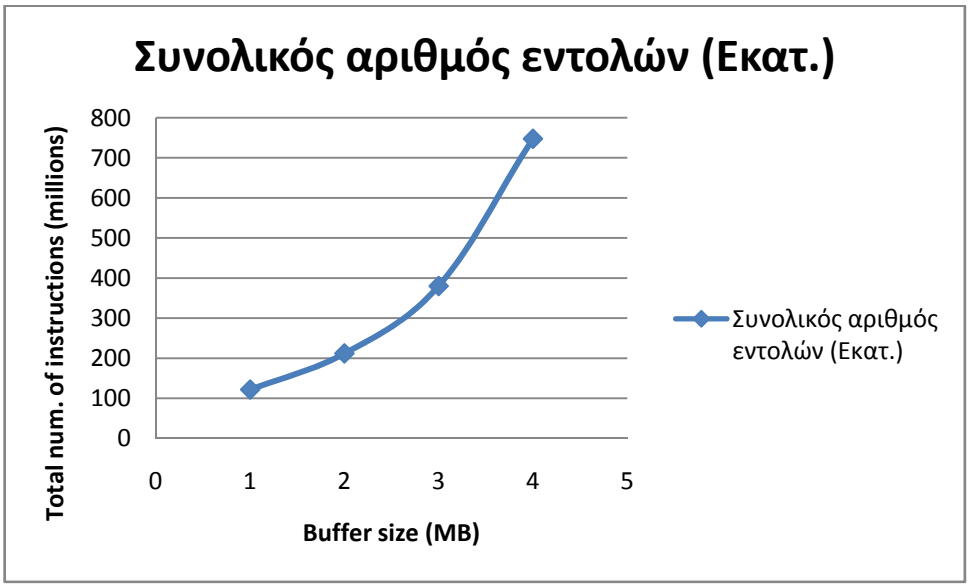
Όπως παρατηρούμε από το πιο πάνω διάγραμμα, όσο αυξάνεται το μέγεθος του buffer αυξάνεται ο χρόνος που χρειάζεται για να ολοκληρωθεί ένα iteration. Αυτό συμβαίνει επειδή αυξάνεται ο αριθμός των πακέτων που έχει να μεταφέρει το πρόγραμμα από τη receive queue στη holding queue, λόγω του αυξημένου μεγέθους του buffer. Επίσης λόγω αυτού αυξάνεται ο αριθμός των ελέγχων που γίνονται συνεπώς αυξάνεται ο αριθμός των λογικών συγκρίσεων και των πράξεων checksum που γίνονται άρα δικαιολογείται έτσι η αύξηση του χρόνου που χρειάζεται για να ολοκληρωθεί ένα iteration.



Για το CPI παρατηρούμε ότι μειώνεται όσο αυξάνεται το μέγεθος του buffer. Αυτό συμβαίνει γιατί διαφοροποιείται ο αριθμός των εντολών που εκτελούνται από το κάθε είδος εντολής. Το CPI των εντολών αυτών διαφέρει συμβάλλοντας με διαφορετικό τρόπο στο συνολικό CPI του προγράμματος.

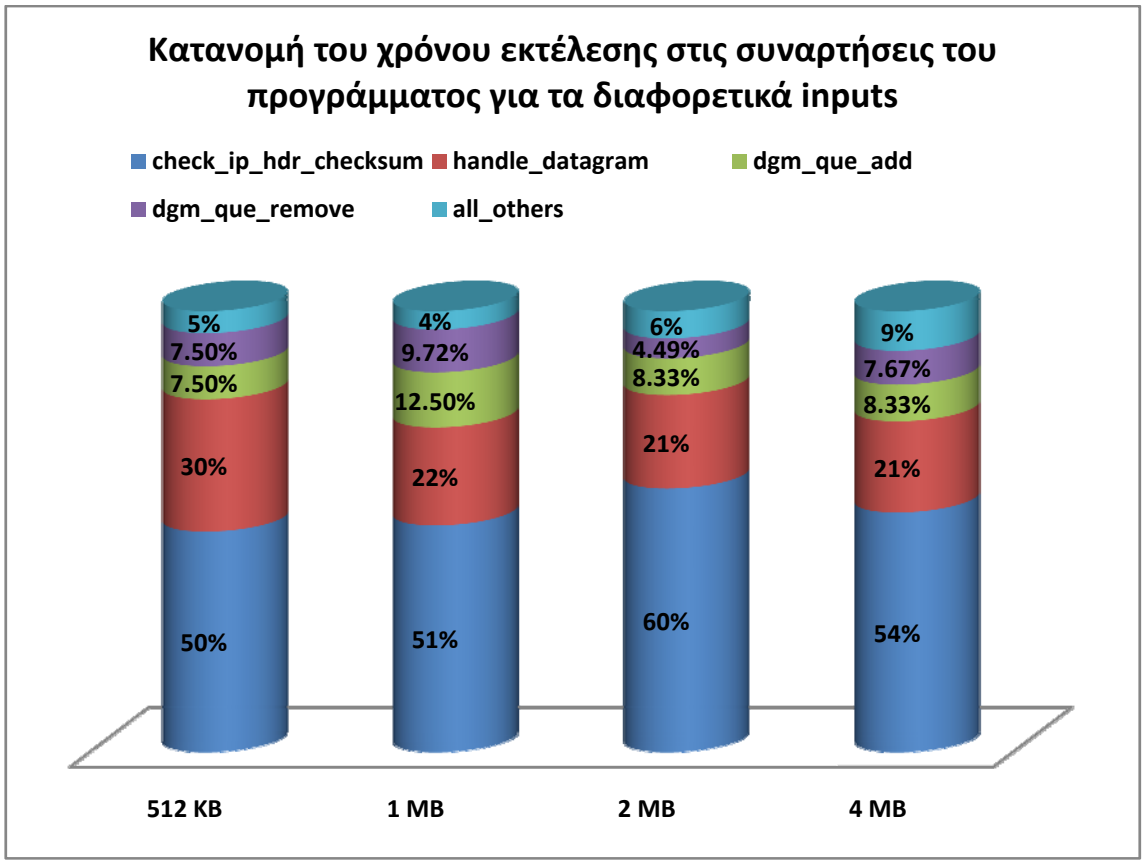


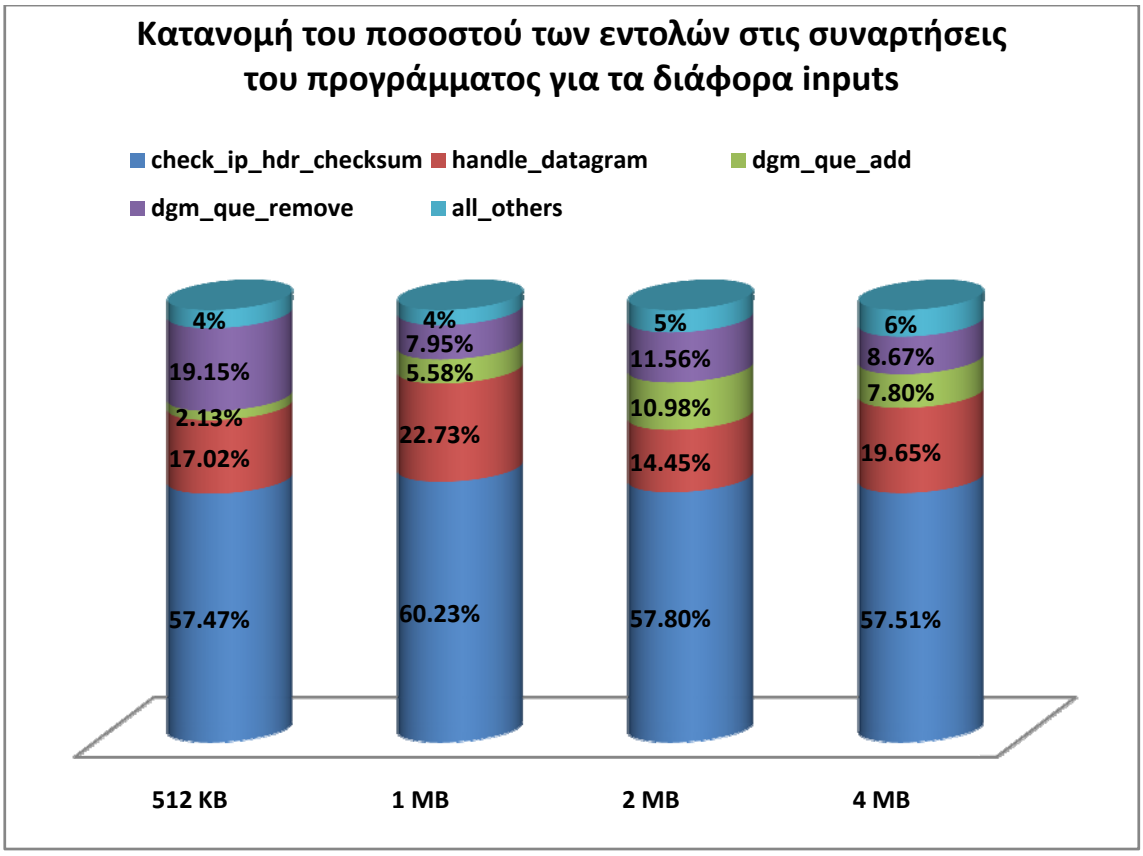
Όπως καταλαβαίνουμε και από τα προηγούμενα διαγράμματα, όσο αυξάνεται το μέγεθος του buffer τόσο αυξάνεται ο συνολικός χρόνος εκτέλεσης του προγράμματος.



Αντίστοιχα με τα αποτελέσματα του πιο πάνω διαγράμματος, όσο αυξάνεται το μέγεθος του buffer αυξάνεται και ο συνολικός αριθμός εντολών που εκτελούνται, αφού αυξάνεται ο αριθμός των δεδομένων που επεξεργάζονται.

Στα πιο κάτω διαγράμματα φαίνεται το ποσοστό χρόνου και εντολών που ανάλωσε η κάθε συνάρτηση του προγράμματος, ανάλογα με το μέγεθος του buffer.





Όπως φαίνεται από τα πιο πάνω διαγράμματα, για όλα τα μεγέθη των buffers, η συνάρτηση που καταναλώνει το μεγαλύτερο ποσοστό του χρόνου και εκτελεί το μεγαλύτερο ποσοστό εντολών, είναι η check_ip_hdr_checksum και ακολουθούν οι συναρτήσεις handle_datagram, dgm_que_add και dgm_que_remove.

Πιο κάτω παρουσιάζεται ο κώδικας των συναρτήσεων στα σημεία που σπαταλείται ο περισσότερος χρόνος εκτέλεσης και που εκτελείται το μεγαλύτερο ποσοστό εντολών.

Συνάρτηση check_ip_hdr_checksum

Η συνάρτηση αυτή κάνει τον έλεγχο του checksum για το header του κάθε πακέτου.

Hot Spots:

1. Το πρώτο hot spot αυτής της συνάρτησης είναι ένας βρόγχος, ο οποίος διατρέχει ανά 2 bits το header του πακέτου υπολογίζοντας παράλληλα το checksum του.

Το σημείο αυτό είναι hot spot γιατί βρίσκεται σε βρόγχο και επειδή εδώ γίνονται πράξεις λογικής σύγκρισης. Επιπλέον έχουμε πρόσβαση στη μνήμη με τη χρήση του δείκτη *addr. Ο κώδικας του σημείου αυτού φαίνεται πιο κάτω.

```
for ( count = ((*(hdrptr)) & 0xf) * 2; count > 0; count--, addr++ )
    {
   /* Pick up the next 2 bytes as a 16-bit value;
    * note that native byte order does NOT matter.  (See RFC1071)
    */
    sum += *addr;
    }
```

Όπως φαίνεται από τη μικροαρχιτεκτονική ανάλυση που έκανα, σε αυτό το σημείο γίνεται ποσοστό προσβάσεων στη μνήμη μεγαλύτερο του 60% για όλα τα μεγέθη input του προγράμματος.

2. Το δεύτερο hot spot αυτής της συνάρτησης βρίσκεται και πάλι σε ένα βρόγχο ο οποίος εφαρμόζει την πράξη του shifting στο άθροισμα που προκύπτει από το προηγούμενο hot spot. Είναι hot spot γιατί έχουμε την πράξη του shifting που σπαταλά πολλούς κύκλους για να εκτελεστεί, όπως επίσης και βρόγχο μέσα στον οποίο εκτελείται η πράξη.

```
while ( sum >> 16 )
      {
      sum = (sum & 0xffff) + (sum >> 16);
      }
```

Τα 2 αυτά σημεία είναι και τα σημεία που σπαταλούν τον περισσότερο από το χρόνο εκτέλεσης του προγράμματος και είναι εδώ επίσης που εκτελείται ο μεγαλύτερος αριθμός εντολών. Ουσιαστικά σε αυτά τα σημεία του κώδικα υλοποιείται ο βασικότερος έλεγχος που γίνεται πάνω στα πακέτα, που είναι ο υπολογισμός του checksum των πακέτων.

Συνάρτηση handle_datagram

Hot Spots:

1. Η συνάρτηση αυτή έχει ένα hot spot. Το hot spot αυτό είναι το σημείο στο οποίο γίνεται ο έλεγχος αν το μήκος του header του πακέτου είναι μεγαλύτερο ή ίσο των 20 bytes. Ο κώδικας του σημείου αυτού φαίνεται πιο κάτω.

```
if ( ((*(byteptr+2)) << 8) + (*(byteptr+3)) < 20 )
```

Είναι hot spot γιατί έχουμε πρόσβαση στη μνήμη μέσω του δείκτη *byteptr, έχουμε πράξη shifting η οποία σπαταλά πολλούς κύκλους για να εκτελεστεί και επίσης έχουμε πράξη λογικής σύγκρισης.

Συνάρτηση dgm_que_remove

Hot spots

1. Η συνάρτηση αυτή έχει ένα hot spot. Το hot spot αυτό είναι το σημείο στο οποίο ο δείκτης που δείχνει στον πρώτο κόμβο της λίστας, μετακινείται και δείχνει στον επόμενο κόμβο της λίστας.

Ο κώδικας του σημείου αυτού φαίνεται πιο κάτω:

```
que->head = que->head->next;
```

Είναι hot spot γιατί έχουμε πρόσβαση στη μνήμη.

Συνάρτηση dgm_que_add

Hot spots

1. Η συνάρτηση αυτή έχει ένα hot spot. Το hot spot αυτό είναι το σημείο στο οποίο αυξάνεται η τιμή της μεταβλητής που κρατά τον αριθμό των κόμβων που υπάρχουν στην ουρά. Ο κώδικας του σημείου αυτού φαίνεται πιο κάτω:

   ```
   que->cnt++;
   ```

Το συγκεκριμένο σημείο είναι hot spot γιατί έχουμε πρόσβαση στη μνήμη, μέσω δείκτη.

5.2 Ανάλυση του benchmark Route Lookup

Εφαρμογή

Το benchmark αυτό υλοποιεί την πιο απλή λειτουργία που εκτελείται στα δίκτυα υπολογιστών. Δηλαδή την αποστολή και παραλαβή πακέτων. Ουσιαστικά υλοποιεί τη λειτουργία κατά την οποία όταν φτάσει στο δρομολογητή ένα πακέτο, του λέει σε ποιο από τα ports του να το προωθήσει ώστε να προχωρήσει παρακάτω στο δίκτυο, μέχρι να φτάσει στον παραλήπτη του.

Περιγραφή του benchmark

Ο κάθε δρομολογητής κρατά έναν πίνακα, ο οποίος περιέχει IP διευθύνσεις (routing table) και τον οποίο διαβάζει κάθε φορά (lookup) που παραλαμβάνει ένα πακέτο για να αποφασίσει σε ποιο port να το προωθήσει, βάσει με τα περιεχόμενα του πίνακα.
Ο αλγόριθμος αυτός υλοποιείται με τη χρήση ενός συμπαγούς δυαδικού δέντρου, του Patricia Tree. Το δέντρο αυτό είναι ένα συμπαγές δυαδικό δέντρο το οποίο επιτρέπει γρήγορες και αποδοτικές αναζητήσεις βάσει συμβολοσειρών απεριόριστου μήκους. Εδώ χρησιμοποιείται γιατί υπάρχουν IP διευθύνσεις οι οποίες δεν έχουν σταθερό μήκος

και έτσι ήταν απαραίτητο να χρησιμοποιηθεί μια δομή η οποία να επιτρέπει αναζητήσεις για οποιοδήποτε μήκος συμβολοσειράς.

Ανάλυση των υπολογιστικών πόρων που εξετάζει το benchmark

Ο κώδικας αυτού του benchmark επαναληπτικά διατρέχει το δέντρο. Συνεπώς εξετάζεται ο χρόνος απόκρισης του επεξεργαστή για ανάκτηση δεδομένων από τη μνήμη και η ικανότητα του να διαχειρίζεται αποδοτικά συχνές Control Transfer Instructions πράξεις.

Input και output του προγράμματος

Η είσοδος του προγράμματος αποτελείται από 2 αρχεία. Το lookups.txt και το routes.txt. Το πρώτο αρχείο αναπαριστά τον αριθμό των πιθανών routes μέσω των οποίων μπορεί να προωθηθεί ένα πακέτο παρακάτω στο δίκτυο. Το δεύτερο αναπαριστά τις αναζητήσεις που μπορούν να γίνουν στο δέντρο μέχρι να βρεθεί το ποιο route πρέπει να ακολουθήσει το πακέτο.

Η έξοδος του προγράμματος είναι ουσιαστικά το σε ποιο route αποφασίστηκε να σταλεί το κάθε πακέτο που παραλήφθηκε από το δρομολογητή.

Τύποι και δομές δεδομένων που χρησιμοποιούνται

Για αυτό το benchmark οι τύποι δεδομένων που χρησιμοποιούνται, είναι τα structures tab_ent και trie_node. Ο πρώτος τύπος δεδομένων περιέχει μια μεταβλητή τύπου e_u32. Ο δεύτερος τύπος δεδομένων, αναπαριστά έναν κόμβο του συμπαγούς δυαδικού δέντρου Patricia Tree μέσω του οποίου υλοποιείται η συνάρτηση route lookup.

Η βασική δομή δεδομένων που χρησιμοποιείται είναι το **Patricia Tree.** Το Patricia Tree είναι ένας ειδικός τύπος συμπαγούς δυαδικού δέντρου, το οποίο έχει ακμές που αναπαριστούνται από σειρές χαρακτήρων ή από διευθύνσεις IP ή από απλούς χαρακτήρες σε αντίθεση με τα άλλα είδη δέντρων. Το PATRICIA αποτελεί ακρωνύμιο των λέξεων: Practical Algorithm to Retrieve Information Coded in Alphanumeric. Οι

ακμές παρουσιάζονται με λεξικογραφική σειρά και μπορούν στο σύνολο τους να σχηματίσουν διάφορες λέξεις. Το Patricia Tree χρησιμεύει για να καθορίζεται από πριν το branching, δηλαδή το ποια διακλάδωση θα ακολουθηθεί.

Structure tab_ent

```
struct tab_ent
{
  e_u32              dst_addr;
};
```

Structure trie_node

```
struct trie_node
{
  e_u32            key;
  e_u32            cmpbit;
  struct trie_node   *llink;
  struct trie_node   *rlink;
};
```

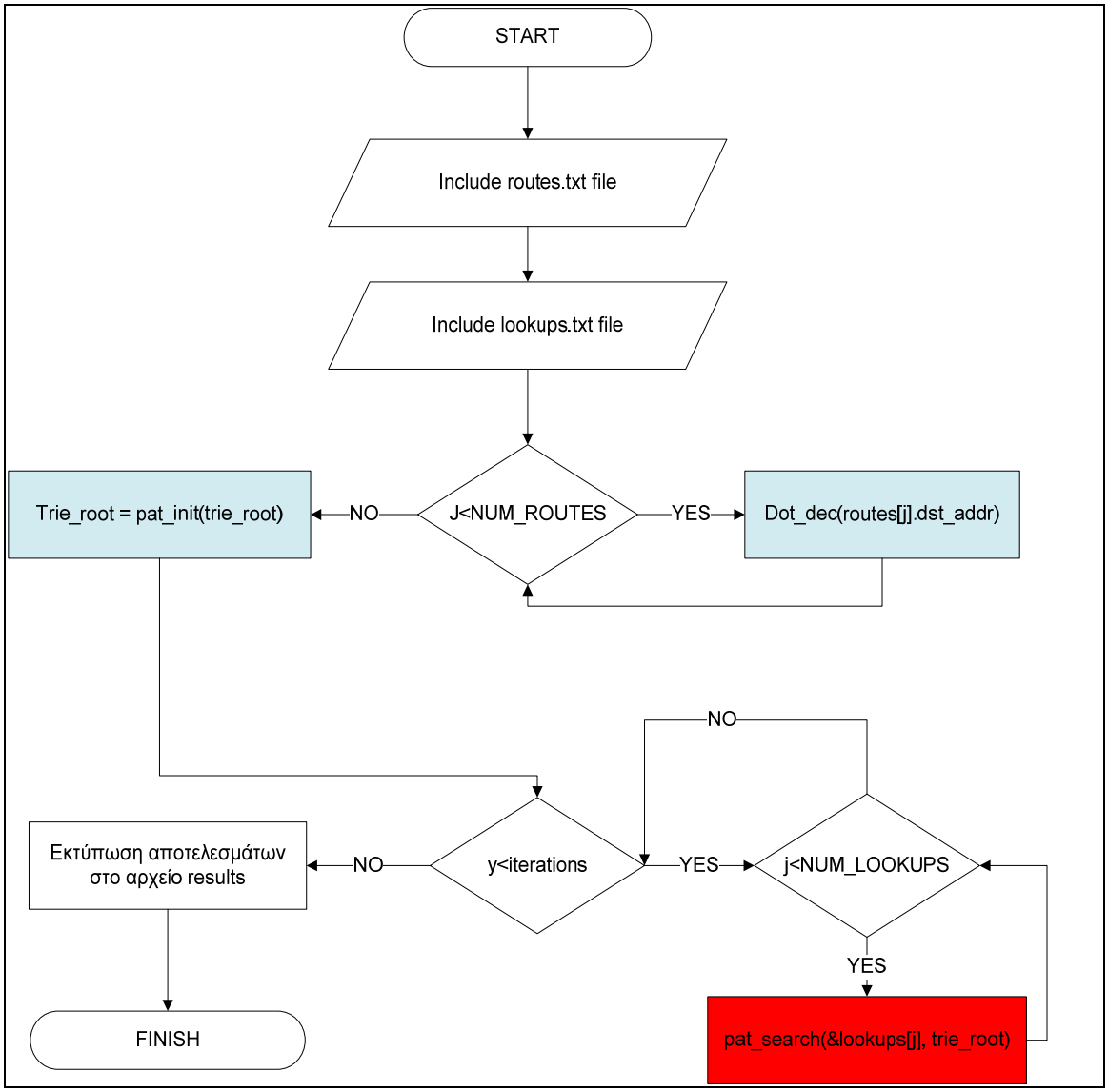
Η μεταβλητή key αναπαριστά το κλειδί του κόμβου που υλοποιείται από αυτή τη δομή. Η μεταβλητή cmpbit αναπαριστά το bit σύγκρισης του κόμβου. Οι μεταβλητές llink και rlink

Γενική περιγραφή των λειτουργιών που εξετάζει ο κώδικας του benchmark

Για κάθε IP πακέτο (route) που παραλαμβάνει ο δρομολογητής, αρχικά μετατρέπει τη διεύθυνση του σε ένα δεκαδικό αριθμό έτσι ώστε να μπορούν να γίνονται οι αναζητήσεις στο Patricia Tree. Έπειτα γίνεται η εισαγωγή αυτών των κόμβων στο Patricia Tree. Στη συνέχεια αρχίζουν οι αναζητήσεις στο Patricia Tree, για να βρεθεί σε ποιο port πρέπει να προωθηθεί το κάθε πακέτο.

Στο πιο κάτω σχήμα παρουσιάζεται ένα ενδεικτικό διάγραμμα ροής που δείχνει τη σειρά κλήσεις των συναρτήσεων του προγράμματος.

Ενδεικτικό διάγραμμα ροής για το benchmark



Εικόνα 5.5: Ενδεικτικό διάγραμμα ροής για το benchmark Route Lookup

Περιγραφή των συναρτήσεων που παρουσιάζονται στο ενδεικτικό διάγραμμα ροής

Η μεταβλητή y είναι μετρητής και μετρά τον αριθμό των iterations που έχουν ολοκληρωθεί.

Συνάρτηση dot_dec

Η συγκεκριμένη συνάρτηση, μετατρέπει μια διεύθυνση IP σε δεκαδική μορφή με τη χρήση της τελείας, για το διαχωρισμό των ακέραιων με τους δεκαδικούς αριθμούς. Η σταθερά NUM_IPS που ισούται με 16 δείχνει τον μέγιστο αριθμό κόμβων που μπορεί να κρατά ο κάθε κόμβος του δέντρου. Ο χαρακτήρας «*» που βρίσκεται στο τέλος της κάθε διεύθυνσης IP είναι ο χαρακτήρας τερματισμού και υποδηλώνει πως φτάσαμε στο τέλος αυτής της διεύθυνσης IP.

Στο τέλος, η συνάρτηση αυτή επιστρέφει τη δεκαδική διεύθυνση την οποία υπολόγισε. Αυτή η συνάρτηση καταναλώνει σημαντικό χρόνο, από το χρόνο εκτέλεσης, γιατί περιέχει πράξεις shifting οι οποίες αναλώνουν σημαντικό ποσοστό κύκλων μηχανής για να εκτελεστούν.

Συνάρτηση pat_init

Η συγκεκριμένη συνάρτηση καλεί τη συνάρτηση pat_insert τόσες φορές όσες και ο αριθμός της σταθεράς NUM_ROUTES. Η συνάρτηση pat_insert είναι η συνάρτηση που κάνει τη εισαγωγή κάποιου κόμβου στο Patricia tree. Έχει πολυπλοκότητα O(m) όπου m είναι ο αριθμός των bits που συγκρίνονται μεταξύ των κόμβων, για να καθοριστεί σε ποια θέση του δέντρου θα μπει ο νέος κόμβος.

Συνεπώς έχει πολυπλοκότητα:

O (n x m)

Όπου:

n: είναι ο αριθμός της σταθεράς NUM_ROUTES και
m: είναι ο αριθμός των bits για τα οποία γίνεται σύγκριση στη συνάρτηση pat_insert που καλείται μέσα στη συνάρτηση pat_init

Συνάρτηση pat_search

Η συνάρτηση αυτή έχει ένα βρόγχο που διατρέχει τους κόμβους του Patricia tree. Η πολυπλοκότητα της είναι O(w) όπου w είναι ο αριθμός των κόμβων του δέντρου. Όπως φαίνεται και από το πιο πάνω διάγραμμα ροής η συγκεκριμένη συνάρτηση καλείται μέσα σε ένα βρόγχο τόσες φορές όσες και ο αριθμός της σταθεράς NUM_LOOKUPS.

Συνοψίζοντας τα πιο πάνω και με βάση το διάγραμμα ροής για το benchmark καταλήγουμε στο συμπέρασμα ότι η πολυπλοκότητα του ισούται με:

O ((n x m) + (y x j x w))

Όπου:

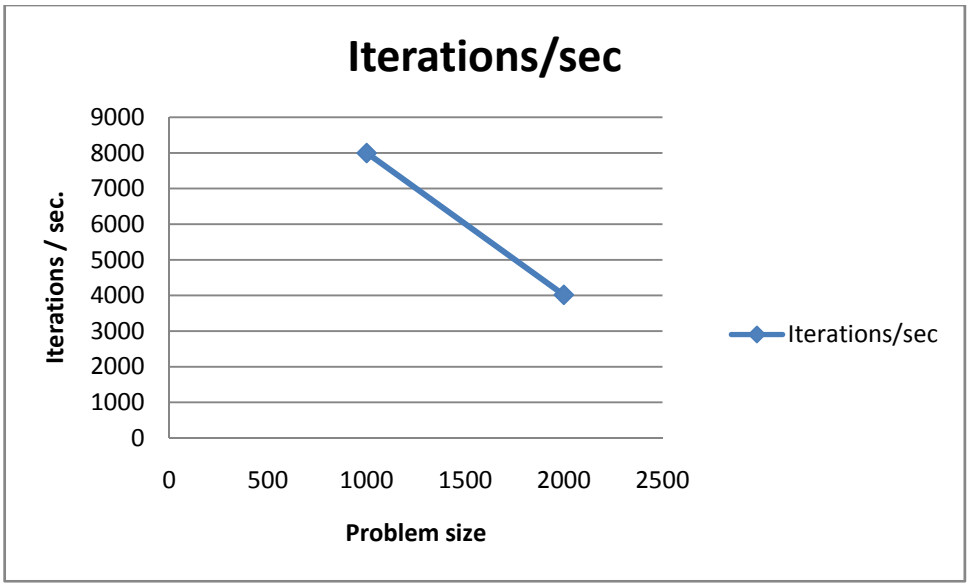
n: είναι ο αριθμός της σταθεράς NUM_ROUTES και
m: είναι ο αριθμός των bits για τα οποία γίνεται σύγκριση στη συνάρτηση pat_insert που καλείται μέσα στη συνάρτηση pat_init
y: είναι ο αριθμός των iterations για τα οποία τρέχει το πρόγραμμα
j: ο αριθμός με τον οποίο ισούται η σταθερά NUM_LOOKUPS
w: ο αριθμός των κόμβων του δέντρου (Patricia tree)

Πιο κάτω παρουσιάζεται ένας συνοπτικός πίνακας αποτελεσμάτων για τους χρόνους εκτέλεσης και τους αριθμούς των εντολών που εκτελέστηκαν για τα τέσσερα διαφορετικά προγράμματα.

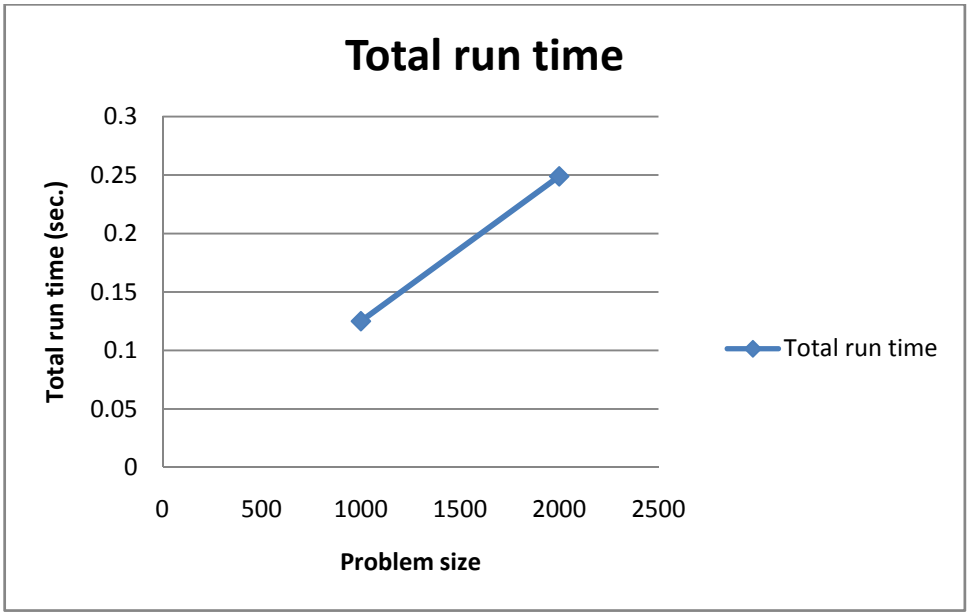
	Route Lookup 1000 looukups - 100 routes	Route Lookup 2000 lookups - 200 routes
Iterations/sec	8000	4016,06
Total run time	0,125	0,249
Time/Iteration	0,000125	0,000249
Time percentage	9,60%	10,08%
Instructions percentage	13,59%	24,97%
CPI	1,392	1,282
Συνολικός χρόνος (Εκατ. κύκλοι)	254,1	554,4
Συνολικός αριθμός εντολών (Εκατ.)	182,7	432,6

Πίνακας 5.3: Συνοπτικός πίνακας που παρουσιάζει τη συμπεριφορά του προγράμματος για κάθε τροποποιημένο input.

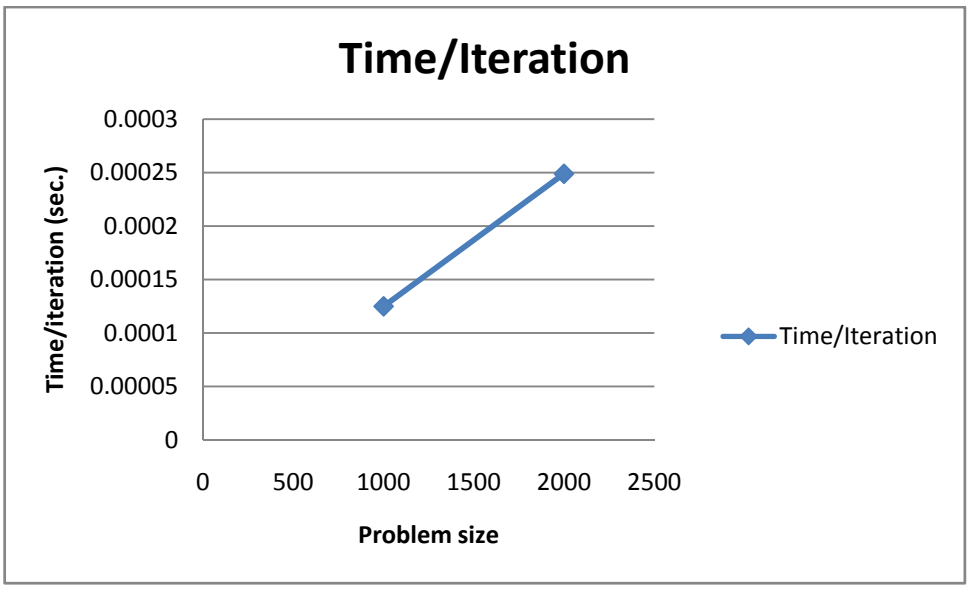
Στα πιο κάτω διαγράμματα φαίνεται η σύγκριση των ποσοστών χρόνου και εντολών για τον πιο πάνω πίνακα.



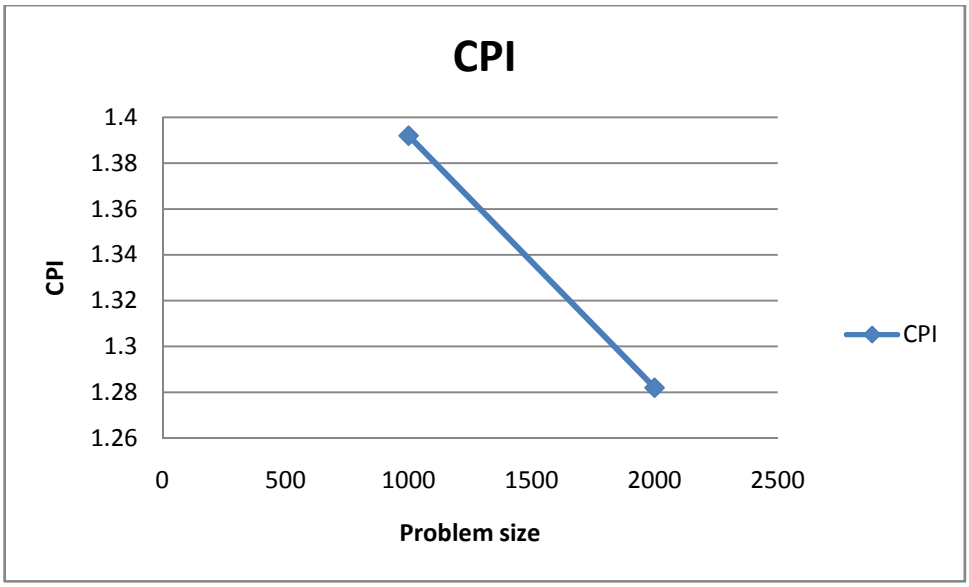
Όπως παρατηρούμε, ο αριθμός των iterations/sec είναι ο διπλάσιος για το route lookup 1000 lookups - 100 routes σε σχέση με το route lookup 2000 lookups - 200 routes. Αυτό συμβαίνει γιατί ο αριθμός των δεδομένων που επεξεργάζεται το συγκεκριμένο πρόγραμμα είναι ο μισός από το αντίστοιχο με τα διπλάσια δεδομένα, συνεπώς σε ένα δευτερόλεπτό μπορούν να ολοκληρωθούν οι διπλάσιοι κύκλοι.



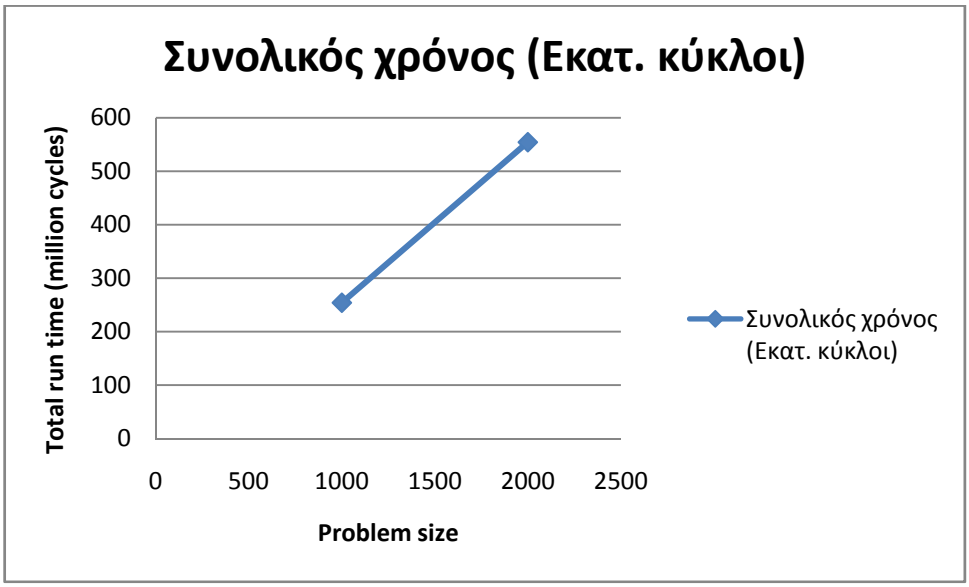
Ο χρόνος εκτέλεσης για το route lookup 2000 lookups - 200 routes είναι ο διπλάσιος από το χρόνο του route lookup 1000 lookups - 100 routes γιατί ο αριθμός των δεδομένων που επεξεργάζονται είναι ο διπλάσιος.

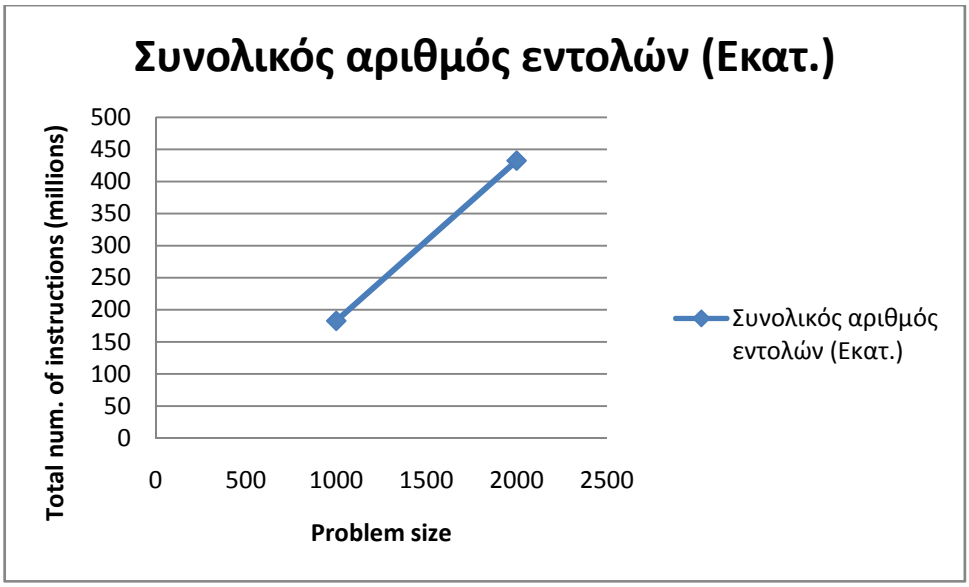


Ο χρόνος ανά iteration είναι ο διπλάσιος για το route lookup 2000 lookups - 200 routes επειδή για ένα iteration εκτελείται ο διπλάσιος αριθμός πράξεων σε κάθε κύκλο.



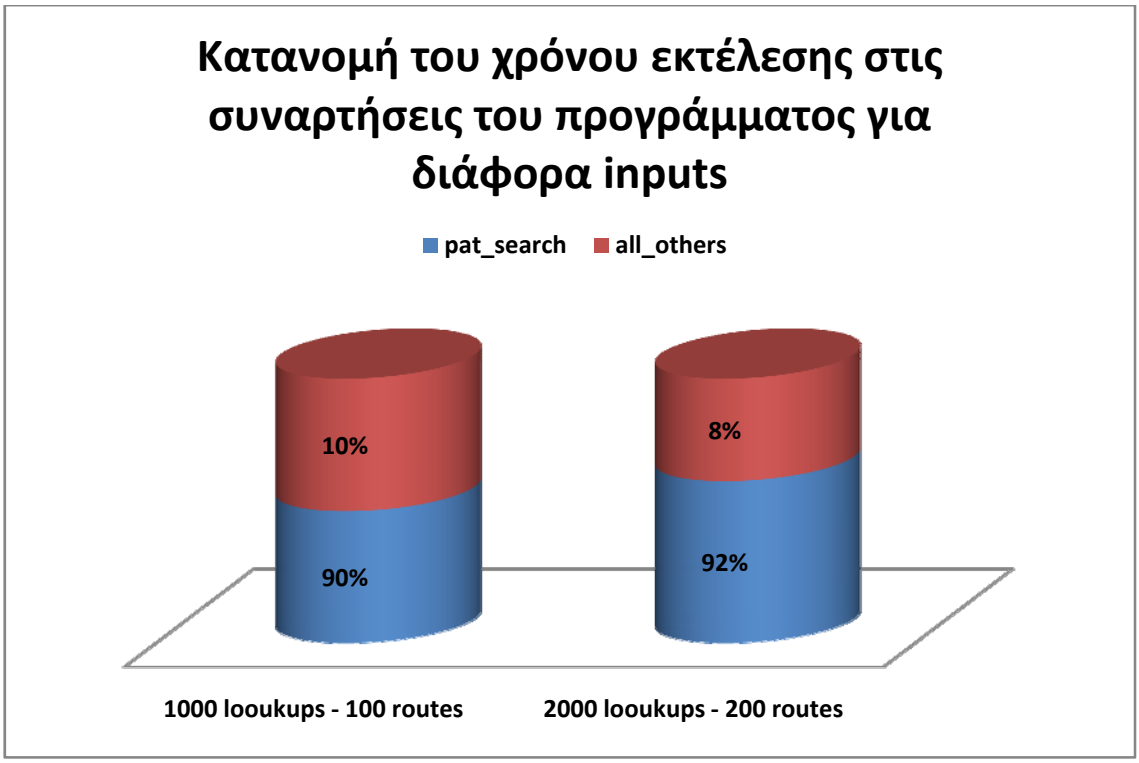
Παρατηρούμε ότι το συνολικό CPI μειώνεται για το benchmark με το μεγαλύτερο input.

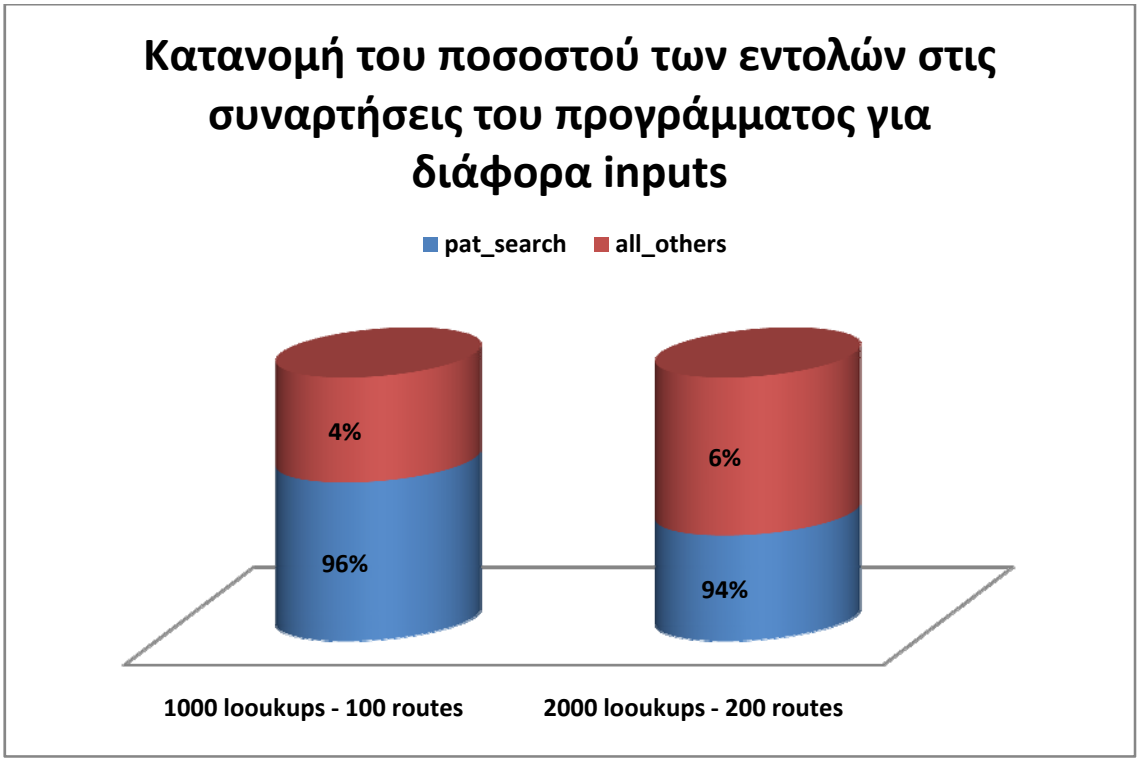




Όπως φαίνεται και από τα πιο πάνω διαγράμματα ο συνολικός αριθμός κύκλων και εντολών αυξάνεται για το πρόγραμμα με το μεγαλύτερο input, λόγω του ότι αυξάνεται ο αριθμός των δεδομένων που επεξεργάζονται.

Στα πιο κάτω διαγράμματα παρουσιάζεται η κατανομή του χρόνου εκτέλεσης και των εντολών στα hot spots του προγράμματος και για τα 2 προγράμματα που έχουν διαφορετικό μέγεθος input.


Κατανομή του χρόνου εκτέλεσης στις συναρτήσεις του προγράμματος για διάφορα inputs
pat_search
all_others
10%
90%
8%
92%
1000 looukups - 100 routes
2000 looukups - 200 routes


Κατανομή του ποσοστού των εντολών στις συναρτήσεις του προγράμματος για διάφορα inputs
pat_search
all_others
4%
96%
6%
94%
1000 looukups - 100 routes
2000 looukups - 200 routes

Όπως παρατηρείται στα πιο πάνω διαγράμματα το μεγαλύτερο ποσοστό χρόνου και εντολών και για τα 2 προγράμματα κατανάλωσε η συνάρτηση pat_search η οποία κάνει αναζητήσεις στο Patricia Tree.

Πιο κάτω παρουσιάζονται ο κώδικας και τα hot spots της συνάρτησης αυτής, όπως προέκυψαν από το εργαλείο VTune.

Συνάρτηση pat_search

Hot spots

1. Το πρώτο hot spot της συνάρτησης είναι σε ένα βρόγχο στον οποίο γίνεται η σύγκριση των bits των διευθύνσεων και με αυτό τον τρόπο διατρέχεται το δέντρο. Ο κώδικας του σημείου αυτού φαίνεται πιο κάτω:

```
while ( this_node->cmpbit > next_node->cmpbit )
```

Είναι hot spot γιατί βρίσκεται μέσα σε βρόγχο, γίνεται πρόσβαση στη μνήμη και επίσης γίνεται πράξη λογικής σύγκρισης.

2. Το δεύτερο hotspot της συνάρτησης βρίσκεται μέσα στο βρόγχο που αναφέρεται πιο πάνω. Σε αυτό το σημείο γίνεται ένας έλεγχος της διεύθυνσης του κόμβου στον οποίο βρίσκεται η εκτέλεση του προγράμματος τη δεδομένη στιγμή που γίνεται ο έλεγχος. Ο κώδικας του σημείου αυτού, φαίνεται πιο κάτω:

```
if ( proute->dst_addr & (0x1 << this_node->cmpbit) )
```

Είναι hot spot γιατί έχουμε πράξη λογικής σύγκρισης, πρόσβαση στη μνήμη και πράξη shifting η οποία χρειάζεται πολλούς κύκλους μηχανής για να εκτελεστεί.

3. Το τρίτο hot spot της συνάρτησης αυτής, βρίσκεται μέσα στο βρόγχο του πρώτου hot spot και μέσα στη συνθήκη ελέγχου του δεύτερου hot spot. Σε αυτό το σημείο διατρέχεται το δέντρο και γίνεται μετακίνηση στο αριστερό ή στο

δεξί παιδί του υφιστάμενου κόμβου, ανάλογα με τη συνθήκη που ισχύει. Ο κώδικας του σημείου αυτού, φαίνεται πιο κάτω:

```
   next_node = this_node->rlink;

   PF4( "Bit %2d: go right from %s for search %s to next %s\n",
      this_node->cmpbit,
      dot_dec( this_node->key ),
      dot_dec( proute->dst_addr ),
      dot_dec( next_node->key ) );
   }
else
   {
   next_node = this_node->llink;
```

Είναι hot spot λόγω του ότι γίνεται πρόσβαση στη μνήμη στο σημείο όπου η εκτέλεση μεταβαίνει στο αριστερό ή στο δεξί παιδί του κόμβου ανάλογα με τη συνθήκη που ισχύει.

Κεφάλαιο 6

Συμπεράσματα

6.1 Τεχνικά Συμπεράσματα

Μέσα από την τριβή που είχα μέσω αυτής της εργασίας με την ανάλυση προγραμμάτων, έχω διαπιστώσει ότι πολλοί παράγοντες είναι αυτοί που καθορίζουν τη συμπεριφορά ενός προγράμματος και το χρόνο εκτέλεσής του. Αρχικά έχω διαπιστώσει πως το μέγεθος και το είδος του input παίζουν σημαντικό ρόλο ως προς το χρόνο εκτέλεσης του προγράμματος. Συνήθως είναι αυτοί οι 2 παράγοντες που καθορίζουν το χρόνο εκτέλεσης του προγράμματος.

Εάν για παράδειγμα ένα πρόγραμμα διαβάζει δεδομένα από αρχείο ή δημιουργεί τα δεδομένα του προβλήματος δυναμικά στη μνήμη του υπολογιστή, υπάρχει μεγάλη διαφορά στους χρόνους εκτέλεσης καθώς και στο CPI αφού οι πρόσβαση σε αρχείο είναι πιο ακριβή.

Επίσης έχω διαπιστώσει ότι το hot spot ενός προγράμματος μπορεί να διαφέρει αναλόγως του input. Στις πλείστες περιπτώσεις εάν υπάρξει αλλαγή στο hot spot, αυτό συμβαίνει λόγω της αύξησης ή μείωσης του input που έχει ως αποτέλεσμα να εκτελούνται διαφορετικές εντολές. Χαρακτηριστικά μπορεί μια αλλαγή στον κώδικα του προγράμματος να προκαλέσει αλλαγή στα ποσοστά των εντολών που χρησιμοποιούνται από το κάθε είδος.

Ειδικότερα για τα EEMBC Networking V2.0 benchmarks έχω διαπιστώσει πως στο σύνολο τους ασχολούνται με την επίδοση του επεξεργαστή όσον αφορά την πρόσβαση στη μνήμη, αφού ως επί το πλείστον, οι διεργασίες των δικτύων που αναπαριστούν, έχουν να κάνουν με πρόσβαση στη μνήμη και με τη διαχείριση των δεδομένων που λαμβάνονται.

6.2 Μη τεχνικά συμπεράσματα

Εκτός των καινούριων γνώσεων που έχω αποκομίσει από αυτή τη διπλωματική εργασία, υπάρχουν και κάποιες μη τεχνικές εμπειρίες τις οποίες έχω αποκομίσει.

Ανάμεσα σε αυτές είναι η απόκτηση της εμπειρίας έρευνας, κάτι που θεωρώ πολύ σημαντικό αφού αυτή η ευκαιρία δε μου έχει ξαναδοθεί στα προηγούμενα χρόνια φοίτησής μου. Επίσης λόγω αυτής της διπλωματικής εργασίας έχω αναπτύξει και άλλες ικανότητες εκτός αυτών των οποίων έχω διδαχτεί από τα μαθήματά μου τα τελευταία τέσσερα χρόνια.

Εκτός αυτών νιώθω πως αυτή η διπλωματική εργασία με έχει βοηθήσει να βελτιωθώ σε επαγγελματικό επίπεδο αυξάνοντας σημαντικά τον επαγγελματισμό μου, κάτι που θα με βοηθήσει στη μετέπειτα πορεία μου. Επιπλέον με βοήθησε στο να αντιμετωπίζω με θάρρος τις όποιες δυσκολίες μου παρουσιαστούν και να πιστεύω στον εαυτό μου, αφού με έχει βοηθήσει να αυξήσω σημαντικά την αυτοπεποίθησή μου.

Βιβλιογραφία

[1] Chang – Burm Cho, Wangyuan Zhang, Tao Li, "Characterizing the Effect of Microarchitecture Design Parameters on Workload Dynamic Behavior"
Intelligent Design of Efficient Architecture Lab (IDEAL)
Department of ECE, University of Florida

[2] David S. Bolme, Michelle Strout and J, Ross Beveridge, "FacePerf: Benchmarks for Face Recognition Algorithms", Colorado State University
Fort Collins, Colorado 80523-1873

[3] L. Hennessy John, Patterson David A., "Computer Architecture: A Quantitative Approach"

[4] L. Hennessy John, Patterson David A., "Computer Organization and design: The Hardware/Software Interface"

[5] Eeckhout Lieven, Hoste Kenneth «Comparing Benchmarks Using Key Michroarchitecture – Independent Characteristics»
ELIS Department, Ghent University, Sint – Pietersnieuwstaat 41, B-9000 Gent, Belgium

[6] EEMBC, Understanding the EEMBC Benchmark Suite

[7] http://www.eembc.org/techlit/datasheets/networking_db.pdf

[8] http://linuxdevices.com/files/article077/embedded_processor_preference_history.jpg

Παράρτημα Α

Οδηγός εγκατάστασης των προγραμμάτων

Για να μπορούμε να εκτελέσουμε τα προγράμματα πρέπει πρώτα να τα εγκαταστήσουμε στον υπολογιστή μας.

1. Τοποθετούμε το φάκελο EEMBC στην τοποθεσία C:\.
2. Ανοίγοντας το φάκελο, βλέπουμε 8 φακέλους. Ο κάθε ένας από τους 7 πρώτους φακέλους αντιστοιχεί σε μια ομάδα από benchmarks ενώ στον τελευταίο φάκελο, τον Test Harness, βρίσκονται οι συναρτήσεις μέσω τον οποίων τρέχουμε τα προγράμματα.
3. Επιλέγουμε ένα από τους φακέλους, στην περίπτωση αυτή, το φάκελο Networking. Ανοίγοντας το φάκελο βλέπουμε 2 φακέλους, το Networking 1.1 και το Networking 2.0. Ο πρώτος αντιστοιχεί στα Networking Version 1, ενώ ο δεύτερος στα Networking Version 2 benchmarks.
4. Ανοίγουμε το φάκελο Networking 2.0. Στα περιεχόμενα του βρίσκεται ένας συμπιεσμένος φάκελος, ο φάκελος networking-2.0R1. Αποσυμπιέζουμε το φάκελο αυτό με τη χρήση του εργαλείου WinZip και εμφανίζεται ο φάκελος με την ίδια ονομασία αποσυμπιεσμένος.
5. Στη συνέχεια μεταβαίνουμε στο φάκελο Test Harness. Ο φάκελος αυτός περιέχει συμπιεσμένους φακέλους. Από αυτούς τους φακέλους επιλέγουμε το φάκελο eembc-2.0R2 και τον αποσυμπιέζουμε.
6. Επιλέγουμε όλα τα περιεχόμενα του συγκεκριμένου φακέλου, και τα αντιγράφουμε με τη χρήση της εντολής copy. Έπειτα επιστρέφουμε στο φάκελο networking-2.0R1 και επικολλούμε τα περιεχόμενα που αντιγράψαμε εκεί.

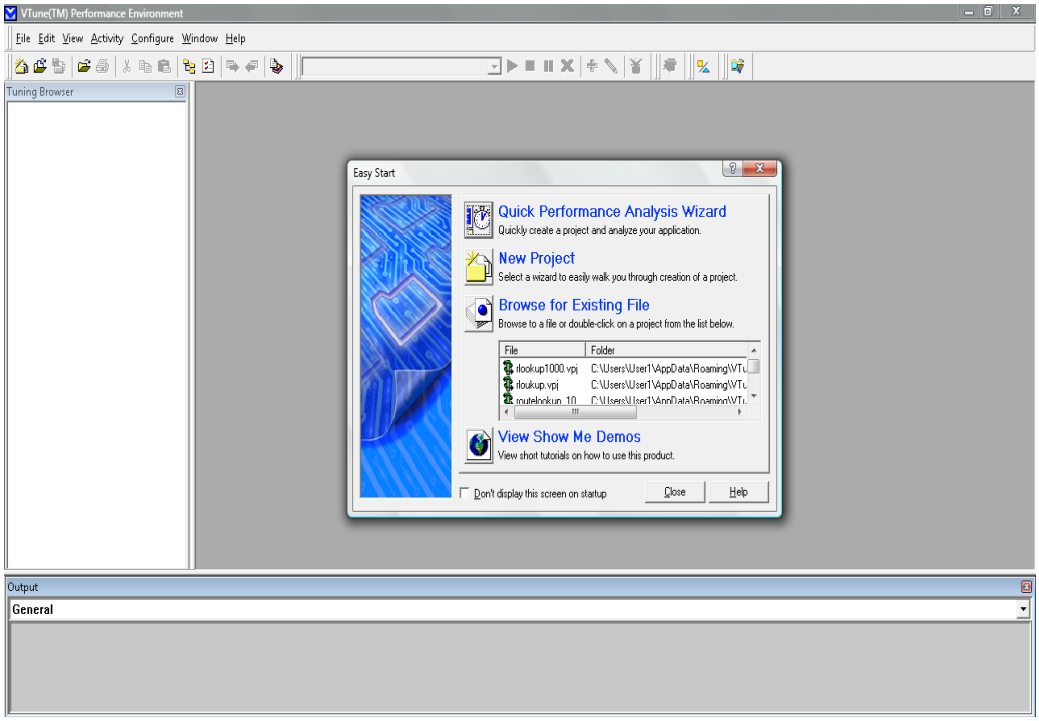
Οδηγός μεταγλώττισης των προγραμμάτων

1. Ανοίγουμε το πρόγραμμα Cygwin
2. Πληκτρολογούμε την εντολή cd C:

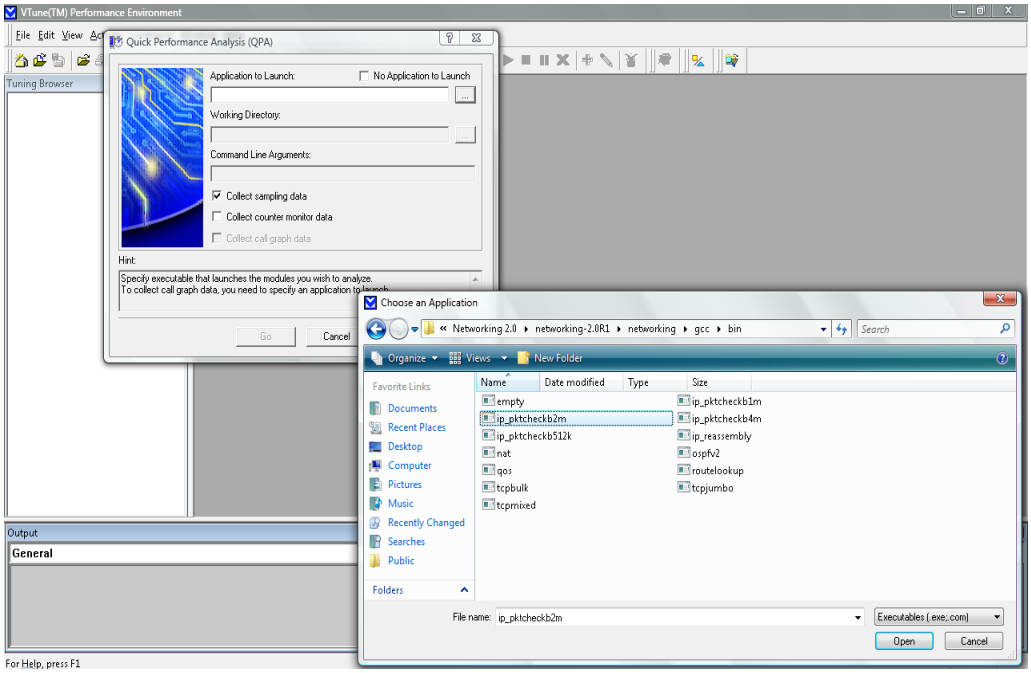
3. Πληκτρολογούμε την εντολή cd EEMBC/Networking/Networking\ 2.0/networking-2.0R1
4. Αφού κάνουμε όλες τις απαραίτητες αλλαγές μέσα στα makefiles που αναφέρονται στο κεφάλαιο 4, πληκτρολογούμε την εντολή make all η οποία θα παράξει τα εκτελέσιμα προγράμματα.

Οδηγός χρήσης του εργαλείου VTune Performance Analyzer

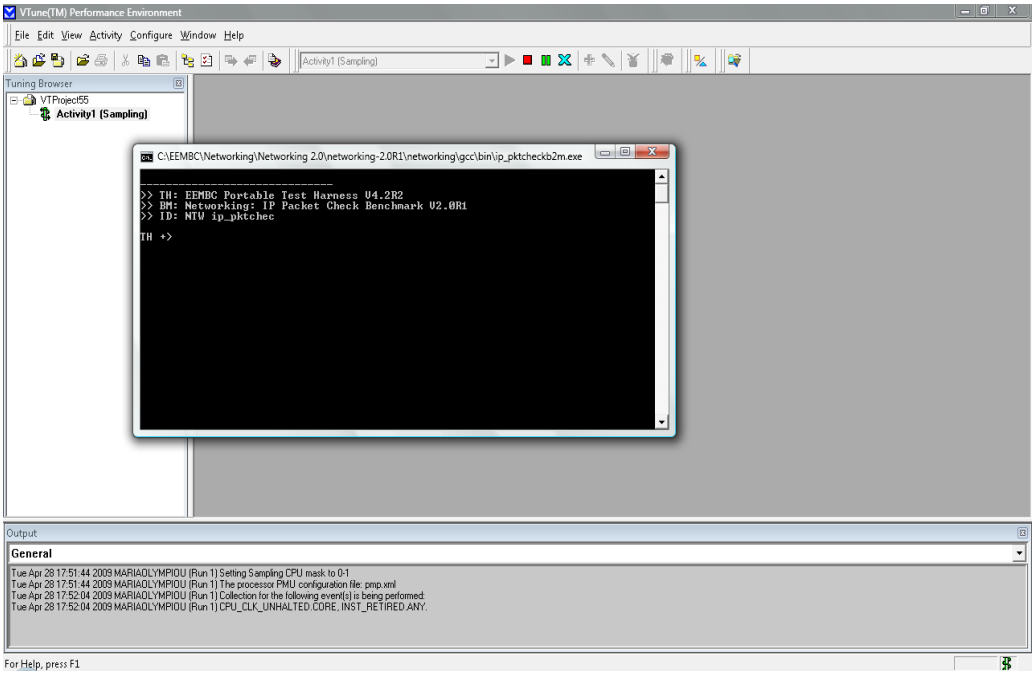
1. Ανοίγουμε το πρόγραμμα Intel VTune Performance Analyzer
2. Έπειτα πατάμε το κουμπί που αντιστοιχεί στο Quick Performance Analysis Wizard



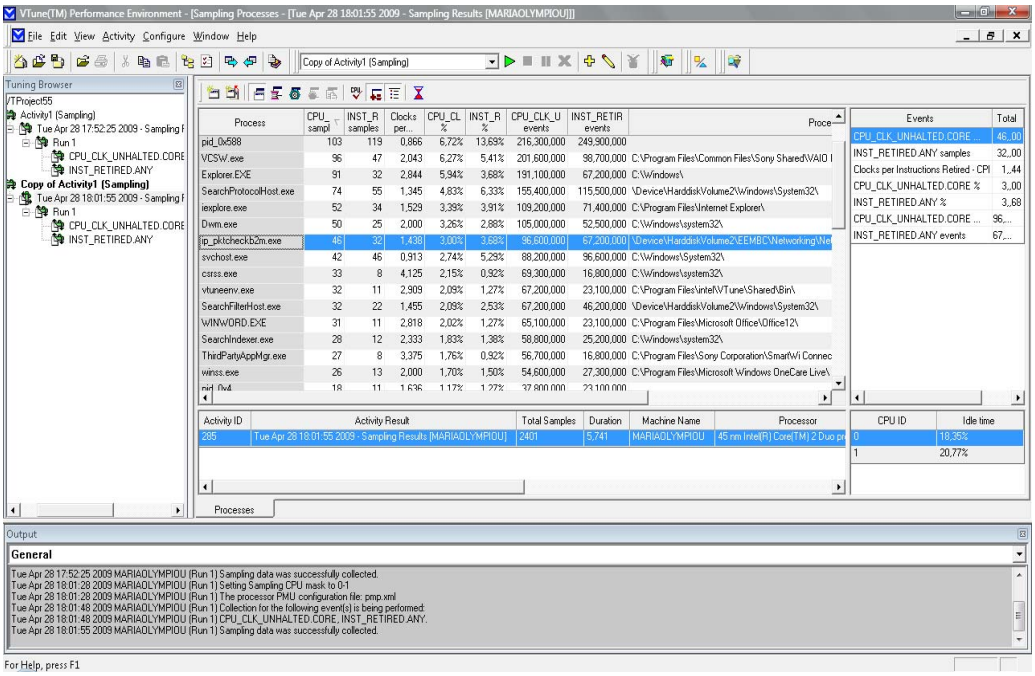
3. Στο παράθυρο που εμφανίζεται επιλέγουμε στο πεδίο Application to Launch την εφαρμογή την οποία θέλουμε να αναλύσουμε. Οι εφαρμογές που μπορούν να αναλυθούν στο VTune είναι μόνο εκείνες που έχουν extension .exe.



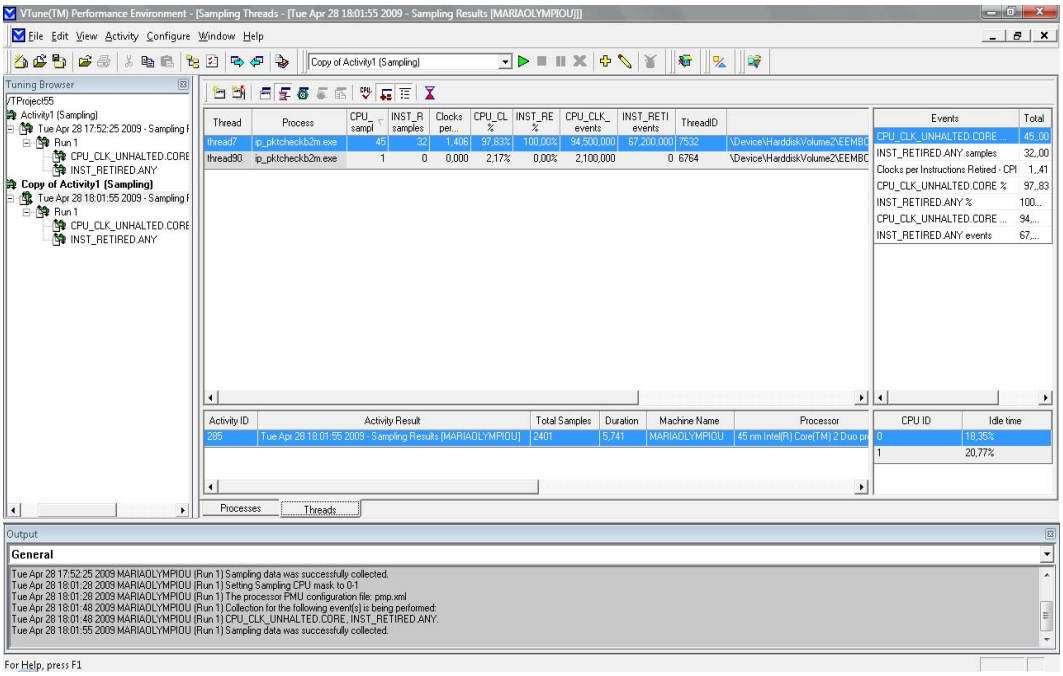
4. Αφού επιλέξουμε την εφαρμογή, πατάμε το κουμπί GO. Αμέσως μετά αρχίζει η διαδικασία δειγματοληψίας. Κατά την διαδικασία αυτή εμφανίζεται ένα Cygwin shell στο οποίο γράφουμε τις εντολές για εκτέλεση του προγράμματος. Για τα εκτελέσιμα αρχεία των EEMBC benchmarks καταχωρούμε τις εντολές που αναφέρονται στο κεφάλαιο 4.



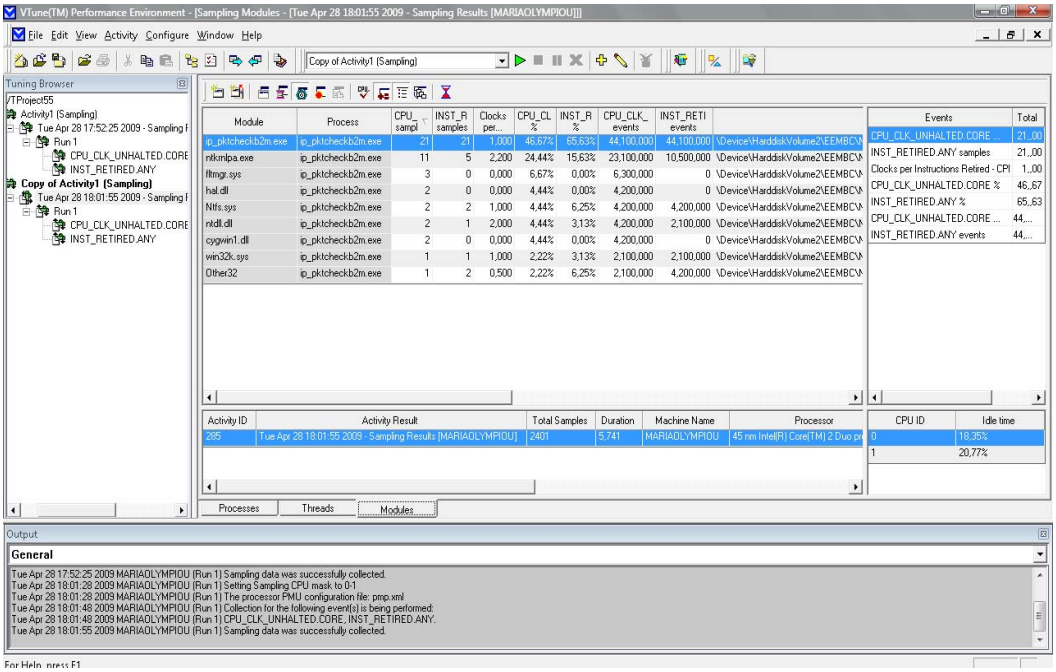
5. Μόλις τελειώσει η δειγματοληψία εμφανίζεται το πιο κάτω παράθυρο. Επιλέγουμε την εφαρμογή που έχουμε αναλύσει, πατώντας 2 φορές πάνω της.



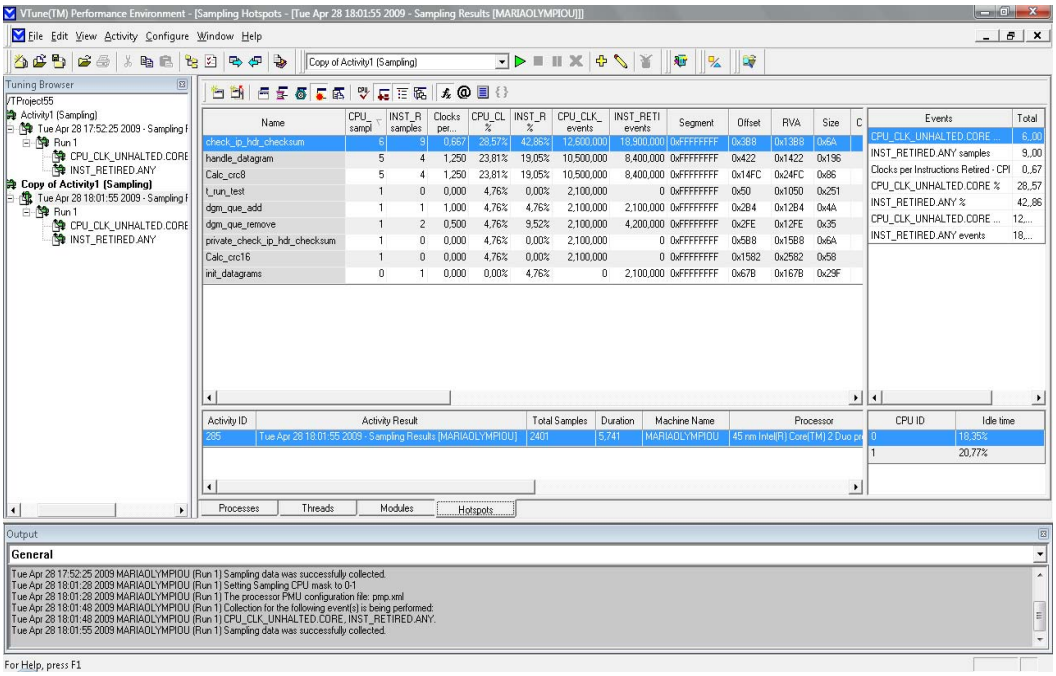
6. Στο επόμενο παράθυρο εμφανίζονται τα threads μέσα στα οποία έτρεξε η εφαρμογή. Επιλέγουμε το thread από το οποίο έχουν επιλεχθεί τα περισσότερα δείγματα, όπως φαίνεται στην εικόνα πιο κάτω.



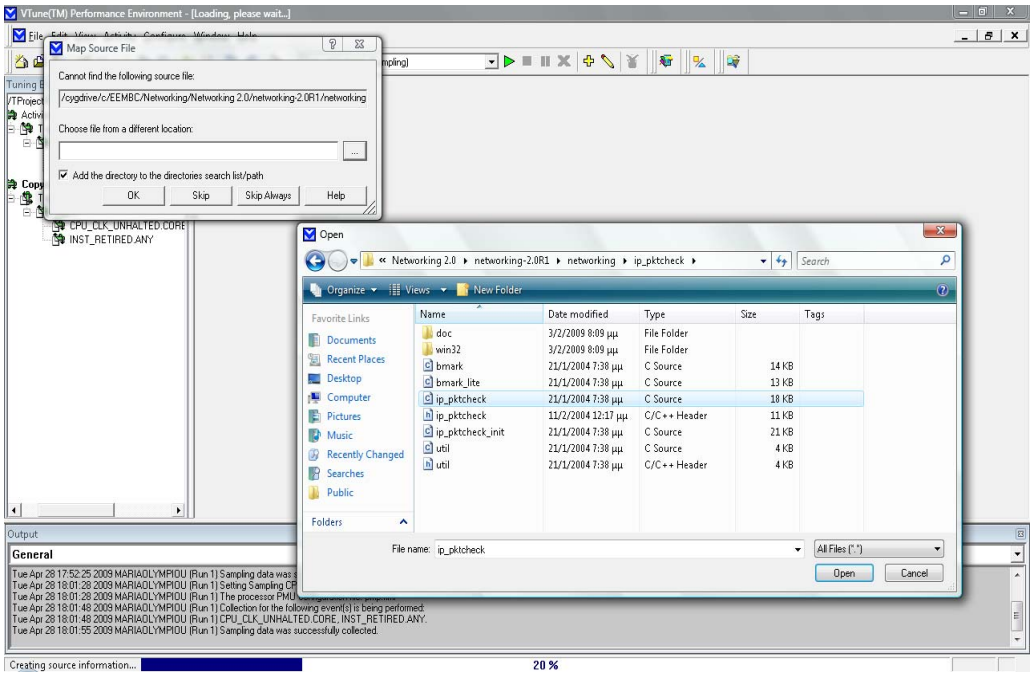
7. Στη συνέχεια εμφανίζονται τα modules που σχετίζονται με την εκτέλεση της εφαρμογής. Επιλέγουμε το module που έχει το ίδιο όνομα με την εφαρμογή που τρέξαμε όπως φαίνεται πιο κάτω.



8. Στο επόμενο παράθυρο εμφανίζονται τα ονόματα των συναρτήσεων που έτρεξαν και φαίνεται η ποσοστιαία κατανομή του χρόνου εκτέλεσης σ' αυτές. Επιλέγουμε μια συνάρτηση για να δούμε τον κώδικά της.

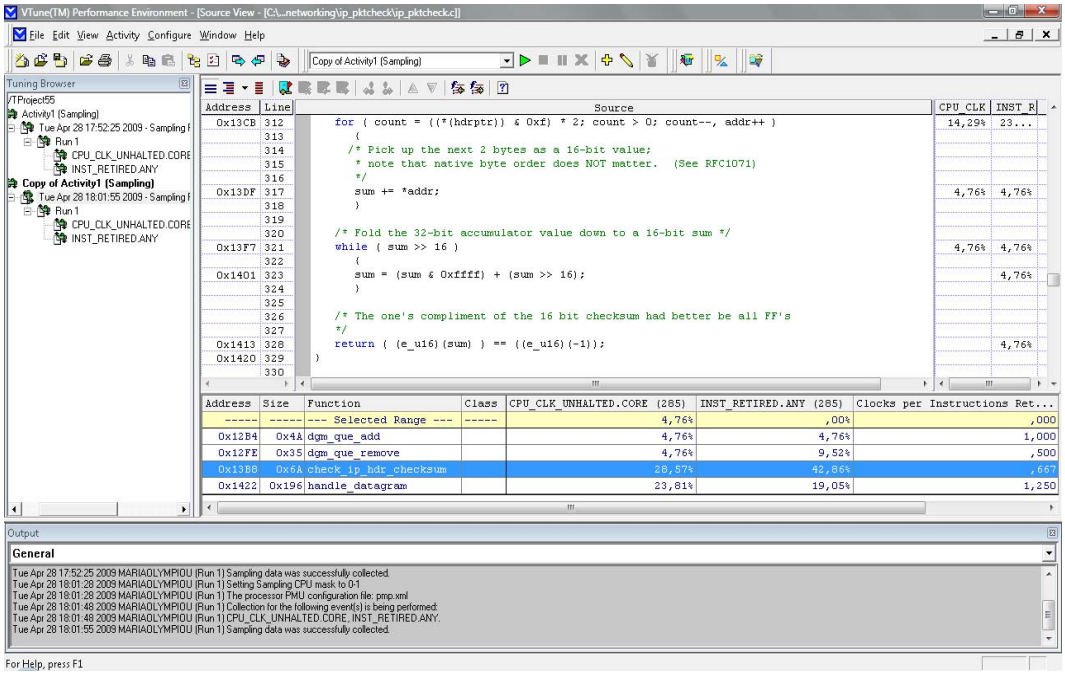


9. Επειδή ο κώδικας και τα εκτελέσιμα αρχεία βρίσκονται σε διαφορετικούς φακέλους, εμφανίζεται ένα παράθυρο όπως αυτό που φαίνεται πιο κάτω:



Σε αυτό μας ζητείται να καθορίσουμε την τοποθεσία στην οποία βρίσκεται ο κώδικας του εκτελέσιμου προγράμματος, επειδή δε βρίσκεται στον ίδιο φάκελο με το εκτελέσιμο αρχείο του προγράμματος.

Μόλις καθορίσουμε από ποιο αρχείο προέρχεται ο κώδικας του προγράμματος, εμφανίζεται ο κώδικας σε ένα καινούριο παράθυρο όπως φαίνεται στην εικόνα πιο κάτω.



10. Για να δούμε το ποσοστό του χρόνου που κατανάλωσε το κάθε σημείο, πατάμε δεξί κλικ και βάζουμε την επιλογή View Event As % of activity. Με αυτό τον τρόπο βλέπουμε το ποσοστό του χρόνου που κατανάλωσε το κάθε σημείο του κώδικα.
11. Για να τρέξουμε ξανά το πρόγραμμα για τον έλεγχο άλλων μετρικών, πατάμε δεξί κλικ, modify activity και επιλέγουμε ποια μετρικά θέλουμε να ελέγξουμε από την επιλογή Configure. Στη συνέχεια πατάμε το κουμπί GO για να ξεκινήσει η δειγματοληψία.