ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΜΕΘΟΔΟΛΟΓΙΑ ΚΑΙ ΕΡΓΑΛΕΙΟ ΛΟΓΙΣΜΙΚΟΥ ΓΙΑ ΤΗΝ ΕΚΤΙΜΗΣΗ ΤΗΣ ΕΠΙΔΟΣΗΣ ΕΝΣΩΜΑΤΩΜΕΝΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΤΟΥ

ΜΙΧΑΗΛ Ε. ΒΗΧΟΥ

Επιβλέπων: Δημήτριος Σούντρης
Επίκουρος Καθηγητής

Αθήνα Νοέμβριος 2011

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΜΕΘΟΔΟΛΟΓΙΑ ΚΑΙ ΕΡΓΑΛΕΙΟ ΛΟΓΙΣΜΙΚΟΥ ΓΙΑ ΤΗΝ ΕΚΤΙΜΗΣΗ ΤΗΣ ΕΠΙΔΟΣΗΣ ΕΝΣΩΜΑΤΩΜΕΝΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΤΟΥ

ΜΙΧΑΗΛ Ε. ΒΗΧΟΥ

Επιβλέπων: Δημήτριος Σούντρης
Επίκουρος Καθηγητής

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή:

Γεώργιος Οικονομάκος Επίκουρος Καθηγητής	Κιαμάλ Πεκμεστζή Καθηγητής	Δημήτριος Σούντρης Επίκουρος Καθηγητής
……	……	……

Αθήνα Νοέμβριος 2011

……
Μιχαήλ Ε. Βήχος
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών

ΠΕΡΙΛΗΨΗ

Τις τελευταίες δεκαετίες έχει παρατηρηθεί αλματώδης αύξηση της παραγωγής ενσωματωμένων συστημάτων. Τα ενσωματωμένα συστήματα παίζουν πλέον κυρίαρχο ρόλο σε πάρα πολλούς τομείς της τεχνολογίας και της βιομηχανίας. Το γεγονός αυτό κάνει επιτακτική την ανάγκη ύπαρξης μεθόδων που διευκολύνουν και επιταχύνουν τη σχεδίαση των ενσωματωμένων συστημάτων αλλά και τον έλεγχο της επίδοσης τους από τα πρώτα στάδια σχεδίασης.

Σε αυτή τη διπλωματική εργασία αναπτύσσεται μία μεθοδολογία που στοχεύει στη πολύ γρήγορη αλλά και με καλή ακρίβεια εκτίμηση της επίδοσης ενσωματωμένων συστημάτων. Η νέα μεθοδολογία στηρίζεται στην μελέτη των Ν. Κρούπη και Δ. Σούντρη [1], οι οποίοι απέδειξαν πειραματικά ότι ο πηγαίος κώδικας μίας εφαρμογής μπορεί να μας δώσει χρήσιμη πληροφορία για την επίδοση του κώδικα μηχανής της εφαρμογής.

Στη νέα μεθοδολογία η εφαρμογή εκτελείται άμεσα σε οποιονδήποτε διαθέσιμο υπολογιστή και λαμβάνονται στατιστικά για την εκτέλεση σε επίπεδο πηγαίου κώδικα. Στη συνέχεια κάθε εντολή του κώδικα μηχανής αντιστοιχίζεται με τη προέλευσή της στον πηγαίο κώδικα με χρήση των πληροφοριών αποσφαλμάτωσης που παράγονται από τον μεταγλωττιστή σε μορφή DWARF [2]. Έτσι γίνεται και εκτίμηση του πλήθους των εκτελέσεών της. Τέλος, γίνεται στατική ανάλυση του χαρακτηρισμένου κώδικα μηχανής και εξάγονται συμπεράσματα για την επίδοση της κρυφής μνήμης εντολών.

Η μεθοδολογία υλοποιήθηκε με την ανάπτυξη ενός συνόλου εργαλείων λογισμικού συμβατών με όλες τις αρχιτεκτονικές που υποστηρίζει ο GNU Compiler [3]. Η ταχύτητα και η ακρίβεια των νέων εργαλείων εξετάστηκε με μία σειρά διεξοδικών πειραμάτων τα οποία έδειξαν ότι η διαδικασία εκτίμησης είναι έως και 20000 φορές ταχύτερη από τις σύγχρονες μεθόδους προσομοίωσης ενώ η μέση τιμή σφάλματος διατηρείται σε πολύ χαμηλά επίπεδα (2,1%).

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ:

Γράφος ροής ελέγχου, Εκτίμηση επίδοσης, Ενσωματωμένο σύστημα, Εργαλείο λογισμικού, Ιεραρχία μνήμης, Κρυφή μνήμη, Κώδικας μηχανής, Μεταγλωττιστής, Πηγαίος κώδικας

ABSTRACT

In the last decades, the production of embedded systems has been raised greatly. Nowadays, embedded systems play a very significant role in the majority of technological and industrial fields. For this reason, it is absolutely necessary new techniques to be created in order to facilitate both the design procedure and the performance evaluation of embedded systems from the very early design stages.

The scope of this diploma thesis is the development of a methodology that manages to make performance estimations for embedded systems very fast but also with good accuracy. The new methodology is based on the work of N. Kroupis and D. Soudris [1] which has shown that the source code of an application can give us useful information about the performance of the machine code.

In the new methodology, the application is executed natively in a host machine and statistics are collected for the execution at the source code level. Then, each machine instruction is mapped to its origin in the source code using the debugging information generated by the compiler in DWARF format [2]. In this way the number of executions of the instruction can also be calculated. Finally, the characterized machine code is statically analysed in order to make estimation for the instruction cache memory performance.

A set of new software tools implementing the methodology has been developed. The new tools are fully compatible with all the architectures supported by the GNU Compiler [3]. The time consumption and the accuracy of the new tools have been evaluated by conducting a thorough set of experiments. The experimental results showed that the tools can achieve high accuracy (average error 2.1%) and a speedup factor up to 20000 with reference to modern simulation methods.

KEYWORDS:

Control flow graph, Performance estimation, Embedded system, Software tool, Memory hierarchy, Cache memory, Machine code, Compiler, Source code

ΕΥΧΑΡΙΣΤΙΕΣ

Η εργασία αυτή δεν θα είχε ολοκληρωθεί χωρίς τη σημαντική βοήθεια κάποιων ανθρώπων τους οποίους θα ήθελα να ευχαριστήσω στη παρούσα παράγραφο.

Αρχικά θα ήθελα να ευχαριστήσω θερμά τον Επίκουρο Καθηγητή κ. Δημήτριο Σούντρη ο οποίος, ως επιβλέπων καθηγητής μου, με καθοδήγησε και με στήριξε σε όλες τις δύσκολες στιγμές της προσπάθειας μου. Καθοριστική ήταν επίσης η βοήθεια του Διδάκτορος κ. Νικόλαου Κρούπη, ο οποίος πρόσφερε απλόχερα πολύτιμη τεχνική βοήθεια και ψυχολογική στήριξη. Η παρούσα διπλωματική εργασία στηρίζεται σε μεγάλο ποσοστό σε δική του μελέτη. Σημαντική ήταν ακόμα η βοήθεια του Διδάκτορος και ερευνητή κ. Αλέξανδρου Μπάρτσα ο οποίος μου αφιέρωσε πολύτιμο χρόνο, τεχνικές συμβουλές και με βοήθησε στη διόρθωση της παρούσας διπλωματικής εργασίας. Ένα μεγάλο «ευχαριστώ» χρωστάω επίσης στον υποψήφιο Διδάκτορα κ. Διονύση Διαμαντόπουλο ο οποίος έκανε καθοριστικές παρατηρήσεις και διορθώσεις και με βοήθησε σε πολλά τεχνικά ζητήματα.

Κυρίως όμως, θέλω να ευχαριστήσω την οικογένειά μου για τη υπομονή που έκανε και τη στήριξη που μου προσέφερε όλα τα χρόνια που αφιέρωσα στις σπουδές μου στο ΕΜΠ.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΚΑΤΑΛΟΓΟΣ ΚΩΔΙΚΑ

1 ΕΙΣΑΓΩΓΗ

Τα τελευταία χρόνια τα ενσωματωμένα συστήματα κάνουν όλο και πιο έντονη τη παρουσία τους στη ζωή μας. Οι τομείς των υπολογιστών, της τηλεφωνίας, της αυτοκινητοβιομηχανίας και τις ψυχαγωγίας δεν είναι παρά λίγοι μόνο από τους κλάδους που παράγουν τεράστιες ποσότητες ενσωματωμένων συστημάτων ετησίως. Αυτά τα νέα και καινοτόμα προϊόντα δημιουργούνται με πολύ γρήγορους ρυθμούς και φτάνουν στα χέρια των καταναλωτών σε πολύ μικρό χρονικό διάστημα. Ενδεικτικά το 2008 κατασκευάστηκαν περίπου 10 δισεκατομμύρια επεξεργαστές εκ των οποίων το 98% για χρήση σε ενσωματωμένα συστήματα [4].

1.1 Περιγραφή του προβλήματος

Για να είναι ανταγωνιστικό ένα νέο προϊόν θα πρέπει μέσα σε πολύ μικρό χρονικό διάστημα να περάσει από το στάδιο σχεδιασμού στο στάδιο παραγωγής και στη συνέχεια στην αγορά και ταυτόχρονα να είναι ποιοτικό, να προσφέρει υψηλές επιδόσεις και να είναι οικονομικό από άποψη ενεργειακής κατανάλωσης. Γίνεται έτσι αντιληπτό ότι είναι απαραίτητο να υπάρχουν μέθοδοι με τις οποίες ο σχεδιαστής θα μπορεί να ελέγχει τη ορθότητα και να μετράει την επίδοση του συστήματος που σχεδιάζει σε κάθε βήμα της σχεδίασης και με μεγάλη ταχύτητα. Διαφορετικά είναι πιθανό να βρεθεί στη δυσάρεστη θέση να χρειαστεί να επανασχεδιάσει μεγάλο κομμάτι του συστήματός προκειμένου αυτό να πληροί τις προδιαγραφές επιδόσεων και κατανάλωσης ενέργειας. Κάτι τέτοιο μεταφράζεται σε απώλεια χρόνου και χρημάτων. Είναι μάλιστα πιθανό το νέο προϊόν να χάσει την εμπορική του αξία λόγω της καθυστερημένης εισόδου του στην αγορά.

Ο έλεγχος της επίδοσης ενός ενσωματωμένου συστήματος από τα πρώτα στάδια της σχεδίασης του δεν είναι πάντα μία εύκολη διαδικασία. Ειδικά στις περιπτώσεις όπου το υλικό και το λογισμικό του συστήματος αναπτύσσονται παράλληλα (συνσχεδιασμός υλικού/λογισμικού), τα προβλήματα είναι ακόμα περισσότερα καθώς θα πρέπει να ερευνηθεί κατά πόσο το λογισμικό και το υλικό έχουν συνδυαστεί αρμονικά και μάλιστα αυτό θα πρέπει να γίνει χωρίς να έχει ολοκληρωθεί ούτε το λογισμικό ούτε το υλικό του συστήματος. Συχνά είναι απαραίτητο να δοκιμαστούν πολλές διαφορετικές αρχιτεκτονικές (αρχιτεκτονικές επεξεργαστών και ιεραρχίες μνήμης) για το υλικό ώστε να εντοπιστεί αυτή που συνδυάζεται βέλτιστα με το

λογισμικό με τρόπο που πληροί τις προδιαγραφές επιδόσεων αλλά και τους οικονομικούς περιορισμούς (architecture exploration).

Μερικοί από τους πιο σημαντικούς παράγοντες που απαρτίζουν την επίδοση ενός ενσωματωμένου συστήματος είναι οι χρόνοι εκτέλεσης των διαφόρων εφαρμογών και η επίδοση της κρυφής μνήμης (ρυθμός ευστοχίας – hit ratio). Στόχος είναι να έχουμε μικρούς χρόνους εκτέλεσης και αυξημένους ρυθμούς ευστοχίας με εύλογη χρησιμοποίηση υπολογιστικών πόρων. Μάλιστα οι αυξημένοι ρυθμοί ευστοχίας της κρυφής μνήμης συντελούν και στους μικρούς χρόνους εκτέλεσης καθώς και στη μειωμένη κατανάλωση ισχύος.

Η πιο διαδεδομένη μέθοδος για την μέτρηση των επιδόσεων μίας εφαρμογής, χωρίς την ανάγκη ύπαρξης του φυσικού υλικού, είναι η προσομοίωση. Η εκτέλεση των εφαρμογών του ενσωματωμένου συστήματος στο υλικό του ενσωματωμένου συστήματος προσομοιώνεται με τη βοήθεια υπολογιστή. Έτσι μπορούμε να ελέγξουμε τη σωστή λειτουργία του συστήματος μας ενώ ταυτόχρονα μπορούμε να κρατήσουμε στατιστικά για την επίδοση της εφαρμογής. Τα στατιστικά της επίδοσης μπορεί να είναι βασικές πληροφορίες όπως το πλήθος των εντολών που εκτελέστηκαν ή πιο εξειδικευμένες πληροφορίες όπως το πλήθος των αστοχιών των διαφόρων επιπέδων της μνήμης cache και η κατανάλωση ενέργειας.

Το αντίτιμο για τη συλλογή εξειδικευμένων στατιστικών είναι πάντα ο χρόνος προσομοίωσης. Ειδικά στη περίπτωση της εξερεύνησης πολλών αρχιτεκτονικών για τον εντοπισμό της βέλτιστης, ο χρόνος που απαιτούν οι πολλαπλές προσομοιώσεις είναι συχνά απαγορευτικός. Δημιουργείτε έτσι η ανάγκη για αναζήτηση νέων μεθόδων και για βελτίωση των υπαρχουσών τεχνικών ώστε η εκτίμηση της επίδοσης ενός ενσωματωμένου συστήματος και η εξερεύνηση πολλαπλών αρχιτεκτονικών να μπορεί να γίνει πιο γρήγορα και με αυξημένη ακρίβεια.

1.2 Σκοπός της παρούσας διπλωματικής εργασίας

Σκοπός της παρούσας εργασίας είναι η υλοποίηση μίας νέας μεθόδου για την εκτίμηση της επίδοσης ενός ενσωματωμένου συστήματος και για τη γρήγορη εξερεύνηση πολλαπλών αρχιτεκτονικών. Η μέθοδος βασίζεται στη μελέτη των N. Κρούπη και Δ. Σούντρη [1] η οποία μας επιτρέπει να εξάγουμε γρήγορα και με πολύ καλή ακρίβεια εκτιμήσεις για το πλήθος των εκτελούμενων εντολών και την επίδοση της κρυφής μνήμης εντολών ταυτόχρονα για πολλές διαφορετικές αρχιτεκτονικές. Η νέα μέθοδος διατηρεί το σημαντικότερο προτέρημα της [1] καθώς δεν απαιτεί προσομοίωση της εκτέλεσης των εφαρμογών και γι’ αυτό επιτυγχάνει έως και 21000 φορές μικρότερους χρόνους περάτωσης από τις μεθόδους προσομοίωσης ενώ το μέσο σφάλμα των εκτιμήσεων είναι μικρότερο από 4% (βλ. κεφάλαιο 5 για αναλυτικά στατιστικά).

1.3 Διάρθρωση της διπλωματικής εργασίας

Η διπλωματική εργασία αναπτύσσεται στη συνέχεια σε 5 κύρια κεφάλαια:

Στο κεφάλαιο 2 γίνεται μία σύντομη επισκόπηση των τεχνικών που έχουν ήδη αναπτυχθεί στο πεδίο της διπλωματικής. Γίνεται μία σύντομη αναφορά στις παραδοσιακές τεχνικές προσομοίωσης καθώς και στις πιο σύγχρονες εναλλακτικές ερευνητικές προσεγγίσεις. Στο τέλος του κεφαλαίου γίνεται μία αναφορά στο ήδη υπάρχον εργαλείο FICA και τη μελέτη των Ν. Κρούπη και Δ. Σούντρη, που αποτελεί τη βάση πάνω στην οποία αναπτύχθηκε η παρούσα διπλωματική εργασία.

Στο κεφάλαιο 3, γίνεται, σε θεωρητικό επίπεδο, αναλυτική περιγραφή των μεθοδολογιών που υλοποιήθηκαν. Στις ενότητες 3.2 έως 3.5 περιγράφεται μία μέθοδος για τη γρήγορη εκτίμηση της γενικής επίδοσης μίας εφαρμογής (πλήθος εκτελούμενων εντολών) και στις ενότητες 3.6 έως 3.8 παρουσιάζεται μία μέθοδος για την εκτίμηση της επίδοσης της κρυφής μνήμης. Στο τέλος του κεφαλαίου περιγράφονται κάποιες περιπτώσεις που απαιτούν ιδιαίτερη προσοχή κατά την εφαρμογή των μεθόδων.

Στο κεφάλαιο 4 περιγράφεται η δομή των εργαλείων που αναπτύχθηκαν και υλοποιούν τις μεθοδολογίες που περιγράφονται στο κεφάλαιο 3. Για κάθε ένα εργαλείο γίνεται περιγραφή των δομικών του μονάδων και των λειτουργιών που αυτές εκτελούν.

Στο κεφάλαιο 5 περιγράφεται σύντομα η διαδικασία που ακολουθήθηκε για την εκτίμηση της ακρίβειας και της ταχύτητας των νέων εργαλείων και στη συνέχεια παρουσιάζονται αναλυτικά τα πειραματικά αποτελέσματα.

Στο κεφάλαιο 6 παρουσιάζονται τα συμπεράσματα που προκύπτουν από τη πειραματική διαδικασία και προτείνονται θέματα για περεταίρω έρευνα καθώς και πιθανές βελτιώσεις που μπορούν να γίνουν στις μεθόδους που αναπτύχθηκαν στα πλαίσια της διπλωματικής εργασίας.

Τέλος στα παραρτήματα δίνονται αναλυτικά εγχειρίδια χρήσης για τα νέα εργαλεία που αναπτύχθηκαν.

2 Υπαρχον Ερευνητικο Έργο

Έχουν αναπτυχθεί διάφορες τεχνικές και εργαλεία για την εκτίμηση της επίδοσης ενσωματωμένων συστημάτων και συγκεκριμένα για την εκτίμηση της επίδοσης της κρυφής μνήμης και για την αναζήτηση των βέλτιστων παραμέτρων της. Οι τεχνικές αυτές βασίζονται είτε σε προσομοίωση της εκτέλεσης των εφαρμογών είτε στην εκτίμηση με χρήση μοντέλων που περιγράφουν τη συμπεριφορά του συστήματος. Η πρώτη κατηγορία (προσομοίωση) προσφέρει υψηλή ακρίβεια στα αποτελέσματα αλλά απαιτεί πολύ χρόνο που σε περιπτώσεις πολύπλοκων εφαρμογών καθίσταται μη εφαρμόσιμη λόγο του πολύ μεγάλου χρόνου περάτωσης. Οι μέθοδοι τις δεύτερης κατηγορίας (εκτίμηση) είναι σαφώς ταχύτερες έχουν όμως μικρότερη ακρίβεια. Στις επόμενες παραγράφους παρουσιάζονται οι κυριότερες μέθοδοι και τα σημαντικότερα υπάρχοντα εργαλεία των δύο κατηγοριών.

2.1 Εργαλεία προσομοίωσης της κρυφής μνήμης εντολών (instruction cache)

Η προσομοίωση της μνήμης cache μπορεί να γίνει είτε προσαρμόζοντας σε προσομοιωτές επεξεργαστών την αναλυτική περιγραφή της (execution-driven simulation), ή χρησιμοποιώντας ξεχωριστούς προσομοιωτές οι οποίοι δέχονται σαν είσοδο ένα αρχείο (trace) που περιέχει όλες τις διευθύνσεις που προσπελάθηκαν από την εφαρμογή (trace-driven simulation). Στη πρώτη περίπτωση κατά τη προσομοίωση της εκτέλεσης παρακολουθείται συνεχώς η κατάσταση της μνήμης cache και υπολογίζεται ο συνολικός αριθμός των αστοχιών της μνήμης (cache misses). Στη δεύτερη περίπτωση το αρχείο διευθύνσεων θα πρέπει να έχει δημιουργηθεί κατά τη προσομοίωση της εφαρμογής ή κατά την άμεση εκτέλεση της (με χρήση κάποιου τρόπου «ανίχνευσης» των διευθύνσεων που προσπελαύνονται). Στη συνέχεια το αρχείο trace δίνεται σαν είσοδος σε έναν αυτοτελή προσομοιωτή της ιεραρχίας μνήμης, ο οποίος υπολογίζει το ποσοστό των αστοχιών.

Ένας σημαντικός αντιπρόσωπος της πρώτης κατηγορίας προσομοιωτών (execution-driven simulators [5]) είναι ο Simplesclalar [6]. Το εν λόγω εργαλείο

δημιουργήθηκε το 1992 από το πανεπιστήμιο του Wisconsin. Πρόκειται για ένα εργαλείο προσομοίωσης και μοντελοποίησης αρχιτεκτονικών. Μπορεί να χρησιμοποιηθεί για τη μοντελοποίηση απλών μέχρι πολύπλοκών αρχιτεκτονικών με πολυεπίπεδα συστήματα μνήμης. Παρέχει ευελιξία στον χρήστη και μπορεί να τροποποιηθεί-επεκταθεί σε περίπτωση που είναι ανάγκη να γίνει κάποια επιπλέον εργασία μοντελοποίησης αρχιτεκτονικής. Αυτά τα χαρακτηριστικά το έχουν κάνει πολύ δημοφιλές στην ερευνητική κοινότητα και έχει καθιερωθεί ως μέτρο αναφοράς.

Από τη δεύτερη κατηγορία προσομοιωτών (trace-driven simulation) ένας από τους πιο ευρέως διαδεδομένους προσομοιωτές είναι ο Dinero [5]. Ο συγκεκριμένος προσομοιωτής δεν μπορεί να προσομοιώσει το σύνολο εντολών κάποιου επεξεργαστή (ISA) αλλά μόνο τη συμπεριφορά της ιεραρχίας μνήμης. Έτσι δεν μπορεί να χρησιμοποιηθεί μόνος του αλλά μόνο σε συνδυασμό με έναν προσομοιωτή του συνόλου εντολών ο οποίος θα αναλάβει και τη δημιουργία του αρχείου trace.

Ένα πιο σύγχρονο εργαλείο για τη προσομοίωση αρχιτεκτονικών διαφόρων επεξεργαστών αλλά και της ιεραρχίας μνήμης είναι το OVP [7]. Το OVP είναι μία πλατφόρμα ανάπτυξης και προσομοίωσης μοντέλων επεξεργαστών αλλά και περιφεριακών. Δίνει στον χρήστη τη δυνατότητα να δημιουργήσει τα δικά του μοντέλα για τις αρχιτεκτονικές που θέλει να προσομοιώσει παρέχοντας ταυτόχρονα και ένα σύνολο έτοιμων μοντέλων για διάφορες δημοφιλείς αρχιτεκτονικές όπως ARM, MIPS, PowerPC και άλλα. Ο χρήστης μπορεί επίσης μπορεί να αναπτύξει ένα δικό του μοντέλο για την ιεραρχία μνήμης και να το ενσωματώσει εύκολα στον προσομοιωτή. Από πλευράς ταχύτητας το OVP έχει πολύ καλές επιδόσεις σε σχέση με τους υπόλοιπους προσομοιωτές.

Σε κάθε περίπτωση, ωστόσο, το κύριο πρόβλημα με τα εργαλεία προσομοίωσης είναι ο μεγάλος χρόνος που απαιτείται για να ολοκληρωθεί η διαδικασία. Ακόμα και γρήγοροι προσομοιωτές όπως το OVP, δεν είναι επαρκείς όταν απαιτείται η προσομοίωση πολλών διαφορετικών αρχιτεκτονικών για την ανεύρεση της βέλτιστης. Ειδικά σε σύγχρονες εφαρμογές που η πολυπλοκότητα είναι ιδιαίτερα αυξημένη και οι προσπελάσεις μνήμης πολλές (π.χ. ψηφιακή επεξεργασία σήματος) ο χρόνος προσομοίωσης γίνεται απαγορευτικός. Για να βελτιωθεί σχετικά ο χρόνος προσομοίωσης μπορεί να δοθούν πιο αφαιρετικές περιγραφές της αρχιτεκτονικής, ελαττώνοντας όμως έτσι την ακρίβεια των αποτελεσμάτων και χωρίς δραματικές αλλαγές στον χρόνο προσομοίωσης.

2.2 Μέθοδοι δημιουργίας trace αρχείων

Ιδιαίτερα στη περίπτωση των “trace-driven” προσομοιωτών υπάρχουν κάποιες επιπλέον δυσκολίες [8]. Η πρώτη αφορά τον τρόπο δημιουργίας του trace αρχείου. Μία μέθοδος για τη καταγραφή όλων των αναφορών στη μνήμη είναι η σύνδεση probes κατευθείαν πάνω στον δίαυλο διευθύνσεων του υπολογιστικού συστήματος

στο οποίο γίνεται η εκτέλεση (external-probe based). Οι διευθύνσεις αποθηκεύονται προσωρινά σε κάποια μνήμη και μόλις αυτή γεμίσει μεταφέρονται σε κάποιο μέσο αποθήκευσης (πχ σκληρό δίσκο) από το οποίο αργότερα θα διαβαστούν για να γίνει η προσομοίωση της μνήμης cache. Η μέθοδος αυτή δεν έχει καμία χρονική επιβάρυνση (εκτέλεση real time) αν ο buffer είναι αρκετά μεγάλος. Αν όμως ο buffer δεν επαρκεί τότε κάθε φορά που τα δεδομένα του μεταφέρονται στο αποθηκευτικό μέσο με κάποιο τρόπο θα πρέπει να αναστέλλεται η λειτουργία του συστήματος, αλλιώς θα έχουμε απώλεια δεδομένων. Η συγκεκριμένη μέθοδος φυσικά απαιτεί τη φυσική ύπαρξη απαιτεί εξειδικευμένο υλικό (hardware) υψηλού κόστους και δεν έχει αρκετή ευελιξία.

Μία άλλη μέθοδος για τη δημιουργία των trace αρχείων είναι η τροποποίηση του μικροκώδικα (microcode) του επεξεργαστή στον οποίο τρέχει η εφαρμογή [9]. Ο μικροκώδικας ενός επεξεργαστή αποτελείται από ένα σύνολο εντολών πάρα πολύ χαμηλού επιπέδου ο οποίες χρησιμοποιούνται για την υλοποίηση των εντολών μηχανής [10]. Είναι αποθηκευμένος σε μία ειδική και πολύ γρήγορη μνήμη η οποία μπορεί να είναι ανάγνωσης μόνο ή ανάγνωσης και εγγραφής. Τροποποιώντας λοιπόν των μικροκώδικα των εντολών προσπέλασης της μνήμης μπορούμε να προκαλέσουμε ως «παρενέργεια» την αποθήκευση των διευθύνσεων που προσπελαύνονται σε μία προκαθορισμένη περιοχή της μνήμης του συστήματος. Αν και η συγκεκριμένη μέθοδος έχει αρκετά πλεονεκτήματα, είναι δύσκολο να χρησιμοποιηθεί πια, καθώς ο μικροκώδικας έχει εκλείψει στους σύγχρονους επεξεργαστές ή αν υπάρχει είναι πολύ δύσκολο να τροποποιηθεί. Φυσικά και αυτή η μέθοδος απαιτεί τη φυσική ύπαρξη του υλικού.

Αν και οι μέθοδοι που περιγράφηκαν στις προηγούμενες παραγράφους έχουν το πλεονέκτημα της πολύ υψηλής ταχύτητας, η πιο δημοφιλής μέθοδος για τη δημιουργία του αρχείου trace είναι μάλλον η χρήση κάποιου προσομοιωτή συνόλου εντολών (ISS). Ένας τέτοιος προσομοιωτής δέχεται ως είσοδο μία εφαρμογή γραμμένη για κάποιον επεξεργαστή «στόχο» (target) διαβάζει μία προς μία τις εντολές και εκτελεί ισοδύναμες λειτουργίες με χρήση του τοπικού επεξεργαστή. Ένας τέτοιος προσομοιωτής μπορεί να αναλάβει να καταγράφει τις διευθύνσεις που προσπελαύνονται στο αρχείο trace. Είναι προφανές ότι αυτή η μέθοδος δεν απαιτεί τη φυσική παρουσία του υλικού. Το πρόβλημα και εδώ είναι ο χρόνος που απαιτεί η προσομοίωση.

Η δημιουργία του trace αρχείου μπορεί επίσης να γίνει και με την επισημείωση του κώδικα μίας εφαρμογής (static code annotation). Συγκεκριμένα, γύρω από τις εντολές προσπέλασης στη μνήμη εισάγονται επιπλέον εντολές για τη καταγραφή των διευθύνσεων. Η τροποποίηση μπορεί να γίνει ή στον κώδικα assembly, στον object κώδικα ή στο τελικό εκτελέσιμο αρχείο (binary). Κάθε επιλογή έχει προτερήματα και μειονεκτήματα. Η μέθοδος αυτή έχει νόημα μόνο όταν ο τοπικός υπολογιστής χρησιμοποιεί το ίδιο σύνολο εντολών με τον υπολογιστή «στόχο» ή αν είναι διαθέσιμο το υλικό του συστήματος ώστε να γίνει πραγματική εκτέλεση των

εφαρμογών πάνω του. Η συγκεκριμένη μέθοδος μπορεί να επιτύχει πολύ μικρούς χρόνους δημιουργίας του trace αρχείου.

Τέλος, μία μέθοδος για τη δημιουργία του trace αρχείου είναι η βήμα προς βήμα εκτέλεση. Υπάρχουν διάφοροι debuggers που διαθέτουν επιλογή για εκτέλεση μίας εφαρμογής εντολή προς εντολή. Μία τέτοια λειτουργία συνήθως παρέχεται από τον ίδιο τον επεξεργαστή. Είναι εύκολο, λοιπόν, ένας τέτοιος debugger να τροποποιηθεί ώστε να καταγράφει μετά από κάθε βήμα τη διεύθυνση που προσπελάθηκε.

Σημαντικό πρόβλημα αποτελεί και το μέγεθος των trace αρχείων. Έχει βρεθεί [1] ότι το τυπικό μέγεθος ενός trace αρχείου μίας εφαρμογής με 1 εκατομμύριο εντολές απαιτεί περίπου 100GB αποθηκευτικού χώρου. Για τον λόγο αυτό έχουν προταθεί διάφορες μέθοδοι δειγματοληψίας μερικών μόνο αναφορών στη μνήμη για τη δημιουργία μικρότερων trace αρχείων. Μία τέτοια τεχνική παρουσιάζεται στο [11].

2.3 Έρευνα πάνω στους προσομοιωτές κρυφής μνήμης

Έχουν γίνει πολλές έρευνες με αντικείμενο τους προσομοιωτές μνήμης cache. Παρακάτω παρουσιάζονται περιληπτικά οι κυριότερες ερευνητικές προσπάθειες.

Το 1990 οι Borg, Kessler και Wall [12] παρουσιάζουν μία τεχνική κατά την οποία η εφαρμογή τροποποιείται κατά τη διάρκεια της σύνδεσης (linking time) ώστε να παράγεται αυτόματα το αρχείο trace κατά την εκτέλεση. Μία διεργασία που εκτελείται παράλληλα με την εφαρμογή (με υψηλότερη προτεραιότητα) επεξεργάζεται άμεσα τα δεδομένα του trace και υπολογίζει την επίδοση της μνήμης cache. Για την επικοινωνία μεταξύ της εφαρμογής και της διεργασίας δεσμεύονται κάποιοι καταχωρητές γενικής χρήσης και έτσι επιτυγχάνεται σημαντική βελτίωση της ταχύτητας. Ωστόσο, οι τροποποίηση που απαιτείται κατά τη σύνδεση δεν είναι δυνατό να γίνει για κάθε εφαρμογή πράγμα που κάνει τη μέθοδο μη εφαρμόσιμη σε κάποιες περιπτώσεις.

Το 1992, ο David Whalley [13] βασιζόμενος στη ιδέα ότι η διεύθυνση κάθε εντολής παραμένει ίδια σε όλη τη διάρκεια της εκτέλεσης ενός προγράμματος, ανέπτυξε τεχνικές ανάλυσης στη φάση της μεταγλώττισης (compile time) για τη μείωση του χρόνου προσομοίωσης. Για την εφαρμογή των τεχνικών που προτείνονται είναι απαραίτητη η τροποποίηση του μεταγλωττιστή.

Το 1995, οι Sugumar & Abraham [14] εισήγαγαν τη χρήση γενικευμένων διονυμικών δέντρων για την αναπαράσταση των παραμέτρων της μνήμης cache. Ανέπτυξαν έναν αλγόριθμο ο οποίος με τη χρήση αυτών των δομών μπορεί να προσομοιώσει ταυτόχρονα και με μία μόνο ανάγνωση του trace αρχείου τη συμπεριφορά πολλών διαφορετικών αρχιτεκτονικών μνήμης. Με τον τρόπο αυτό η αναζήτηση της βέλτιστης αρχιτεκτονικής επιταχύνεται σημαντικά. Προσπάθειες για

αναζήτηση της βέλτιστης αρχιτεκτονικής με ένα πέρασμα έγιναν αργότερα και από τους Janapsatya κλπ. [15] και Viana κλπ. [16].

Το 1998 οι Schnar και Larus παρουσίασαν τον προσομοιωτή FastSim [17]. Στον FastSim είχαν ενσωματωθεί γνωστές τεχνικές βελτιστοποίησης από τις γλώσσες προγραμματισμού και την αρχιτεκτονικής επεξεργαστών: το memoization και η υποθετική εκτέλεση εντολών. Η ενσωμάτωση αυτών τον τεχνικών είχε ως αποτέλεσμα την αύξηση της ταχύτητας κατά 4.9–11.9 φορές.

Το 2002, ο Nohl και η ομάδα του παρουσιάζουν στο [18] μία τεχνική προσομοίωσης για βελτίωση της ταχύτητας και της ευελιξίας με την ονομασία (just in time cache compiled simulation). Η τεχνική τους ενσωματώθηκε στη πλατφόρμα σχεδίασης επεξεργαστών LISA. Η ευελιξία που παρέχει είναι ικανοποιητική όμως οι ταχύτητες δεν είναι αρκετά υψηλές για την διερεύνηση αρχιτεκτονικών.

Το 2004 οι Ghosh και Grivargis [19], παρουσίασαν μία αναλυτική μέθοδο εξερεύνησης των παραμέτρων της μνήμης cache. Η μέθοδος απαιτεί σαν είσοδο, εκτός από το trace αρχείο με τις αναφορές στη μνήμη, την επιθυμητή επίδοση της μνήμης cache (μέγιστος αριθμός αστοχιών) και ένας αλγόριθμος εντοπίζει το σύνολο τον αρχιτεκτονικών μνήμης που ικανοποιούν τις απαιτήσεις. Δεν γίνεται κανένα είδος προσομοίωσης και αυτό έχει ως αποτέλεσμα μικρούς χρόνους εξερεύνησης για μικρές εφαρμογές. Όσο όμως το μέγεθος των εφαρμογών αυξάνει, οι χρόνοι πλησιάζουν σε αυτούς της προσομοίωσης.

2.4 Τεχνικές βασισμένες στη μοντελοποίηση της μνήμης cache

Αρκετή ερεύνα έχει γίνει στο τομέα της εκτίμησης επίδοσης των μνημών cache με χρήση μοντέλων. Όπως αναφέρεται στο [1] οι μέθοδοι μοντελοποίησης των μνημών cache μπορούν να χωριστούν σε 4 κατηγορίες: τη στατιστική μοντελοποίηση [20], την υβριδική (στατιστική – αναλυτική) μοντελοποίηση [21], τις εξισώσεις αστοχιών μνήμης cache [22] και τη στατική προσομοίωση [23]. Παρακάτω γίνεται μία σύντομη αναφορά στις σημαντικότερες προσπάθειες για κάθε μία από τις τέσσερεις κατηγορίες.

2.4.1 Στατιστική μοντελοποίηση

Η τεχνική που παρουσιάζεται στο [20] βασίζεται σε έναν προσομοιωτή αρχιτεκτονικής υπολογιστή (HLS) στον οποίο όμως δεν προσομοιώνεται η εκτέλεση ολόκληρης της εφαρμογής, αλλά μόνο ένα δείγμα συμβολικού κώδικα το οποίο δημιουργείται από τη στατιστική ανάλυση του κώδικα της εφαρμογής. Τα βήματα που ακολουθούνται είναι τα εξής:

1. Η εφαρμογή μεταγλωττίζεται και δημιουργείται το τελικό εκτελέσιμο αρχείο (binary).

2. Προσομοιώνεται η εκτέλεση του εκτελέσιμου αρχείου σε έναν τροποποιημένο προσομοιωτή Simplescalar. Κατά τη προσομοίωση συλλέγονται στατιστικά δεδομένα που αφορούν τις αποστάσεις μεταξύ των εντολών που απασχολούν κοινές μονάδες του επεξεργαστή.

3. Στη συνέχεια προσομοιώνεται η εκτέλεση του εκτελέσιμου αρχείου σε έναν κανονικό προσομοιωτή Simplescalar και συλλέγονται στατιστικά δεδομένα για των branch predictors και της μνήμης cache.

4. Από τα στατιστικά δεδομένα που έχουν συλλεχθεί παράγεται ο συμβολικός κώδικας. Η διαφορά του συμβολικού αυτού κώδικα από τον συμβατικό κώδικα είναι ότι οι εντολές του, αντί για πραγματικές παραμέτρους, έχουν στατιστικές παραμέτρους που παρέχουν πληροφορίες για τις λειτουργικές μονάδες του επεξεργαστεί που θέτονται σε λειτουργία, για τη συμπεριφορά της μνήμης cache, και για τις εξαρτήσεις με άλλες εντολές.

5. Ο συμβολικός κώδικας εκτελείται στον προσομοιωτή HLS και γίνεται εκτίμηση της επίδοσης της εφαρμογής.

Καθώς ο HLS είναι ένας στατιστικός προσομοιωτής, το βήμα 5 θα πρέπει να εκτελεστεί αρκετές φορές ώστε να έχουμε μικρή τυπική απόκλιση στα αποτελέσματα, κάτι που σημαίνει ότι απαιτείται αρκετή ώρα για να προκύψει μία ικανοποιητική εκτίμηση. Επίσης, χρονική επιβάρυνση προσθέτουν και οι προσομοιώσεις των βημάτων 2 και 3.

2.4.2 Υβριδική (στατιστική - αναλυτική) μοντελοποίηση

Οι Eeckhout & De Bosschere [21] παρατήρησαν σε μία εφαρμογή τα διάφορα χαρακτηριστικά της λειτουργίας των καταχωρητών (πλήθος προσπελάσεων, διάρκεια ζωής του στιγμιότυπου, χρήσιμη διάρκεια ζωής του στιγμιότυπου, ηλικία του στιγμιότυπου) ακολουθούν συναρτήσεις κατανομής πιθανότητας που ταιριάζουν στον νόμο δύναμης:

$$P\ X = x\ = ax^{-b} \qquad \textbf{(2.1)}$$

Η συγκεκριμένη κατανομή χαρακτηρίζεται πλήρως από δύο παραμέτρους (τις a και b). Έτσι για κάθε εφαρμογή μπορεί να κατασκευαστεί ένα στατιστικό μοντέλο με χρήση λίγων μόνο συντελεστών. Αυτό σημαίνει ότι ο χώρος των διαφόρων παραμέτρων σχεδίασης μπορεί να εξερευνηθεί αποδοτικά και για πολλές εφαρμογές με απλή αλλαγή των εν λόγω συντελεστών. Ωστόσο οι συντελεστές αυτοί θα πρέπει να έχουν πρώτα υπολογιστεί με στατιστική ανάλυση της κάθε εφαρμογής.

2.4.3 Εξισώσεις αστοχιών μνήμης cache

Στο [22] παρουσιάζεται ένα αναλυτικό μοντέλο για τη στατική εκτίμηση της επίδοσης της μνήμης cache βασισμένο στα χαρακτηριστικά των εμφωλευμένων βρόχων. Οι Vera & Xue ανέπτυξαν αναλυτικές εξισώσεις που μπορούν να

εκτιμήσουν με καλή ακρίβεια τη συμπεριφορά της μνήμης cache χωρίς να απαιτείται κανενός είδους προσομοίωση ή εκτέλεση της εφαρμογής. Ξεχωριστές εξισώσεις δίνονται για τον υπολογισμό των replacement misses και των cold misses, ενώ προτείνονται δύο αλγόριθμοι για την επίλυση των εξισώσεων.

2.4.4 Στατική προσομοίωση

Οι Mueller και Whaley [23] προτείνουν έναν στατικό προσομοιωτή. Με στατική ανάλυση του κώδικα μπορούν να βρεθούν οι εντολές που θα προκαλούν πάντα αστοχίες ή ευστοχίες. Στη συνέχεια η εφαρμογή εκτελείται και καταμετρητές μετρούν πόσες φορές εκτελέσθηκε κάθε κομμάτι του κώδικα. Ταυτόχρονα γίνεται δυναμική προσομοίωση τις μνήμης cache για τις εντολές που δεν ήταν δυνατόν να γίνει στατική προσομοίωση. Τελικά ο αριθμός των αστοχιών μπορεί να προκύψει από τα αποτελέσματα της στατικής και της δυναμικής προσομοίωσης και από τις τιμές των καταμετρητών. Για τη χρήση της εν λόγω τεχνικής απαιτούνται τροποποιήσεις στον compiler.

2.5 Το εργαλείο FICA

Το 2009 παρουσιάστηκε ένα νέο εργαλείο το οποίο μπορεί να κάνει πολύ γρήγορη εκτίμηση της επίδοσης της ιεραρχίας μνήμης αντλώντας πληροφορίες από τον πηγαίο κώδικα. Οι Ν. Κρούπης και Δ. Σούντρης [1] στηρίχτηκαν στη παραδοχή ότι ο γράφος ροής ελέγχου (Control Flow Graph - CFG) ενός προγράμματος είναι ο ίδιος στον πηγαίο και στον τελικό κώδικα. Εισάγοντας λοιπόν μετρητές σε κάθε διακλάδωση του προγράμματος (απευθείας στον πηγαίο κώδικα), μεταγλωττίζοντας και εκτελώντας τον κώδικα μπορούμε να συλλέξουμε πληροφορίες για τον αριθμό τον εκτελέσεων του κάθε basic block του προγράμματος. Στη συνέχεια από τον assembly κώδικα της εφαρμογής δημιουργείται ένας γράφος ροής ελέγχου στον οποίο αντιστοιχίζονται οι τιμές των μετρητών. Με μία επαναληπτική διαδικασία υπολογίζεται ο αριθμός των εκτελέσεων κάθε basic block και έπειτα εκτιμάται στατικά (με τη χρήση αναλυτικών εξισώσεων που παρουσιάζονται στο [24]) για κάθε basic block ο αριθμός των αστοχιών που θα επιφέρει. Πολλαπλασιάζοντας, λοιπόν με τον αριθμό των εκτελέσεων μπορούμε να υπολογίσουμε τον συνολικό αριθμό των αστοχιών και μάλιστα για διαφορετικά μεγέθη μνήμης cache ταυτόχρονα. Η συγκεκριμένη μέθοδος επιφέρει αποτελέσματα με αρκετά καλή ακρίβεια και με ταχύτητες από 10 έως 13000 μεγαλύτερες (σε σχέση με τη προσομοίωση με χρήση του Simplescalar). Σε αυτή τη μέθοδο στηρίζεται και η παρούσα διπλωματική εργασία.

3 Η Προτεινομενη Μεθοδος

Τα πειράματα των Ν. Κρούπη και Δ. Σούντρη έδειξαν ότι η μέθοδός τους για την εκτίμηση της επίδοσης της κρυφής μνήμης εντολών μπορεί να πετύχει πολύ υψηλή ακρίβεια σε πολύ μικρό χρονικό διάστημα [1]. Αυτό δείχνει ότι αξίζει να ασχοληθούμε με την τελειοποίηση της μεθόδου και την δημιουργία μίας λειτουργικής υλοποίησής της. Πολλά στοιχεία της νέας μεθοδολογίας που παρουσιάζεται σε αυτή τη διπλωματική λαμβάνονται αυτούσια από την ήδη υπάρχουσα δουλειά, άλλα είναι απλές τροποποιήσεις και άλλα είναι εντελώς νέα. Ωστόσο, σε αυτή την αναφορά η νέα μεθοδολογία θα παρουσιαστεί από το μηδέν για λόγους συνοχής και πληρότητας.

Σε αυτό το κεφάλαιο θα παρουσιαστούν αναλυτικά οι νέες τεχνικές που αναπτύχθηκαν. Πρόκειται για τεχνικές οι οποίες αποσκοπούν στην εκτίμηση της επίδοσης που παρουσιάζει η εκτέλεση μίας εφαρμογής σε ένα ενσωματωμένο σύστημα. Συγκεκριμένα οι παράγοντες της επίδοσης που μπορούν να εκτιμηθούν με τις νέες μεθόδους είναι το **πλήθος των εκτελούμενων εντολών** και η επίδοση της ιεραρχίας μνήμης σε όρους **ρυθμού ευστοχίας (hit rate)**.

Στην ενότητα 3.1 θα δούμε την ορολογία που θα χρησιμοποιηθεί κατά τη περιγραφή των τεχνικών καθώς και τις παραδοχές που λαμβάνονται υπόψη. Στις ενότητες 3.2 έως 3.6 θα παρουσιαστεί μία μεθοδολογία με την οποία μπορούμε να έχουμε μία εκτίμηση για το πλήθος εκτελέσεων κάθε εντολής του κώδικα μηχανής που προορίζεται για ένα ενσωματωμένο σύστημα, χωρίς να απαιτείται προσομοίωση της εκτέλεσης της εφαρμογής. Στη συνέχεια στις ενότητες 3.7 και 3.8 θα παρουσιαστεί μία τεχνική για την εκτίμηση της επίδοσης της κρυφής μνήμης εντολών με βάση τα αποτελέσματα της πρώτης μεθοδολογίας.

3.1 Ορολογία και παραδοχές

Ένα σημαντικό πλεονέκτημα των μεθοδολογιών που αναπτύχθηκαν είναι ότι δεν απαιτούν τη φυσική παρουσία κάποιου συστήματος ίδιας αρχιτεκτονικής με το σύστημα για το οποίο θέλουμε να κάνουμε την εκτίμηση επίδοσης, ούτε τη χρήση κάποιου είδους προσομοιωτή. Αυτό που απαιτείται είναι η ύπαρξη του πηγαίου

κώδικα της εφαρμογής που προορίζεται για το ενσωματωμένο σύστημα σε γλώσσα C. Υποθέτουμε επίσης ότι είναι διαθέσιμος ένας υπολογιστής εξοπλισμένος με έναν μεταγλωττιστή της γλώσσας C για την αρχιτεκτονική του ενσωματωμένου συστήματος και με έναν μεταγλωττιστή της γλώσσας C για τη δική του αρχιτεκτονική. Λίγο παρακάτω (Πίνακας 3.1) διασαφηνίζεται η έννοια κάποιων όρων που θα χρησιμοποιηθούν κατά κόρον στη περιγραφή των μεθόδων.

Πίνακας 3.1: Επεξήγηση ειδικής ορολογίας

ΟΡΟΛΟΓΙΑ	ΕΠΕΞΗΓΗΣΗ
Host υπολογιστής:	Ο υπολογιστής στον οποίο εκτελείτε η διαδικασία εκτίμησης. Δεν είναι απαραίτητο η αρχιτεκτονική του να είναι ίδια με αυτή του ενσωματωμένου συστήματος.
Target επεξεργαστής:	Ο επεξεργαστής που χρησιμοποιεί το ενσωματωμένο σύστημα.
Native compiler:	Ο μεταγλωττιστής για την αρχιτεκτονική του host υπολογιστή.
Cross compiler:	Ο μεταγλωττιστής για την αρχιτεκτονική του target επεξεργαστή.
Άλμα, jump	Η μετάβαση του ελέγχου εκτέλεσης ενός προγράμματος από ένα σημείο Α σε ένα σημείο Β όπου τα Α και Β δεν είναι διαδοχικά.
Υπό συνθήκη άλμα, Conditional jump	Το άλμα του οποίου η εκτέλεση εξαρτάται από μία λογική συνθήκη.
Προέλευση άλματος, Πηγή άλματος, Jump origin:	Το σημείο στο οποίο διακόπτεται η σειριακή ροή ενός προγράμματος. Η διεύθυνση μίας εντολής άλματος.
Προορισμός άλματος, Στόχος άλματος, Jump target:	Η διεύθυνση του σημείου στο οποίο γίνεται η μετάβαση ελέγχου κατά την εκτέλεση του άλματος.
Basic block:	Το βασικό στοιχείο συνεχούς εκτέλεσης του κώδικα. Είναι ένα κομμάτι κώδικα το οποίο ικανοποιεί τις εξής συνθήκες [25]. 1) Έχει μόνο ένα σημείο εισόδου.

	Καμία εντολή του εκτός της πρώτης δεν είναι στόχος κάποιας εντολής άλματος. 2) Έχει μόνο ένα σημείο εξόδου. Μόνο η τελευταία εντολή μπορεί να είναι εντολή άλματος.
Target basic block:	Ένα basic block του κώδικα μηχανής του target επεξεργαστή.
Source basic block:	Ένα basic block του πηγαίου κώδικα.
Γράφος Ροής Ελέγχου (Γ.Ρ.Ε), Control Flow Graph (C.F.G.):	Ένας κατευθυνόμενος γράφος που αναπαριστά τη ροή ενός προγράμματος. Κόμβοι του γράφου είναι τα basic blocks του προγράμματος και ακμές τα άλματα. Επιπλέον υπάρχουν δύο ειδικοί κόμβοι που αναπαριστούν το σημείο έναρξης και λήξης του προγράμματος.
Arcs:	Τα άλματα (jumps) ως στοιχεία ενός Γ.Ρ.Ε. (οι ακμές του Γ.Ρ.Ε.).
Target Γ.Ρ.Ε., Target C.F.G.:	Ο Γ.Ρ.Ε. του κώδικα μηχανής για τον target επεξεργαστή.
Source Γ.Ρ.Ε., Source C.F.G.	Ο Γ.Ρ.Ε. του πηγαίου κώδικα μίας εφαρμογής.
Επίσημειωμένος Γράφος Ροής Ελέγχου (Ε.Γ.Ρ.Ε), Annotated Control Flow Graph (A.C.F.G):	Ο Γ.Ρ.Ε του οποίου οι κόμβοι περιέχουν επιπλέον πληροφορίες για τα αντίστοιχα basic blocks όπως: • Ο αριθμός εκτελέσεων • Η διεύθυνσή τους στη μνήμη • Το μέγεθός τους Ενώ οι ακμές περιέχουν πληροφορίες για το πλήθος εκτελέσεων των αντίστοιχων αλμάτων.
Target Ε.Γ.Ρ.Ε.	Ο Επίσημειωμένος Γράφος Ροής Ελέγχου για τον κώδικα μηχανής για τον target επεξεργαστή.

Επειδή οι μεθοδολογίες που έχουν αναπτυχθεί δεν χρησιμοποιούν σε κανένα σημείο τον κώδικα μηχανής του host υπολογιστή, η αναφορές μας σε αυτό θα είναι σπάνιες. Έτσι για λόγους συντομίας, όπου αναφέρεται ο όρος «κώδικας μηχανής» θα εννοείται ο κώδικας μηχανής του target επεξεργαστή, εκτός αν αναφέρεται ρητώς ότι πρόκειται για τον κώδικα μηχανής του host υπολογιστή.

3.2 Η βασική ιδέα

Η βασική ιδέα της μεθόδου είναι να στηριχτούμε στις πληροφορίες που μπορούμε να αντλήσουμε από τον πηγαίο κώδικα μίας εφαρμογής για να εξάγουμε συμπεράσματα για τον κώδικα μηχανής (το τελικό αποτέλεσμα της διαδικασίας μεταγλώττισης). Κάτι που μπορούμε να θεωρήσουμε βέβαιο είναι ότι ο ίδιος πηγαίος κώδικας θα πρέπει να έχει ακριβώς τα ίδια λειτουργικά αποτελέσματα ανεξάρτητα από την αρχιτεκτονική στην οποία θα εκτελεσθεί. Σε αυτήν τη θεώρηση υπάρχουν κάποιες εξαιρέσεις τις οποίες θα προσπαθήσουμε να καλύψουμε στην ενότητα 3.9. Φυσικά σε καμία περίπτωση δεν θα μπορούσαμε να υποστηρίξουμε ότι συμβαίνει το ίδιο και για την επίδοση της εφαρμογής όταν αυτή εκτελείται σε διαφορετικές αρχιτεκτονικές.

Αν θεωρήσουμε ότι εκτελούμε την εφαρμογή μας σε μία αρχιτεκτονική Α και συλλέγουμε πληροφορίες για τη συμπεριφορά του πηγαίου κώδικα, όπως το πόσες φορές εκτελέστηκε κάθε γραμμή του, τότε σίγουρα ξέρουμε ποια θα είναι η συμπεριφορά του πηγαίου κώδικα αν αυτός εκτελεσθεί σε μία αρχιτεκτονική Β. Θα είναι ακριβώς η ίδια (εκτός των εξαιρέσεων που περιγράφουμε στην ενότητα 3.9). Ίσως όμως να μπορούμε να κάνουμε μία εκτίμηση και για το ποια θα είναι η επίδοση της εφαρμογής μας όταν αυτή εκτελεσθεί στην αρχιτεκτονική Β.

Η επιπλέον πληροφορία που χρειαζόμαστε για να κάνουμε την εκτίμησή μας, είναι μία αντιστοίχηση ανάμεσα στον πηγαίο κώδικα και στον κώδικα μηχανής για την αρχιτεκτονική Β. Αν με κάποιο τρόπο μπορούμε να ξέρουμε ποια λειτουργία του πηγαίου κώδικα εκτελεί κάθε μία εντολή του κώδικα μηχανής, τότε μπορούμε να υποστηρίξουμε ότι η εν λόγω εντολή μηχανής θα εκτελεσθεί τόσες φορές όσες θα εκτελεσθεί και η αντίστοιχη λειτουργία του πηγαίου κώδικα στην αρχιτεκτονική Α (η οποιαδήποτε άλλη). Έπειτα, εξάγοντας από τον κώδικα μηχανής τον Γ.Ρ.Ε. και γνωρίζοντας τον αριθμό εκτελέσεων κάθε μίας εντολής του μπορούμε να δημιουργήσουμε τον target Ε.Γ.Ρ.Ε.. Στη συνέχεια, με περεταίρω ανάλυση του target Ε.Γ.Ρ.Ε μπορούμε να κάνουμε επιπλέον εκτιμήσεις επίδοσης. Τον τρόπο θα τον περιγράψουμε στις ενότητες 3.7 και 3.8.

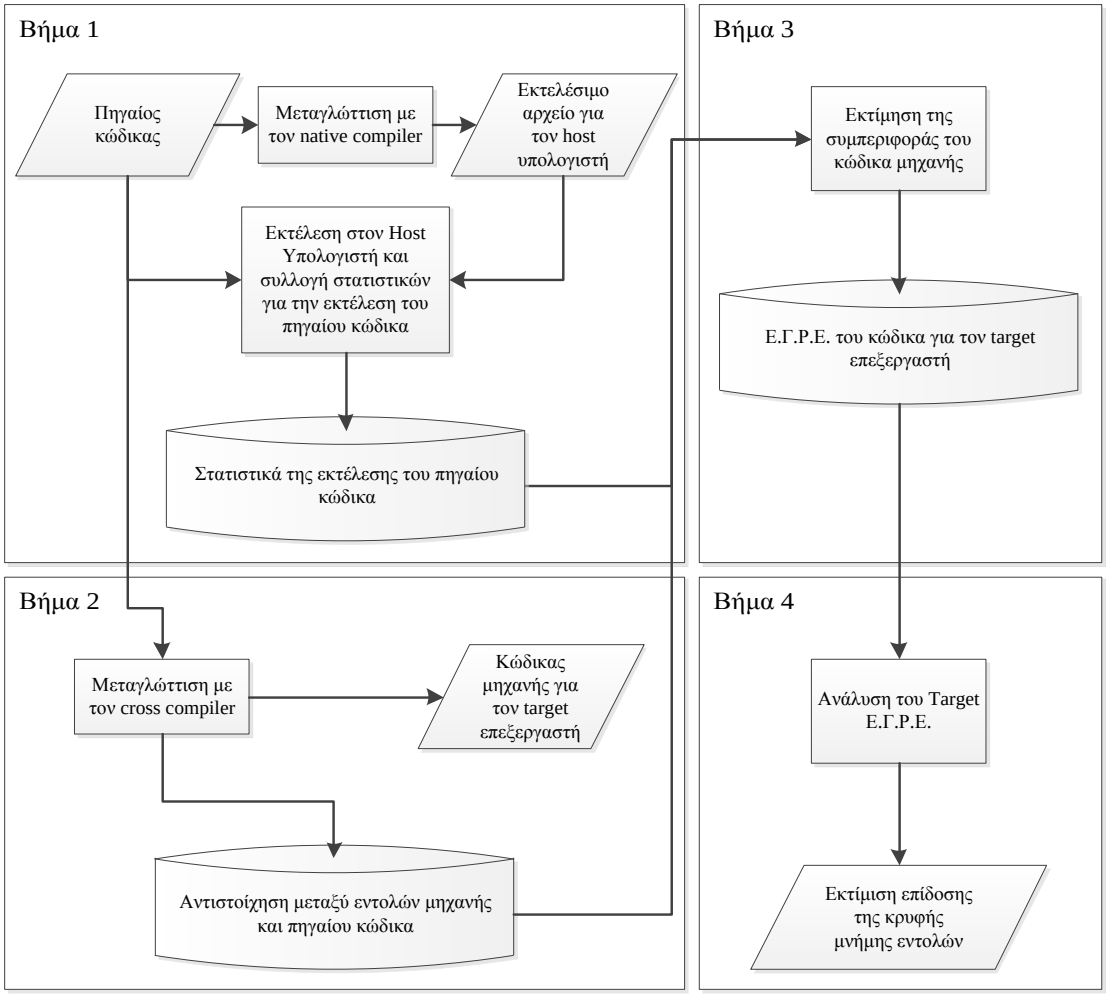
Σύμφωνα με τα παραπάνω, τα βασικά βήματα της νέας μεθοδολογίας έχουν ως εξής:

1. Εκτέλεση της εφαρμογής στον host υπολογιστή και συλλογή πληροφοριών για την εκτέλεση του πηγαίου κώδικα.

2. Αντιστοίχηση του πηγαίου κώδικα με τον κώδικα μηχανής του target processor.

3. Εκτίμηση της συμπεριφοράς του κώδικα μηχανής με βάση την αντιστοίχιση του βήματος 2 και τις πληροφορίες που συλλέχθηκαν στο βήμα 1 (επισημείωση του target Γ.Ρ.Ε.).

4. Ανάλυση του target Ε.Γ.Ρ.Ε για περαιτέρω εκτίμηση της επίδοσης (επίδοση μνήμης cache).



Σχήμα 3.1: Τα βήματα της μεθοδολογίας

Στη συνέχεια στην ενότητα 3.3 περιγράφουμε αναλυτικά το βήμα 1, στις ενότητες 3.4 έως 3.6 τα βήματα 2 και 3 και στις ενότητες 3.7 και 3.8 το βήμα 4.

3.3 Εκτέλεση της εφαρμογής και χαρακτηρισμός του πηγαίου κώδικα

Στο πρώτο στάδιο της μεθόδου θα πρέπει να γίνει ο «χαρακτηρισμός του πηγαίου κώδικα». Με την φράση αυτή εννοούμε ότι με κάποιον τρόπο πρέπει να συλλέξουμε πληροφορίες για το πλήθος εκτελέσεων κάθε μίας δομικής μονάδας του πηγαίου κώδικα. Αυτό θα πρέπει να γίνει με μεταγλώττιση της εφαρμογής μας με τον native compiler και άμεση εκτέλεση στον host υπολογιστή όπως έχει ήδη αναφερθεί.

3.3.1 Ορισμός δομικής μονάδας εκτέλεσης πηγαίου κώδικα

Το ερώτημα που τίθεται στο συγκεκριμένο σημείο είναι το τι θα θεωρήσουμε ως βασική δομική μονάδα του πηγαίου κώδικα. Έχοντας υπόψη ότι ο πηγαίος κώδικας είναι γραμμένος στη γλώσσα C, κάποιες πιθανές επιλογές θα μπορούσαν να ήταν:

- Τα basic blocks του πηγαίου κώδικα
- Τα statements (βασικές εντολές) της γλώσσας C
- Τα primary expressions (βασικές εκφράσεις) της γλώσσας C
- Η κάθε γραμμή κώδικα

Η προδιαγραφές που θέλουμε να πληροί η βασική δομική μονάδα που θα επιλέξουμε είναι οι εξής:

1. Κάθε εκτέλεση της δομικής μονάδας θα πρέπει να είναι συνεχείς και ίδια με τις άλλες εκτελέσεις. Δεν θα μπορούσαμε να ορίσουμε π.χ. ως δομική μονάδα μία selection statement (εντολή if) καθώς το κομμάτι κώδικα που θα εκτελεστεί εξαρτάται από την συνθήκη. Έτσι μπορεί μία εκτέλεση της εντολής if να οδηγήσει στην εκτέλεση της εντολής μηχανής A και μία άλλη εκτέλεση να οδηγήσει στην εκτέλεση της εντολής μηχανής B. Αυτό είναι προφανές ότι κάνει αδύνατη την αντιστοίχιση της δομικής μονάδας με τις εντολές του κώδικα μηχανής.

2. Θα πρέπει να υπάρχει τρόπος να καταμετρηθεί το πλήθος εκτελέσεων κάθε δομικής μονάδας. Αν π.χ. επιλεχθούν ως βασικές μονάδες τα basic blocks του πηγαίου κώδικα, το πλήθος εκτελέσεών τους μπορεί να υπολογιστεί εισάγοντας εντολές καταμέτρησης στην αρχή κάθε basic block [1].

3. Θα πρέπει να υπάρχει τρόπος να αντιστοιχηθεί κάθε εντολή μηχανής (του target επεξεργαστή) με μία από τις δομικές μονάδες του πηγαίου κώδικα.

Θα εξετάσουμε τώρα τις πιθανές επιλογές ώστε να βρούμε ποια ικανοποιεί καλύτερα τις προδιαγραφές.

Τα basic block του πηγαίου κώδικα φαίνεται εκ πρώτης όψεως να αποτελούν μία καλή επιλογή. Το κριτήριο 1 ικανοποιείται εξ' ορισμού ενώ το κριτήριο 2 θα μπορούσε να εκπληρωθεί με την εισαγωγή εντολών καταμέτρησης στον πηγαίο κώδικα. Το πρόβλημα είναι η αντιστοίχιση των εντολών μηχανής και των basic blocks (κριτήριο 3). Αν θεωρήσουμε ότι ο Γ.Ρ.Ε. του κώδικα μηχανής είναι πανομοιότυπος με τον Γ.Ρ.Ε. του πηγαίου κώδικα [1], τότε λύνουμε αυτομάτως το πρόβλημα αντιστοίχισης. Ωστόσο η υπόθεση αυτή δεν είναι πάντα αληθής. Τις περισσότερες φορές ο μεταγλωττιστής θα τροποποιήσει σημαντικά τον Γ.Ρ.Ε. της εφαρμογής στα διάφορα στάδια της μεταγλώττισης, κάτι που κάνει την αντιστοίχιση των basic blocks στους δύο Γ.Ρ.Ε αδύνατη χωρίς την τροποποίηση του μεταγλωττιστή.

Οι βασικές εντολές της C από την άλλη δεν ικανοποιούν το κριτήριο 1. Μία εντολή της γλώσσας C μπορεί να εμπεριέχει πολλαπλά basic blocks. Αυτό σημαίνει ότι η εκτέλεση της δεν είναι σίγουρο ότι θα είναι συνεχής και ίδια σε κάθε περίπτωση. Ένα παράδειγμα τέτοιων εντολών αποτελούν οι selection statements (εντολές if) όπως αναφέραμε παραπάνω στο κριτήριο 2.

Οι primary expressions τέλος δεν είναι σίγουρο ότι θα αποτελούν πάντα εκτελέσιμο κώδικα. Είναι πιθανό μία έκφραση της C να μην αντιστοιχίζεται σε καμία εντολή μηχανής (π.χ. σταθερές εκφράσεις). Αυτό, ωστόσο, δεν αποτελεί σημαντικό πρόβλημα. Το σημαντικότερο είναι ότι όπως και με τα basic blocks του πηγαίου κώδικα είναι δύσκολη η αντιστοίχιση με τον κώδικα μηχανής.

Η δομική μονάδα που επιλέχθηκε στη μεθοδολογία που αναπτύχθηκε είναι η γραμμή κώδικα. Η επιλογή αυτή ικανοποιεί πολύ καλά τις προδιαγραφές 2 και 3. Για την καταμέτρηση του πλήθους των εκτελέσεων κάθε γραμμής του πηγαίου κώδικα υπάρχουν αρκετά έτοιμα εργαλεία (coverage profilers) [26], ενώ όπως θα δούμε στην ενότητα 3.4 η αντιστοίχιση των εντολών μηχανής του target επεξεργαστή με τις γραμμές πηγαίου κώδικα μπορεί να γίνει με χρήση του debugging format DWARF. Αυτό που δεν είναι σαφές στη παρούσα φάση είναι το πώς ικανοποιείται η προδιαγραφή 1, αφού κανείς δεν μας απαγορεύει να γράψουμε ακόμα και ένα ολόκληρο πρόγραμμα σε μία μόνο γραμμή κώδικα C. Προς το παρόν υποθέστε ότι μπορούμε με κάποιο τρόπο να διασφαλίσουμε τη προδιαγραφή 1 για κάθε γραμμή κώδικα. Το πώς μπορεί να γίνει αυτό θα αναλυθεί στην ενότητα 3.5.

3.3.2 Καταμέτρηση του πλήθους εκτελέσεων γραμμών (το εργαλείο gcov)

Αφού επιλέξαμε ως δομική μονάδα του πηγαίου κώδικα τη γραμμή κώδικα, θα πρέπει τώρα να ορίσουμε και τον τρόπο με τον οποίο θα γίνει η καταμέτρηση των εκτελέσεων κάθε γραμμής. Όπως αναφέραμε και στην προηγούμενη ενότητα, για τον σκοπό αυτό έχουν αναπτυχθεί διάφορα εργαλεία. Στα πλαίσια αυτής της διπλωματικής εργασίας χρησιμοποιήθηκε το εργαλείο gcov [27].

Το gcov είναι ένα εργαλείο συμβατό με τον compiler gcc (που χρησιμοποιείται και στην παρούσα εργασία). Το gcov λειτουργεί αναλύοντας τον Γ.Ρ.Ε. του πηγαίου

κώδικα κατά τη φάση της μεταγλώττισης και δημιουργώντας ένα αρχείο που τον περιγράφει (.gcda). Παράλληλα, εισάγονται σε επιλεγμένα arcs του Γ.Ρ.Ε. της εφαρμογής εντολές καταμέτρησης με τις οποίες στη φάση εκτέλεσης της εφαρμογής γίνεται καταμέτρηση του πλήθους εκτελέσεων των αλμάτων. Κατά την έξοδο της εφαρμογής οι τιμές των μετρητών αποθηκεύονται σε ένα νέο αρχείο (.gcno). Μετά την έξοδο της εφαρμογής το gcov συνδυάζει τις πληροφορίες από τα αρχεία .gcda και .gcno και παράγει ένα αρχείο που περιέχει τον πηγαίο κώδικα επισημειωμένο με το πλήθος εκτελέσεων κάθε γραμμής κώδικα (.gcov).

Όπως αναφέρθηκε, το gcov εισάγει εντολές καταμέτρησης σε επιλεγμένα arcs. Δεν εισάγονται εντολές καταμέτρησης σε όλα τα arcs για να είναι μικρότερο το μέγεθος του εξαγόμενου εκτελέσιμου αλλά κυρίως για να είναι ταχύτερη η εκτέλεση του προγράμματος. Η επιλογή των arcs στα οποία θα εισαχθούν εντολές καταμέτρησης γίνεται με τέτοιον τρόπο ώστε να είναι πάντα δυνατό να υπολογισθούν και οι τιμές εκτελέσεων των arcs στα οποία δεν εισάγονται εντολές καταμέτρησης με μετέπειτα ανάλυση.

Συγκεκριμένα, για να επιλεχθούν τα arcs που θα εισαχθούν καταμετρητές, αναζητείται και κατασκευάζεται ένα συνδετικό δέντρο [28] του Γ.Ρ.Ε. του πηγαίου κώδικα. Στη συνέχεια εισάγονται εντολές καταμέτρησης σε όλα τα arcs που δεν περιέχονται στο συνδετικό δέντρο [29]. Η μέθοδος αυτή διασφαλίζει ότι με μία επαναληπτική διαδικασία, θα μπορούν μετά την εκτέλεση τις εφαρμογής να υπολογισθούν οι εκτελέσεις όλων των arcs του Γ.Ρ.Ε. Φυσικά, εφόσον έχουν υπολογισθεί οι εκτελέσεις όλων των arcs είναι εύκολο να υπολογισθούν και οι εκτελέσεις των basic blocks απλά αθροίζοντας τα πλήθη εκτελέσεων όλων των εισερχόμενων arcs για κάθε basic block.

Για να χρησιμοποιήσουμε το gcov απαιτείται να ενεργοποιήσουμε στον native compiler (gcc) τις σημαίες –fprofile-arcs και –ftest-coverage. Στη συνέχεια εκτελούμε κανονικά την εφαρμογή μας και μετά εκτελούμε την εντολή “gcov <filename>” όπου <filename> είναι το όνομα του αρχείου πηγαίου κώδικα. Παρακάτω φαίνεται ένα απλό παράδειγμα πηγαίου κώδικα σε γλώσσα C (Κώδικας 3.1) και ένα παράδειγμα τυπικής εξόδου του gcov (αρχείο .gcov) (Κώδικας 3.2) για τον εν λόγω κώδικα.

```
int main(){
    int i,j,k;
    j=1;k=1;
    if(j==0)
        k=0;
    for(i=0;i<=100;i++){
        j+=i;
        k+=j;
    }
    return j;
}
```

Κώδικας 3.1: Ένα παράδειγμα απλού πηγαίου κώδικα

```
        -:    0:Source:test.c
        -:    0:Graph:test.gcno
        -:    0:Data:test.gcda
        -:    0:Runs:1
        -:    0:Programs:1
        1:    1:int main(){
        -:    2:    int i,j,k;
        1:    3:    j=1;k=1;
        1:    4:    if(j==0)
    #####:    5:        k=0;
      102:    6:    for(i=0;i<=100;i++){
      101:    7:        j+=i;
      101:    8:        k+=j;
        -:    9:    }
        1:   10:    return j;
        -:   11:}
```

Κώδικας 3.2: Ένα παράδειγμα τυπικής εξόδου του gcov

Όπως φαίνεται παραπάνω ένα αρχείο gcov δομείται σε τρείς στήλες οι οποίες διαχωρίζονται με τον χαρακτήρα ‘:’. Στη τρίτη στήλη εμφανίζεται ο πηγαίος κώδικας της εφαρμογής μας με αναλλοίωτη τη μορφοποίησή του, στη δεύτερη στήλη εμφανίζεται ο αριθμός γραμμής του πηγαίου κώδικα και στη πρώτη στήλη εμφανίζεται το πλήθος εκτελέσεων της αντίστοιχης γραμμής. Οι πρώτες γραμμές του gcov αρχείου περιέχουν επιπλέον ενημερωτικές πληροφορίες οι οποίες διακρίνονται από τον πηγαίο κώδικα της εφαρμογής μας καθώς «μαρκάρονται» με αριθμό γραμμής 0.

Η πρώτη στήλη μπορεί να περιέχει είτε τον χαρακτήρα ‘-‘, είτε τη σήμανση ‘#####’, είτε έναν αριθμό. Ο χαρακτήρας ‘-‘ δηλώνει ότι η αντίστοιχη γραμμή του πηγαίου κώδικα δεν περιέχει καμία εκτελέσιμη εντολή, η σήμανση ‘#####’ δηλώνει ότι η αντίστοιχη γραμμή δεν εκτελέστηκε καμία φορά, ενώ ένας αριθμός δηλώνει το πλήθος εκτελέσεων της γραμμής.

Όπως φαίνεται στο παραπάνω παράδειγμα (Κώδικας 3.2), το gcov μας δίνει πληροφορία για το πλήθος εκτελέσεων σε επίπεδο γραμμής (αυτή εξάλλου είναι και η δομική μονάδα που επιλέξαμε στην ενότητα 3.3.1). Το gcov ωστόσο παρέχει τη δυνατότητα να δίνει πληροφορίες για το πλήθος εκτελέσεων πολλαπλών basic block που μπορεί να αποκρύπτονται σε μία γραμμή πηγαίου κώδικα. Στο παράδειγμά μας, στη γραμμή 6 του πηγαίου κώδικα εμφανίζεται μία εντολή “for” η οποία προφανώς αποτελείται από περισσότερα από ένα basic blocks, που το κάθε ένα έχει διαφορετικό πλήθος εκτελέσεων. Η εντολή “i=0” εκτελείται μία φορά, η έκφρασή “i<=100” αποτιμάται 102 φορές και η εντολή “ i++” εκτελείται 101 φορές. Το gcov όπως φαίνεται μας παρουσιάζει μόνο τον μεγαλύτερο αριθμό εκτελέσεων. Αν θέλουμε να πάρουμε πληροφορία σε επίπεδο basic block από το gcov μπορούμε να χρησιμοποιήσουμε τη σημαία “–a”. Τότε η έξοδος του gcov θα μοιάζει με τη παρακάτω (Κώδικας 3.3).

```
        -:    0:Source:test.c
        -:    0:Graph:test.gcno
        -:    0:Data:test.gcda
        -:    0:Runs:1
        -:    0:Programs:1
        1:    1:int main(){
        -:    2:    int i,j,k;
        1:    3:    j=1;k=1;
        1:    4:    if(j==0)
        1:    4-block  0
    #####:    5:        k=0;
    $$$$$:    5-block  0
      102:    6:    for(i=0;i<=100;i++){
        1:    6-block  0
      101:    6-block  1
      102:    6-block  2
      101:    7:        j+=i;
      101:    8:        k+=j;
        -:    9:    }
        1:   10:    return j;
        1:   10-block  0
        -:   11:}
```

Κώδικας 3.3: Έξοδος του gcov με πληροφορία σε επίπεδο basic block

Όπως φαίνεται (Κώδικας 3.3) με τη χρήση της σημαίας "-a" το gcov παράγει επιπλέον γραμμές που μας δίνουν πληροφορία για το πλήθος εκτελέσεων των διακριτών basic block για κάθε μία «μη τετριμμένη» γραμμή πηγαίου κώδικα. Σε αυτές τις γραμμές ο διαχωρισμός της δεύτερης και της τρίτης στήλης γίνεται με τον χαρακτήρα '-' αντί του ':'. Η σειρά που εμφανίζονται οι γραμμές αντιστοιχούν στη σειρά που εμφανίζονται τα basic blocks μετά την «επέκταση» της γραμμής. Εδώ η σήμανση $$$$$ έχει την ίδια έννοια με τη σήμανση ##### (καμία εκτέλεση) αλλά αναφέρεται σε basic block αντί για γραμμή κώδικα.

Όπως έχει ήδη αναφερθεί, στην ενότητα 3.5 θα εξηγήσουμε με ποιο τρόπο μπορούμε να εξασφαλίσουμε ότι κάθε μία γραμμή του πηγαίου κώδικα δεν εμπεριέχει περισσότερα από ένα basic blocks. Έτσι η δυνατότητα που παρέχει το gcov να δίνει πληροφορίες σε επίπεδο basic block δεν είναι απαραίτητη. Θα χρησιμοποιηθεί όμως επικουρικά όπως θα δούμε στην ενότητα 0.

Με τη χρήση του gcov, λοιπόν, μπορούμε να συλλέξουμε πληροφορία για το πλήθος εκτελέσεων κάθε γραμμής πηγαίου κώδικα της εφαρμογής μας. Στην επόμενη ενότητα θα δούμε πως μπορούμε να αντιστοιχίσουμε τις εντολές μηχανής με τις γραμμές πηγαίου κώδικα και κατ' επέκταση να αντιστοιχίσουμε και τους αριθμούς εκτελέσεων.

3.4 Αντιστοίχιση του κώδικα μηχανής με τον πηγαίο κώδικα με χρήση του format DWARF

Όπως αναφέρθηκε στην ενότητα 3.2 η προτεινόμενη μέθοδος αποτελείται από 4 βασικά βήματα εκ' των οποίων το βήμα 2 είναι η αντιστοίχιση κάθε μίας εντολής μηχανής του target επεξεργαστή με τη γραμμή πηγαίου κώδικα στην οποία αντιστοιχεί. Αυτή η απαίτηση έχει υπάρξει και στο παρελθόν για διαφορετικούς λόγους. Ο σημαντικότερος είναι η αποσφαλμάτωση του πηγαίου κώδικα (debugging). Πολλές φορές όταν κάνουμε αποσφαλμάτωση σε μία εφαρμογή χρησιμοποιούμε κάποιο εργαλείο αποσφαλμάτωσης (debugger), το οποίο εκτελεί την εφαρμογή μας σε ένα ελεγχόμενο περιβάλλον και μας δίνει λεπτομερείς πληροφορίες για την εκτέλεση της. Σε αυτές τις περιπτώσεις είναι πολύ χρήσιμο ο debugger να γνωρίζει την αντιστοίχιση μεταξύ των εντολών μηχανής και του πηγαίου κώδικα, ώστε αν εντοπισθεί κάποιο σφάλμα να μπορεί να ενημερώσει τον προγραμματιστή για τη πηγή του σφάλματος στον πηγαίο κώδικα. Γι αυτόν το σκοπό έχουν αναπτυχθεί διάφορα format με τα οποία η ζητούμενη πληροφορία (και άλλες χρήσιμες πληροφορίες) μπορεί να ενσωματωθεί στο εκτελέσιμο αρχείο της εφαρμογής. Ένα πολύ διαδεδομένο format είναι το DWARF [2] το οποίο υποστηρίζεται και από τα gnu εργαλεία.

Το format DWARF είναι άμεσα συνδεδεμένο με τον τύπο αρχείων ELF (Executable and Linkable Format [30]) όμως είναι σχεδιασμένο ώστε να είναι ανεξάρτητο από τον τύπο αρχείου του εκτελέσιμου. Αποτελείται από διάφορες δομές δεδομένων για την αναπαράσταση των διαφόρων στοιχείων του πηγαίου κώδικα (μεταβλητές, συναρτήσεις κτλ.) για ποικίλες γλώσσες προγραμματισμού. Το στοιχείο του DWARF format, όμως, που θα φανεί χρήσιμο σε αυτή την εργασία είναι ο πίνακας που αντιστοιχεί τις εντολές του κώδικα μηχανής με της γραμμές του πηγαίου κώδικα. Ένας εύκολος τρόπος να διαβάσουμε αυτή την αντιστοίχηση από ένα αρχείο

```
a.exe:     file format elf-i386

Decoded dump of debug contents of section .debug_line:

CU: test.c:
File name                            Line number    Starting address
test.c                                         1            0x4013c0
test.c                                         1            0x4013c9
test.c                                         3            0x4013ce
test.c                                         4            0x4013de
test.c                                         5            0x4013e5
test.c                                         6            0x4013ed
test.c                                         7            0x4013f7
test.c                                         8            0x4013ff
test.c                                         6            0x401407
test.c                                         6            0x40140b
test.c                                        10            0x401412
test.c                                        11            0x401416
```

Σχήμα 3.2: Έξοδος της εντολής objdump -WL

ELF που περιέχει πληροφορίες αποσφαλμάτωσης στο format DWARF, είναι η χρήση του εργαλείου objdump (μέλος της σουίτας GNU Binutils [31]). Στο Σχήμα 3.2 βλέπουμε μία τυπική έξοδο της εντολής objdump –WL με την οποία ζητούμε από το εργαλείο objdump να μας εμφανίσει την αντιστοιχία του κώδικα μηχανής και των γραμμών πηγαίου κώδικα για το εκτελέσιμο που παράχθηκε από τον παράδειγμα κώδικα που χρησιμοποιήσαμε στην προηγούμενη ενότητα (Κώδικας 3.1).

Όπως βλέπουμε στο παραπάνω παράδειγμα (Σχήμα 3.2) η έξοδος του objdump μας ενημερώνει για τον τύπο του εκτελέσιμου αρχείου (εδώ elf-i386) και στη συνέχεια μας εμφανίζει την αντιστοιχία του κώδικα μηχανής με τον πηγαίο κώδικα για κάθε ξεχωριστό αρχείο πηγαίου κώδικα (εδώ έχουμε μόνο ένα). Για κάθε αρχείο παρουσιάζονται τρεις στήλες. Η πρώτη στήλη περιέχει το όνομα του αρχείου, η δεύτερη τους αριθμούς γραμμών πηγαίου κώδικα και η τρίτη διευθύνσεις του κώδικα μηχανής. Κάθε νέα γραμμή στη έξοδο του objdump υποδηλώνει ότι όλες οι εντολές που περιέχονται στο φάσμα διευθύνσεων που ξεκινά από τη διεύθυνση που αναγράφεται στη τρέχουσα γραμμή (τρίτη στήλη) και τελειώνει πριν τη διεύθυνση της επόμενης γραμμής προέρχονται από την γραμμή πηγαίου κώδικα που εμφανίζεται στη τρέχουσα γραμμή (δεύτερη στήλη).

Η αντιστοίχηση μεταξύ του κώδικα μηχανής και του πηγαίου κώδικα παράγεται φυσικά από τον cross compiler (για τον gcc πρέπει να δοθεί η παράμετρος -g) και μεταφέρεται από αυτόν στον assembler και στη συνέχεια στον linker ώστε να υπάρχει και στο τελικό εκτελέσιμο αρχείο. Ο cross compiler μεταδίδει την πληροφορία στον assembler μέσω κάποιων εξειδικευμένων directives. Στη περίπτωση του GCC (GNU Compiler Collection) [3] τα directives αυτά είναι τα “.file” και “.loc”. Η σύνταξη των δύο αυτών directives δίνεται συνοπτικά παρακάτω (Πίνακας 3.2).

Με το directive “.file” ο compiler συνδέει το όνομα ενός αρχείου πηγαίου κώδικα με ένα μοναδικό αριθμό (θετικό ακέραιο). Από την εμφάνιση ενός directive “.file” και ύστερα, τα directives “.loc” θα μπορούν να χρησιμοποιούν τον αριθμό αυτό για να αναφέρονται στο συγκεκριμένο αρχείο.

Με το directive “.loc” δηλώνεται ότι όλες οι εντολές μηχανής που περιέχονται μεταξύ αυτού του directive και του επόμενου “.loc” directive προέρχονται από μία συγκεκριμένη γραμμή ενός συγκεκριμένου αρχείου (του οποίου το id δίνεται).

Παρακάτω (Κώδικας 3.4), δίνεται η έξοδος του GCC (assembly κώδικας) για τον ενδεικτικό πηγαίο κώδικα που χρησιμοποιούμε (Κώδικας 3.1) με τονισμένα τα directives “.file” και “.loc”. Στον compiler δόθηκαν οι παράμετροι –g –dA και –dp με τις οποίες ενεργοποιούμε τη παραγωγή πληροφορίας debugging σε μορφή DWARF καθώς και κάποιων άλλων χρήσιμων πληροφοριών. Στην έξοδο του GCC φαίνεται ο τρόπος που χρησιμοποιούνται τα directives “.file” και “.loc” και διακρίνεται και η ύπαρξη κάποιων επιπλέον πληροφοριών που αφορούν την έναρξη και λήξη των basic blocks καθώς και το μέγεθος των εντολών.

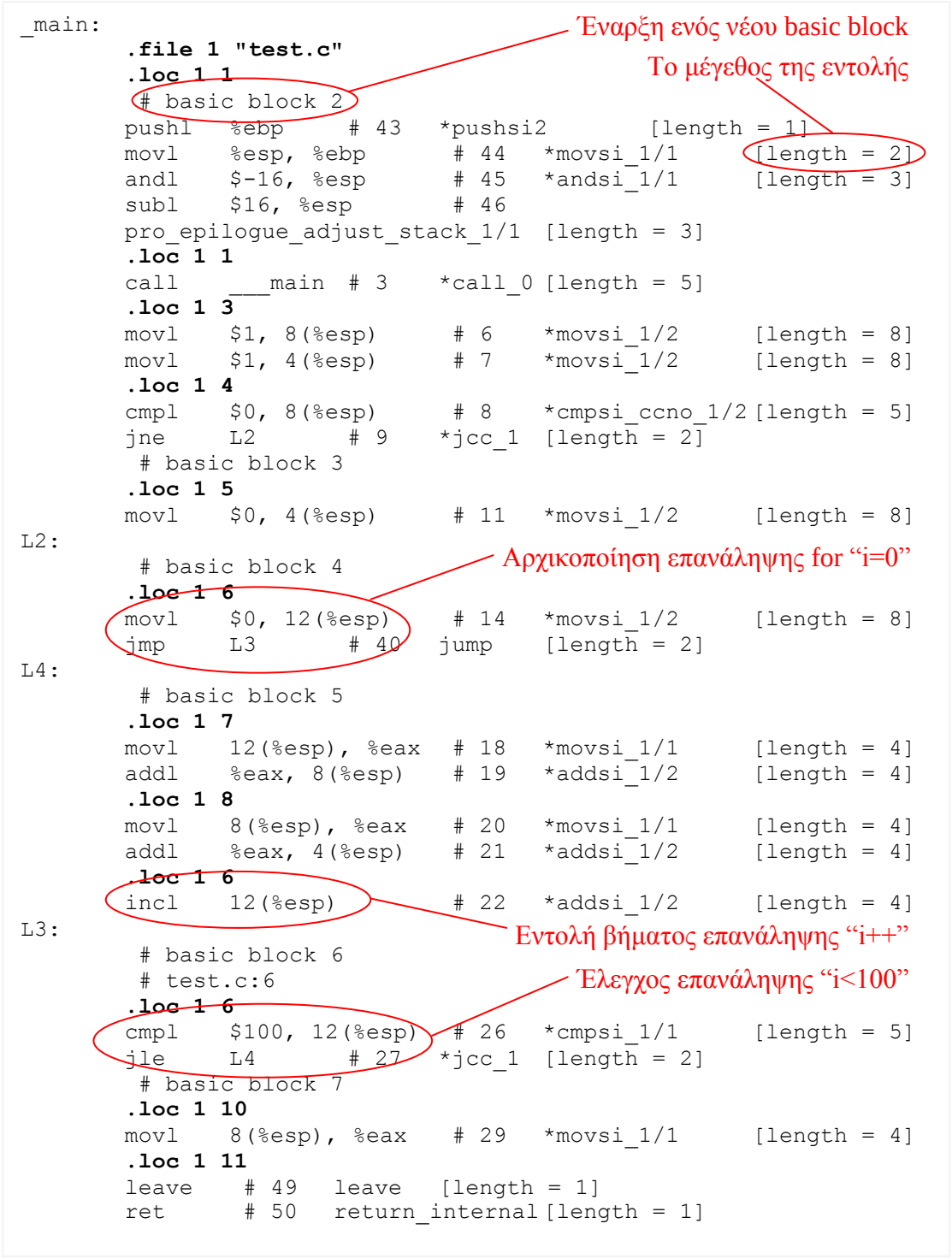
DIRECTIVE	ΠΑΡΆΜΕΤΡΟΣ	ΕΠΕΞΉΓΗΣΗ
.file <file_id> <filename>	<file_id>:	Ένας μοναδικός αριθμός id που αντιστοιχίζεται στο αρχείο με όνομα <filename>.
	<filename>:	Το όνομα του αρχείου που αντιστοιχίζεται στον αριθμό id <file_id>.
.loc <file_id> <file_line>	<file_id>:	Το id ενός δηλωμένου με το directive .file αρχείου.
	<file_line>:	Ο αριθμός μίας γραμμής στο αρχείο πηγαίου κώδικα με id <file_id>.

Πίνακας 3.2: Η σύνταξη των directives .file και .loc του GCC

3.5 Προεπεξεργασία πηγαίου κώδικα

Στις μέχρι τώρα περιγραφές έχει μείνει σκοτεινό ένα σημείο: πως διασφαλίζεται ότι μία γραμμή κώδικα ικανοποιεί το κριτήριο 1 για τη δομική μονάδα πηγαίου κώδικα; Πως δηλαδή είμαστε βέβαιοι ότι μία γραμμή κώδικα δεν εμπεριέχει πολλαπλά basic blocks με διαφορετικούς αριθμούς εκτελέσεων;

Δυστυχώς το κριτήριο 1 δεν ικανοποιείται σχεδόν ποτέ για συνηθισμένα προγράμματα. Ας μελετήσουμε το δικό μας παράδειγμα κώδικα (Κώδικας 3.1). Όπως φαίνεται από τα ".loc" directives στην έξοδο του GCC (Κώδικας 3.4) η εντολή αρχικοποίησης, η εντολή ελέγχου και η εντολή βήματος της for επανάληψης (γραμμή 6) δηλώνεται ότι προέρχονται από τη γραμμή 6 χωρίς όμως να υπάρχει πληροφορία για το basic block από το οποίο προέρχονται. Έτσι δεν μπορούμε να αντιστοιχίσουμε τις εντολές με τους σωστούς αριθμούς εκτελέσεων.




```
_main:
        .file 1 "test.c"
        .loc 1 1
         # basic block 2
        pushl   %ebp      # 43   *pushsi2        [length = 1]
        movl    %esp, %ebp      # 44   *movsi_1/1      [length = 2]
        andl    $-16, %esp      # 45   *andsi_1/1      [length = 3]
        subl    $16, %esp       # 46
        pro_epilogue_adjust_stack_1/1  [length = 3]
        .loc 1 1
        call    ___main  # 3    *call_0 [length = 5]
        .loc 1 3
        movl    $1, 8(%esp)     # 6    *movsi_1/2      [length = 8]
        movl    $1, 4(%esp)     # 7    *movsi_1/2      [length = 8]
        .loc 1 4
        cmpl    $0, 8(%esp)     # 8    *cmpsi_ccno_1/2 [length = 5]
        jne     L2       # 9    *jcc_1  [length = 2]
         # basic block 3
        .loc 1 5
        movl    $0, 4(%esp)     # 11   *movsi_1/2      [length = 8]
L2:
         # basic block 4
        .loc 1 6
        movl    $0, 12(%esp)    # 14   *movsi_1/2      [length = 8]
        jmp     L3       # 40   jump    [length = 2]
L4:
         # basic block 5
        .loc 1 7
        movl    12(%esp), %eax  # 18   *movsi_1/1      [length = 4]
        addl    %eax, 8(%esp)   # 19   *addsi_1/2      [length = 4]
        .loc 1 8
        movl    8(%esp), %eax   # 20   *movsi_1/1      [length = 4]
        addl    %eax, 4(%esp)   # 21   *addsi_1/2      [length = 4]
        .loc 1 6
        incl    12(%esp)        # 22   *addsi_1/2      [length = 4]
L3:
         # basic block 6
         # test.c:6
        .loc 1 6
        cmpl    $100, 12(%esp)  # 26   *cmpsi_1/1      [length = 5]
        jle     L4       # 27   *jcc_1  [length = 2]
         # basic block 7
        .loc 1 10
        movl    8(%esp), %eax   # 29   *movsi_1/1      [length = 4]
        .loc 1 11
        leave    # 49   leave   [length = 1]
        ret      # 50   return_internal [length = 1]
```

Κώδικας 3.4: Assembly κώδικας του cross compiler (GCC) με ενεργοποιημένες τις παραμέτρους –g, –dA και -dp

Οι επαναλήψεις, δεν είναι οι μόνες εντολές που παρουσιάζουν αυτό το πρόβλημα. Εντολές επιλογής (if) με σύνθετες συνθήκες παρουσιάζουν επίσης το ίδιο πρόβλημα. Τέτοιες εντολές καταλήγουν επίσης στη δημιουργία πολλαπλών basic blocks για την αποτίμηση των λογικών εκφράσεων. Παρακάτω (Κώδικας 3.5) δίνεται ένα παράδειγμα μίας σύνθετης εντολής επιλογής και ύστερα (Κώδικας 3.6) παρουσιάζεται ο τρόπος που επεκτείνεται αυτή η εντολή από το front end του

μεταγλωττιστή. Η επέκταση του κώδικα πάρθηκε από τον GCC με χρήση της σημαίας –fdump-tree-cfg.

```
if(d=((a=((b|c)||(d>2)))&&(f==e)))
    return 1;
```

Κώδικας 3.5: Ένα παράδειγμα εντολής if με σύνθετη συνθήκη.

```
  D.1196 = b | c;
  if (D.1196 != 0)
      goto <D.1193>;
  if (d > 2)
      goto <D.1193>;
  else
      goto <D.1194>;
  <D.1193>:;
  iftmp.2 = 1;
  goto <D.1195>;
  <D.1194>:;
  iftmp.2 = 0;
  <D.1195>:;
  a = iftmp.2;
  if (a == 0)
      goto <D.1190>;
  if (f != e)
      goto <D.1190>;
  iftmp.1 = 1;
  goto <D.1191>;
  <D.1190>:;
  iftmp.1 = 0;
  <D.1191>:;
  d = iftmp.1;
  if (d != 0) {
      D.1197 = 1;
      return D.1197;
  }
```

Κώδικας 3.6: Επέκταση της σύνθετης εντολής if (Κώδικας 3.5)

Παρατηρούμε σε πόσο σύνθετο κώδικα επεκτείνεται μία μόνο γραμμή του πηγαίου κώδικα. Δημιουργούνται νέες εντολές διακλάδωσης και χρησιμοποιούνται νέες προσωρινές μεταβλητές. Είναι προφανές ότι σε αυτή την εντολή επιλογής (Κώδικας 3.5) δεν μπορεί να αντιστοιχηθεί ένας μόνο αριθμός εκτελέσεων.

Άλλα παραδείγματα που προκαλούν ανάλογά προβλήματα είναι οι σύνθετες λογικές εκφράσεις (Κώδικας 3.7) και οι υπό συνθήκη εκφράσεις (Κώδικας 3.8).

```
a = b || c && d;
```

Κώδικας 3.7: Ανάθεση λογικής έκφρασης.

```
a = b==c ? d : e ;
```

Κώδικας 3.8: Ανάθεση μίας υπό συνθήκη έκφρασης.

Ο τρόπος που αντιμετωπίζεται το πρόβλημα, στα πλαίσια της διπλωματικής εργασίας, είναι η επεξεργασία του πηγαίου κώδικα και η μεταφορά του σε ένα υποσύνολο της γλώσσας C που δεν περιέχει εντολές επιλογής με σύνθετες συνθήκες (Κώδικας 3.5) ούτε δομές επανάληψης. Οι συγκεκριμένες δομές υλοποιούνται με τη χρήση απλών εντολών if (με απλές συνθήκες) και εντολών goto. Επίσης, μετά την συντακτική επεξεργασία του κώδικα εισάγονται αλλαγές γραμμών ανάμεσα από τις εντολές ώστε να διασφαλιστεί ότι κάθε γραμμή περιέχει μία μόνο εντολή C η οποία είναι αρκετά απλή για να καταλήξει σε ένα μόνο basic block.

Ωστόσο, ο τρόπος που γίνεται η συντακτική μετατροπή του κώδικα πρέπει να είναι τέτοιος ώστε να μην αλλοιώνει τον τελικό κώδικα μηχανής. Μία τροποποίηση του πηγαίου κώδικα μπορεί να οδηγήσει σε διαφοροποιημένο τελικό κώδικα ακόμα και αν ο τροποποιημένος πηγαίος κώδικας είναι πανομοιότυπος σημασιολογικά με τον αρχικό κώδικα.

Μετά από εξερεύνηση διάφορων συντακτικών μετασχηματισμών και με χρήση του compiler gcc βρέθηκε ότι οι μετασχηματισμοί που αλλοιώνουν λιγότερο τον τελικό κώδικα είναι αυτοί που παρουσιάζονται παρακάτω (Σχήμα 3.2 έως Σχήμα 3.7).

Οι τρείς πρώτοι μετασχηματισμοί (Σχήμα 3.2 έως Σχήμα 3.5) αφορούν τους βρόχους επανάληψης, ενώ οι επόμενοι (Σχήμα 3.6 και Σχήμα 3.7) αφορούν μετασχηματισμούς των εντολών επιλογής. Όπως φαίνεται και στα σχήματα οι μετασχηματισμοί απαιτούν τη δημιουργία labels τα οποία πρέπει να διασφαλιστεί ότι έχουν μοναδικό όνομα.

Οι συντακτικοί αυτοί μετασχηματισμοί εφαρμόζονται αναδρομικά. Αν π.χ. ένας βρόχος do-while που έχει μία σύνθετη συνθήκη ελέγχου επεκταθεί σύμφωνα με τον μετασχηματισμό που παρουσιάζεται στο Σχήμα 3.3, θα δημιουργηθεί μία νέα σύνθετη εντολή επιλογής η οποία επίσης θα μετασχηματιστεί με τον αντίστοιχο μετασχηματισμό. Στους μετασχηματισμούς των εντολών επιλογής τα a, b, ..., z μπορεί να είναι σύνθετες εκφράσεις οι οποίες επίσης θα επεκταθούν αναδρομικά.

Φυσικά αν μέσα σε έναν βρόχο επανάληψης υπάρχει κάποια εντολή break ή continue θα πρέπει να αντικατασταθεί με μία εντολή “goto” προς το κατάλληλο σημείο του παραγόμενου κώδικα. Τέλος αν στον μετασχηματισμό εντολών επιλογής με or έκφραση για συνθήκη (Σχήμα 3.6) δεν υπάρχει το τμήμα “else” στον αρχικό κώδικα, παραλείπονται τα “goto <end_label>”, “<false_stmt>” και “<end_label>”. Επίσης αν δεν υπάρχει το τμήμα “else” στον αρχικό κώδικα του μετασχηματισμού εντολών επιλογής με and έκφραση για συνθήκη (Σχήμα 3.7), παραλείπεται το “<false_stmt>”.

Αρχικός κώδικας:

```
do
    <loop_content>
while (<guard>);
```

Τροποποιημένος κώδικας:

```
<start_label>:
<loop_content>
<check_label>:
if (<guard>)
    goto <start_label>;
<end_label>:
```

Σχήμα 3.3: Συντακτικός μετασχηματισμός των βρόχων do-while

Αρχικός κώδικας:

```
while(<guard>)
    <loop_content>
```

Τροποποιημένος κώδικας:

```
goto <check_label>;
<start_label>:
<loop_content>
<check_label>:
if (<guard>)
    goto <start_label>;
<end label>:
```

Σχήμα 3.4: Συντακτικός μετασχηματισμός των βρόχων while

Αρχικός κώδικας:

```
for(<init>;<guard>;<step>)
    <loop_content>
```

Τροποποιημένος κώδικας:

```
<init>;
goto <check_label>;
<start_label>:
<loop_content>
<step_label>:
<step>;
<check_label>:
if (<guard>)
    goto <start_label>;
<end_label>:
```

Σχήμα 3.5: Συντακτικός μετασχηματισμός των βρόχων for

Αρχικός κώδικας:

```
if(a||b||...||z)
    <true_stmt>
else
    <false_stmt>
```

Τροποποιημένος κώδικας:

```
if(a)
    goto <true_label>;
if(b)
    goto <true_label>;
.
.
.
if(!z)
    goto <false_label>;
<true_label>:
<true_stmt>
goto <end_label>
<false_label>:
<false_stmt>
<end_label>:
;
```

Σχήμα 3.6: Συντακτικός μετασχηματισμός εντολών επιλογής (if) με έκφραση or για συνθήκη

Αρχικός κώδικας:

```
if(a&&b&&...&&z)
    <true_stmt>
else
    <false_stmt>
```

Τροποποιημένος κώδικας:

```
if(!a)
    goto <false_label>;
if(!b)
    goto <false_label>;
.
.
.
if(z)
    goto <true_label>;
<false_label>:
<false_stmt>
goto <end_label>
<true_label>:
<true_stmt>
<end_label>:
;
```

Σχήμα 3.7: Συντακτικός μετασχηματισμός εντολών επιλογής (if) με έκφραση and για συνθήκη

Φυσικά πριν τους συντακτικούς μετασχηματισμούς ο κώδικας θα πρέπει να έχει πρώτα περάσει από τον προεπεξεργαστή της C. Με αυτόν τον τρόπο διασφαλίζεται ότι θα έχουν επεκταθεί όλες οι μακροεντολές που επίσης μπορεί να αποκρύπτουν πολλαπλά basic blocks σε μία γραμμή.

Παρακάτω (Κώδικας 3.9) δίνεται το παράδειγμα πηγαίου κώδικα της ενότητας 3.3 (Κώδικας 3.1) μετά τους συντακτικούς μετασχηματισμούς.

```
int main ( ) {
    int i , j , k ;
    j = 1 ;
    k = 1 ;
    {
        if ( j == 0 )
            k = 0 ;
    }
    {
        i = 0 ;
        goto __lbl_2__ ;
        __lbl_0__ :
        {
            j += i ;
            k += j ;
        }
        __lbl_1__ :
        i ++ ;
        __lbl_2__ :
        {
            if ( i <= 100 )
                goto __lbl_0__ ;
        }
        __lbl_3__ :
        ;
    }
    return j ;
}
```

Κώδικας 3.9: Το παράδειγμα πηγαίου κώδικα (Κώδικας 3.1) μετά τους συντακτικούς μετασχηματισμούς.

block. Έτσι είναι βέβαιο ότι μπορούμε να αντιστοιχίσουμε έναν αριθμό εκτελέσεων σε κάθε μία γραμμή του πηγαίου κώδικα και κατ' επέκταση να αντιστοιχίσουμε έναν αριθμό μοναδικό εκτελέσεων σε κάθε basic block του τελικού κώδικα μηχανής (βλ. ενότητα 3.4).

3.6 Παραγωγή του Γ.Ρ.Ε. και χαρακτηρισμός των αλμάτων

Στις προηγούμενες ενότητες δείξαμε πως μπορούμε να αντιστοιχίσουμε έναν αριθμό εκτελέσεων σε κάθε εντολή του τελικού assembly κώδικα, χωρίς να κάνουμε

κάποιου είδους προσομοίωση. Ωστόσο, για να προχωρήσουμε σε περεταίρω ανάλυση της επίδοσης (π.χ. επίδοση της μνήμης cache) είναι απαραίτητο να έχουμε μία αναπαράσταση του target Ε.Γ.Ρ.Ε. Η ανάγκη αυτή δημιουργεί δύο νέα προβλήματα. Το πρώτο αφορά τη κατασκευή του target Γ.Ρ.Ε. και το δεύτερο την επισημείωση των ακμών του Γ.Ρ.Ε. με το πλήθος των εκτελέσεών τους (για τα basic blocks η λύση έχει δοθεί στην ενότητα 3.4). Στις δύο επόμενες ενότητες περιγράφεται η προσπάθεια που έγινε να λυθούν αυτά τα δύο προβλήματα με τρόπο που να κάνει εύκολη την εφαρμογή της συνολικής μεθόδου ανεξάρτητα από την αρχιτεκτονική συνόλου εντολών (ISA) του ενσωματωμένου συστήματος.

3.6.1 Παραγωγή του Γ.Ρ.Ε. για διαφορετικές αρχιτεκτονικές συνόλου εντολών

Ο προφανής τρόπος για τη δημιουργία του target Γ.Ρ.Ε. είναι να δημιουργηθεί ένας parser για την γλώσσα μηχανής του ενσωματωμένου συστήματος. Ο parser αυτός δεν είναι αναγκαίο να είναι πλήρης. Αρκεί να αναγνωρίζονται οι εντολές διακλάδωσης ώστε να μπορεί να εξαχθεί το σχήμα του Γ.Ρ.Ε.. Ωστόσο, αν θέλουμε η μέθοδός μας να είναι εφαρμόσιμη για πολλές διαφορετικές αρχιτεκτονικές συνόλου εντολών, θα πρέπει να κατασκευαστούν πολλοί διαφορετικοί parser.

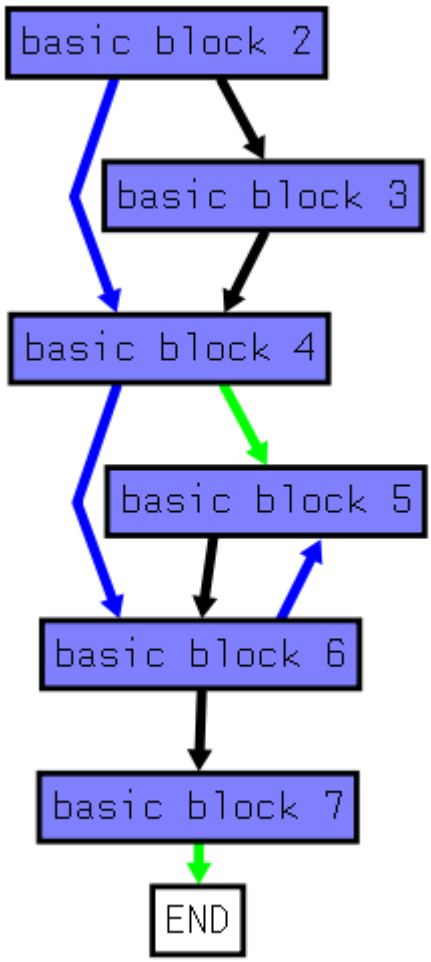
Στα πλαίσια της συγκεκριμένης εργασίας για την επίλυση του προβλήματος κατασκευής του target Γ.Ρ.Ε. εκμεταλλευτήκαμε κάποιες επιπλέον δυνατότητες του compiler GCC. Με τη χρήση αυτών των δυνατοτήτων εξαλείφεται η ανάγκη να κατασκευαστεί ένας ή παραπάνω parsers και δίνεται η δυνατότητα να εφαρμοστεί η μέθοδος σε διάφορες αρχιτεκτονικές συνόλου εντολών χωρίς να απαιτείται καμία τροποποίηση.

Το backend τμήμα του GCC χρησιμοποιεί για τη παραγωγή του τελικού κώδικα μία μορφή εσωτερικής αναπαράστασης που ονομάζεται RTL [32]. Η RTL αναπαράσταση υφίσταται πολλούς μετασχηματισμούς μέχρι να μετατραπεί στον τελικό κώδικα assembly. Ο GCC παρέχει τη δυνατότητα εξαγωγής του σχήματος του Γ.Ρ.Ε. για τα διάφορα στάδια του RTL κώδικα (για λόγους debugging) σε μία μορφή ανεξάρτητη της αρχιτεκτονικής συνόλου εντολών. Το τελευταίο στάδιο της RTL περιγραφής πριν τη παραγωγή του τελικού κώδικα έχει προφανώς ακριβώς τον ίδιο Γ.Ρ.Ε. με τον τελικό assembly κώδικα. Η ιδέα, λοιπόν, είναι αντί να κατασκευαστεί ένας parser για κάθε μία αρχιτεκτονική, να κατασκευαστεί ένας γενικευμένος parser που θα διαβάζει τα αρχεία αναπαράστασης της RTL που εξάγει ο GCC.

Μετά από πειράματα, βρέθηκε ότι το στάδιο από το οποίο μπορούμε να εξάγουμε την αναπαράσταση της RTL και αυτή να είναι πάντα πανομοιότυπη με τον τελικό κώδικα assembly είναι μετά τη στοίχιση (alignment) των εντολών της RTL αναπαράστασης. Για να πάρουμε μία αναπαράσταση αυτού του σταδίου θα πρέπει να δοθούν στον GCC οι παράμετροι –fdump-rtl-alignments και –dv. Με τις παραμέτρους αυτές ο GCC δημιουργεί δύο νέα αρχεία. Το ένα περιέχει τον RTL κώδικα αυτούσιο και το άλλο μία αναπαράσταση του Γ.Ρ.Ε. σε μία γενικευμένη γλώσσα αναπαράστασης γράφων που ονομάζεται gdl [33]. Από το δεύτερο αρχείο μπορεί να εξαχθεί ο target Γ.Ρ.Ε. και έπειτα να αντιστοιχίσουμε σε κάθε basic block

του το πλήθος εκτελέσεων των αντίστοιχων εντολών. Η αντιστοίχηση μεταξύ των basic block του τελικού assembly κώδικα και των basic block που εμφανίζονται στην gdl αναπαράσταση είναι πολύ εύκολη καθώς και στις δύο περιπτώσεις οι αντίστοιχες εντολές εμφανίζονται ακριβώς με την ίδια σειρά.

Παρακάτω (Σχήμα 3.8) βλέπουμε μία γραφική αναπαράσταση του gdl αρχείου που παράγει ο cross compiler για το παράδειγμα πηγαίου κώδικα (Κώδικας 3.1). Παρατηρήστε ότι η αρίθμηση των basic blocks είναι αυτή που χρησιμοποιείται και στον assembly κώδικα (Κώδικας 3.4).



Σχήμα 3.8: Γραφική αναπαράσταση του target Γ.Ρ.Ε. που παράγει ο cross compiler για το παράδειγμα πηγαίου κώδικα (Κώδικας 3.1)

3.6.2 Υπολογισμός των πλήθους εκτελέσεων των ακμών του Γ.Ρ.Ε.

Οι Ν. Κρούπης και Δ. Σούντρης [1], στηριζόμενοι στην υπόθεση ότι ο Γ.Ρ.Ε. του πηγαίου κώδικα είναι πανομοιότυπος με τον Γ.Ρ.Ε. του κώδικα μηχανής θεώρησαν ότι κάθε ακμή (άλμα) του target Γ.Ρ.Ε. μπορεί να αντιστοιχηθεί με μία ακμή του πηγαίου κώδικα. Στην παρούσα εργασία δεν θεωρούμε δεδομένο ότι ο target Γ.Ρ.Ε. είναι πανομοιότυπος με τον Γ.Ρ.Ε. του πηγαίου κώδικα. Έτσι πρέπει να βρεθεί ένας τρόπος να υπολογισθούν οι εκτελέσεις κάθε ακμής με βάση τους αριθμούς εκτελέσεων των basic blocks.

Για να προχωρήσουμε στην ανάλυση της μεθόδου που χρησιμοποιήθηκε σε αυτή την εργασία, θα πρέπει πρώτα να δώσουμε κάποιους ορισμούς συμβόλων που θα χρησιμοποιηθούν στη συνέχεια.

Έστω ότι αντιστοιχίζουμε σε κάθε basic block του Γ.Ρ.Ε. έναν φυσικό αριθμό i. Στον παρακάτω πίνακα (Πίνακας 3.3) ορίζουμε τα σύμβολα που θα χρησιμοποιήσουμε στην ανάλυση μας.

ΣΥΜΒΟΛΟ	**ΕΠΕΞΗΓΗΣΗ**
B:	Το σύνολο όλων των basic blocks που περιέχει ο Γ.Ρ.Ε. (δηλαδή ένα σύνολο φυσικών αριθμών).
e_i:	Ο αριθμός εκτελέσεων του basic block i.
$j_{i,n}$:	Ο αριθμός εκτελέσεων άλματος από το basic block i στο basic block n.
I_i:	Το σύνολο των basic blocks για τα οποία υπάρχει άλμα που ξεκινά από αυτά και καταλήγει στο basic block i.
O_i:	Το σύνολο των basic blocks για τα οποία υπάρχει άλμα που ξεκινά από το basic block i και καταλήγει σε αυτά.

Πίνακας 3.3: Ορισμός συμβόλων για την ανάλυση του Γ.Ρ.Ε.

Είναι βέβαιο ότι για κάθε κόμβο του Γ.Ρ.Ε. ισχύουν οι σχέσεις (3.1) και (3.2). Ισχύει, δηλαδή ότι για κάθε basic block ο συνολικός αριθμός εκτελέσεων των εισερχόμενων στο basic block αλμάτων είναι ίσος με τον αριθμό εκτελέσεων του basic block και τον συνολικό αριθμό εκτελέσεων των εξερχόμενων από το basic block αλμάτων.

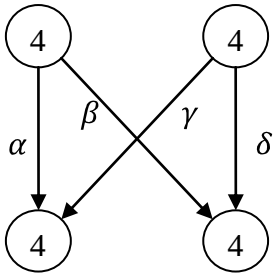
$$\sum_{n \in O_i} j_{i,n} = e_i \qquad \textbf{(3.1)}$$

$$\sum_{n \in I_i} j_{n,i} = e_i \qquad \textbf{(3.2)}$$

Επομένως, αν θέλουμε να υπολογίσουμε τους αριθμούς εκτελέσεων των αλμάτων του Γ.Ρ.Ε. θα πρέπει να επιλύσουμε το σύστημα (3.3).

$$\sum_{n \in O_i} j_{i,n} = e_i$$
$$\sum_{n \in I_i} j_{n,i} = e_i \qquad \forall i \in B \tag{3.3}$$

Το σύστημα (3.3), δυστυχώς δεν έχει πάντα μοναδική λύση. Αυτό σημαίνει ότι σε κάποιες περιπτώσεις ο ακριβής υπολογισμός των εκτελέσεων των ακμών του Γ.Ρ.Ε. είναι αδύνατος. Στο παρακάτω σχήμα (Σχήμα 3.9) βλέπουμε ένα παράδειγμα γράφου που παρουσιάζει το παραπάνω πρόβλημα.



Σχήμα 3.9: Παράδειγμα γράφου που δεν μπορούν να υπολογιστούν οι εκτελέσεις των ακμών του

Το σύστημα που πρέπει να επιλυθεί για τον υπολογισμό των α, β, γ και δ του παραπάνω σχήματος (Σχήμα 3.9) είναι το παρακάτω (3.4). Αν προσπαθήσετε να λύσετε το σύστημα (3.4) θα δείτε ότι η λύση του δεν είναι μοναδική.

$$\begin{aligned} \alpha + \beta &= 4 \\ \gamma + \delta &= 4 \\ \alpha + \gamma &= 4 \\ \beta + \delta &= 4 \end{aligned} \tag{3.4}$$

Τέτοιου είδους γράφοι, ευτυχώς, εμφανίζονται πολύ σπάνια σε πραγματικές εφαρμογές.

Στα πλαίσια αυτής της εργασίας αναπτύχθηκε ένας αλγόριθμος για τον γρήγορο υπολογισμό των εκτελέσεων ακμών. Ο αλγόριθμος που αναπτύχθηκε δεν κάνει αυστηρή αλγεβρική επίλυση του συστήματος και γι αυτό είναι αρκετά ταχύς. Στηρίζεται στην ιδέα να υπολογιστούν σταδιακά οι αριθμοί εκτελέσεων των ακμών ξεκινώντας από αυτούς που είναι πολύ εύκολο να υπολογιστούν.

Ο αριθμός εκτελέσεων μίας ακμής (άλματος) του Γ.Ρ.Ε. μπορεί να υπολογιστεί άμεσα αν ισχύει ένα ή περισσότερα από τα παρακάτω κριτήρια.

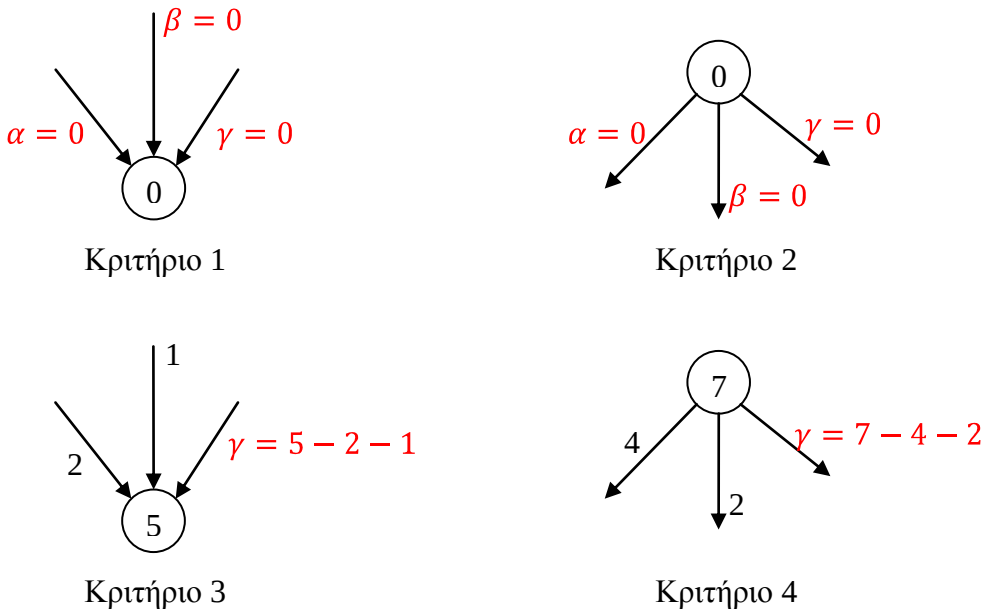
1. Το basic block στο οποίο καταλήγει η ακμή έχει αριθμό εκτελέσεων 0. Τότε είναι σίγουρο ότι και το άλμα δεν έχει εκτελεστεί καμία φορά.

2. Το basic block από το οποίο ξεκινά η ακμή έχει αριθμό εκτελέσεων 0. Τότε είναι σίγουρο ότι και το άλμα δεν έχει εκτελεστεί καμία φορά.

3. Η ακμή είναι η μόνη της οποίας δεν έχουν υπολογιστεί οι εκτελέσεις και καταλήγει στο συγκεκριμένο basic block. Τότε ο αριθμός εκτελέσεών της είναι ίσος με τον αριθμό εκτελέσεων του basic block μείων τον συνολικό αριθμό εκτελέσεων των υπόλοιπων ακμών που καταλήγουν σε αυτό.

4. Η ακμή είναι η μόνη της οποίας δεν έχουν υπολογιστεί οι εκτελέσεις και ξεκινά από το συγκεκριμένο basic block. Τότε ο αριθμός εκτελέσεών της είναι ίσος με τον αριθμό εκτελέσεων του basic block μείων τον συνολικό αριθμό εκτελέσεων των υπόλοιπων ακμών που ξεκινάν από αυτό.

Παρακάτω (Σχήμα 3.10) δίνεται από ένα παράδειγμα για κάθε κριτήριο.



Σχήμα 3.10: Κριτήρια για τον άμεσο υπολογισμό εκτελέσεων των ακμών.

Κάθε φορά που υπολογίζεται ο αριθμός εκτελέσεων μίας ακμής μπορούν να εμφανιστούν νέες ακμές που ικανοποιούν κάποιο από τα κριτήρια. Έτσι σταδιακά μπορούν να υπολογιστούν οι αριθμοί εκτελέσεων όλων των ακμών. Με βάση αυτή τη λογική αναπτύχθηκε ο αλγόριθμος για τον υπολογισμό των εκτελέσεων των αλμάτων του Γ.Ρ.Ε. (Κώδικας 3.10).

Φυσικά ο αλγόριθμος αυτός δεν είναι βέβαιο ότι μπορεί να υπολογίσει σε κάθε περίπτωση τους αριθμούς εκτελέσεων όλων των ακμών του Γ.Ρ.Ε για τους λόγους που αναλύσαμε νωρίτερα (ύπαρξη πολλαπλών πιθανών λύσεων των αλγεβρικών συστημάτων εξισώσεων). Οι περιπτώσεις, ωστόσο, αυτές είναι εξαιρετικά σπάνιες. Ενδεικτικά, σε 1748 διαφορετικές δοκιμές που έγιναν (βλ. κεφάλαιο 5), ο αλγόριθμος δεν απέτυχε ούτε σε μία. Στη περίπτωση, ωστόσο, που ο αλγόριθμος αποτυγχάνει, τότε οι ακμές για τις οποίες υπάρχουν πολλαπλές λύσεις χαρακτηρίζονται με βάση κάποιον ευριστικό-προσεγγιστικό αλγόριθμο [34]. Ο ευριστικός αλγόριθμος που χρησιμοποιεί η μέθοδος υποθέτει ότι για δύο οι περισσότερες ακμές εξερχόμενες από

έναν κόμβο ο αριθμός εκτελέσεών τους είναι ίσος με τον αριθμό εκτελέσεων του κόμβου διά το πλήθος των εξερχόμενων ακμών. Έτσι όταν ο κύριος αλγόριθμος (Κώδικας 3.10) δεν μπορεί να συνεχίσει, χρησιμοποιείται ο παραπάνω βοηθητικός αλγόριθμος ώστε να μπορέσει να συνεχίσει την εκτέλεση του ο κύριος αλγόριθμος.

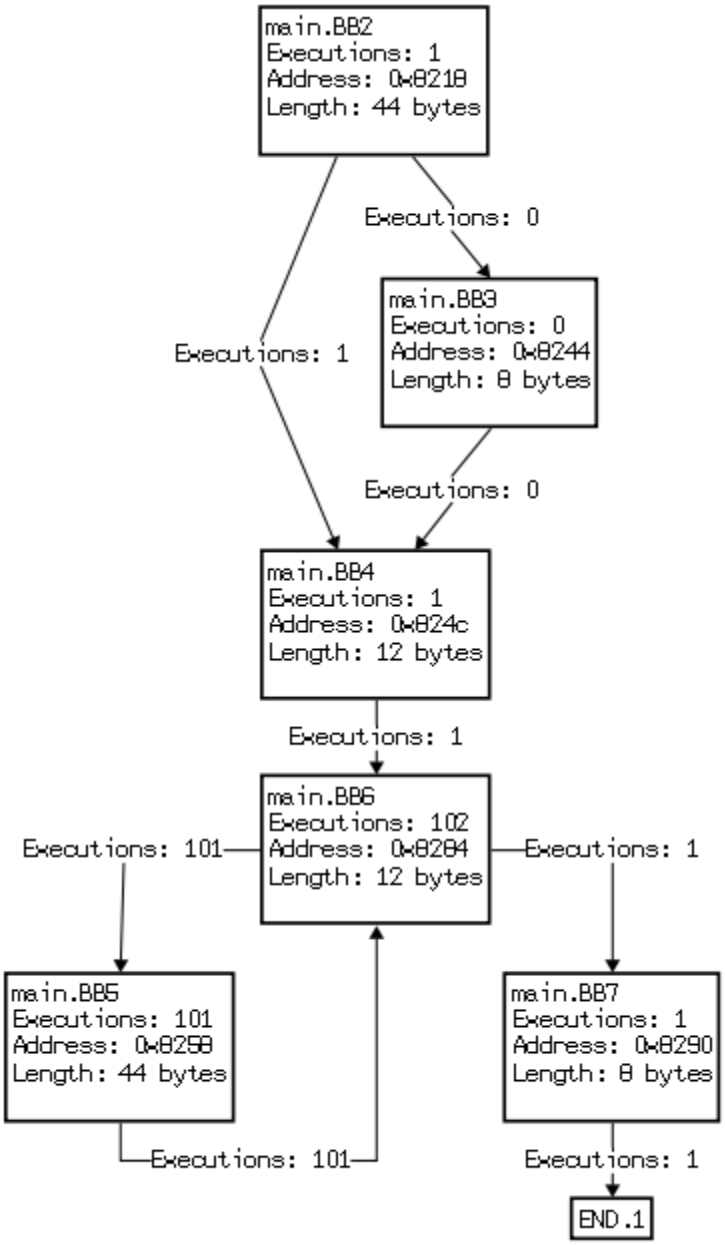
3.6.3 Επισημείωση του target Γ.Ρ.Ε. με πληροφορίες για μέγεθος και τις διευθύνσεις των basic blocks.

Ο target Ε.Γ.Ρ.Ε. θα πρέπει να περιέχει πληροφορίες για το μέγεθος και τη διεύθυνση του κάθε basic block. Οι πληροφορίες αυτές μπορούν να εξαχθούν από το τελικό αρχείο assembly που παράγει ο cross compiler. Όπως φαίνεται η χρήση ενεργοποίηση των παραμέτρων “-dA” και “-dp” έχει ως αποτέλεσμα ο cross compiler να υποσημειώνει κάθε assembly εντολή με το μέγεθός της σε byte (βλ. Κώδικας 3.4). Με βάση τα μεγέθη των εντολών μπορούν φυσικά να υπολογιστούν και οι σχετικές διευθύνσεις τους.

```
Calculate_Jump_Executions(CFG)
  for each jump j ∈ CFG.jumps
    j.executions ← UNINITIALISED
  for each basic block bb ∈ CFG.basic_blocks
    bb.in_jump_executions ← bb.executions
    bb.out_jump_executions ← bb.executions
    bb.uninit_in_jump_num ← bb.in_jump_num
    bb.uninit_out_jump_num ← bb.out_jump_num
  loop until no change occurs
    for each basic block bb ∈ CFG.basic_blocks:
     bb.uninit_in_jump_num = 1 ∨ bb.in_jump_executions = 0
      for each jump j ∈ bb.in_jumps: j.executions = UNINITIALISED
        j.executions ← bb.in_jump_executions
        origin_bb ← j.origin_basic_block
        origin_bb.uninit_out_jump_num ←
        origin_bb.uninit_out_jump_num − 1
        origin_bb.out_jump_executions ←
         origin_bb.out_jump_executions - j.executions
        bb.in_jump_executions ← 0
        bb.uninit_in_jump_num ← 0
    for each basic block bb ∈ CFG.basic_blocks:
     bb.uninit_out_jump_num = 1 ∨ bb.out_jump_executions = 0
      for each jump j ∈ bb.out_jumps:
       j.executions = UNINITIALISED
        j.executions ← bb.out_jump_executions
        target_bb ← j.origin_basic_block
        target_bb.uninit_in_jump_num ←
        target_bb.uninit_in_jump_num − 1
        target_bb.in_jump_executions ←
         target_bb.in_jump_executions - j.executions
        bb.out_jump_executions ← 0
        bb.uninit_out_jump_num ← 0
```

Κώδικας 3.10: Αλγόριθμος υπολογισμού των εκτελέσεων των αλμάτων του Γ.Ρ.Ε.

Παρακάτω (Σχήμα 3.11) φαίνεται μία γραφική αναπαράσταση από τον target Ε.Γ.Ρ.Ε που δημιούργησε η μέθοδος από τον πηγαίο κώδικα (Κώδικας 3.1) με χρήση cross compiler για την αρχιτεκτονική armv4.



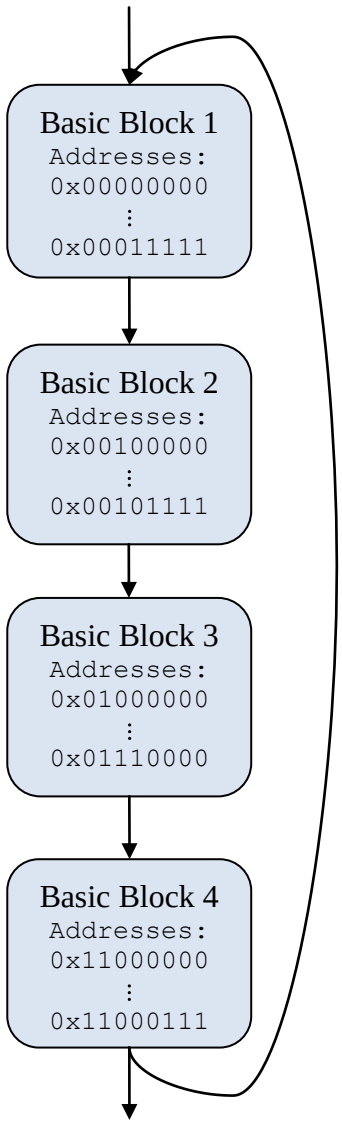
Σχήμα 3.11: Ο target Ε.Γ.Ρ.Ε. του παραδείγματος πηγαίου κώδικα (Κώδικας 3.1) για αρχιτεκτονική armv4

3.7 Εκτίμηση των αστοχιών σε έναν απλό βρόχο

Στις προηγούμενες ενότητες περιγράψαμε βήμα-βήμα τον τρόπο κατασκευής του target Ε.Γ.Ρ.Ε.. Σε αυτή και τις επόμενες ενότητες θα περιγράψουμε μία μέθοδο με την οποία, αναλύοντας τον target Ε.Γ.Ρ.Ε,. μπορούμε να βγάλουμε συμπεράσματα για την επίδοση της μνήμης cache. Συγκεκριμένα σε αυτή την ενότητα θα δείξουμε

πως μπορούν να υπολογιστούν τα cache misses που επιφέρει ένας απλός βρόχος με ανάλυση του Ε.Γ.Ρ.Ε. του. Η μέθοδος που παρουσιάζεται είναι στην ουσία μία γενίκευση των αναλυτικών τύπων που παρουσιάζονται στη μελέτη των Λιβέρη Ν. κτλ. [24].

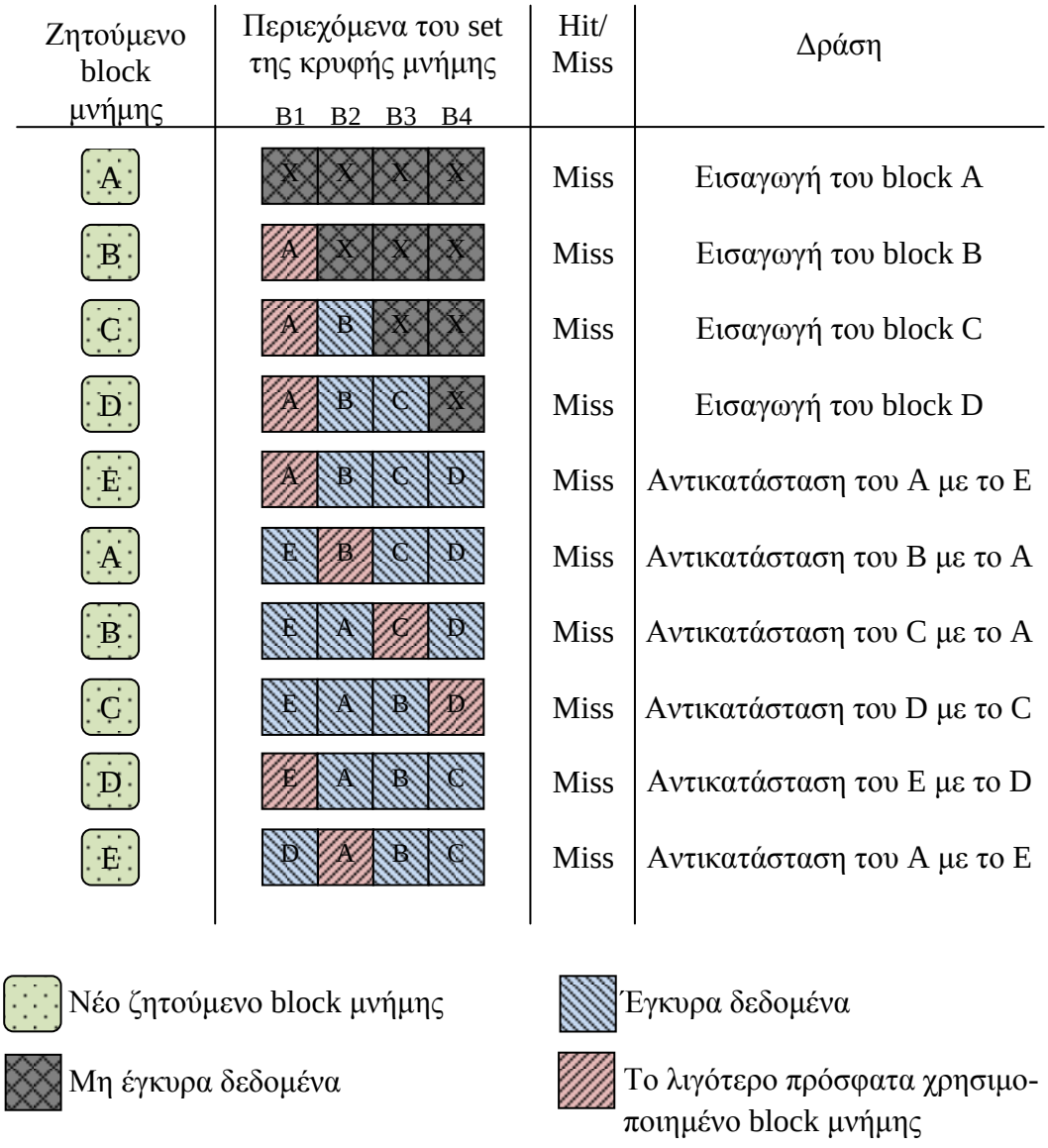
Αρχικά υποθέτουμε ότι έχουμε έναν απλό βρόχο στου οποίου το σώμα δεν εκτελείται καμία διακλάδωση. Τα basic block που εκτελούνται στο σώμα του βρόχου είναι τα ίδια σε κάθε επανάληψη και εκτελούνται σειριακά (αποτελούν μονοπάτι). Δεν είναι απαραίτητο τα basic blocks να έχουν διαδοχικές διευθύνσεις αλλά θεωρούμε δεδομένο ότι οι διευθύνσεις τους αυξάνονται όσο προχωράμε προς το τέλος του σώματος του βρόχου (όπως ο βρόχος στο Σχήμα 3.12). Το ζητούμενο είναι να υπολογίσουμε πόσα misses θα προκαλέσει η εκτέλεση αυτού του βρόχου σε μία συσχετιστική (associative) κρυφή μνήμη εντολών με LRU πολιτική αντικατάστασης.



Σχήμα 3.12: Ένα παράδειγμα απλού βρόχου

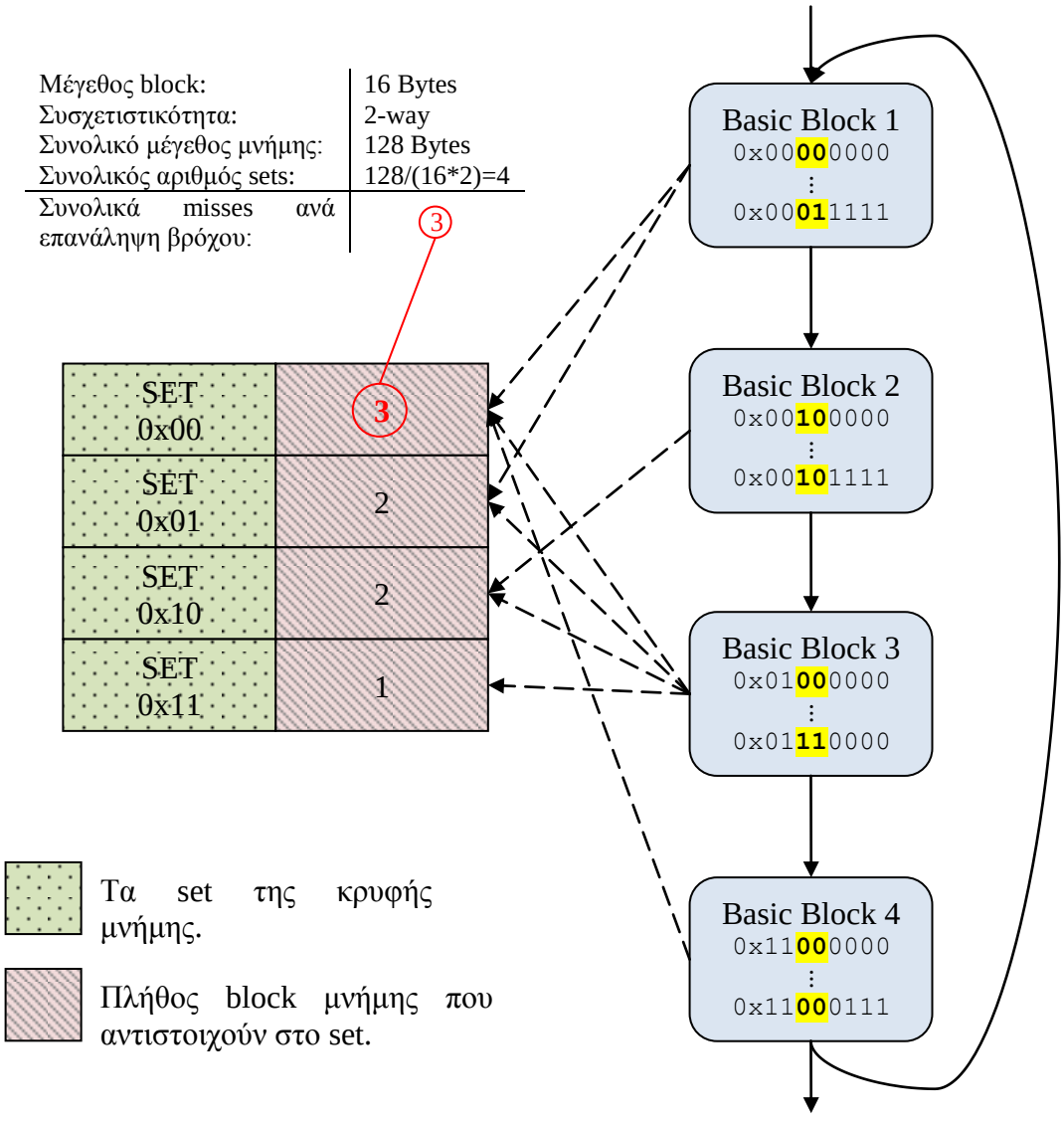
Για να αναπτύξουμε την μέθοδο, θα ξεκινήσουμε επικεντρωνόμενοι στη συμπεριφορά ενός συνόλου (set) μίας κρυφής μνήμης με βαθμό συσχετικότητας (associativity) A. Υποθέτουμε επίσης ότι το σώμα του βρόχου περιέχει N block μνήμης τα οποία αντιστοιχούν στο συγκεκριμένο set.

Είναι εύκολο να συμπεράνουμε πως αν N≤A, τότε με εξαίρεση την πρώτη εκτέλεση το βρόχου (compulsory misses), σε όλες τις επόμενες εκτελέσεις του βρόχου η ζήτηση των N basic blocks που αντιστοιχούν στο εν λόγω set δεν θα προκαλέσει κανένα cache miss. Αυτό συμβαίνει επειδή και τα N block μνήμης χωράν στο set, επομένως μετά τη πρώτη εκτέλεση του βρόχου θα παραμείνουν όλα διαθέσιμα για τις επόμενες εκτελέσεις.



Σχήμα 3.13: Συμπεριφορά ενός set μίας 4-way συσχετιστικής μνήμης κατά την εκτέλεση ενός βρόχου με 5 block μνήμης που αντιστοιχούν στο set.

Αντίθετα, αν N>A, τότε, μετά την ζήτηση των πρώτων A blocks μνήμης, θα χρειαστεί να απορριφτούν τα πρώτα block που εισήχθησαν στο set (τα λιγότερο πρόσφατα χρησιμοποιημένα) ώστε να πάρουν τη θέση του τα νέα. Όταν στην επόμενη εκτέλεση του βρόχου ζητηθούν τα block που απορρίφτηκαν τότε και αυτά με τη σειρά τους θα διαγράψουν τα block που πρόκειται να ζητηθούν στη συνέχεια. Αυτή η κυκλική διαγραφή των blocks μνήμης έχει ως αποτέλεσμα να καταλήγουν όλες οι αιτήσεις που αντιστοιχούν στο συγκεκριμένο set σε cache misses. Στο Σχήμα 3.13 φαίνεται ενδεικτικά η συμπεριφορά ενός set μίας 4-way συσχετιστικής μνήμης κατά την εκτέλεση ενός βρόχου με 5 block μνήμης που αντιστοιχούν στο set. Δείτε πως όλες οι αιτήσεις καταλήγουν σε cache miss.



Σχήμα 3.14: Υπολογισμός των συνολικών cache misses του βρόχου (Σχήμα 3.12)

Σύμφωνα με τα παραπάνω, για να υπολογίσουμε το συνολικό αριθμό των cache misses που θα επιφέρει κάθε επανάληψη του βρόχου, αρκεί να υπολογίσουμε τον πλήθος των block μνήμης που αντιστοιχούν σε κάθε set της κυφής μνήμης εντολών. Αφού το κάνουμε αυτό μένει να αναζητήσουμε τα sets στα οποία αντιστοιχούν περισσότερα από A (βαθμός συσχετικότητας) blocks μνήμης και να αθροίσουμε το

πλήθος των blocks που αντιστοιχούν σε αυτά. Ένα παράδειγμα δίνεται σχηματικά στο Σχήμα 3.14. Φυσικά αν πολλαπλασιάσουμε με το πλήθος επαναλήψεων του βρόχου θα υπολογίσουμε και το πλήθος των συνολικών instruction cache misses που επέφερε η εκτέλεση του βρόχου.

Παρακάτω (Κώδικας 3.11) δίνεται ο αλγόριθμος υπολογισμού των instruction cache misses ενός απλού βρόχου σε ψευδοκώδικα. Ο αλγόριθμος αυτός μπορεί να εφαρμοστεί αναδρομικά για την εκτίμηση της επίδοσης ενός πολυεπίπεδου συστήματος κρυφής μνήμης. Αρκεί να τον τρέξουμε ξανά για το ανώτερο επίπεδο μνήμης δίνοντας του ως είσοδο το μονοπάτι που αποτελείται από τα block μνήμης που επέφεραν cache miss.

```
//BLOCK_SIZE:    Το μέγεθος των block της μνήμης σε bytes
//ASSOCIATIVITY: Ο βαθμός συσχετικότητας της μνήμης
//CACHE_SIZE:    Το μέγεθος της κρυφής μνήμης

CACHE_SET_NUMBER:= CACHE_SIZE/(BLOCK_SIZE*ASSOCIATIVITY)
LINE_OFFSET:= log2(BLOCK_SIZE)
LINE_MASK:= BLOCK_SIZE-1
SET_MASK:= log2(CACHE_SET_NUMBER)

Calculate_Unique_Path_Misses(path)
  for i in 0 to CACHE_SET_NUMBER-1
    cache_set_demands[i] ← 0
  last_covered_address ← -1
  for each basic block bb ∈ path.basic_blocks
    first_set ← max(bb.address,last_covered_address+1)
    first_set ← first_set >> LINE_OFFSET
    last_covered_address ← bb.address + bb.length − 1
    last_covered_address ← last_covered_address ∨ LINE_MASK
    last_set ← last_covered_address >> LINE_OFFSET
    for i in first_set to last_set
      cache_set_demands[i ∧ SET_MASK] ←
       cache_set_demands[i ∧ SET_MASK] + 1
  misses ← 0
  for i in 0 to CACHE_SET_NUMBER - 1
    if cache_set_demands[i] > ASSOCIATIVITY
      misses ← misses + cache_set_demands[i]
  misses ← misses * path.executions
  return misses
```

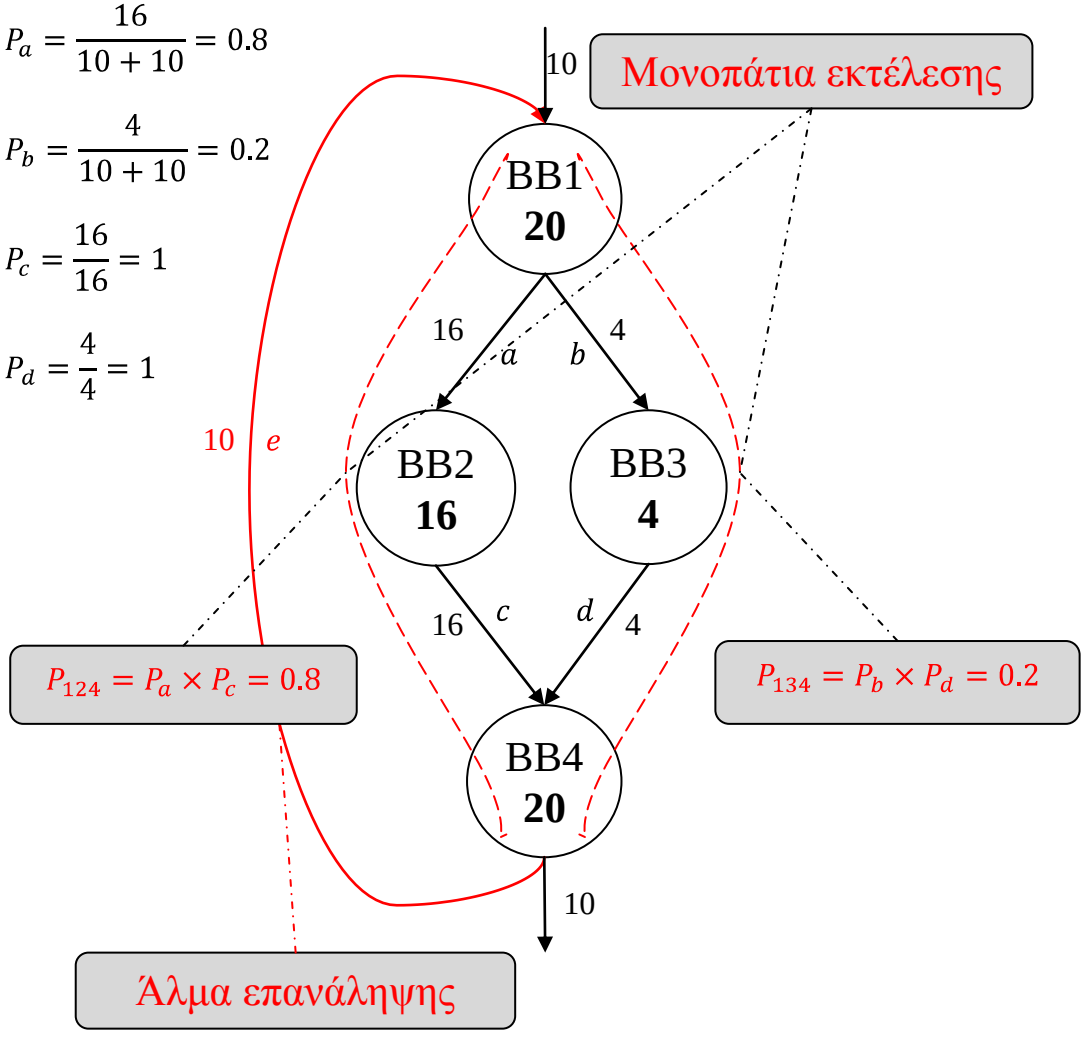
Κώδικας 3.11: Αλγόριθμος υπολογισμού των instruction cache misses ενός απλού βρόχου

3.8 Εντοπισμός βρόχων και μοναδιαίων μονοπατιών και υπολογισμός του συνολικού αριθμού αστοχιών

Αφού αναπτύξαμε έναν αλγόριθμο για τον υπολογισμό των cache misses που επιφέρει ένας απλός βρόχος, θα πρέπει τώρα να διευρύνουμε τη μέθοδο για τον υπολογισμό των misses που επιφέρει μία πλήρης εφαρμογή. Ο τεχνική που περιγράφεται σε αυτή την ενότητα είναι ιδιαίτερα απλοϊκή αλλά πολύ γρήγορη και όπως θα δούμε στο κεφάλαιο 5 έχει αρκετά καλή ακρίβεια.

Κάθε βρόχος του target E.Γ.Ρ.Ε. εξετάζεται ανεξάρτητα από τους υπόλοιπους (σαν να ήταν ένα αυτόνομο πρόγραμμα) και στο τέλος αθροίζονται τα συνολικά cache misses όλων των βρόχων του προγράμματος. Για κάθε βρόχο εντοπίζονται όλα τα μονοπάτια εκτέλεσης και υπολογίζεται ο αριθμός εκτελέσεων του κάθε μονοπατιού. Στη συνέχεια, για κάθε μονοπάτι, εφαρμόζεται ο αλγόριθμος που περιγράφηκε στην ενότητα 3.7 (Κώδικας 3.11) για να βρεθούν τα συνολικά misses που προκαλεί η εκτέλεση του βρόχου.

Ο εντοπισμός των βρόχων δεν γίνεται με βάση κάποιον αλγόριθμο αναζήτησης κύκλων σε γράφους, αφού κάτι τέτοιο θα ήταν εξαιρετικά χρονοβόρο σε πολύπλοκους γράφους. Αντιθέτως ο εντοπισμός βρόχων γίνεται απλά αναζητώντας στον target E.Γ.Ρ.Ε. ακμές με κατεύθυνση από μία μεγαλύτερη διεύθυνση μνήμης σε μία μικρότερη. Δεδομένου, ότι η εκτέλεση του κώδικα μηχανής γίνεται κατά κανόνα από τις μικρότερες προς τις μεγαλύτερες διευθύνσεις, δεν είναι δυνατό να υπάρξει κάποιος βρόχος χωρίς την ύπαρξη ενός άλματος από υψηλότερη προς χαμηλότερη διεύθυνση. Αυτό το άλμα θα το ονομάζουμε άλμα (ακμή) επανάληψης. Ως πλήθος επαναλήψεων του κάθε βρόχου θεωρούμε τον αριθμό εκτελέσεων του άλματος επανάληψης.



Σχήμα 3.15: Ένας βρόχος και τα μονοπάτια εκτέλεσής του

Για κάθε βρόχο θα πρέπει να εντοπιστούν όλα τα μονοπάτια εκτέλεσης και για κάθε μονοπάτι εκτέλεσης να υπολογιστεί ο αριθμός εκτελέσεών του. Στο Σχήμα 3.15 δίνεται το παράδειγμα ενός βρόχου. Με κόκκινη συνεχή γραμμή φαίνεται το άλμα επανάληψης και με κόκκινες διακεκομμένες γραμμές φαίνονται τα δύο μονοπάτια εκτέλεσης.

Για να γίνει περισσότερο κατανοητός ο τρόπος με τον οποίο υπολογίζεται ο αριθμός εκτελέσεων ενός μονοπατιού, θα πρέπει να εισάγουμε δύο νέους όρους: τη πιθανότητα εκτέλεσης άλματος και τη πιθανότητα εκτέλεσης μονοπατιού [1].

Ως πιθανότητα εκτέλεσης του άλματος από τον κόμβο i στον κόμβο n για τον βρόχο L ορίζεται (βλ. Πίνακας 3.3 για συμβολισμούς):

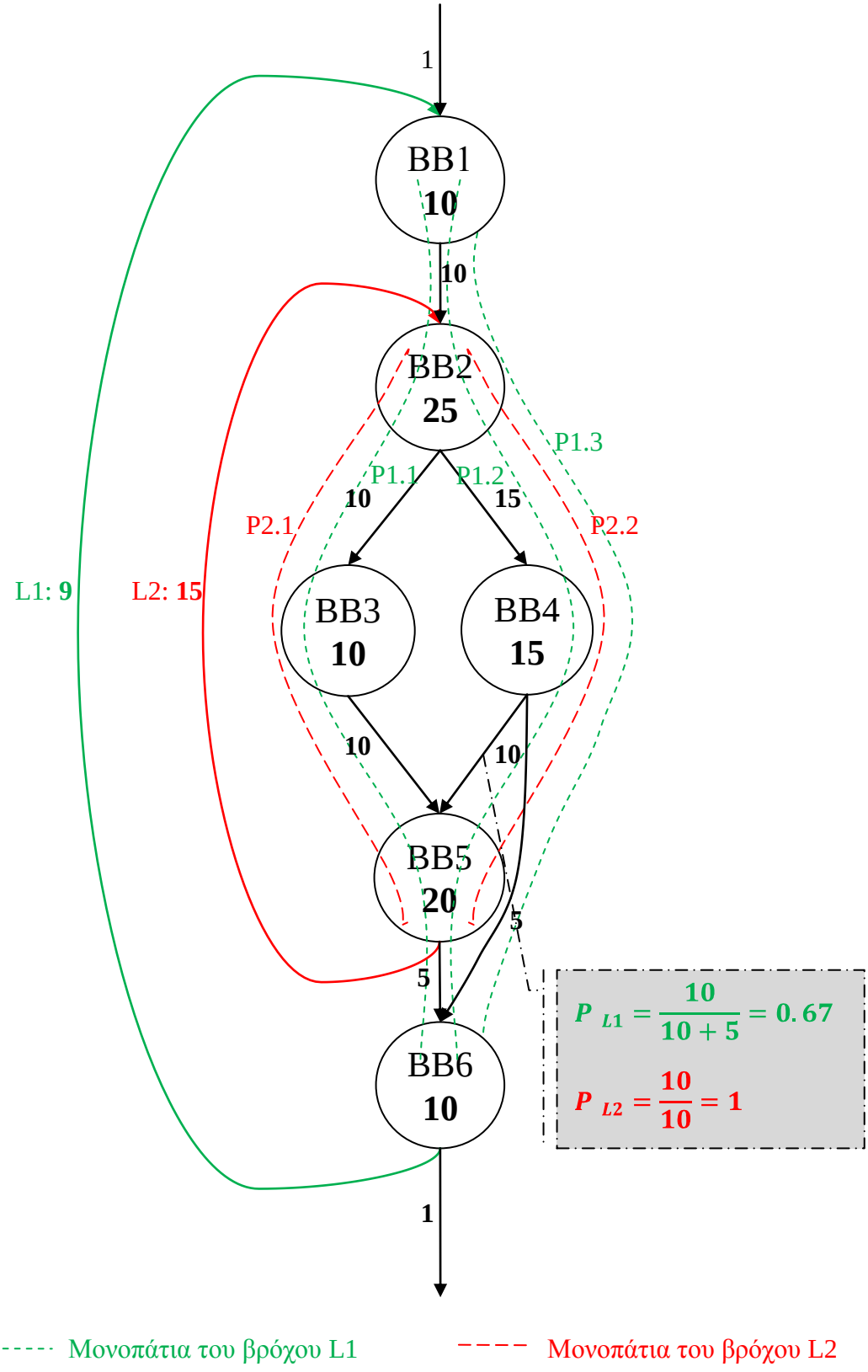
$$P_{i,n\ L} = \frac{j_{i,n}}{\sum_{m \in N_{L,i}} j_{i,m}} \quad \textbf{(3.5)}$$

Όπου $N_{L,i}$ είναι το σύνολο των κόμβων για του οποίους υπάρχει άλμα που ξεκινάει από τον κόμβο i και καταλήγει σε αυτούς **και οι οποίοι ανήκουν στο σώμα του βρόχου L**. Ο τελευταίος περιορισμός είναι σημαντικός και χρειάζεται προσοχή. Όπως θα δούμε στη συνέχεια η πιθανότητα εκτέλεσης μίας ακμής μπορεί να είναι διαφορετική ανάλογα με τον βρόχο στον οποίο θα θεωρήσουμε ότι ανήκει. Στο Σχήμα 3.15 φαίνονται οι υπολογισμοί των πιθανοτήτων για τις ακμές a, b, c και d.

Η πιθανότητα εκτέλεσης ενός μονοπατιού, τώρα, είναι το γινόμενο των πιθανοτήτων εκτέλεσης όλων των ακμών που περιέχει. Στο Σχήμα 3.15 φαίνονται οι υπολογισμοί των πιθανοτήτων για τα μονοπάτια 1→2→4 και 1→3→4.

Σε περίπτωση που έχουμε εμφωλευμένους βρόχους, ένα μονοπάτι του εσωτερικού βρόχου αποτελεί ταυτόχρονα και κομμάτι ενός μονοπατιού του εξωτερικού βρόχου. Αυτό το κομμάτι θα πρέπει να εξετάζεται ξεχωριστά για τον εσωτερικό και τον εξωτερικό βρόχο.

Στο Σχήμα 3.16 βλέπουμε έναν Γ.Ρ.Ε. με δύο βρόχους όπου ο ένας (L2) είναι εμφωλευμένος στο εσωτερικό του άλλου (L1). Στο σχήμα φαίνονται επίσης με πράσινες διακεκομμένες γραμμές τα τρία μονοπάτια εκτέλεσης του βρόχου L1 και με κόκκινες διακεκομμένες γραμμές τα δύο μονοπάτια εκτέλεσης του βρόχου L2. Ένα σημείο που έχει ενδιαφέρον στο συγκεκριμένο παράδειγμα, είναι η ακμή 4→5. Το ενδιαφέρον με αυτή την ακμή, είναι ότι παρουσιάζει διαφορετική πιθανότητα εκτέλεσης ανάλογα με τον βρόχο στον οποίο θεωρούμε ότι ανήκει. Έτσι κατά την ανάλυση του βρόχου L1 η ακμή αυτή θα έχει πιθανότητα 0.67, αφού η ακμή 4→6 είναι μέρος του βρόχου και ο αριθμός εκτελέσεών της θα προστεθεί στον παρανομαστή της πιθανότητας. Κατά την ανάλυση όμως του βρόχου L2 η ακμή αυτή θα έχει πιθανότητα εκτέλεσης ίση με 1 αφού τώρα η ακμή 4→6 δεν ανήκει στο βρόχο.



Σχήμα 3.16: Δύο εμφωλευμένοι βρόχοι και τα μονοπάτια εκτέλεσής τους

Αφού υπολογίσουμε την πιθανότητα εκτέλεσης κάθε μοναδιαίου μονοπατιού, τότε θεωρούμε ότι ο αριθμός επαναλήψεων του μονοπατιού ισούται με την

πιθανότητα εκτέλεσής του επί τον αριθμό επαναλήψεων του βρόχου. Έτσι για το μονοπάτι 1→2→4 από το Σχήμα 3.15 το πλήθος εκτελέσεων είναι 10×0.8 = 8 ενώ για το μονοπάτι 1→3→4 το πλήθος εκτελέσεων είναι 10×0.2 = 2.

Ωστόσο, στο Σχήμα 3.15 φαίνεται ξεκάθαρα ότι το μονοπάτι 1→2→4→ έχει εκτελεστεί 16 φορές αντί για 8 που μόλις υπολογίσαμε ενώ το μονοπάτι 1→3→4 έχει εκτελεστεί 2 φορές αντί για 2 που υπολογίσαμε. Αν παρατηρήσετε προσεκτικά θα δείτε ότι η πρώτη ακμή του γράφου έχει αριθμό εκτελέσεων 10 (όπως και η τελευταία). Αυτό σημαίνει ότι ο γράφος αυτό αποτελεί κομμάτι του σώματος ενός άλλου βρόχου που δεν απεικονίζεται στο σχήμα. Έτσι οι υπόλοιπες εκτελέσεις των μονοπατιών θα υπολογιστούν κατά την ανάλυση του εξωτερικού βρόχου που δεν φαίνεται στο σχήμα. Τέλος να επισημάνουμε ότι ένα ολοκληρωμένο πρόγραμμα θεωρείται ως ένας βρόχος με μόνο μία επανάληψη.

Σύμφωνα με τα παραπάνω δημιουργήθηκε ένας αλγόριθμος ο οποίος αναλαμβάνει να εντοπίσει όλα τα μοναδιαία μονοπάτια ενός βρόχου, καθώς και τον αριθμό επαναλήψεών τους. Ο αλγόριθμος αυτός στηρίζεται στον αλγόριθμο αναζήτησης γράφων DFS [35]. Παρακάτω (Κώδικας 3.12) δίνεται στη μορφή ψευδοκώδικα.

```
//Το σημείο εκκίνησης του αλγορίθμου.
Detect_Loop_Unique_Paths(loop)
        first_bb ← loop.first_basic_block
        for each jump j: j.origin_basic_block = first_bb
                If j.target > j.origin
                        Visit_Basic_Block(loop,j.target_block,1)

//Υπολογισμός του συνολικού αριθμού εκτελέσεων των αλμάτων που //ξεκινάν
από το basic block "bb" και ανήκουν στο βρόχο "loop".
Intraloop_Jumps_Executions(bb,loop)
        result ← 0
        for each jump j: j.source_basic_block = bb
                if j.target > j.origin _? j.origin ≤ loop.last_addr
                        result ← result + j.executions
        return result

//Η υπορουτίνα επίσκεψης basic block (DFS).
Visit_Basic_Block(loop,bb,prop)
        if bb.address > loop.last_addr
                return
        if prop = 0
                return
        if bb = loop.last_basic_block
                Unique_Path_Detected(loop.exections * prop)
                Return
        ije ← Intraloop_Jumps_Executions(bb,loop)
        if ije = 0
                return
        for each jump j: j.source_basic_block = bb
                if j.target > j.origin
                        next_bb ← j.source_target_block
                        next_bb.prev_in_path ← bb
                        new_prop ← prop * j.executions/ije
                        Visit_Basic_Block(next_bb)
```

Κώδικας 3.12: Αλγόριθμος εντοπισμού των μονοπατιών εκτέλεσης ενός βρόχου και υπολογισμού των επαναλήψεών τους

Αφού λοιπόν με τη χρήση του προηγούμενου αλγορίθμου εντοπιστούν όλα τα μονοπάτια εκτέλεσης και υπολογιστούν οι αριθμοί επανάληψής τους, τότε μπορούμε να εφαρμόσουμε τη μέθοδο που περιγράφτηκε στην ενότητα 3.7 για κάθε μονοπάτι εκτέλεσης, κάθε βρόχου της εφαρμογής, να αθροίσουμε τα αποτελέσματα για να δημιουργήσουμε μία εκτίμηση για την συνολική επίδοση της κρυφής μνήμης εντολών.

3.9 Περιπτώσεις που απαιτούν ιδιαίτερη προσοχή

Σε αυτή την ενότητα θα περιγράψουμε εν συντομία κάποιες περιπτώσεις που μπορεί να κάνουν τη μέθοδο που περιγράφθηκε στις προηγούμενες ενότητες να παράγει εκτιμήσεις με σημαντική απόκλιση. Θα προταθούν επίσης τρόποι για να αντιμετωπιστούν τα προβλήματα αυτά.

Κάτι που υπονοήθηκε σιωπηλά κατά τη περιγραφή της μεθοδολογίας είναι ότι η εφαρμογή για την οποία γίνεται η εκτίμηση είναι εξ' ολοκλήρου διαθέσιμη σε μορφή πηγαίου κώδικα. Αν δεν έχουμε τον πηγαίο κώδικα είναι αδύνατο να μεταγλωττίσουμε την εφαρμογή μας για δύο ξεχωριστές αρχιτεκτονικές (host υπολογιστής και target επεξεργαστής) όπως απαιτεί η μεθοδολογία. Φυσικά είναι εξαιρετικά σπάνιο μία εφαρμογή να είναι εξ' ολοκλήρου διαθέσιμη σε μορφή πηγαίου κώδικα, αφού οι περισσότερες εφαρμογές χρησιμοποιούν τουλάχιστον την standard βιβλιοθήκη της γλώσσας C, η οποία είναι προ-μεταγλωττισμένη.

Η χρήση βιβλιοθηκών της C, ωστόσο, δεν κάνει σε καμία περίπτωση μη εφαρμόσιμη τη μεθοδολογία μας. Αυτό οφείλεται στο ότι κάθε μεταγλωττιστής (native και cross) έχει τη δική του προ-μεταγλωττισμένη βιβλιοθήκη της C η οποία είναι κατάλληλη για την αρχιτεκτονική του μεταγλωττιστή. Έτσι η εφαρμογή μπορεί να μεταγλωττιστεί χωρίς προβλήματα και για τις δύο αρχιτεκτονικές. Αν μάλιστα οι συναρτήσεις της βιβλιοθήκης της C έχουν μικρό ποσοστό εκτέλεσης σε σχέση με τον συνολικό κώδικα, τότε η έλλειψη του πηγαίου κώδικά έχει ως αποτέλεσμα μία μικρή και ασήμαντη απόκλιση. Βεβαίως, αν οι συναρτήσεις βιβλιοθήκης έχουν μεγάλο ποσοστό εκτέλεσης, τότε και η απόκλιση στα αποτελέσματα της μεθόδου θα είναι μεγαλύτερη.

Κάποιες περιπτώσεις όμως χρειάζονται ιδιαίτερη προσοχή, και μία τέτοια είναι πχ. η συνάρτηση qsort. Η συνάρτηση qsort δέχεται ως όρισμα έναν δείκτη σε μία δική μας συνάρτηση, την οποία καλεί όποτε χρειάζεται να κάνει κάποια σύγκριση. Το μεγαλύτερο ποσοστό εκτέλεσης της συνάρτησης qsort αποτελείται από κλήσεις στη δική μας συνάρτηση σύγκρισης, κάτι που σημαίνει ότι λογικά δεν θα πρέπει να δημιουργήσει σημαντικό πρόβλημα στη μέθοδό μας. Υπάρχει ωστόσο η περίπτωση η υλοποίηση της qsort για έναν εκ των δύο μεταγλωττιστές να είναι αποδοτικότερη, με αποτέλεσμα να καλεί λιγότερες φορές τον κώδικά μας. Αυτό μπορεί να δημιουργήσει σημαντικές αποκλίσεις στα αποτελέσματα τις μεθόδου. Έτσι σε τέτοιες περιπτώσεις

θα πρέπει να διασφαλίζεται ότι οι δύο εκδώσεις των προ-μεταγλωττισμένων συναρτήσεων είναι ισοδύναμες από πλευράς απόδοσης. Ανάλογη προσοχή χρειάζονται και οι συναρτήσεις των οποίων το αποτέλεσμα δεν εξαρτάται μόνο από τη κατάσταση του προγράμματος, αλλά και από τη κατάσταση του συστήματος. Τέτοιες συναρτήσεις, είναι πχ. οι συναρτήσεις που ζητούν την ώρα από το σύστημα. Αυτές οι συναρτήσεις μπορεί να προκαλέσουν διαφορετική συμπεριφορά στην εφαρμογή όταν αυτή τρέξει στον host υπολογιστή από τη συμπεριφορά που θα είχε αν έτρεχε στο ενσωματωμένο σύστημα.

Ένα άλλο πρόβλημα που μπορεί να δημιουργηθεί από την χρήση διαφορετικών εκδόσεων της standard βιβλιοθήκης της C έχει να κάνει με την αντιστοίχηση του πηγαίου κώδικα και του κώδικα μηχανής μέσω του format dwarf. Όπως έχει περιγραφεί στις προηγούμενες ενότητες, ο ίδιος πηγαίος κώδικας χρησιμοποιείται και για τη παραγωγή του εκτελέσιμου που θα τρέξει στον host υπολογιστή, αλλά και για την παραγωγή του τελικού assembly κώδικα για τον target επεξεργαστή. Ωστόσο, πριν τη φάση της μεταγλώττισης ο πηγαίος κώδικας περνάει από τον προ-επεξεργαστή της C ο οποίος αναλαμβάνει να ενσωματώσει στον πηγαίο κώδικα τα περιεχόμενα των header files. Σε περίπτωση που αυτά τα header files δεν είναι όμοια για τους δύο μεταγλωττιστές, οι δύο εκδώσεις του προ-επεξεργασμένου κώδικα που προκύπτουν δεν είναι όμοια στοιχισμένες, με αποτέλεσμα να μη γίνεται σωστή αντιστοίχιση των αριθμών εκτελέσεων των εντολών μηχανής Το πρόβλημα αυτό μπορεί να λυθεί με χρήση των ενδεικτικών γραμμής που χρησιμοποιεί ο CPP (gnu pre-processor). Ο CPP στην έξοδό του εισάγει κάποιους δείκτες (identifiers) που αντιστοιχίζουν τις γραμμές του επεξεργασμένου πηγαίου κώδικα με τις γραμμές του αρχικού πηγαίου κώδικα. Με χρήση αυτών των δεικτών μπορεί να διορθωθεί η αντιστοίχιση γραμμών των δύο αρχείων και να αποφευχθεί το πρόβλημα που περιγράφηκε.

4 Υλοποιηση της Μεθοδου

Οι μεθοδολογία που περιγράφθηκε στο κεφάλαιο 3 υλοποιήθηκε με την ανάπτυξη τριών ξεχωριστών εργαλείων. Τα εργαλεία αυτά ονομάστηκαν csa (C syntax analyzer), ccov (cross coverage) και canal (cache analyzer).

Το πρώτο (csa) είναι ένας συντακτικός αναλυτής της γλώσσας C. Σκοπός του είναι να δέχεται ως είσοδο ένα αρχείο πηγαίου κώδικα C και να εφαρμόζει σε αυτό τους συντακτικούς μετασχηματισμούς που περιγράφηκαν στην ενότητα 3.5.

Το δεύτερο (ccov) έχει ως σκοπό τη δημιουργία του target E.Γ.Ρ.Ε. με χρήση των μεθόδων που περιγράφηκαν στις ενότητες 3.3, 3.4 και 3.6 (βήματα 2 και 3 της μεθόδου). Ο πηγαίος κώδικας στον οποίο εφαρμόζει το ccov τις μεθοδολογίες, είναι ο μετασχηματισμένος κώδικας που παράγει το csa ενώ ο target E.Γ.Ρ.Ε. που παράγεται μπορεί να εξαχθεί σε διάφορες μορφές (π.χ. XML [36]).

Τέλος, το canal δέχεται ως είσοδο τον target E.Γ.Ρ.Ε. που εξάγει το ccov και με χρήση των μεθόδων που περιγράφονται στις ενότητες 3.7 και 3.8 κάνει εκτιμήσεις για την επίδοση διαφόρων αρχιτεκτονικών κρυφής μνήμης εντολών ταυτόχρονα (βήμα 4 της μεθόδου). Στις επόμενες ενότητες θα αναφερθούμε σύντομα στη δομή αυτών των εργαλείων.

4.1 Csa: Ένας συντακτικός μετατροπέας για τη γλώσσα C

Το εργαλείο csa αποτελείται από 7 βασικές δομικές μονάδες οι οποίες στη συνέχεια παρατίθενται αλφαβητικά και περιγράφονται συνοπτικά.

c_expr: Η μονάδα που διαχειρίζεται την εσωτερική αναπαράσταση του συντακτικού δέντρου των αρχείων C. Παρέχει συναρτήσεις για τη δημιουργία, αντιγραφή, καταστροφή και εμφάνιση των συντακτικών εκφράσεων της γλώσσας C.

c_parser: Ένας συντακτικός αναλυτής (parser) προγραμμάτων C γραμμένος για το εργαλείο bison [37]. Η γραμματική της γλώσσας C έχει παρθεί από το [38] και

έχει τροποποιηθεί ελαφρώς (βλ. ΠΑΡΑΡΤΗΜΑ Ι για συμβατότητα). Η μονάδα c_parser λειτουργεί σε συνεργασία με τη μονάδα c_scanner.

c_scanner: Ένας λεκτικός αναλυτής της γλώσσας C γραμμένος για το εργαλείο flex [39]. Αναγνωρίζει όλο το λεξιλόγιο της ANSI C καθώς και κάποιες δεσμευμένες λέξεις της υλοποίησης του gcc (βλ. ΠΑΡΑΡΤΗΜΑ Ι για συμβατότητα).

csa: Η μονάδα που αναλαμβάνει τη διεπαφή με τον χρήστη, την παραμετροποίηση και την δρομολόγηση των διαδικασιών .

error: Η μονάδα που διαχειρίζεται την εμφάνιση μηνυμάτων σφαλμάτων και προειδοποιήσεων προς τον χρήστη.

memory: Η μονάδα διαχείρισης μνήμης. Παρέχει συναρτήσεις για τη δέσμευση και την ελευθέρωση μνήμης με κάποιες επιπλέον λειτουργίες από αυτές των συναρτήσεων malloc και free.

symbol_table: Ο πίνακας συμβόλων του συντακτικού αναλυτή. Μία δομή δεδομένων στην οποία αποθηκεύονται οι ιδιότητες των συμβόλων που είναι απαραίτητες για τη λειτουργία του αναλυτή.

syntax_transform: Η μονάδα που αναλαμβάνει την εφαρμογή των συντακτικών μετασχηματισμών που περιγράφονται στην ενότητα 3.5.

4.2 Ccov: Ένα εργαλείο για τον χαρακτηρισμό του Γ.Ρ.Ε. του κώδικα μηχανής

Το εργαλείο ccov αποτελείται από 14 βασικές δομικές μονάδες οι οποίες στη συνέχεια παρατίθενται αλφαβητικά και περιγράφονται συνοπτικά.

asm_level: Κάποιες φορές, λόγω αστοχίας της μεθόδου, μπορεί να βρεθεί ότι εντολές του ίδιου basic block του κώδικα μηχανής έχουν διαφορετικούς αριθμούς εκτελέσεων. Σε αυτή τη περίπτωση θα πρέπει να γίνει διόρθωση των τιμών εκτελέσεων. Η μονάδα asm_level παρέχει ένα σύνολο εναλλακτικών αλγορίθμων για την επίλυση του προβλήματος (βλ. σελ. 105 για τους διαθέσιμους αλγόριθμους).

asm_parser: Η μονάδα που αναλαμβάνει την ανάγνωση του κώδικα μηχανής και των επιπλέον πληροφοριών που παράγει ο gcc από τα αρχεία assembly (βλ. ενότητα 3.4).

aux: Μία βοηθητική μονάδα που παρέχει δομές και συναρτήσεις χρήσιμες σε διάφορες άλλες μονάδες του εργαλείου (πχ. λίστες ονομάτων αρχείων).

ccov: Η μονάδα που αναλαμβάνει τη διεπαφή με τον χρήστη, την παραμετροποίηση των υπολοίπων μονάδων και τη δρομολόγηση των διαδικασιών.

ccov_printer: Η μονάδα που αναλαμβάνει τη δημιουργία των βασικών αρχείων εξόδου του εργαλείου (βλ. παράρτημα: II.III.I).

cfg: Η εσωτερική αναπαράσταση του Ε.Γ.Ρ.Ε. Παρέχονται δομές και συναρτήσεις για τη δημιουργία, καταστροφή και τροποποίηση του Ε.Γ.Ρ.Ε.

cfg_printer: Η μονάδα που αναλαμβάνει τη δημιουργία των αρχείων περιγραφής του Ε.Γ.Ρ.Ε.. (βλ. παράρτημα: II.III.II και II.III.III).

error: Η μονάδα που διαχειρίζεται την εμφάνιση μηνυμάτων σφαλμάτων και προειδοποιήσεων προς τον χρήστη.

gcov_level: Όπως και στα basic block του κώδικα μηχανής, έτσι και στον πηγαίο κώδικα μπορεί να βρεθεί ότι μία γραμμή έχει παραπάνω από έναν αριθμούς εκτελέσεων (πχ. αν δεν γίνει χρήση του csa). Η μονάδα παρέχει ένα σύνολο αλγορίθμων για την επίλυση του προβλήματος (βλ. σελ. 105 για τους διαθέσιμους αλγόριθμους).

gcov_parser: Η μονάδα που αναλαμβάνει την άντληση πληροφοριών από τα αρχεία gcov (βλ. ενότητα 3.3.2).

hash_table: Η μονάδα με την οποία αντιστοιχίζονται ονόματα (ids) σε διάφορα αντικείμενα (πχ. αναπαραστάσεις γράφων συναρτήσεων, αρχείων, συμβόλων κτλ.).

memory: Η μονάδα διαχείρισης μνήμης. Παρέχει συναρτήσεις για τη δέσμευση και την ελευθέρωση μνήμης με κάποιες επιπλέον λειτουργίες από αυτές των συναρτήσεων malloc και free.

object_symbol_parser: Η μονάδα που αναλαμβάνει την ανάγνωση της εξόδου του εργαλείου nm για την εξαγωγή των διευθύνσεων των συναρτήσεων. (βλ. παράμετρος symbol_list σελ. 105).

vcg_parser: Η μονάδα vcg_parser υλοποιήθηκε με χρήση του εργαλείου bison [37] και αναλαμβάνει την ανάγνωση της περιγραφής του Γ.Ρ.Ε. από τα αρχεία vcg (βλ. ενότητα 3.6.1). Πρόκειται στην ουσία για έναν συντακτικό αναλυτή της γλώσσας gdl [33] με κάποιες τροποποιήσεις ώστε να αναγνωρίζει τις επιπλέον πληροφορίες που εισάγει ο gcc στα αρχεία vcg. Η μονάδα vcg_parser λειτουργεί σε συνεργασία με τη μονάδα vcg_scanner.

vcg_scanner: Η μονάδα που αναλαμβάνει την ανάγνωση των λεκτικών μονάδων από τα αρχεία vcg. Είναι υλοποιημένη με χρήση του εργαλείου flex [39].

4.3 Canal: Ένα εργαλείο για γρήγορη εκτίμηση της επίδοσης της κρυφής μνήμης εντολών

Το εργαλείο canal είναι υλοποιημένο με τη γλώσσα C++ και χρησιμοποιεί τη βιβλιοθήκη TinyXML [40] για την ανάγνωση αρχείων XML [36] . Αποτελείται από 9 κλάσεις οι οποίες στη συνέχεια παρατίθενται αλφαβητικά και περιγράφονται συνοπτικά.

ArchitectureExplorer: Η βασική κλάση του εργαλείου. Αρχικοποιείται από ένα cxml αρχείο (βλ. παράρτημα III.II) και δημιουργεί την αναπαράσταση των αρχιτεκτονικών που περιγράφονται στο αρχείο. Συνδέει τις αρχιτεκτονικές με τα αντίστοιχα gxml αρχεία (βλ. παράρτημα II.III.II), δρομολογεί τη διαδικασία εκτίμησης επίδοσης και παρουσιάζει τα αποτελέσματα.

CacheModel: Η κλάση που υλοποιεί το μοντέλο μίας μονάδας κρυφής μνήμης. Αρχικοποιείται από ένα XML στοιχείο τύπου "cache" (βλ. παράρτημα III.II) και παρέχει μεθόδους για απαίτηση δεδομένων και υπολογισμό των αστοχιών με βάση τη μέθοδο που περιγράφθηκε στις ενότητες 3.7 και 3.8.

CfgBasicBlock: Η κλάση που υλοποιεί το basic block του Ε.Γ.Ρ.Ε.. Αρχικοποιείται από ένα στοιχείο τύπου basic block κάποιου αρχείου gxml (βλ. παράρτημα II.III.II).

CfgFile: Η κλάση που υλοποιεί τον Ε.Γ.Ρ.Ε. ενός αρχείου gxml (βλ. παράρτημα II.III.II). Αρχικοποιείται .άμεσα από ένα αρχείο gxml και παρέχει μεθόδους για αναζήτηση συναρτήσεων, και τον υπολογισμό γενικών στατιστικών (πχ. συνολικός αριθμός εντολών).

CfgFunction: Η κλάση που υλοποιεί τον Ε.Γ.Ρ.Ε. μίας συνάρτησης. Αρχικοποιείται από ένα XML στοιχείο τύπου function (βλ. παράρτημα II.III.II).και παρέχει συναρτήσεις για την αναζήτηση basic block, τον εντοπισμό βρόχων και άλλες λειτουργίες.

CfgLoop: Η κλάση που υλοποιεί έναν βρόχο του Ε.Γ.Ρ.Ε. Παρέχει μεθόδους για διάσχιση όλων των μονοπατιών εκτέλεσης (βλ. ενότητα 3.8) και για τον υπολογισμό διαφόρων ιδιοτήτων του βρόχου.

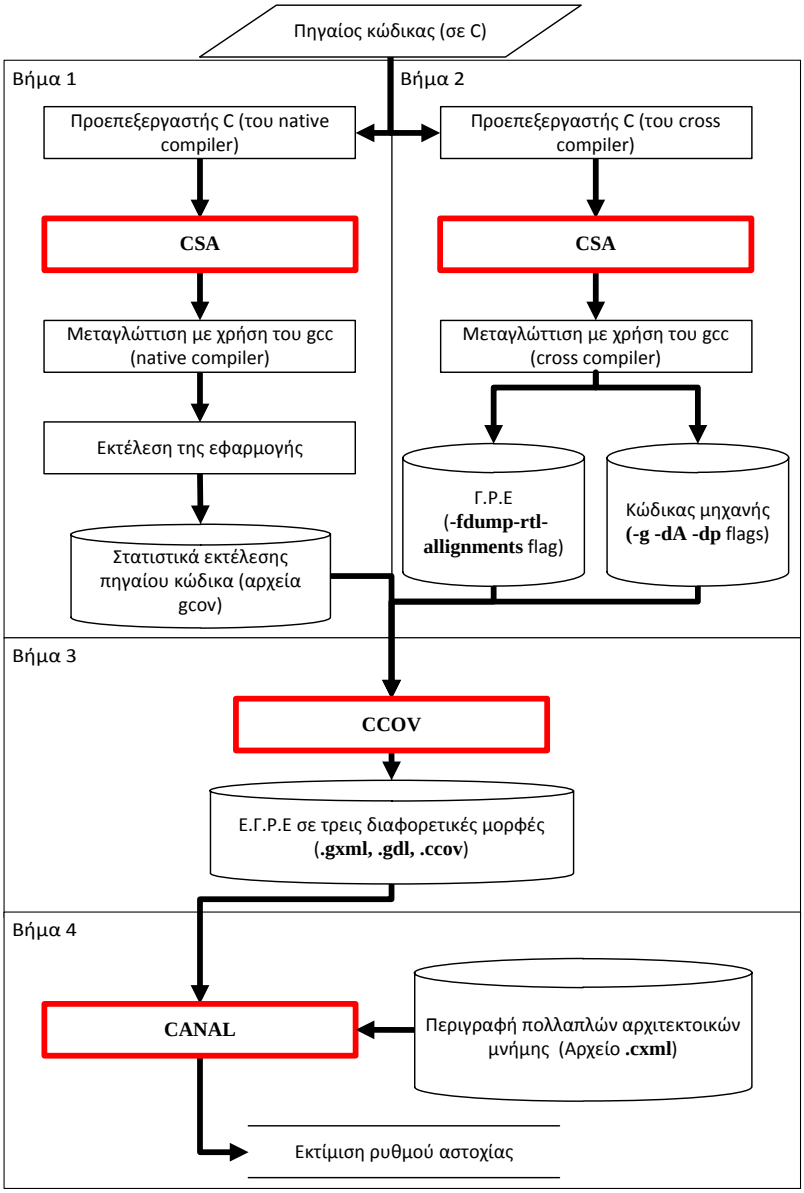
CfgUniquePath: Η κλάση που περιγράφει ένα μοναδιαίο μονοπάτι εκτέλεσης. Χαρακτηρίζεται από έναν αριθμό εκτελέσεων και παρέχει μεθόδους για τη διάσχιση των basic block του μονοπατιού.

ErrorHandler: Η κλάση που διαχειρίζεται την εμφάνιση μηνυμάτων σφαλμάτων και προειδοποιήσεων προς τον χρήστη

MemoryArchitecture: Η κλάση που υλοποιεί μία ιεραρχία μνήμης. Αρχικοποιείται από ένα XML στοιχείο τύπου "memory_architecture" (βλ. παράρτημα III.II). Αναλαμβάνει να δημιουργήσει τις συνδέσεις μεταξύ των μονάδων κρυφής μνήμης που περιέχει καθώς και να δρομολογήσει τις αιτήσεις δεδομένων.

4.4 Συνδυάζοντας τα εργαλεία

Η συνολική διαδικασία εκτίμησης που περιγράφθηκε στα προηγούμενα κεφάλαια είναι σχετικά πολύπλοκη για να εκτελεσθεί βήμα-βήμα χρησιμοποιώντας τα εργαλεία που υλοποιήθηκαν. Έτσι για τη διευκόλυνση του χρήστη αναπτύχθηκε ένα Makefile (βλ. εργαλείο gnu make [41]) το οποίο αυτοματοποιεί την διαδικασία. Παρακάτω δίνεται ένα σχήμα (Σχήμα 4.1) που περιγράφει τη διαδικασία που αυτοματοποιεί το Makefile. Με κόκκινο περίγραμμα διαχωρίζονται τα νέα εργαλεία. Στο σχήμα φαίνεται, επίσης, η αντιστοίχηση της διαδικασίας με τα βήματα της μεθοδολογίας που περιγράφθηκαν στην ενότητα 3.2.



Σχήμα 4.1: Αυτοματοποιημένη διαδικασία εκτίμησης με χρήση του εργαλείου GNU Make

5 Πειραματικα Αποτελεσματα

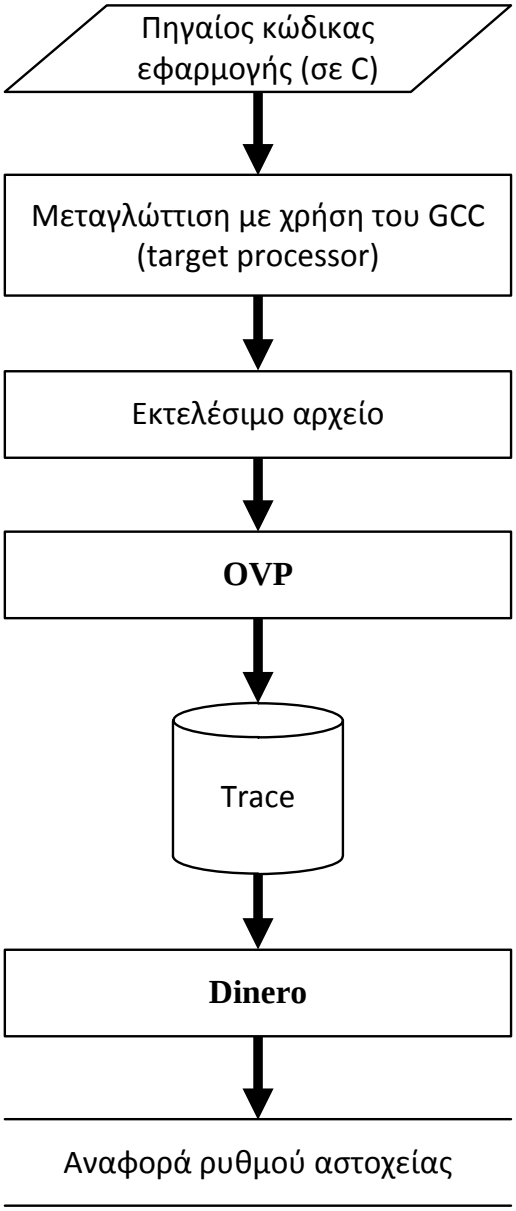
Για να αποτιμηθεί η ακρίβεια και η ταχύτητα της νέας μεθοδολογίας, ήταν απαραίτητο να πραγματοποιηθούν κάποιες μετρήσεις. Για το σκοπό αυτό τα νέα εργαλεία χρησιμοποιήθηκαν για την εκτίμηση της επίδοσης ενός συνόλου εφαρμογών σε 4 διαφορετικές αρχιτεκτονικές. Τα αποτελέσματα συγκρίθηκαν με τα αποτελέσματα προσομοιώσεων για τον υπολογισμό της ακρίβειας αλλά και της εξοικονόμησης χρόνου που πετυχαίνει η μέθοδος.

5.1 Πειραματική διαδικασία

Για τις προσομοιώσεις των εκτελέσεων των εφαρμογών χρησιμοποιήθηκε η πλατφόρμα OVP [7], η οποία θεωρείται αρκετά γρήγορη και παρέχει αρκετά μοντέλα αρχιτεκτονικών. Η συγκεκριμένη πλατφόρμα όπως περιγράψαμε και στο κεφάλαιο 2 παρέχει εκτός από τη δυνατότητα προσομοίωσης της αρχιτεκτονικής ενός επεξεργαστή και τη δυνατότητα προσομοίωσης της ιεραρχίας μνήμης. Ωστόσο το μοντέλο της ιεραρχίας μνήμης πρέπει να αναπτυχθεί εκ του μηδενός από το χρήστη. Για το λόγο αυτό, για την προσομοίωση της ιεραρχίας μνήμης προτιμήθηκε να χρησιμοποιηθεί ένα έτοιμο και δοκιμασμένο εργαλείο, το Dinero IV [5].

Έτσι το OVP ανέλαβε τη λειτουργία της προσομοίωσης της αρχιτεκτονικής των επεξεργαστών και τη παραγωγή ενός αρχείου trace το οποίο δόθηκε στο Dinero ως είσοδος (βλ. trace-driven simulation στο κεφάλαιο 2.1). Το αρχείο trace που παράχθηκε από το OVP ήταν διαδικής μορφής, δηλαδή της μορφής που προκαλεί τη μικρότερη δυνατή διακίνηση δεδομένων και απαιτεί τους λιγότερους υπολογιστικούς πόρους για να γραφτεί και να αναγνωστεί (βλ. εγχειρίδιο χρήση Dinero [5]). Το OVP και το Dinero συνδέθηκαν μεταξύ τους μέσω pipes (βλ. named pipes [42]) κάτι που σημαίνει ότι το αρχείο trace δεν χρειάστηκε να αποθηκευτεί σε κάποια μονάδα μόνιμης αποθήκευσης (πχ. σκληρός δίσκος) και έτσι εξοικονομήθηκε πολύτιμος χρόνος. Γενικά έγινε κάθε προσπάθεια ώστε η διαδικασία προσομοίωσης να είναι όσο το δυνατόν ταχύτερη και να λειτουργήσει ως ένα αυστηρό μέτρο σύγκρισης για την επίδοση των νέων εργαλείων.

Η διαδικασία της προσομοίωσης φαίνεται επιγραμματικά στο Σχήμα 5.1. Η διαδικασία εκτίμησης με χρήση των νέων εργαλείων ήταν αυτή που δίνεται στο Σχήμα 4.1.



Σχήμα 5.1: Διαδικασία προσομοίωσης με χρήση του OVP και του Dinero IV

5.2 Επιλογή επεξεργαστών, εφαρμογών benchmark και αρχιτεκτονικών μνήμης για τη πειραματική διαδικασία.

Οι επεξεργαστές που επιλέχθηκαν για τη διεξαγωγή των πειραμάτων είναι οι ARM7TDMI, ARM CORTEX A5, ARM CORTEX A9 και ARM CORTEX M4. Οι επεξεργαστές αυτοί επιλέχθηκαν ως ένα αντιπροσωπευτικό δείγμα σύγχρονων επεξεργαστών με ευρεία χρήση σε ποικίλους τομείς και πληθώρα εφαρμογών.

Επιλέχθηκαν συνολικά 18 εφαρμογές benchmark με στόχο να καλύψουν ένα μεγάλο εύρος εφαρμογών. Παρακάτω δίνεται ένας πίνακας με μία σύντομη περιγραφή των benchmark που χρησιμοποιήθηκαν.

BENCHMARK	**ΣΥΝΤΟΜΟΓΡΑΦΙΑ**	**ΠΕΔΙΟ ΕΦΑΡΜΟΓΩΝ**
1D Wavelet Transformation	WAVELET	Επεξεργασία εικόνας
Adaptive differential pulse-code modulation decoder	ADPCM_DEC	Επεξεργασία σήματος
Adaptive differential pulse-code modulation encoder	ADPCM_ENC	Επεξεργασία σήματος
Blowfish Decoder	BLOWFISH_DEC	Κρυπτογραφία
Blowfish Encoder	BLOWFISH_ENC	Κρυπτογραφία
Cavity Detector	CAVITY	Επεξεργασία εικόνας
EEMBC CoreMark	COREMARK	Έλεγχος λειτουργικότητας επεξεργαστών
Fast Fourier Transformation	FFT	Επεξεργασία σήματος
Full Search	FS	Εκτίμηση κίνησης
Hierarchical Search	HS	Εκτίμηση κίνησης
Jpeg Decoder	DJPEG	Αποκωδικοποιητής jpeg
Jpeg Encoder	CJPEG	Κωδικοποιητής jpeg
Lame mp3 Encoder	LAME	Κωδικοποιητής mp3
Parallel Hierarchical One Dimensional Search	PHODS	Εκτίμηση κίνησης
Secure Hash Algorithm	SHA	Ασφάλεια επικοινωνιών - Κρυπτογραφία
Smallest Univalue Segment Assimilating Nucleus	SUSAN	Επεξεργασία εικόνας
Spiral Search	SS	Εκτίμηση κίνησης
Three Step Logarithmic Step	3SLOG	Εκτίμηση κίνησης

Πίνακας 5.1: Τα benchmarks που χρησιμοποιήθηκαν για την αποτίμηση των νέων εργαλείων

Τέλος οι αρχιτεκτονικές μνήμης που επιλέχθηκαν για προσομοίωση, είναι όλοι οι δυνατοί συνδυασμοί των παρακάτω παραμέτρων (συνολικά 23 συνδυασμοί):

Μέγεθος κρυφής μνήμης: 64 bytes, 128 bytes, 256 bytes, 512 bytes

Μέγεθος block μνήμης: 8 bytes, 32 bytes

Βαθμός συσχετικότητας: 1 (άμεσης αντιστοίχισης), 2, 4

Οι αρχιτεκτονικές που δοκιμάστηκαν είναι ενός επιπέδου, όμως και για μονάδες μνήμης υψηλότερων επιπέδων εφαρμόζεται αναδρομικά η ίδια ακριβώς μεθοδολογία, επομένως μπορούμε να θεωρήσουμε ότι τα αποτελέσματα της αποτίμησης είναι αντιπροσωπευτικά.

5.3 Αποτελέσματα πειραματικής διαδικασίας

5.3.1 Εκτίμηση πλήθους εκτελούμενων εντολών

Στους επόμενους πίνακες περιγράφεται η ακρίβεια που πέτυχαν τα νέα εργαλεία στην εκτίμηση του αριθμού εντολών που εκτελέσθηκαν για κάθε benchmark και επεξεργαστή. Το συνολικό μέσο σφάλμα είναι **3,2%**.

Benchmark	Αριθμός εντολών	Εκτίμηση αριθμού εντολών	Σφάλμα %
WAVELET	46309168	46309173	0,0
ADPCM_DEC	60535754	59816682	1,2
ADPCM_ENC	79198062	80196062	1,3
BLOWFISH_DEC	70417136	73175967	3,9
BLOWFISH_ENC	69497122	71684753	3,1
CAVITY	9537142308	9537142309	0,0
COREMARK	1182512655	1148511675	2,9
FFT	911227	973133	6,8
FS	549773604	553268277	0,6
HS	46177577	47629348	3,1
DJPEG	53247641	52783625	0,9
CJPEG	217470175	210756240	3,1
LAME	287470175	266811139	7,2
PHODS	47427876	49072437	3,5
SHA	35789376	36616273	2,3
SUSAN	217470175	210756240	3,1
SS	519260147	553268277	6,5
3SLOG	42283127	44777806	5,9
Μέση τιμή σφάλματος			**3,1**

Πίνακας 5.2: Εκτιμώμενοι αριθμοί εντολών για τον επεξεργαστή ARM7TDMI

Benchmark	Αριθμός εντολών	Εκτίμηση αριθμού εντολών	Σφάλμα %
WAVELET	48528799	48548819	0,0
ADPCM_DEC	62261283	62172565	0,1
ADPCM_ENC	79211826	79521058	0,4
BLOWFISH_DEC	58236356	54995138	5,6
BLOWFISH_ENC	57604785	54371485	5,6
CAVITY	8493352462	8390852246	1,2
COREMARK	1182191063	1206141529	2,0
FFT	948429	948322	0,0
FS	505059638	519095541	2,8
HS	43161468	42638842	1,2
DJPEG	43307771	41662332	3,8
CJPEG	162721599	158707801	2,5
LAME	317467298	358024922	12,8
PHODS	42875038	42590867	0,7
SHA	36443421	35300823	3,1
SUSAN	4623477	4711418	1,9
SS	464326531	482765204	4,0
3SLOG	35317208	36705619	3,9
Μέση τιμή σφάλματος			**2,9**

Πίνακας 5.3: Εκτιμώμενοι αριθμοί εντολών για τον επεξεργαστή ARM CORTEX A5

Benchmark	Αριθμός εντολών	Εκτίμηση αριθμού εντολών	Σφάλμα %
WAVELET	48528799	48548819	0,0
ADPCM_DEC	62261283	61536516	1,2
ADPCM_ENC	79198062	80196062	1,3
BLOWFISH_DEC	58236356	54992061	5,6
BLOWFISH_ENC	57604785	54368408	5,6
CAVITY	8493352462	8390852246	1,2
COREMARK	1182191225	1258078144	6,4
FFT	948429	948322	0,0
FS	505059638	519095541	2,8
HS	43164636	42638842	1,2
DJPEG	43307771	40289379	7,0
CJPEG	162721599	158707801	2,5
LAME	317467298	357975567	12,8
PHODS	48528799	42875038	11,7
SHA	36443421	37322863	2,4
SUSAN	4623477	4711418	1,9
SS	463102936	482765204	4,3
3SLOG	36540767	36705619	0,5
Μέση τιμή σφάλματος			**3,8**

Πίνακας 5.4: Εκτιμώμενοι αριθμοί εντολών για τον επεξεργαστή ARM CORTEX A9

Benchmark	Αριθμός εντολών	Εκτίμηση αριθμού εντολών	Σφάλμα %
WAVELET	46091687	43837472	4,9
ADPCM_DEC	62546114	61536516	1,6
ADPCM_ENC	79898299	79520389	0,5
BLOWFISH_DEC	58236356	50665360	13,0
BLOWFISH_ENC	51731263	49721987	3,9
CAVITY	8493352462	8390852246	1,2
COREMARK	1202564200	1253666084	4,3
FFT	948429	948322	0,0
FS	911691428	937293782	2,8
HS	37225194	36372349	2,3
DJPEG	41203094	40141469	2,6
CJPEG	142059445	139176150	2,0
LAME	267157069	256314641	4,1
PHODS	55496042	55521776	0,1
SHA	26323829	27187023	3,3
SUSAN	4623678	4711782	1,9
SS	464937125	482765201	3,8
3SLOG	37835277	36705639	3,0
Μέση τιμή σφάλματος			**3,1**

Πίνακας 5.5: Εκτιμώμενοι αριθμοί εντολών για τον επεξεργαστή ARM CORTEX M4

5.3.2 Εκτίμηση ρυθμού αστοχίας (συνοπτικά)

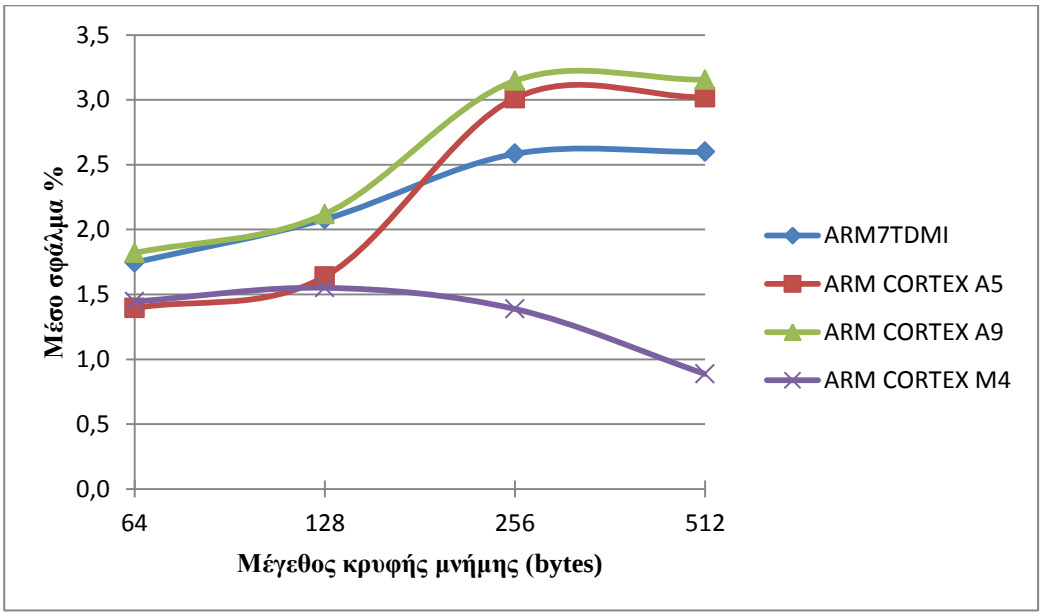
Στους επόμενους πίνακες περιγράφεται συνοπτικά η ακρίβεια που πέτυχαν τα νέα εργαλεία στη εκτίμηση του ρυθμού αστοχίας της κρυφής μνήμης. Παρουσιάζονται επίσης διαγράμματα στα οποία δίνεται το μέσο σφάλμα των εργαλείων ως συνάρτηση των χαρακτηριστικών της κρυφής μνήμης (μέγεθος, μέγεθος block μνήμης, βαθμός συσχετικότητας). Το συνολικό μέσο σφάλμα στην εκτίμηση του ρυθμού αστοχίας είναι **2,1%**.

Benchmark	ARM7TDMI	ARM CORTEX A5	ARM CORTEX A9	ARM CORTEX M4	Μέση τιμή για το benchmark
WAVELET	0,1	0,2	0,2	2,6	**0,8**
ADPCM_DEC	0,3	0,4	0,5	2,3	**0,9**
ADPCM_ENC	0,5	0,3	1,5	0,4	**0,7**
BLOWFISH_DEC	9,3	6,5	6,6	2,3	**6,2**
BLOWFISH_ENC	9,3	6,6	6,7	1,9	**6,1**
CAVITY	1,5	0,4	0,4	0,4	**0,7**
COREMARK	1,2	3,5	4,0	1,0	**2,4**
FFT	0,6	3,8	3,8	1,1	**2,3**
FS	0,4	0,7	0,7	0,5	**0,6**
HS	1,7	1,9	1,9	1,1	**1,7**
DJPEG	1,0	1,8	2,3	1,5	**1,7**
CJPEG	3,1	3,1	3,7	1,3	**2,8**
LAME	3,4	3,3	3,3	1,7	**2,9**
PHODS	1,6	0,7	2,6	0,6	**1,4**
SHA	1,6	1,3	2,1	2,5	**1,9**
SUSAN	3,1	5,4	5,4	1,3	**3,8**
SS	1,2	0,9	1,0	0,2	**0,8**
3SLOG	1,4	1,0	0,5	0,4	**0,8**
Μέση τιμή για τον επεξεργαστή	**2,3**	**2,3**	**2,6**	**1,3**	**2,1**

Πίνακας 5.6: Μέσα σφάλματα εκτίμησης ρυθμού αστοχίας για κάθε benchmark και επεξεργαστή

Μέγεθος κρυφής μνήμης (bytes)	Σφάλμα εκτίμησης ρυθμού αστοχίας %				
	ARM7TDMI	ARM CORTEX A5	ARM CORTEX A9	ARM CORTEX M4	Μέση τιμή
64	1,7	1,4	1,8	1,4	**1,6**
128	2,1	1,6	2,1	1,6	**1,8**
256	2,6	3,0	3,1	1,4	**2,5**
512	2,6	3,0	3,2	0,9	**2,4**

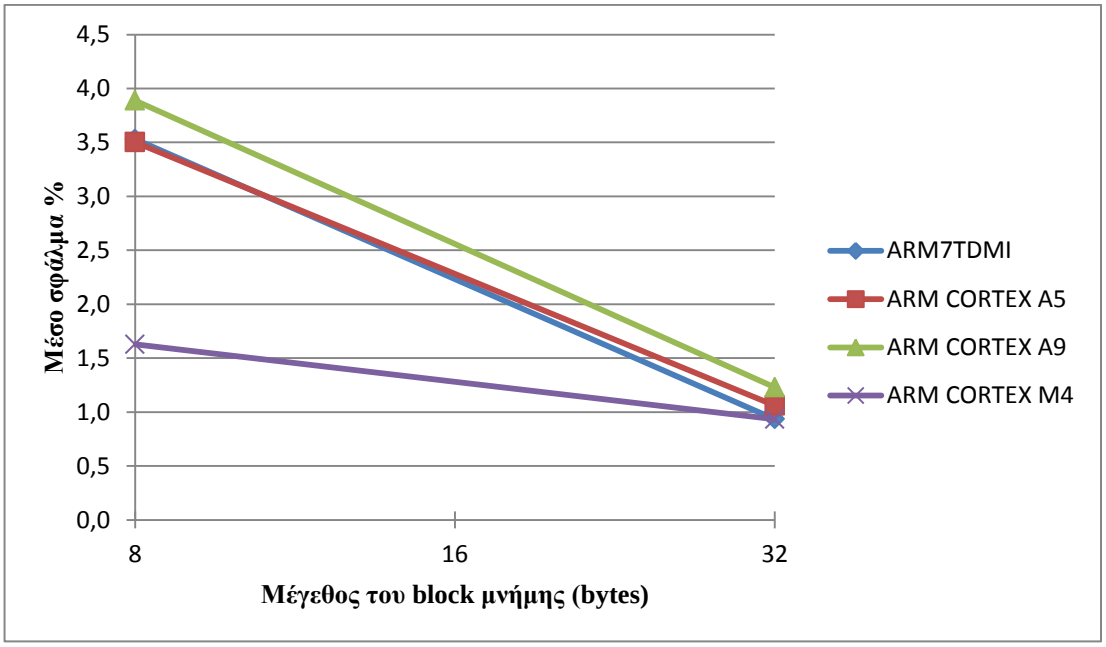
Πίνακας 5.7: Το σφάλμα εκτίμησης ρυθμού αστοχίας ως συνάρτηση του μεγέθους της κρυφής μνήμης



Σχήμα 5.2: Το σφάλμα εκτίμησης ρυθμού αστοχίας ως συνάρτηση του μεγέθους της κρυφής μνήμης

Μέγεθος block κρυφής μνήμης (bytes)	Σφάλμα εκτίμησης ρυθμού αστοχίας %				
	ARM7TDMI	ARM CORTEX A5	ARM CORTEX A9	ARM CORTEX M4	Μέση τιμή
8	3,5	3,5	3,9	1,6	**3,1**
32	0,9	1,1	1,2	0,9	**1,0**

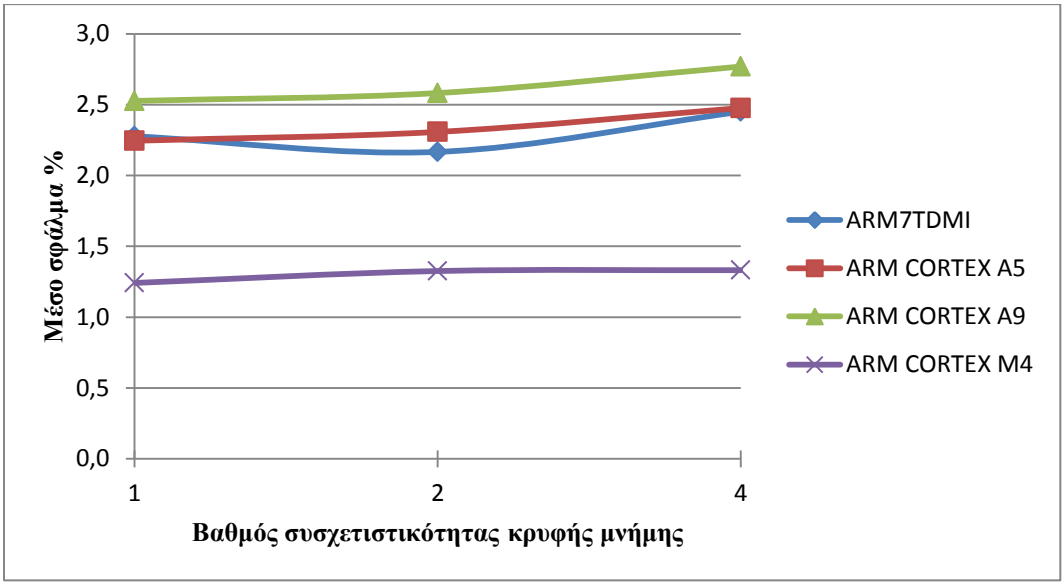
Πίνακας 5.8: Το σφάλμα εκτίμησης ρυθμού αστοχίας ως συνάρτηση του μεγέθους block της κρυφής μνήμης



Σχήμα 5.3: Το σφάλμα εκτίμησης ρυθμού αστοχίας ως συνάρτηση του μεγέθους block της κρυφής μνήμης

Βαθμός συσχετικότητας	Σφάλμα εκτίμησης ρυθμού αστοχίας %				
	ARM7TDMI	ARM CORTEX A5	ARM CORTEX A9	ARM CORTEX M4	Μέση τιμή
1	2,3	2,2	2,5	1,2	**2,1**
2	2,2	2,3	2,6	1,3	**2,1**
4	2,5	2,5	2,8	1,3	**2,3**

Πίνακας 5.9: Το σφάλμα εκτίμησης ρυθμού αστοχίας ως συνάρτηση του βαθμού συσχετικότητας της μνήμης



Σχήμα 5.4: Το σφάλμα εκτίμησης ρυθμού αστοχίας ως συνάρτηση του βαθμού συσχετικότητας της μνήμης

5.3.3 Μετρήσεις ταχύτητας των νέων εργαλείων

Στους επόμενους πίνακες παρουσιάζεται η ταχύτητα των νέων εργαλείων σε σύγκριση με τον συνδυασμό εργαλείων προσομοίωσης OVP-Dinero. Σύμφωνα με τα πειραματικά αποτελέσματα τα νέα εργαλεία πέτυχαν ταχύτητες έως και 21248 φορές υψηλότερες από τα εργαλεία OVP και Dinero. Οι χρόνοι εκτίμησης περιέχουν και το άθροισμα των χρόνων μεταγλώττισης και για τις δύο αρχιτεκτονικές

Benchmark	ARM7TDMI			ARM CORTEX A5		
	Χρόνος προσομοίωσης OVP-DINERO (s)	Χρόνος εκτίμησης νέων εργαλείων (s)	Αύξηση ταχύτητας	Χρόνος προσομοίωσης OVP-DINERO (s)	Χρόνος εκτίμησης νέων εργαλείων (s)	Αύξηση ταχύτητας
WAVELET	246,613	0,845	291,8	301,972	0,953	316,9
ADPCM_DEC	417,003	0,602	692,7	330,725	0,704	469,8
ADPCM_ENC	391,664	0,594	659,4	938,286	0,724	1296,0
BLOWFISH_DEC	505,918	1,84	275,0	510,747	2,129	239,9
BLOWFISH_ENC	458,585	1,86	246,6	430,738	2,097	205,4
CAVITY	43622,256	3,248	13430,5	68195,001	3,344	20393,2
COREMARK	5728,523	2,614	2191,5	5226,334	2,99	1747,9
FFT	105,445	0,416	253,5	53,476	0,537	99,6
FS	2557,918	0,588	4350,2	2829,74	0,671	4217,2
HS	259,523	0,609	426,1	245,303	0,689	356,0
DJPEG	440,265	18,029	24,4	461,347	18,745	24,6
CJPEG	1298,589	25,782	50,4	1458,322	27,466	53,1
LAME	7216,224	17,694	407,8	7512,833	22,479	334,2
PHODS	245,743	0,469	524,0	260,623	0,56	465,4
SHA	210,845	0,729	289,2	226,109	0,851	265,7
SUSAN	45,081	1,934	23,3	300,347	14,62	20,5
SS	2298,981	0,657	3499,2	2598,774	0,741	3507,1
3SLOG	215,29	0,472	456,1	227,031	0,58	391,4
Μέση αύξηση ταχύτητας			**1560,6**			**1911,3**
Μέγιστη αύξηση ταχύτητας			**13430,5**			**20393,2**

Πίνακας 5.10: Αύξηση ταχύτητας των νέων εργαλείων σε σχέση με τον προσομοίωση OVP-DINERO για τους επεξεργαστές ARM7TDMI και ARM CORTEX A5

Benchmark	ARM CORTEX A9			ARM CORTEX M4		
	Χρόνος προσομοίωσης OVP-DINERO (s)	Χρόνος εκτίμησης νέων εργαλείων (s)	Αύξηση ταχύτητας	Χρόνος προσομοίωσης OVP-DINERO (s)	Χρόνος εκτίμησης νέων εργαλείων (s)	Αύξηση ταχύτητας
WAVELET	300,006	0,911	329,3	304,738	0,993	306,9
ADPCM_DEC	356,943	0,681	524,1	328,946	0,756	435,1
ADPCM_ENC	750,6	0,709	1058,7	932,582	0,834	1118,2
BLOWFISH_DEC	739,797	2,06	359,1	516,576	2,43	212,6
BLOWFISH_ENC	818,131	2,205	371,0	436,511	2,237	195,1
CAVITY	54674,48	3,293	16603,2	68207,712	3,21	21248,5
COREMARK	4805,734	3,065	1567,9	5247,813	2,84	1847,8
FFT	79,588	0,538	147,9	53,002	0,521	101,7
FS	1121,78	0,624	1797,7	2840,798	0,672	4227,4
HS	427,365	0,642	665,7	248,258	0,678	366,2
DJPEG	555,75	18,73	29,7	467,141	18,123	25,8
CJPEG	1200,674	27,362	43,9	1442,287	26,416	54,6
LAME	7289,024	22,364	325,9	7532,619	21,442	351,3
PHODS	245,602	0,533	460,8	259,622	0,534	486,2
SHA	539,207	0,814	662,4	227,151	0,82	277,0
SUSAN	119,912	14,493	8,3	300,635	15,1	19,9
SS	2617,136	0,704	3717,5	2582,724	0,741	3485,5
3SLOG	218,86	0,527	415,3	228,191	0,592	385,5
Μέση αύξηση ταχύτητας			**1616,0**			**1952,5**
Μέγιστη αύξηση ταχύτητας			**16603,2**			**21248,5**

Πίνακας 5.11: Αύξηση ταχύτητας των νέων εργαλείων σε σχέση με τον προσομοίωση OVP-DINERO για τους επεξεργαστές ARM CORTEX A9 και ARM CORTEX M4

5.3.4 Αναλυτικά αποτελέσματα εκτίμησης ρυθμού αστοχίας

Στους πίνακες των επόμενων σελίδων παρουσιάζονται αναλυτικά τα αποτελέσματα της πειραματικής διαδικασίας (εκτίμηση ρυθμού αστοχίας) για όλα τα benchmark, τους επεξεργαστές και τις αρχιτεκτονικές μνήμης. Για λόγους εξοικονόμησης χώρου χρησιμοποιούνται οι παρακάτω συμβολισμοί:

S: Το συνολικό μέγεθος της κρυφής μνήμης

BS: Το μέγεθος του block μνήμης

A: Ο βαθμός συσχετικότητας της μνήμης

MR: Ρυθμός αστοχίας όπως αυτός προκύπτει από τη προσομοίωση

EMR: Εκτιμούμενος ρυθμός αστοχίας από τα νέα εργαλεία

ER: Σφάλμα εκτίμησης %

S	BS	A	ARM7TDMI			ARM CORTEX A5			ARM CORTEX A9			ARM CORTEX M4		
			MR	EMR	ER	MR	EMR	ER	MR	EMR	ER	MR	EMR	ER
64	8	1	42,4	42,4	0,0	43,4	43,9	0,5	43,4	43,9	0,5	30,6	35,5	4,9
		2	43,8	43,8	0,0	45,4	45,9	0,5	45,4	45,9	0,5	34,8	40,1	5,3
		4	46,2	46,2	0,0	46,5	47,0	0,5	46,5	47,0	0,5	36,3	39,0	2,7
	32	1	14,4	14,4	0,0	14,1	14,3	0,2	14,1	14,3	0,2	10,3	15,5	5,2
		2	15,7	15,7	0,0	15,3	15,5	0,2	15,3	15,5	0,2	11,0	16,2	5,3
128	8	1	22,3	21,8	0,5	24,7	24,8	0,1	24,7	24,8	0,1	8,6	14,8	6,2
		2	28,3	27,8	0,5	30,5	30,6	0,1	30,5	30,6	0,1	12,3	20,4	8,1
		4	32,0	31,7	0,3	35,2	35,4	0,2	35,2	35,4	0,2	16,8	23,1	6,4
	32	1	8,4	8,4	0,0	8,7	8,9	0,2	8,7	8,9	0,2	3,8	6,9	3,1
		2	9,8	9,8	0,0	10,4	10,6	0,2	10,4	10,6	0,2	5,3	8,8	3,5
		4	11,4	11,5	0,0	11,3	11,5	0,2	11,3	11,5	0,2	5,5	9,9	4,4
256	8	1	0,9	0,9	0,1	0,9	1,0	0,1	0,9	1,0	0,1	0,5	1,1	0,6
		2	0,9	0,9	0,0	1,1	0,9	0,2	1,1	0,9	0,2	0,5	1,2	0,7
		4	0,9	0,8	0,1	1,2	1,0	0,3	1,2	1,0	0,3	0,5	1,3	0,9
	32	1	0,5	0,6	0,1	0,5	0,6	0,1	0,5	0,6	0,1	0,3	0,7	0,4
		2	0,6	0,6	0,0	0,6	0,7	0,0	0,6	0,7	0,0	0,2	0,7	0,5
		4	0,4	0,4	0,0	0,5	0,5	0,0	0,5	0,5	0,0	0,2	0,5	0,3
512	8	1	0,5	0,3	0,2	0,5	0,4	0,1	0,5	0,4	0,1	0,2	0,4	0,2
		2	0,5	0,2	0,4	0,5	0,1	0,3	0,5	0,1	0,3	0,3	0,2	0,1
		4	0,5	0,2	0,3	0,5	0,2	0,3	0,5	0,2	0,3	0,4	0,1	0,2
	32	1	0,4	0,4	0,0	0,2	0,2	0,0	0,2	0,2	0,0	0,2	0,2	0,0
		2	0,2	0,1	0,1	0,2	0,1	0,0	0,2	0,1	0,0	0,1	0,1	0,0
		4	0,2	0,1	0,1	0,1	0,1	0,0	0,1	0,1	0,0	0,1	0,1	0,0
Μέσο σφάλμα:					**0,1**			**0,2**			**0,2**			**2,6**

Πίνακας 5.12: Τα πειραματικά αποτελέσματα του benchmark WAVELET

S	BS	A	ARM7TDMI			ARM CORTEX A5			ARM CORTEX A9			ARM CORTEX M4		
			MR	EMR	ER	MR	EMR	ER	MR	EMR	ER	MR	EMR	ER
64	8	1	53,8	54,4	0,6	55,1	54,8	0,2	55,1	55,4	0,4	36,0	39,3	3,3
		2	53,8	54,4	0,6	55,1	54,8	0,2	55,1	55,4	0,4	36,0	39,3	3,3
		4	53,8	54,4	0,6	55,1	54,8	0,2	55,1	55,4	0,4	36,0	39,3	3,3
	32	1	18,1	18,3	0,2	17,1	17,6	0,5	17,1	17,8	0,7	11,0	12,2	1,3
		2	18,1	18,3	0,2	17,1	17,6	0,5	17,1	17,8	0,7	11,0	12,2	1,3
128	8	1	53,1	53,6	0,6	55,1	54,8	0,2	55,1	55,4	0,4	32,8	37,2	4,4
		2	53,8	54,4	0,6	55,1	54,8	0,2	55,1	55,4	0,4	34,7	39,3	4,5
		4	53,8	54,4	0,6	55,1	54,8	0,2	55,1	55,4	0,4	36,0	39,3	3,3
	32	1	18,1	18,3	0,2	17,1	17,6	0,5	17,1	17,8	0,7	11,0	12,2	1,3
		2	18,1	18,3	0,2	17,1	17,6	0,5	17,1	17,8	0,7	11,0	12,2	1,3
		4	18,1	18,3	0,2	17,1	17,6	0,5	17,1	17,8	0,7	11,0	12,2	1,3
256	8	1	40,1	39,4	0,8	42,3	41,0	1,3	42,3	41,5	0,8	8,8	12,2	3,4
		2	46,4	46,8	0,5	50,2	48,6	1,6	50,2	49,1	1,1	12,4	17,3	4,9
		4	50,8	51,3	0,5	54,6	54,0	0,6	54,6	54,5	0,0	16,7	26,1	9,3
	32	1	18,1	18,3	0,2	16,5	17,6	1,1	16,5	17,8	1,3	4,4	6,7	2,3
		2	18,1	18,3	0,2	17,1	17,6	0,5	17,1	17,8	0,7	6,6	10,0	3,4
		4	18,1	18,3	0,2	17,1	17,6	0,5	17,1	17,8	0,7	11,0	12,2	1,3
512	8	1	0,1	0,0	0,1	0,1	0,0	0,1	0,1	0,0	0,1	0,0	0,0	0,0
		2	0,1	0,0	0,1	0,1	0,0	0,1	0,1	0,0	0,1	0,1	0,0	0,1
		4	0,1	0,0	0,1	0,1	0,0	0,1	0,1	0,0	0,1	0,1	0,0	0,1
	32	1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
		2	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
		4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
Μέσο σφάλμα:					**0,3**			**0,4**			**0,5**			**2,3**

Πίνακας 5.13: Τα πειραματικά αποτελέσματα του benchmark ADPCM_DEC

S	BS	A	ARM7TDMI			ARM CORTEX A5			ARM CORTEX A9			ARM CORTEX M4		
			MR	EMR	ER	MR	EMR	ER	MR	EMR	ER	MR	EMR	ER
64	8	1	54,0	54,3	0,3	54,2	53,9	0,3	54,9	53,5	1,4	36,9	37,1	0,1
		2	54,0	54,3	0,3	54,2	53,9	0,3	54,9	53,5	1,4	36,9	37,1	0,1
		4	54,0	54,3	0,3	54,2	53,9	0,3	54,9	53,5	1,4	36,9	37,1	0,1
	32	1	16,5	16,8	0,3	17,2	17,1	0,1	17,7	17,0	0,8	11,2	12,0	0,9
		2	16,5	16,8	0,3	17,2	17,1	0,1	17,7	17,0	0,8	11,2	12,0	0,9
128	8	1	53,8	54,2	0,4	54,2	53,9	0,3	54,6	53,5	1,1	36,2	36,4	0,1
		2	54,0	54,3	0,3	54,2	53,9	0,3	54,9	53,5	1,4	36,9	37,1	0,1
		4	54,0	54,3	0,3	54,2	53,9	0,3	54,9	53,5	1,4	36,9	37,1	0,1
	32	1	16,5	16,8	0,3	17,2	17,1	0,1	17,7	17,0	0,8	11,2	12,0	0,9
		2	16,5	16,8	0,3	17,2	17,1	0,1	17,7	17,0	0,8	11,2	12,0	0,9
		4	16,5	16,8	0,3	17,2	17,1	0,1	17,7	17,0	0,8	11,2	12,0	0,9
256	8	1	47,9	47,9	0,0	47,6	47,2	0,5	48,9	46,8	2,1	24,2	24,0	0,2
		2	52,3	51,8	0,6	52,4	51,9	0,5	52,4	51,5	0,9	29,6	29,4	0,2
		4	53,8	53,5	0,3	54,2	53,8	0,4	53,9	53,3	0,5	32,2	32,3	0,1
	32	1	15,3	16,1	0,8	16,7	16,6	0,1	17,7	16,4	1,3	8,6	10,3	1,7
		2	16,5	16,8	0,3	17,2	17,1	0,1	17,7	17,0	0,8	11,2	12,0	0,9
		4	16,5	16,8	0,3	17,2	17,1	0,1	17,7	17,0	0,8	11,2	12,0	0,9
512	8	1	22,2	23,5	1,3	21,7	21,5	0,2	22,3	21,3	1,0	0,1	0,0	0,1
		2	24,3	25,6	1,3	25,3	24,2	1,2	27,7	24,0	3,7	0,1	0,0	0,1
		4	24,7	24,0	0,7	19,0	17,2	1,8	23,9	17,1	6,9	0,1	0,0	0,1
	32	1	7,8	7,7	0,1	7,8	7,7	0,1	7,8	7,7	0,1	0,0	0,0	0,0
		2	11,7	11,5	0,2	11,7	11,6	0,1	11,7	11,5	0,2	0,0	0,0	0,0
		4	9,8	12,3	2,5	13,3	13,3	0,1	16,0	13,1	2,9	0,0	0,0	0,0
Μέσο σφάλμα:					**0,5**			**0,3**			**1,5**			**0,4**

Πίνακας 5.14: Τα πειραματικά αποτελέσματα του benchmark ADPCM_ENC

S	BS	A	ARM7TDMI			ARM CORTEX A5			ARM CORTEX A9			ARM CORTEX M4		
			MR	EMR	ER	MR	EMR	ER	MR	EMR	ER	MR	EMR	ER
64	8	1	49,9	39,9	10,0	49,4	50,3	0,9	49,4	49,1	0,3	34,2	36,6	2,5
		2	51,2	41,6	9,6	51,0	52,0	1,0	51,0	50,8	0,1	38,1	44,9	6,8
		4	51,2	41,6	9,6	51,0	52,0	1,0	51,0	50,8	0,1	38,1	45,4	7,2
	32	1	14,2	13,0	1,2	15,1	16,0	0,9	15,1	16,0	0,9	10,5	9,4	1,1
		2	14,2	13,0	1,2	15,1	16,0	0,9	15,1	16,0	0,9	11,0	10,4	0,6
128	8	1	39,1	29,3	9,7	33,8	30,6	3,2	33,8	28,3	5,5	19,3	19,3	0,0
		2	44,0	34,2	9,8	39,4	37,3	2,1	39,4	35,0	4,4	20,8	22,6	1,8
		4	45,6	36,5	9,0	40,4	40,6	0,3	40,4	39,5	0,9	21,6	25,7	4,1
	32	1	11,7	10,5	1,2	11,5	12,1	0,6	11,5	12,1	0,6	6,4	7,9	1,6
		2	12,5	11,3	1,2	13,0	13,7	0,7	13,0	13,7	0,7	7,4	6,2	1,2
		4	12,5	11,3	1,2	13,0	13,7	0,7	13,0	13,7	0,7	7,0	5,4	1,6
256	8	1	25,8	5,1	20,6	24,5	7,5	17,0	24,5	7,5	17,0	16,2	15,6	0,6
		2	25,8	5,1	20,7	22,6	5,5	17,1	22,6	5,5	17,1	16,3	20,8	4,5
		4	25,8	5,0	20,8	22,6	5,5	17,1	22,6	5,5	17,1	16,3	20,3	4,0
	32	1	6,7	3,0	3,6	7,9	3,5	4,4	7,9	3,5	4,4	4,3	1,6	2,7
		2	6,7	3,9	2,8	7,4	3,0	4,4	7,4	3,0	4,4	4,4	6,9	2,6
		4	6,7	5,6	1,1	6,0	1,5	4,5	6,0	1,5	4,5	4,4	5,6	1,3
512	8	1	25,7	3,2	22,5	24,5	4,0	20,5	24,5	4,0	20,5	16,2	16,7	0,5
		2	25,8	4,8	21,0	22,6	3,0	19,7	22,6	3,0	19,7	16,2	19,3	3,1
		4	25,8	5,0	20,7	22,6	5,0	17,7	22,6	5,0	17,7	16,2	15,9	0,3
	32	1	6,6	0,9	5,8	7,8	2,7	5,1	7,8	2,7	5,1	4,3	4,1	0,2
		2	6,7	1,3	5,4	6,0	1,1	4,9	6,0	1,1	4,9	4,3	6,3	1,9
		4	6,7	1,3	5,4	6,0	1,5	4,5	6,0	1,5	4,5	4,3	1,7	2,6
Μέσο σφάλμα:					**9,3**			**6,5**			**6,6**			**2,3**

Πίνακας 5.15: Τα πειραματικά αποτελέσματα του benchmark BLOWFISH_DEC

S	BS	A	ARM7TDMI			ARM CORTEX A5			ARM CORTEX A9			ARM CORTEX M4		
			MR	EMR	ER	EMR	EMR	ER	MR	EMR	ER	MR	EMR	ER
64	8	1	49,6	39,9	9,7	49,9	50,9	0,9	49,9	49,7	0,3	39,0	45,0	6,0
		2	51,0	41,6	9,3	51,5	52,6	1,1	51,5	51,4	0,1	43,5	42,6	0,9
		4	51,0	41,6	9,3	51,5	52,6	1,1	51,5	51,4	0,1	43,5	43,3	0,2
	32	1	14,4	13,3	1,1	14,6	15,6	0,9	14,6	15,6	0,9	12,9	10,6	2,4
		2	14,4	13,3	1,1	14,6	15,6	0,9	14,6	15,6	0,9	12,9	13,9	1,0
128	8	1	37,8	28,3	9,6	34,2	30,9	3,2	34,2	28,6	5,6	22,8	25,5	2,7
		2	42,1	32,5	9,6	39,9	37,7	2,1	39,9	35,4	4,5	25,0	27,9	2,8
		4	45,2	36,4	8,8	40,8	41,1	0,3	40,8	39,9	0,9	24,3	25,8	1,5
	32	1	11,8	10,7	1,1	11,0	11,7	0,7	11,0	11,7	0,7	7,2	7,2	0,0
		2	12,6	11,5	1,1	12,5	13,3	0,8	12,5	13,3	0,8	8,3	8,3	0,0
		4	12,6	11,5	1,1	12,5	13,3	0,8	12,5	13,3	0,8	10,5	10,3	0,2
256	8	1	25,9	5,1	20,8	24,8	7,6	17,2	24,8	7,6	17,2	18,3	22,2	3,9
		2	26,0	5,1	20,9	22,9	5,6	17,3	22,9	5,6	17,3	18,3	21,2	2,8
		4	26,0	5,0	20,9	22,9	5,6	17,3	22,9	5,6	17,3	18,3	24,2	5,9
	32	1	6,7	3,1	3,7	6,9	2,4	4,4	6,9	2,4	4,4	4,9	6,7	1,9
		2	6,7	4,0	2,8	5,9	1,4	4,5	5,9	1,4	4,5	4,9	7,4	2,5
		4	6,7	5,7	1,0	5,9	1,4	4,5	5,9	1,4	4,5	4,9	5,2	0,3
512	8	1	25,9	3,0	22,9	24,7	4,0	20,7	24,8	4,0	20,7	18,2	19,9	1,7
		2	25,9	4,6	21,4	22,9	3,0	19,9	22,9	3,0	19,9	18,2	19,2	1,0
		4	25,9	5,0	20,9	22,9	5,0	17,8	22,9	5,0	17,9	18,3	20,8	2,5
	32	1	6,7	0,9	5,8	6,9	1,6	5,3	6,9	1,6	5,3	4,8	3,8	1,1
		2	6,7	1,3	5,4	5,9	0,9	5,1	5,9	0,9	5,1	4,8	2,2	2,7
		4	6,7	1,3	5,4	5,9	1,4	4,5	5,9	1,4	4,5	4,9	4,4	0,4
Μέσο σφάλμα:					**9,3**			**6,6**			**6,7**			**1,9**

Πίνακας 5.16: Τα πειραματικά αποτελέσματα του benchmark BLOWFISH_ENC

S	BS	A	ARM7TDMI			ARM CORTEX A5			ARM CORTEX A9			ARM CORTEX M4		
			MR	EMR	ER	MR	EMR	ER	MR	EMR	ER	MR	EMR	ER
64	8	1	51,0	51,2	0,3	50,9	48,5	2,5	50,9	48,5	2,5	83,1	84,1	0,9
		2	52,3	52,6	0,3	51,1	51,6	0,5	51,1	51,6	0,5	84,7	84,7	0,1
		4	52,3	52,6	0,3	56,9	52,8	4,1	56,9	52,8	4,1	84,9	84,7	0,2
	32	1	16,3	16,5	0,3	16,1	15,7	0,4	16,1	15,7	0,4	26,8	26,2	0,6
		2	16,3	16,5	0,3	16,4	16,3	0,1	16,4	16,3	0,1	27,0	26,2	0,8
128	8	1	31,2	20,2	11,0	15,8	14,8	1,0	15,8	14,8	1,0	64,8	64,2	0,6
		2	36,4	29,7	6,7	21,5	21,5	0,0	21,5	21,5	0,0	77,6	75,3	2,3
		4	40,4	39,8	0,6	30,6	30,4	0,2	30,6	30,4	0,2	76,4	75,3	1,1
	32	1	11,9	11,1	0,8	8,6	8,7	0,1	8,6	8,7	0,1	24,3	23,7	0,6
		2	13,8	14,3	0,5	12,8	12,6	0,1	12,8	12,6	0,1	24,4	24,3	0,1
		4	13,8	14,3	0,5	14,6	14,5	0,1	14,6	14,5	0,1	24,9	24,9	0,0
256	8	1	13,5	14,4	0,9	3,9	3,8	0,2	3,9	3,8	0,2	14,7	14,1	0,5
		2	17,9	12,4	5,5	0,9	0,9	0,0	0,9	0,9	0,0	20,9	20,7	0,2
		4	22,2	23,0	0,8	1,1	1,1	0,0	1,1	1,1	0,0	20,3	19,6	0,7
	32	1	4,7	1,9	2,8	2,2	2,1	0,1	2,2	2,1	0,1	7,8	7,7	0,0
		2	5,8	4,6	1,3	1,3	1,2	0,0	1,3	1,2	0,0	11,8	11,4	0,4
		4	6,9	5,5	1,3	0,3	0,3	0,0	0,3	0,3	0,0	17,5	17,2	0,4
512	8	1	0,3	0,3	0,1	0,0	0,0	0,0	0,0	0,0	0,0	1,0	1,0	0,0
		2	0,1	0,0	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,5	0,5	0,0
		4	0,1	0,0	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,9	0,8	0,0
	32	1	0,1	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,3	0,3	0,0
		2	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,1	0,1	0,0
		4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,2	0,2	0,0
Μέσο σφάλμα:					**1,5**			**0,4**			**0,4**			**0,4**

Πίνακας 5.17: Τα πειραματικά αποτελέσματα του benchmark CAVITY

S	BS	A	ARM7TDMI			ARM CORTEX A5			ARM CORTEX A9			ARM CORTEX M4		
			MR	EMR	ER	EMR	EMR	ER	MR	EMR	ER	MR	EMR	ER
64	8	1	40,0	37,3	2,6	45,7	44,5	1,2	45,7	42,7	3,0	27,5	27,4	0,1
		2	43,6	39,4	4,3	46,5	47,0	0,5	46,5	45,1	1,4	28,5	27,1	1,4
		4	43,9	41,0	2,9	47,4	48,6	1,2	47,4	46,6	0,9	29,6	28,2	1,4
	32	1	14,0	13,1	0,9	16,7	17,2	0,5	16,9	16,5	0,4	10,4	10,2	0,2
		2	15,2	13,6	1,6	17,7	18,2	0,6	17,9	17,5	0,4	10,4	9,7	0,8
128	8	1	27,6	24,3	3,3	36,0	29,4	6,6	36,0	28,2	7,8	12,8	10,9	1,9
		2	28,9	27,1	1,8	40,0	33,9	6,1	40,0	32,5	7,5	13,8	13,9	0,0
		4	32,0	29,2	2,8	41,6	36,7	4,9	41,6	35,2	6,4	15,6	16,2	0,5
	32	1	10,8	9,5	1,3	12,8	12,3	0,5	13,0	11,8	1,2	6,5	6,5	0,0
		2	12,6	11,1	1,5	13,7	14,4	0,6	14,2	13,8	0,4	7,2	7,9	0,7
		4	12,8	11,5	1,3	13,8	14,7	0,9	14,5	14,1	0,4	8,2	7,6	0,6
256	8	1	4,8	3,9	0,9	12,3	7,3	5,0	12,3	7,0	5,3	6,4	7,2	0,8
		2	1,1	1,1	0,0	13,6	3,5	10,2	13,6	3,3	10,3	5,6	6,3	0,7
		4	0,4	0,4	0,0	14,9	0,6	14,3	14,9	0,6	14,3	5,0	3,7	1,3
	32	1	2,3	2,4	0,0	5,2	4,6	0,6	5,9	4,4	1,4	3,3	2,2	1,0
		2	2,1	1,7	0,4	5,8	3,3	2,6	6,6	3,1	3,5	3,3	1,8	1,5
		4	2,2	1,5	0,8	6,1	3,0	3,1	7,0	2,8	4,1	3,4	4,3	0,9
512	8	1	0,4	0,4	0,1	7,4	1,9	5,5	7,3	1,8	5,5	3,2	0,0	3,2
		2	0,2	0,1	0,1	5,7	0,3	5,4	5,7	0,3	5,4	1,7	0,0	1,7
		4	0,1	0,0	0,1	5,4	0,0	5,4	5,5	0,0	5,5	1,0	0,0	1,0
	32	1	0,4	0,4	0,0	2,8	0,9	1,9	3,9	0,9	3,0	1,8	0,0	1,8
		2	0,5	0,5	0,0	2,0	0,7	1,3	2,7	0,7	2,1	1,6	0,0	1,6
		4	0,5	0,0	0,5	2,0	0,0	2,0	2,4	0,0	2,4	0,9	0,0	0,9
Μέσο σφάλμα:					**1,2**			**3,5**			**4,0**			**1,0**

Πίνακας 5.18: Τα πειραματικά αποτελέσματα του benchmark COREMARK

S	BS	A	ARM7TDMI			ARM CORTEX A5			ARM CORTEX A9			ARM CORTEX M4		
			MR	EMR	ER	MR	EMR	ER	MR	EMR	ER	MR	EMR	ER
64	8	1	50,4	49,7	0,7	50,4	56,5	6,1	50,4	56,5	6,1	36,4	35,0	1,4
		2	50,5	50,4	0,1	50,5	57,1	6,6	50,5	57,1	6,6	36,6	35,5	1,1
		4	50,5	50,6	0,1	50,5	57,3	6,8	50,5	57,3	6,8	36,6	36,5	0,1
	32	1	13,6	13,9	0,3	13,7	15,3	1,6	13,7	15,3	1,6	9,9	10,4	0,5
		2	13,6	13,9	0,3	13,6	15,3	1,7	13,6	15,3	1,7	9,9	8,2	1,7
128	8	1	49,3	49,1	0,2	49,2	55,6	6,4	49,2	55,6	6,4	35,4	37,0	1,6
		2	49,7	48,7	1,0	49,4	54,7	5,3	49,4	54,7	5,3	35,3	35,2	0,2
		4	50,1	48,1	2,0	50,1	54,7	4,7	50,1	54,7	4,7	35,4	36,8	1,4
	32	1	13,3	13,4	0,1	13,2	14,8	1,6	13,2	14,8	1,6	9,6	8,7	0,9
		2	13,4	13,4	0,0	13,3	14,7	1,4	13,3	14,7	1,4	9,7	10,1	0,4
		4	13,4	13,4	0,0	13,3	14,9	1,6	13,3	14,9	1,6	9,9	9,6	0,3
256	8	1	51,8	49,9	1,9	48,3	51,9	3,6	48,3	51,9	3,6	33,5	31,9	1,6
		2	51,1	51,2	0,1	48,8	52,6	3,8	48,8	52,6	3,8	34,8	33,6	1,2
		4	51,1	51,6	0,4	47,8	53,9	6,1	47,8	53,9	6,1	34,8	35,2	0,5
	32	1	13,3	13,0	0,3	12,7	13,8	1,1	12,7	13,8	1,1	9,5	11,1	1,7
		2	13,5	13,4	0,0	13,0	14,1	1,1	13,0	14,1	1,1	9,2	7,9	1,3
		4	13,1	13,3	0,2	12,4	14,3	2,0	12,4	14,3	2,0	9,2	10,2	1,0
512	8	1	22,8	20,3	2,5	26,0	31,9	5,8	26,0	31,9	5,8	3,7	5,0	1,3
		2	30,4	30,3	0,1	35,6	47,6	12,0	35,6	47,6	12,0	5,4	7,1	1,6
		4	42,1	44,0	1,9	47,7	50,5	2,8	47,7	50,5	2,8	8,6	7,1	1,5
	32	1	6,6	6,4	0,2	7,2	8,9	1,7	7,2	8,9	1,7	1,2	3,0	1,9
		2	8,7	9,6	0,8	9,7	13,4	3,7	9,7	13,4	3,7	1,7	0,2	1,5
		4	12,3	11,7	0,7	12,3	13,1	0,8	12,3	13,1	0,8	2,2	3,2	1,0
Μέσο σφάλμα:					**0,6**			**3,8**			**3,8**			**1,1**

Πίνακας 5.19: Τα πειραματικά αποτελέσματα του benchmark FFT

S	BS	A	ARM7TDMI			ARM CORTEX A5			ARM CORTEX A9			ARM CORTEX M4		
			MR	EMR	ER	MR	EMR	ER	MR	EMR	ER	EMR	EMR	ER
64	8	1	51,5	50,6	0,9	51,7	50,5	1,1	51,7	50,5	1,1	42,4	42,1	0,4
		2	51,5	50,6	0,9	51,6	50,2	1,5	51,6	50,2	1,5	42,5	42,1	0,3
		4	51,5	50,6	0,9	51,6	50,2	1,5	51,6	50,2	1,5	42,5	42,2	0,3
	32	1	13,6	13,4	0,2	14,7	14,4	0,4	14,7	14,4	0,4	11,4	11,1	0,3
		2	13,5	13,4	0,1	14,8	14,4	0,4	14,8	14,4	0,4	11,4	11,1	0,3
128	8	1	51,2	50,5	0,7	51,4	50,4	1,0	51,4	50,4	1,0	42,3	41,6	0,7
		2	51,4	50,5	0,9	51,5	50,0	1,5	51,5	50,0	1,5	42,3	41,6	0,7
		4	51,4	50,4	0,9	51,5	50,0	1,5	51,5	50,0	1,5	42,3	41,6	0,7
	32	1	13,5	13,3	0,3	14,7	14,3	0,4	14,7	14,3	0,4	11,3	11,0	0,3
		2	13,5	13,1	0,3	14,6	14,2	0,4	14,6	14,2	0,4	11,3	11,1	0,2
		4	13,5	13,1	0,3	14,6	14,2	0,4	14,6	14,2	0,4	11,3	11,1	0,2
256	8	1	37,0	36,1	0,9	33,9	32,2	1,7	33,9	32,2	1,7	40,7	40,6	0,1
		2	47,4	46,9	0,5	46,2	44,8	1,4	46,2	44,8	1,4	42,0	41,6	0,4
		4	49,3	49,4	0,0	50,4	49,2	1,2	50,4	49,2	1,2	42,2	41,6	0,6
	32	1	10,4	10,1	0,3	11,1	10,8	0,3	11,1	10,8	0,3	11,2	10,9	0,4
		2	13,4	13,2	0,1	14,4	14,1	0,3	14,4	14,1	0,3	11,2	10,9	0,4
		4	13,4	13,3	0,1	14,5	14,2	0,3	14,5	14,2	0,3	11,3	10,9	0,4
512	8	1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	8,3	7,3	1,0
		2	0,1	0,0	0,1	0,1	0,0	0,1	0,1	0,0	0,1	11,8	11,0	0,8
		4	0,1	0,0	0,1	0,0	0,0	0,0	0,0	0,0	0,0	19,3	17,4	1,9
	32	1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	2,8	2,5	0,3
		2	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	4,1	3,7	0,4
		4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	6,4	5,8	0,6
Μέσο σφάλμα:					**0,4**			**0,7**			**0,7**			**0,5**

Πίνακας 5.20: Τα πειραματικά αποτελέσματα του benchmark FS

S	BS	A	ARM7TDMI			ARM CORTEX A5			ARM CORTEX A9			ARM CORTEX M4		
			MR	EMR	ER	MR	EMR	ER	MR	EMR	ER	MR	EMR	ER
64	8	1	49,5	49,4	0,1	50,0	47,3	2,7	50,0	47,3	2,7	38,3	36,1	2,2
		2	50,1	50,3	0,2	50,2	47,3	2,9	50,2	47,3	2,9	38,4	36,1	2,2
		4	51,0	50,3	0,7	50,3	47,6	2,7	50,3	47,6	2,7	38,3	36,2	2,2
	32	1	14,1	14,0	0,2	14,4	14,0	0,4	14,4	14,0	0,4	11,5	10,6	0,9
		2	14,3	14,0	0,2	14,3	14,1	0,2	14,3	14,1	0,2	11,5	10,7	0,8
128	8	1	47,5	45,5	2,0	48,5	45,8	2,7	48,5	45,8	2,7	36,7	35,3	1,3
		2	47,6	45,7	1,9	48,6	46,1	2,5	48,6	46,1	2,5	36,8	35,0	1,8
		4	47,7	45,9	1,8	49,5	46,4	3,1	49,5	46,4	3,1	37,3	35,4	1,9
	32	1	13,2	12,5	0,6	13,6	12,8	0,7	13,6	12,8	0,7	10,5	10,2	0,3
		2	13,1	12,5	0,6	13,6	12,9	0,8	13,6	12,9	0,8	10,8	10,3	0,5
		4	13,1	12,5	0,6	13,7	13,0	0,7	13,7	13,0	0,7	10,8	10,3	0,5
256	8	1	46,6	43,4	3,2	44,4	41,1	3,3	44,4	41,1	3,3	17,6	14,2	3,4
		2	47,4	44,1	3,4	46,1	44,6	1,5	46,1	44,6	1,5	19,1	16,6	2,5
		4	47,4	44,0	3,4	46,0	44,6	1,4	46,0	44,6	1,4	28,9	27,4	1,6
	32	1	12,8	12,0	0,8	13,2	12,5	0,7	13,2	12,5	0,7	5,9	4,9	1,0
		2	12,8	11,9	0,9	12,8	12,3	0,5	12,8	12,3	0,5	7,8	7,3	0,5
		4	12,9	11,9	1,1	12,9	12,2	0,7	12,9	12,2	0,7	9,8	9,6	0,2
512	8	1	5,4	2,4	3,1	4,3	0,8	3,5	4,3	0,8	3,5	0,4	0,1	0,3
		2	7,8	3,4	4,4	4,1	0,0	4,1	4,1	0,0	4,1	0,5	0,0	0,5
		4	10,7	5,5	5,2	5,4	0,0	5,4	5,4	0,0	5,4	0,6	0,0	0,6
	32	1	3,2	1,9	1,4	1,4	0,3	1,1	1,4	0,3	1,1	0,1	0,1	0,1
		2	4,6	2,7	1,9	1,5	0,0	1,5	1,5	0,0	1,5	0,2	0,0	0,2
		4	7,0	4,4	2,6	1,5	0,0	1,5	1,5	0,0	1,5	0,2	0,0	0,2
Μέσο σφάλμα:					**1,7**			**1,9**			**1,9**			**1,1**

Πίνακας 5.21: Τα πειραματικά αποτελέσματα του benchmark HS

S	BS	A	ARM7TDMI			ARM CORTEX A5			ARM CORTEX A9			ARM CORTEX M4		
			MR	EMR	ER	MR	EMR	ER	MR	EMR	ER	MR	EMR	ER
64	8	1	50,9	52,2	1,3	48,0	51,1	3,1	48,0	52,8	4,8	35,8	36,0	0,3
		2	50,9	52,4	1,4	48,1	51,3	3,2	48,1	53,1	4,9	37,8	40,6	2,7
		4	50,9	52,4	1,4	48,1	51,2	3,1	48,3	53,0	4,7	37,8	38,4	0,7
	32	1	14,6	14,9	0,3	13,6	15,4	1,8	13,5	15,9	2,4	11,3	10,5	0,8
		2	14,6	15,0	0,3	13,6	15,4	1,8	13,5	15,9	2,4	11,4	13,8	2,5
128	8	1	48,8	49,3	0,5	35,5	37,8	2,3	35,6	39,1	3,5	19,4	20,6	1,3
		2	50,7	51,6	0,9	39,1	41,3	2,2	39,0	42,7	3,7	20,2	21,8	1,6
		4	50,5	51,4	0,9	40,3	42,5	2,2	40,3	43,9	3,6	21,4	20,2	1,2
	32	1	14,6	14,8	0,3	11,0	13,4	2,4	10,9	13,8	2,9	6,7	5,8	0,9
		2	14,6	14,9	0,3	12,2	13,9	1,7	12,1	14,3	2,2	7,3	7,0	0,4
		4	14,6	14,9	0,3	13,5	15,2	1,7	13,4	15,7	2,3	8,5	12,1	3,6
256	8	1	19,5	18,7	0,8	17,6	17,0	0,5	17,5	17,6	0,1	13,2	11,4	1,7
		2	22,4	22,0	0,4	18,5	18,1	0,5	18,6	18,7	0,1	14,5	17,6	3,2
		4	24,7	25,0	0,3	20,0	19,3	0,6	20,3	20,0	0,3	16,2	16,5	0,4
	32	1	7,1	6,6	0,5	5,5	5,8	0,3	5,3	6,0	0,8	4,2	6,8	2,6
		2	8,7	7,6	1,0	5,8	6,6	0,8	5,5	6,8	1,3	4,5	1,8	2,7
		4	10,5	8,6	1,9	6,0	6,7	0,7	5,9	6,9	1,1	5,1	6,7	1,6
512	8	1	13,4	11,1	2,3	12,0	8,8	3,2	12,0	9,1	2,9	11,0	9,9	1,1
		2	12,9	10,1	2,8	12,0	8,6	3,5	11,9	8,8	3,1	10,6	10,9	0,3
		4	12,4	9,2	3,2	11,9	8,5	3,4	11,9	8,8	3,1	10,0	9,3	0,7
	32	1	3,9	3,7	0,2	3,4	2,4	1,0	3,3	2,5	0,8	3,3	5,5	2,2
		2	3,7	3,4	0,3	3,4	2,3	1,0	3,3	2,4	0,9	3,1	4,3	1,2
		4	3,4	2,6	0,8	3,5	2,5	1,0	3,4	2,5	0,9	2,8	1,8	1,0
Μέσο σφάλμα:					**1,0**			**1,8**			**2,3**			**1,5**

Πίνακας 5.22: Τα πειραματικά αποτελέσματα του benchmark DJPEG

S	BS	A	ARM7TDMI			ARM CORTEX A5			ARM CORTEX A9			ARM CORTEX M4		
			MR	EMR	ER	MR	EMR	ER	MR	EMR	ER	MR	EMR	ER
64	8	1	47,2	43,9	3,3	42,7	42,4	0,2	45,0	42,4	2,5	26,6	24,7	1,9
		2	51,2	48,3	2,8	45,2	44,9	0,3	47,2	44,9	2,4	23,9	26,6	2,8
		4	51,1	48,4	2,7	47,5	47,8	0,4	48,9	47,8	1,1	26,9	28,5	1,6
	32	1	15,1	15,7	0,6	17,4	16,9	0,5	18,1	16,9	1,2	12,3	15,9	3,5
		2	15,7	15,8	0,1	18,1	17,9	0,3	18,7	17,9	0,9	14,7	14,8	0,1
128	8	1	38,4	33,5	4,9	28,9	24,6	4,3	29,5	24,6	4,9	20,2	23,7	3,6
		2	36,6	32,3	4,3	26,6	22,2	4,4	27,1	22,2	4,9	17,8	20,2	2,4
		4	37,8	33,4	4,4	26,4	21,7	4,7	26,2	21,7	4,5	17,8	20,0	2,2
	32	1	12,9	12,7	0,2	12,2	11,1	1,1	13,9	11,1	2,8	7,4	8,1	0,7
		2	12,8	13,5	0,6	13,3	12,6	0,7	13,9	12,6	1,3	5,8	7,0	1,1
		4	13,5	15,5	2,0	12,8	11,9	0,9	13,4	11,9	1,6	5,9	7,6	1,7
256	8	1	21,4	15,8	5,7	19,9	13,7	6,2	21,0	13,7	7,4	14,0	12,5	1,5
		2	22,5	16,3	6,1	19,5	12,0	7,5	19,6	12,0	7,6	11,1	10,3	0,8
		4	22,0	15,6	6,5	21,8	15,1	6,7	22,0	15,1	7,0	11,1	11,4	0,3
	32	1	6,5	4,7	1,7	6,5	4,2	2,3	7,6	4,2	3,4	5,3	4,5	0,8
		2	6,9	5,2	1,7	6,0	3,7	2,3	6,1	3,7	2,4	3,7	3,8	0,1
		4	6,4	4,7	1,7	6,6	4,5	2,1	6,9	4,5	2,4	3,6	4,5	0,9
512	8	1	12,4	7,5	4,9	13,2	6,7	6,5	13,3	6,7	6,5	5,7	6,6	0,9
		2	12,9	7,5	5,4	13,3	7,3	5,9	13,4	7,3	6,0	6,2	7,8	1,6
		4	12,9	7,5	5,5	13,5	7,3	6,2	13,6	7,3	6,3	7,6	7,8	0,3
	32	1	3,6	1,9	1,7	4,1	1,8	2,4	4,3	1,8	2,6	2,2	1,8	0,4
		2	3,8	1,9	1,9	4,2	1,9	2,3	4,3	1,9	2,4	2,4	2,0	0,4
		4	3,8	1,9	1,9	4,2	1,9	2,3	4,3	1,9	2,4	2,8	2,0	0,8
Μέσο σφάλμα:					**3,1**			**3,1**			**3,7**			**1,3**

Πίνακας 5.23: Τα πειραματικά αποτελέσματα του benchmark CJPEG

S	BS	A	ARM7TDMI			ARM CORTEX A5			ARM CORTEX A9			ARM CORTEX M4		
			MR	EMR	ER	MR	EMR	ER	MR	EMR	ER	MR	EMR	ER
64	8	1	66,1	70,2	4,1	50,7	50,4	0,2	50,8	50,5	0,4	40,4	42,0	1,7
		2	72,6	71,3	1,4	51,5	50,9	0,6	51,6	50,9	0,7	42,0	41,6	0,4
		4	64,6	71,3	6,8	51,6	51,0	0,5	51,6	51,0	0,6	41,4	42,0	0,6
	32	1	17,8	20,0	2,1	14,6	14,0	0,6	14,6	13,8	0,8	12,7	12,8	0,0
		2	17,1	20,0	2,9	14,6	14,0	0,6	14,6	13,9	0,7	12,9	15,7	2,8
128	8	1	50,5	62,0	11,5	38,6	44,0	5,4	38,7	44,0	5,3	28,2	31,9	3,7
		2	54,7	63,8	9,1	42,2	46,4	4,3	42,2	46,4	4,2	28,7	27,8	0,9
		4	54,8	65,8	10,9	44,4	46,9	2,5	44,4	46,9	2,5	29,9	28,3	1,5
	32	1	20,4	18,0	2,4	11,7	12,6	0,9	11,8	12,4	0,6	11,4	10,2	1,2
		2	19,2	18,9	0,4	12,6	13,2	0,6	13,1	13,1	0,0	10,3	10,3	0,1
		4	16,5	19,3	2,9	13,0	13,6	0,7	13,7	13,4	0,3	10,0	12,5	2,5
256	8	1	43,2	44,7	1,5	26,2	33,7	7,5	26,1	33,7	7,5	20,0	23,8	3,8
		2	45,2	44,7	0,5	26,8	33,9	7,1	26,8	33,9	7,1	20,3	23,4	3,1
		4	41,4	45,3	3,9	26,7	34,4	7,6	26,8	34,4	7,6	20,5	21,9	1,4
	32	1	11,2	13,0	1,8	7,3	9,1	1,8	7,3	9,1	1,8	5,7	8,0	2,4
		2	13,4	13,2	0,3	7,5	9,3	1,7	7,5	9,3	1,7	5,7	8,7	3,0
		4	12,9	14,0	1,1	7,8	9,5	1,8	7,7	9,5	1,8	5,8	8,5	2,6
512	8	1	39,3	41,5	2,2	22,5	31,0	8,5	22,5	31,0	8,5	15,5	17,8	2,4
		2	42,9	42,5	0,4	23,2	31,7	8,5	23,2	31,8	8,5	17,2	17,9	0,6
		4	42,5	42,7	0,3	23,6	31,9	8,3	23,7	31,9	8,2	17,6	17,9	0,3
	32	1	15,8	11,2	4,6	6,1	8,3	2,2	6,2	8,3	2,2	4,5	6,7	2,2
		2	16,5	11,4	5,1	6,4	8,5	2,1	6,4	8,5	2,1	4,9	6,7	1,7
		4	9,8	11,5	1,6	6,4	8,5	2,1	6,4	8,5	2,1	5,1	6,5	1,4
Μέσο σφάλμα:					**3,4**			**3,3**			**3,3**			**1,7**

Πίνκας 5.24: Τα πειραματικά αποτελέσματα του benchmark LAME

S	BS	A	ARM7TDMI			ARM CORTEX A5			ARM CORTEX A9			ARM CORTEX M4		
			MR	EMR	ER	MR	EMR	ER	MR	EMR	ER	MR	EMR	ER
64	8	1	50,8	49,0	1,8	49,2	49,6	0,4	43,5	49,2	5,8	40,7	40,8	0,1
		2	51,1	49,3	1,7	49,6	50,0	0,4	43,8	49,7	5,8	40,8	40,9	0,1
		4	51,0	49,3	1,7	49,7	50,1	0,4	43,9	49,8	5,8	40,8	40,9	0,1
	32	1	13,6	13,1	0,5	13,2	13,3	0,1	11,7	13,2	1,6	11,2	11,2	0,0
		2	13,6	13,1	0,4	13,3	13,4	0,1	11,7	13,3	1,6	11,2	11,2	0,0
128	8	1	49,7	47,9	1,7	48,3	48,4	0,0	42,7	48,1	5,3	39,4	39,3	0,1
		2	49,7	47,9	1,7	48,4	48,7	0,3	42,7	48,4	5,6	39,5	39,5	0,1
		4	49,7	47,9	1,7	48,4	48,7	0,3	42,7	48,4	5,6	39,5	39,5	0,1
	32	1	13,2	12,7	0,5	12,8	13,0	0,1	11,3	12,9	1,5	10,9	10,9	0,1
		2	13,1	12,7	0,4	12,8	13,0	0,1	11,3	12,9	1,5	10,9	11,0	0,1
		4	13,1	12,7	0,4	12,8	13,0	0,1	11,3	12,9	1,5	11,0	11,0	0,1
256	8	1	37,3	33,9	3,4	29,9	27,4	2,5	26,4	27,2	0,8	28,6	27,3	1,3
		2	47,2	44,0	3,1	39,3	37,7	1,6	34,7	37,5	2,8	33,9	32,6	1,3
		4	49,0	46,4	2,6	48,1	47,0	1,1	42,5	46,7	4,2	36,1	34,7	1,4
	32	1	11,2	10,4	0,8	8,4	7,8	0,6	7,4	7,7	0,3	9,8	7,7	2,1
		2	13,5	12,9	0,6	11,8	11,5	0,3	10,4	11,4	1,0	10,7	10,6	0,1
		4	13,5	13,1	0,4	12,8	12,8	0,0	11,3	12,7	1,4	10,7	10,7	0,0
512	8	1	2,5	0,6	1,9	1,6	0,7	0,9	1,4	0,7	0,7	1,5	1,0	0,5
		2	3,5	0,0	3,5	2,2	0,0	2,2	1,9	0,0	1,9	1,9	0,0	1,9
		4	4,7	0,0	4,7	3,2	0,0	3,2	2,8	0,0	2,8	3,0	0,0	3,0
	32	1	0,7	0,2	0,5	0,5	0,3	0,3	0,5	0,3	0,2	0,5	0,3	0,2
		2	0,9	0,0	0,9	0,7	0,0	0,7	0,6	0,0	0,6	0,6	0,0	0,6
		4	1,2	0,0	1,2	1,1	0,0	1,1	1,0	0,0	1,0	0,9	0,0	0,9
Μέσο σφάλμα:					**1,6**			**0,7**			**2,6**			**0,6**

Πίνακας 5.25: Τα πειραματικά αποτελέσματα του benchmark PHODS

S	BS	A	ARM7TDMI			ARM CORTEX A5			ARM CORTEX A9			ARM CORTEX M4		
			MR	EMR	ER	MR	EMR	ER	MR	EMR	ER	MR	EMR	ER
64	8	1	48,8	46,6	2,3	50,9	50,8	0,1	50,9	48,0	2,9	42,7	38,5	4,2
		2	49,0	46,8	2,2	51,7	51,7	0,1	51,7	48,9	2,9	45,0	40,8	4,2
		4	49,4	47,2	2,2	51,7	51,7	0,1	51,7	48,9	2,9	45,0	40,8	4,2
	32	1	14,8	13,9	0,9	14,9	14,0	1,0	14,9	13,2	1,7	13,5	13,1	0,4
		2	15,1	14,1	1,0	14,9	14,0	1,0	14,9	13,2	1,7	13,7	13,4	0,3
128	8	1	33,5	31,0	2,5	35,6	34,5	1,2	35,6	32,6	3,0	8,3	3,6	4,7
		2	43,8	41,4	2,4	45,1	44,6	0,5	45,1	42,2	2,9	10,8	5,4	5,4
		4	48,4	46,2	2,3	49,3	49,0	0,3	49,3	46,4	2,9	15,8	9,0	6,9
	32	1	12,3	11,6	0,7	12,3	10,3	2,0	12,3	9,7	2,6	4,0	4,4	0,4
		2	14,2	13,5	0,7	14,1	13,1	1,0	14,1	12,4	1,7	5,4	6,7	1,3
		4	14,2	13,5	0,7	14,1	13,1	1,0	14,1	12,4	1,7	8,2	11,1	2,9
256	8	1	2,6	0,0	2,6	2,9	0,0	2,9	2,9	0,0	2,9	3,1	0,0	3,1
		2	2,6	0,0	2,6	2,9	0,0	2,9	2,9	0,0	2,9	3,0	0,0	3,0
		4	2,6	0,0	2,6	2,9	0,0	2,9	2,9	0,0	2,9	3,0	0,0	3,0
	32	1	0,7	0,0	0,7	0,9	0,0	0,9	0,9	0,0	0,9	0,9	0,0	0,9
		2	0,7	0,0	0,7	0,9	0,0	0,9	0,9	0,0	0,9	0,9	0,0	0,9
		4	0,7	0,0	0,7	0,9	0,0	0,9	0,9	0,0	0,9	0,9	0,0	0,9
512	8	1	2,6	0,0	2,6	2,9	0,0	2,9	2,9	0,0	2,9	3,0	0,0	3,0
		2	2,6	0,0	2,6	2,9	0,0	2,9	2,9	0,0	2,9	3,0	0,0	3,0
		4	2,6	0,0	2,6	2,9	0,0	2,9	2,9	0,0	2,9	3,0	0,0	3,0
	32	1	0,7	0,0	0,7	0,8	0,0	0,8	0,8	0,0	0,8	0,8	0,0	0,8
		2	0,7	0,0	0,7	0,8	0,0	0,8	0,8	0,0	0,8	0,8	0,0	0,8
		4	0,7	0,0	0,7	0,8	0,0	0,8	0,8	0,0	0,8	0,8	0,0	0,8
Μέσο σφάλμα:					**1,6**			**1,3**			**2,1**			**2,5**

Πίνακας 5.26: Τα πειραματικά αποτελέσματα του benchmark SHA

S	BS	A	ARM7TDMI			ARM CORTEX A5			ARM CORTEX A9			ARM CORTEX M4		
			MR	EMR	ER	MR	EMR	ER	MR	EMR	ER	EMR	MR	ER
64	8	1	47,2	43,9	3,3	48,4	56,9	8,5	48,1	56,4	8,3	48,4	48,4	0,1
		2	51,2	48,3	2,8	48,2	57,0	8,8	48,2	56,9	8,7	48,2	46,9	1,3
		4	51,1	48,4	2,7	50,1	58,6	8,5	49,5	57,9	8,4	50,1	50,6	0,5
	32	1	15,1	15,7	0,6	13,8	16,0	2,2	13,9	15,9	2,0	13,8	13,9	0,1
		2	15,7	15,8	0,1	13,9	16,2	2,2	13,9	16,0	2,1	13,9	13,1	0,9
128	8	1	38,4	33,5	4,9	46,7	54,8	8,1	46,4	54,6	8,1	46,7	45,2	1,6
		2	36,6	32,3	4,3	45,4	53,7	8,4	45,4	53,7	8,4	45,4	51,5	6,2
		4	37,8	33,4	4,4	45,4	53,7	8,4	45,4	53,7	8,4	45,4	51,7	6,3
	32	1	12,9	12,7	0,2	12,7	14,9	2,2	12,7	14,5	1,8	12,7	13,8	1,1
		2	12,8	13,5	0,6	12,4	14,6	2,2	11,7	14,0	2,3	12,4	12,7	0,3
		4	13,5	15,5	2,0	12,9	15,1	2,2	11,7	13,9	2,3	12,9	13,5	0,6
256	8	1	21,4	15,8	5,7	46,3	54,4	8,1	46,0	54,1	8,1	46,3	45,9	0,4
		2	22,5	16,3	6,1	45,0	53,4	8,4	45,0	53,4	8,4	45,0	45,0	0,0
		4	22,0	15,6	6,5	45,1	53,5	8,4	45,1	53,5	8,4	45,1	43,4	1,7
	32	1	6,5	4,7	1,7	12,6	14,6	2,0	12,6	14,4	1,8	12,6	13,7	1,1
		2	6,9	5,2	1,7	11,6	13,9	2,3	11,6	13,9	2,3	11,6	13,1	1,5
		4	6,4	4,7	1,7	11,7	13,9	2,3	11,7	13,9	2,3	11,7	13,0	1,3
512	8	1	12,4	7,5	4,9	44,7	53,1	8,4	44,7	53,1	8,4	44,7	44,7	0,1
		2	12,9	7,5	5,4	44,8	53,2	8,4	44,8	53,2	8,4	44,8	44,6	0,2
		4	12,9	7,5	5,5	44,8	53,2	8,4	44,8	53,2	8,4	44,8	42,5	2,3
	32	1	3,6	1,9	1,7	11,5	13,8	2,3	11,5	13,8	2,3	11,5	12,2	0,7
		2	3,8	1,9	1,9	11,5	13,8	2,3	11,5	13,8	2,3	11,5	12,3	0,8
		4	3,8	1,9	1,9	11,5	13,8	2,3	11,5	13,8	2,3	11,5	10,0	1,6
Μέσο σφάλμα:					**3,1**			**5,4**			**5,4**			**1,3**

Πίνακας 5.27: Τα πειραματικά αποτελέσματα του benchmark SUSAN

S	BS	A	ARM7TDMI			ARM CORTEX A5			ARM CORTEX A9			ARM CORTEX M4		
			MR	EMR	ER	MR	EMR	ER	MR	EMR	ER	MR	EMR	ER
64	8	1	51,3	49,5	1,7	51,8	50,1	1,8	52,0	50,1	1,9	30,0	30,5	0,6
		2	51,3	49,6	1,7	51,9	49,8	2,0	52,0	49,8	2,2	30,6	30,6	0,1
		4	51,3	49,6	1,7	51,9	49,8	2,0	52,0	49,8	2,2	30,6	30,6	0,1
	32	1	15,4	14,5	0,8	14,7	14,3	0,4	14,7	14,3	0,5	9,8	9,5	0,3
		2	15,4	14,6	0,8	14,8	14,3	0,4	14,8	14,3	0,5	9,8	9,5	0,3
128	8	1	50,4	48,9	1,5	51,1	49,4	1,7	51,2	49,4	1,8	24,5	23,2	1,3
		2	51,1	49,5	1,6	51,7	49,8	2,0	51,9	49,8	2,1	28,5	29,1	0,6
		4	51,2	49,5	1,8	51,8	49,8	2,1	52,0	49,8	2,2	30,2	30,5	0,3
	32	1	15,3	14,4	1,0	14,7	14,2	0,4	14,7	14,2	0,5	9,8	9,5	0,3
		2	15,3	14,4	1,0	14,7	14,1	0,6	14,7	14,1	0,6	9,8	9,5	0,3
		4	15,3	14,4	1,0	14,7	14,1	0,6	14,7	14,1	0,6	9,8	9,5	0,3
256	8	1	38,0	35,2	2,8	28,8	26,9	1,9	28,8	26,9	1,9	0,1	0,0	0,1
		2	46,2	43,3	3,0	39,4	37,7	1,7	39,5	37,7	1,8	0,1	0,0	0,1
		4	52,0	48,7	3,3	46,1	44,1	2,0	46,2	44,1	2,1	0,2	0,0	0,2
	32	1	13,1	12,2	0,9	9,9	9,4	0,6	9,9	9,4	0,6	0,1	0,0	0,0
		2	15,2	14,3	0,9	14,4	13,8	0,6	14,4	13,8	0,6	0,1	0,0	0,1
		4	15,3	14,3	1,0	14,5	14,1	0,4	14,5	14,1	0,5	0,1	0,0	0,1
512	8	1	0,1	0,0	0,1	0,1	0,0	0,1	0,1	0,0	0,1	0,0	0,0	0,0
		2	0,1	0,0	0,1	0,2	0,0	0,2	0,2	0,0	0,2	0,0	0,0	0,0
		4	0,2	0,0	0,2	0,2	0,0	0,2	0,2	0,0	0,2	0,0	0,0	0,0
	32	1	0,1	0,0	0,0	0,1	0,0	0,0	0,1	0,0	0,0	0,0	0,0	0,0
		2	0,1	0,0	0,1	0,1	0,0	0,1	0,1	0,0	0,1	0,0	0,0	0,0
		4	0,1	0,0	0,1	0,1	0,0	0,1	0,1	0,0	0,1	0,0	0,0	0,0
Μέσο σφάλμα:					**1,2**			**0,9**			**1,0**			**0,2**

Πίνακας 5.28: Τα πειραματικά αποτελέσματα του benchmark SS

S	BS	A	ARM7TDMI			ARM CORTEX A5			ARM CORTEX A9			ARM CORTEX M4		
			MR	EMR	ER	MR	EMR	ER	MR	EMR	ER	MR	EMR	ER
64	8	1	51,9	48,5	3,4	52,1	49,9	2,2	50,3	49,9	0,4	29,7	30,5	0,8
		2	52,1	49,1	3,1	52,4	50,6	1,8	50,6	50,6	0,0	29,5	30,2	0,7
		4	52,5	49,6	2,9	52,8	51,0	1,8	51,1	51,0	0,0	29,5	30,4	0,9
	32	1	13,6	14,4	0,8	16,4	15,9	0,5	15,8	15,9	0,0	9,4	9,8	0,4
		2	13,7	14,5	0,8	16,4	15,9	0,5	15,9	15,9	0,0	9,3	9,8	0,5
128	8	1	49,5	46,2	3,3	44,2	41,8	2,3	42,7	41,8	0,9	1,3	0,7	0,6
		2	50,9	47,3	3,7	47,8	44,8	3,0	46,2	44,8	1,4	1,6	0,0	1,6
		4	51,0	47,3	3,8	51,5	48,5	3,0	49,8	48,5	1,3	2,2	0,0	2,2
	32	1	13,3	14,1	0,8	15,9	15,4	0,5	15,4	15,4	0,0	4,1	4,0	0,1
		2	13,2	13,8	0,5	15,8	15,0	0,8	15,3	15,0	0,3	5,8	5,6	0,2
		4	13,4	13,8	0,4	15,8	15,0	0,8	15,3	15,0	0,3	9,2	9,3	0,2
256	8	1	1,8	0,6	1,3	1,2	1,0	0,3	1,2	1,0	0,3	0,1	0,0	0,0
		2	2,2	0,0	2,1	1,6	0,0	1,6	1,6	0,0	1,6	0,1	0,0	0,1
		4	3,4	0,0	3,4	2,5	0,0	2,5	2,5	0,0	2,4	0,1	0,0	0,1
	32	1	3,7	3,4	0,3	0,6	0,5	0,1	0,6	0,5	0,1	0,0	0,0	0,0
		2	5,1	4,6	0,5	0,7	0,0	0,7	0,7	0,0	0,6	0,0	0,0	0,0
		4	8,5	7,7	0,9	1,1	0,0	1,1	1,1	0,0	1,1	0,0	0,0	0,0
512	8	1	0,1	0,0	0,1	0,1	0,0	0,0	0,1	0,0	0,0	0,0	0,0	0,0
		2	0,1	0,0	0,1	0,1	0,0	0,1	0,1	0,0	0,1	0,0	0,0	0,0
		4	0,1	0,0	0,1	0,1	0,0	0,1	0,1	0,0	0,1	0,0	0,0	0,0
	32	1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
		2	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
		4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
Μέσο σφάλμα:					**1,4**			**1,0**			**0,5**			**0,4**

Πίνακας 5.29: Τα πειραματικά αποτελέσματα του benchmark 3SLOG

6 ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΕΣ ΕΠΕΚΤΑΣΕΙΣ

Στο κεφάλαιο αυτό γίνεται ένας απολογισμός των αποτελεσμάτων της διπλωματικής εργασίας. Σχολιάζονται τα ποιοτικά χαρακτηρίστηκα των νέων εργαλείων και τα πειραματικά αποτελέσματα που παρουσιάστηκαν στο κεφάλαιο 5. Στη συνέχεια γίνεται μία αναφορά στις δυνατότητες επέκτασης των εργαλείων και σε κάποια ανοικτά προβλήματα των οποίων η επίλυση θα μπορούσε να βελτιώσει τα χαρακτηριστικά της μεθόδου.

6.1 Συμπεράσματα

Στα προηγούμενα κεφάλαια περιγράφθηκε μία μέθοδος ανάλυσης εφαρμογών η οποία αποσκοπεί στην πολύ γρήγορη, αλλά και με αρκετά καλή ακρίβεια, εκτίμηση της επίδοσης ενσωματωμένων συστημάτων. Στηριζόμενοι στη πληροφορία που μπορεί να μας δώσει ο πηγαίος κώδικας μίας εφαρμογής για την εκτέλεση του κώδικα μηχανής αποφύγαμε τις κλασσικές μεθόδους προσομοίωσης και έτσι ο χρόνος εκτίμησης μειώθηκε δραματικά (έως και περισσότερο από **20000 φορές).**

Η μέθοδος που παρουσιάστηκε στηρίχτηκε στη μελέτη των Ν. Κρούπη και Δ. Σούντρη [1] οι οποίοι απέδειξαν πειραματικά ότι η άντληση πληροφορίας από τον πηγαίο κώδικα μπορεί να οδηγήσει σε πολύ γρήγορη και αρκετά ακριβή εκτίμηση της επίδοσης ενός ενσωματωμένου συστήματος. Στη παρούσα εργασία έγινε μία προσπάθεια να βελτιωθεί η μέθοδος [1] και να αναπτυχθούν ολοκληρωμένα και αποδοτικά εργαλεία λογισμικού που την υλοποιούν.

Οι βασικές αλλαγές που έγιναν από τη παρούσα εργασία στη μέθοδο [1] είναι ο τρόπος με τον οποίο αντιστοιχίζεται ο Γ.Ρ.Ε. του κώδικα μηχανής στον Γ.Ρ.Ε. του πηγαίου κώδικα και ο τρόπος που υπολογίζεται το πλήθος αστοχιών ενός απλού βρόχου εκτέλεσης. Στη νέα μέθοδο, ο Γ.Ρ.Ε. του κώδικα μηχανής δεν θεωρείται όμοιος με τον Γ.Ρ.Ε. του πηγαίου κώδικα, αλλά αντιστοιχίζεται με αυτόν με χρήση

των πληροφοριών που μπορεί να μας δώσει ο μεταγλωττιστής, ενώ η μέθοδος υπολογισμού των αστοχιών τροποποιήθηκε, ώστε να γίνει πιο ακριβής και να υποστηρίξει πλήρως και συσχετιστικές μνήμες. Μία σημαντική καινοτομία αποδείχτηκε, επίσης, ο συντακτικός μετασχηματισμός του πηγαίου κώδικα, ο οποίος σύμφωνα με τα αποτελέσματα των πειραμάτων βελτιώνει σημαντικά την ακρίβεια εκτίμησης.

Τα νέα εργαλεία που αναπτύχθηκαν υλοποιούν αποδοτικά τη μεθοδολογία και παρέχουν ευελιξία, ενώ τηρούν τις προδιαγραφές για περεταίρω επέκταση (πχ. για εφαρμογές πολλών νημάτων). Με τη χρήση τους μπορεί να γίνει εύκολα εκτίμηση της επίδοσης μίας εφαρμογής ανεξάρτητα από το πώς δομείται ή το πόσο περίπλοκη είναι αυτή η εφαρμογή. Όλη η διαδικασία είναι αυτοματοποιημένη και ο χρήστης χρειάζεται να κάνει μόνο τη βασική παραμετροποίηση. Ένα από τα σημαντικότερα πλεονεκτήματα των νέων εργαλείων, είναι ότι υποστηρίζουν όλες τις αρχιτεκτονικές που υποστηρίζουν τα GNU εργαλεία χωρίς να χρειάζονται εξειδικευμένη παραμετροποίηση.

Ένα άλλο μεγάλο πλεονέκτημα της μεθόδου είναι ότι με μία μόνο εκτέλεση της εφαρμογής μπορεί να γίνει εκτίμηση της επίδοσης για πολλαπλές αρχιτεκτονικές επεξεργαστών και κρυφής μνήμης. Έτσι η μέθοδος καθίσταται ικανή να αναζητήσει την βέλτιστη αρχιτεκτονική από ένα σύνολο πολλών διαθέσιμων επιλογών με μεγάλη ταχύτητα. Το πρόβλημα αυτό είναι πολύ μεγάλης σημασίας όταν το λογισμικό και το υλικό αναπτύσσονται παράλληλα με σκοπό να συνεργάζονται όσο πιο αποδοτικά γίνεται.

Όπως φαίνεται από τα αποτελέσματα που παρουσιάζονται στο κεφάλαιο 5 η ακρίβεια των νέων εργαλείων δοκιμάστηκε σε πληθώρα διαφορετικών περιπτώσεων (συνολικά 1656 διαφορετικές περιπτώσεις) και αποδείχτηκε πάρα πολύ καλή. Σκοπός της μεθόδου δεν είναι η εξαγωγή αποτελεσμάτων απόλυτης ακρίβειας, αλλά η πολύ γρήγορη εξαγωγή αποτελεσμάτων καλής ακρίβειας. Με βάση αυτήν την απαίτηση, η τιμή μέσου σφάλματος **2,1%** και η αύξηση της ταχύτητας έως και **20000** φορές σε σχέση με τις συμβατικές μεθόδους προσομοίωσης είναι παραπάνω από ικανοποιητικά αποτελέσματα και χρήζουν τη προσπάθεια επιτυχή.

6.2 Επεκτάσεις και μελλοντική έρευνα

6.2.1 Επεκτείνοντας τα εργαλεία για εφαρμογές πολλών νημάτων

Τα εργαλεία που αναπτύχθηκαν παρέχουν μία υποδομή για την εκτίμηση της επίδοσης πολύ-νηματικών εφαρμογών. Τα αρχεία τύπου cxml (βλ. παράρτημα III.II) υποστηρίζουν τη περιγραφή αρχιτεκτονικών πολλών επεξεργαστικών πυρήνων όπου κάθε πυρήνας μπορεί να αντιστοιχηθεί σε Ε.Γ.Ρ.Ε. διαφορετικών εφαρμογών.

Για να υποστηριχτούν πλήρως οι πολύ-νηματικές εφαρμογές θα πρέπει να γίνουν, ωστόσο, κάποιες επεκτάσεις. Αρχικά ο μηχανισμός χαρακτηρισμού του πηγαίου κώδικα θα πρέπει να αλλάξει, καθώς το εργαλείο gcov που χρησιμοποιείται δεν υποστηρίζει πολύ-νηματικές εφαρμογές. Θα πρέπει να αναζητηθεί ένα εργαλείο-αντικαταστάτης του gcov, το οποίο όμως να είναι συμβατό με κάποια πλατφόρμα πολύ-νηματικών εφαρμογών (πχ. OpenMP [43]). Εναλλακτικά, μπορεί να επεκταθεί το εργαλείο csa ώστε να αναλάβει και τον χαρακτηρισμό του πηγαίου κώδικα. Το εργαλείο csa μπορεί να επεκταθεί ώστε να εισάγει εντολές καταμέτρησης στα απαραίτητα ημεία του πηγαίου κώδικα (βλ. τρόπο λειτουργίας του gcov, ενότητα 3.3.2). Στη συνέχεια θα εκτελείται ο τροποποιημένος αυτός κώδικας αντί του αρχικού πηγαίου κώδικα στον host υπολογιστή και θα αποθηκεύονται οι τιμές των μετρητών σε ένα αρχείο (ανάλογο του αρχείου gcov).

Το csa μπορεί, επίσης, να επεκταθεί, ώστε να αναγνωρίζει τα βασικά directives της πλατφόρμας OpenMP [43] (ή όποιας άλλης έχει επιλεχθεί). Επιπλέον θα πρέπει να παράγει εντολές καταμέτρησης που ελέγχουν πιο είναι το νήμα υπό του οποίου εκτελούνται. Έτσι για κάθε νήμα της εφαρμογής θα δημιουργείται και ένα ξεχωριστό αρχείο ανάλογο των αρχείων gcov. Τα διακριτά αυτά αρχεία θα συνδέονται με τους αντίστοιχους επεξεργαστικούς πυρήνες μέσου του μηχανισμού που παρέχουν τα cxml αρχεία.

Στη συνέχεια θα πρέπει να γίνουν πειράματα ώστε να μετρηθεί η ακρίβεια του υπάρχοντος αλγόριθμου εκτίμησης αστοχιών για πολύ-νηματικές εφαρμογές. Αν τα αποτελέσματα δεν είναι ικανοποιητικά, θα πρέπει να τροποποιηθεί ο πυρήνας του αλγορίθμου. Θα μπορούσε πχ. να ενσωματωθεί στη διαδικασία κάποιο στατιστικό μοντέλο της συμπεριφοράς των συστημάτων πολλών επεξεργαστικών πυρήνων (βλ. [44]).

6.2.2 Συμβατότητα με περισσότερους μεταγλωττιστές

Το εργαλείο ccov διαβάζει τη πληροφορία αντιστοίχισης του κώδικα μηχανής με τον πηγαίο κώδικα καθώς και άλλες χρήσιμες πληροφορίες από τα αρχεία assembly κώδικα που παράγει ο gcc. Οι μορφή στην οποία παρουσιάζονται αυτές οι πληροφορίες είναι μοναδική και συμβατή μόνο με τον gcc. Υπάρχουν ωστόσο κάποιες έτοιμες βιβλιοθήκες (βλ. BFD [45]) με τη χρήση των οποίων οι παραπάνω πληροφορίες μπορούν να εξαχθούν απευθείας από το εκτελέσιμο αρχείο μίας εφαρμογής. Με τη χρήση της βιβλιοθήκης BFD μάλιστα μπορεί να εξαχθεί από το εκτελέσιμο αρχείο και ο Γ.Ρ.Ε. της εφαρμογής, κάτι που θα μπορούσε να αντικαταστήσει τη μέθοδο που χρησιμοποιείται στη μεθοδολογία που παρουσιάστηκε (χρήση αρχείων vcg).

Με την αξιοποίηση της βιβλιοθήκης BFD, τα εργαλεία θα μπορούσαν να γίνουν πλήρως ανεξάρτητα από τον μεταγλωττιστή gcc και να παρείχαν ακόμα μεγαλύτερη ευελιξία.

6.2.3 Άλλες επεκτάσεις και ανοικτά προβλήματα

Κάποιες άλλες βελτιώσεις και επεκτάσεις που θα μπορούσαν να γίνουν στα εργαλεία είναι οι παρακάτω.

Το εργαλείο csa δεν παρέχει συντακτικούς μετασχηματισμούς για υπό συνθήκη εκφράσεις (τελεστής ?) και πράξεις με λογικές εκφράσεις. Η υλοποίηση αυτών των μετασχηματισμών θα μπορούσε ίσως να βελτιώσει την ακρίβεια εκτίμησης των εργαλείων. Θα πρέπει λοιπόν να γίνει μελέτη ώστε να βρεθεί ποιοι μετασχηματισμοί πετυχαίνουν τη καλύτερη ακρίβεια στα αποτελέσματα και να ενσωματωθούν στο csa.

Ακόμα, θα μπορούσε να γίνει μελέτη ώστε να επεκταθεί η χρήση των εργαλείων και για την εκτίμηση της κατανάλωσης ενέργειας. Αυτό μπορεί να γίνει με χρήση μοντέλων κατανάλωσης ενέργειας (βλ. [46]).

Τέλος, θα ήταν καλό να υλοποιηθεί και ένα γραφικό περιβάλλον για πιο εύκολη χρήση των εργαλείων.

ΠΑΡΑΡΤΗΜΑ Ι *ΕΓΧΕΙΡΙΔΙΟ ΧΡΗΣΗΣ ΤΟΥ ΕΡΓΑΛΕΙΟΥ CSA*

Ι.Ι Η βασική εντολή εκτέλεσης του csa

Η βασική εντολή εκτέλεσης του εργαλείου csa είναι η:

```
csa [OPTION]… [-o OUTPUTFILE] SOURCEFILE
```

- `OPTION: Παράμετρος του προγράμματος`
- `OUTPUTFILE: Το όνομα του αρχείου εξόδου`
- `SOURCEFILE: Το όνομα του αρχείου εισόδου`

Με την εντολή αυτή το αρχείο `SOURCEFILE` διαβάζεται, υφίσταται τους συντακτικούς μετασχηματισμούς που καθορίζονται από τις παραμέτρους `OPTION` και ο τροποποιημένος κώδικας αποθηκεύεται στο αρχείο `OUTPUTFILE`. Οι σειρά με την οποία δίνονται οι παράμετροι δεν επηρεάζει τη λειτουργία του εργαλείου.

Ι.ΙΙ Η είσοδος του csa

Το csa μπορεί να διαβάσει την είσοδό του είτε από ένα αρχείο είτε από την standard είσοδο (πχ. πληκτρολόγιο). Αν επιθυμούμε να χρησιμοποιήσουμε την standard είσοδο θα πρέπει να δώσουμε στο csa τη παράμετρο –i και να μη δώσουμε κανένα όνομα αρχείου εισόδου (`SOURCEFILE`). Διαφορετικά θα πρέπει να δώσουμε ένα όνομα αρχείου εισόδου. Αν το csa δεν μπορέσει για κάποιο λόγο (πχ. το αρχείο δεν υπάρχει) να ανοίξει το αρχείο εισόδου, τότε θα εμφανίσει το αντίστοιχο μήνυμα σφάλματος και θα τερματίσει τη λειτουργία του.

Η είσοδος πρέπει να είναι ένα έγκυρο πρόγραμμα σε γλώσσα C. Επειδή το εργαλείο δεν έχει ενσωματωμένο προ-επεξεργαστή της C, ο κώδικας εισόδου θα πρέπει να είναι ήδη επεξεργασμένος από έναν ξεχωριστό προ-επεξεργαστή (π.χ. cpp). Το csa είναι σε μεγάλο βαθμό συμβατό με το πρότυπο ISO c99, ενώ υποστηρίζει και κάποιες επεκτάσεις του gnu πρότυπου για τη γλώσσα [47]. Οι βασικότερες επεκτάσεις είναι οι παρακάτω:

- Αναγνώριση της λέξης κλειδί `__attribute__`.
- Αναγνώριση της λέξης κλειδί `__extension__`.
- Υποστήριξη του τελεστή `typeof()`.

- Υποστήριξη σύνθετων εντολών (compound statements) στο ρόλο εκφράσεων εντός παρενθέσεων.

- Υποστήριξη εμφωλευμένων δομών (`struct`) και ενώσεων (`union`) χωρίς όνομα.

Ένα σημείο που θέλει προσοχή, είναι η εντολές "`asm`". Επειδή το csa προορίζεται να επεξεργάζεται κώδικα που πρόκειται να μεταγλωττιστεί για πολλές διαφορετικές αρχιτεκτονικές, οι εντολές "`asm`" αγνοούνται με την εμφάνιση μίας προειδοποίησης (warning) προς τον χρήστη.

I.III Η έξοδος του csa

Η το αποτέλεσμα της εκτέλεσης του csa μπορεί είτε να αποθηκευτεί σε ένα αρχείο, είτε να εμφανιστεί κατευθείαν στην standard έξοδο. Για να αποθηκευτεί η έξοδος σε αρχείο θα πρέπει να δοθεί στο csa η παράμετρος `-o` ακολουθούμενη από το όνομα του αρχείου εξόδου `(OUTPUTFILE)`. Σε περίπτωση που το αρχείο δεν μπορεί να δημιουργηθεί για κάποιον λόγο (π.χ. δεν έχουμε την άδεια), τότε εμφανίζεται το αντίστοιχο μήνυμα σφάλματος και η εκτέλεση του csa τερματίζεται. Αν επιθυμούμε να εμφανισθεί ο παραγόμενος κώδικας στην standard έξοδο, τότε αρκεί να παραληφθεί εντελώς η παράμετρος αρχείου εξόδου `(-o OUTPUTFILE)`.

Εφόσον η είσοδος του csa αποτελεί έγκυρο κώδικα, η έξοδος του csa αποτελεί επίσης έγκυρο κώδικα της γλώσσας C.

I.IV Παράμετροι συντακτικών μετασχηματισμών

Οι παράμετροι (`OPTION`) με τις οποίες καθορίζονται οι συντακτικοί μετασχηματισμοί που θα γίνουν στον κώδικα εισόδου είναι οι εξής:

- `-l` : Επέκταση όλων των βρόχων του κώδικα.

- `-s` : Επέκταση όλων των εντολών επιλογής (`if`).

Οι επεκτάσεις των βρόχων και των εντολών επιλογής γίνονται σύμφωνα με τους μετασχηματισμούς που παρουσιάζονται στα σχήματα της ενότητας 3.5 (βλ. σελ. 43). Σε περίπτωση που δεν δοθεί καμία παράμετρος συντακτικού μετασχηματισμού, η έξοδος του προγράμματος θα είναι όμοια με την είσοδό του, αλλά με τυποποιημένη στοίχιση και ακριβώς μία εντολή σε κάθε γραμμή. Παρακάτω (Κώδικας I-1) δίνεται ένα δείγμα της στοίχισης που εφαρμόζει το csa στον κώδικα εξόδου.

```c
int main ( ) {
    int i ;
    for ( i = 0 ; 1 < 100 ; i ++ )
        {
            printf ( "%d\n" , i ) ;
        }
    return 0 ;
}
```

Κώδικας I-1: Η στοίχιση που εφαρμόζει το csa στον κώδικα C.

I.V Διαχείριση των header files

Όπως αναφέραμε στην ενότητα I.II ο κώδικας εισόδου του csa πρέπει να έχει πρώτα περάσει από έναν ξεχωριστό προ-επεξεργαστή. Αυτό σημαίνει ότι ο κώδικας του αρχικού αρχείου θα έχει ενσωματωθεί με τον κώδικα των header files που εισήχθησαν από τον προ-επεξεργαστή. Ωστόσο, ο χρήστης μπορεί να επιλέξει αν θέλει ο κώδικας από τα header files που εισήχθησαν να μεταφερθεί και στην έξοδο αυτούσιος ή να εμφανιστεί ως ένα include directive (`#include`) όπως και στον αρχικό πηγαίο κώδικα. Εάν είναι επιθυμητό να εμφανιστεί το include directive αντί ολόκληρου του κώδικα θα πρέπει να δοθεί στο csa η παράμετρος `-f`.

Η παράμετρος αυτή μπορεί να φανεί χρήσιμη για την επίλυση του προβλήματος αλλοίωσης της αρίθμησης γραμμών του πηγαίου κώδικα όταν αυτός μεταγλωττίζεται για δύο διαφορετικές αρχιτεκτονικές (βλ. σελ. 62). Εφόσον ο κώδικας των header files δεν αντιγράφετε αυτούσιος στην έξοδο του csa δεν μπορεί να αλλοιώσει την αρίθμηση γραμμών του κώδικα.

Η παράμετρος `-f`, ωστόσο, θα πρέπει να χρησιμοποιείται με ιδιαίτερη προσοχή καθώς σε ορισμένες περιπτώσεις μπορεί να οδηγήσει σε αλλοίωση της λειτουργικότητας του κώδικα. Πολλές φορές το περιεχόμενο του λειτουργικού κώδικα σε ένα header file εξαρτάται από τις τιμές κάποιων ορισμών "`#define`" (πχ. περιέχει εντολές "`ifdef`"). Έτσι ο προγραμματιστής πριν συμπεριλάβει (`include`) στον κώδικά του ένα header file, μπορεί να έχει κάνει κάποιους ορισμούς "`#define`" που καθορίζουν τη λειτουργικότητα του header file. Όταν ο αρχικός κώδικας περάσει από τον προ-επεξεργαστή της C όλοι οι "`define`" ορισμοί εξαλείφονται ενώ ο κώδικας του header file παίρνει τη τελική του μορφή (η οποία πλέον δεν εξαρτάται από κάποιον "`#define`" ορισμό). Αν αυτός ο κώδικας όμως περάσει από το csa με χρήση της παραμέτρου `-f` τότε η τελική μορφή του κώδικα από το header file θα αντικατασταθεί και πάλι από την αρχική της μορφή η οποία εξαρτάται από τους "`#define`" ορισμούς. Οι ορισμοί του προγραμματιστή όμως έχουν απαλειφθεί από τον προ-επεξεργαστή με αποτέλεσμα η λειτουργικότητα του κώδικα από το header file να μην είναι πλέον εγγυημένη.

Για τον λόγο αυτό το csa παρέχει έναν εναλλακτικό τρόπο για τη διόρθωση του προβλήματος αλλοίωσης της αρίθμησης γραμμών. Αν δοθεί η παράμετρος `-p`, ο

κώδικας των header files δεν παραλείπεται από την έξοδο, εισάγονται, όμως, στον κώδικα εξόδου ενδείκτες αρίθμησης γραμμής (line directives, "`#line`") με τους οποίους επιδιορθώνεται η αρίθμηση γραμμών και διατηρείται ανεξάρτητη από το μέγεθος του κώδικα των header files.

I.VI Διαχείριση σφαλμάτων και προειδοποιήσεων

Σε περίπτωση που ο κώδικας εισόδου έχει κάποιο συντακτικό σφάλμα, το csa θα εμφανίσει ένα μήνυμα που ενημερώνει για την ακριβή θέση του χωρίς όμως να δίνει καμία περιγραφή του. Ο χρήστης παροτρύνεται να χρησιμοποιήσει τον μεταγλωττιστή για τον έλεγχο ορθότητας της εφαρμογής του. Για οποιοδήποτε άλλο σφάλμα το csa δίνει ακριβή περιγραφή.

I.VII Συνοπτική περιγραφή όλων των παραμέτρων του csa

Παρακάτω δίνονται συνοπτικά οι λειτουργίες κάθε μίας παραμέτρου που αναγνωρίζεται από το csa.

ΠΑΡΆΜΕΤΡΟΣ	ΛΕΙΤΟΥΡΓΊΑ
`-h`	Εμφάνιση σύντομης βοήθειας προς τον χρήστη.
`-i`	Ανάγνωση του κώδικα από την standard είσοδο.
`-l`	Ενεργοποίηση των συντακτικών μετασχηματισμών βρόχων.
`-s`	Ενεργοποίηση των συντακτικών μετασχηματισμών εντολών `if`.
`-c`	Κράτηση για μελλοντική επέκταση.
`-e`	Κράτηση για μελλοντική επέκταση.
`-p`	Επιδιόρθωση της αρίθμησης του κώδικα με χρήση line directives.
`-f`	Αφαίρεση του κώδικα των header files από την έξοδο.

Πίνακας I-1: Συνοπτική περιγραφή των παραμέτρων του csa

ΠΑΡΑΡΤΗΜΑ ΙΙ: *ΕΓΧΕΙΡΙΔΙΟ ΧΡΗΣΗΣ ΤΟΥ ΕΡΓΑΛΕΙΟΥ CCOV*

ΙΙ.Ι Η βασική εντολή εκτέλεσης του ccov

Η βασική εντολή εκτέλεσης του εργαλείου csa είναι η:

```
Ccov [OPTION]... [-o SINGLE_OUTPUT_FILE]
     -c COMMENT_SYMBOL GCOV_FILE... ASM_FILE... [VCG_FILE...]
```

- OPTION: Παράμετρος του προγράμματος.
- SINGLE_OUTPUT_FILE: Το όνομα του αρχείου εξόδου.
- COMMENT_SYMBOL: Το σύμβολο που χρησιμοποιείται για την παράθεση σχολίων στον assembly κώδικα του cross compiler.
- GCOV_FILE: Αρχείο gcov με πληροφορίες για ένα αρχείο πηγαίου κώδικα της εφαρμογής.
- ASM_FILE: Ένα αρχείο assembly, έξοδος του cross compiler.
- VCG_FILE: Ένα αρχείο περιγραφής του Γ.Ρ.Ε. κάποιου αρχείου assembly της εφαρμογής.

Η βασική λειτουργία του ccov είναι να διαβάζει τον κώδικα μηχανής των αρχείων ASM_FILE και τις πληροφορίες από τα GCOV_FILE και συνδυάζοντας τες να παράγει πληροφορία για το πλήθος εκτελέσεων κάθε εντολής του κώδικα μηχανής. Επιπλέον, με τη κατάλληλη παραμετροποίηση, το ccov μπορεί να διαβάσει τον Γ.Ρ.Ε. της εφαρμογής από τα αρχεία VCG_FILE και να τον αναπαράγει επισημειωμένο με διάφορες χρήσιμες πληροφορίες (όπως το πλήθος εκτελέσεων των basic blocks και των αλμάτων). Απαραίτητη παράμετρος για τη λειτουργία του ccov είναι η παράμετρος `-c COMMENT_SYMBOL` με την οποία δηλώνεται το σύμβολο που χρησιμοποιεί ο assembly κώδικας που παράγεται από τον cross compiler για την έναρξη σχολίων (πχ. για ARM: `-c @`).

ΙΙ.ΙΙ Περιγραφή των αρχείων εισόδου

Οι βασικοί τύποι αρχείων εισόδου του ccov, είναι τα assembly αρχεία και τα gcov αρχεία. Αν θέλουμε να χρησιμοποιήσουμε το ccov για την εξαγωγή πληροφορίας σχετικά με το πλήθος εκτελέσεων των εντολών ενός αρχείου foo.s, τότε θα πρέπει να δώσουμε ως παραμέτρους στο ccov το όνομα του assembly αρχείου (foo.s) και το όνομα του αρχείου gcov που περιέχει πληροφορίες για τον αρχείο πηγαίου κώδικα από το οποίο παράχθηκε το foo.s (foo.c.gcov). Τα δύο αρχεία πρέπει να έχουν όμοιο κύριο μέρος ονόματος και να διαφέρουν μόνο στη κατάληξη. Το ccov αναγνωρίζει

τους τύπους αρχείων από τη κατάληξή τους (.c.gcov και .s) και τα αντιστοιχίζει μεταξύ τους μέσω του κύριου μέρους του ονόματός τους.

Τα αρχεία assembly (.s) θα πρέπει οπωσδήποτε να έχουν παραχθεί με χρήση των παραμέτρων –g –dA και -dp στον gcc. Οι παράμετροι αυτοί επισημειώνουν τα assembly αρχεία με τις απαραίτητες πληροφορίες που χρησιμοποιεί το ccov για να παράγει αποτελέσματα. Τα αρχεία gcov θα πρέπει να έχουν παραχθεί μόνο με τη προαιρετική χρήση της παραμέτρου –a στο εργαλείο gcov.

Εάν θέλουμε να παραχθεί από το ccov και ο Ε.Γ.Ρ.Ε. για κάθε αρχείο assembly, θα πρέπει αναγκαστικά να τροφοδοτήσουμε το ccov με ένα ακόμα τύπο αρχείων (`VCG_FILE`). Τα αρχεία αυτά μπορούν να παραχθούν από τον gcc με χρήση των παραμέτρων –fdump-rtl-<pass> και –dv. Με τις παραμέτρους αυτές ο gcc παράγει ένα επιπλέον αρχείο που περιγράφει τον Γ.Ρ.Ε. της εσωτερικής αναπαράστασης RTL στο στάδιο <pass>. Για να συμπίπτει αυτή η αναπαράσταση με τον Γ.Ρ.Ε. του assembly κώδικα, θα πρέπει να επιλεχθεί το τελικό στάδιο της RTL αναπαράστασης. Μία επιλογή που έχει δοκιμαστεί και λειτουργεί σωστά, είναι η επιλογή του σταδίου alignments (-fdump-rtl-alignments). Το αρχείο που παράγεται από τον gcc με αυτές τις παραμέτρους έχει όνομα της μορφής:

```
<main_name>.c.<pass_number>r.<pass_name>.vcg
```

Όπου <main_name> είναι το κύριο όνομα του αρχείου, <pass_number> ο αριθμός του σταδίου της εσωτερικής αναπαράστασης RTL και <pass_name> το όνομα της εσωτερικής αναπαράστασης RTL. Ένα παράδειγμα ονόματος θα μπορούσε να ήταν το foo.c.207r.alignments.vcg. Το αρχείο αυτό, πριν δοθεί ως είσοδος στο ccov θα πρέπει να μετονομασθεί στη μορφή

```
<main_name>.vcg
```

Δηλαδή στο παράδειγμά μας θα πρέπει να μετονομασθεί σε foo.vcg, ώστε να μπορεί να αναγνωρισθεί από το gcov και να αντιστοιχηθεί με το σωστό αρχείο assembly. Οι παράμετρος –fdump-rtl-<pass> έχει ως αποτέλεσμα και τη δημιουργία ενός επιπλέον αρχείου με όνομα της μορφής

```
<main_name>.c.<pass_number>r.<pass_name>
```

Το αρχείο αυτό δεν είναι χρήσιμο στο ccov και μπορεί να διαγραφθεί. Για περισσότερες πληροφορίες μπορείτε να ανατρέξετε στο εγχειρίδιο χρήσης του gcc [3].

II.III Περιγραφή των αρχείων εξόδου

Οι βασικοί τύποι αρχείων που παράγει το ccov είναι τρείς:

1. Τα αρχεία ccov (.ccov) είναι αρχεία που περιέχουν πληροφορία για πλήθος εκτελέσεων κάθε μίας εντολής του κώδικα μηχανής.

2. Τα αρχεία gxml (.gxml) είναι μία αναπαράσταση του target Ε.Γ.Ρ.Ε. σε μορφή xml.

3. Τα αρχεία gdl (.gdl) περιέχουν επίσης μία αναπαράσταση του Ε.Γ.Ρ.Ε . σε μία μορφή κατάλληλη για γραφική απεικόνιση [33]. Τα αρχεία αυτά μπορούν να απεικονισθούν γραφικά με χρήση εργαλείων όπως το xvcg [48] και το aiSee [49].

Οι τύποι αρχείων .ccov και .gxml είναι νέοι τύποι αρχείων και γι' αυτό περιγράφονται αναλυτικά στις επόμενες ενότητες.

```
-:    -:Assembly file :test.cross.s
-:    -:Gcov files :test.cross.c.gcov .
-:    -:Source files :test.native.c .
-:    -:Instructions :32
-:    -:Basic blocks :6
-:    -:Instructions' size :128
-:    -:Instructions Executed :4299
-:    -:Bytes transfered :17196
0:    1:.file "test.cross.c"
0:    2:.section .debug_abbrev,"",%progbits
0:    3:.Ldebug_abbrev0:
0:    4:.section .debug_info,"",%progbits
0:    5:.Ldebug_info0:
0:    6:.section .debug_line,"",%progbits
0:    7:.Ldebug_line0:
0:    8:.text
0:    9:.Ltext0:
0:   10:.align 2
0:   11:.global main
0:   12:.type main, %function
0:   13:main:
0:   14:.LFB2:
0:   15:.file 1 "test.cross.c"
0:   16:@ test.cross.c:1
0:   17:.loc 1 1 0
0:   18:@ args = 0, pretend = 0, frame = 12
0:   19:@ frame_needed = 1, uses_anonymous_args = 0
0:   20:@ basic block 2
0:   20:
3:   21:mov ip, sp @ 61 *arm_movsi_insn/1 [length = 4]
0:   22:.LCFI0:
3:   23:stmfd sp!, {fp, ip, lr, pc} @ 62 *push_multi [length = 4]
0:   24:.LCFI1:
3:   25:sub fp, ip, #4 @ 63 *arm_addsi3/2 [length = 4]
0:   26:.LCFI2:
3:   27:sub sp, sp, #12 @ 64 *arm_addsi3/2 [length = 4]
0:   28:.LCFI3:
0:   29:@ test.cross.c:3
0:   30:.loc 1 3 0
3:   31:mov r3, #1 @ 5 *arm_movsi_insn/1 [length = 4]
3:   32:str r3, [fp, #-20] @ 6 *arm_movsi_insn/5 [length = 4]
                           .
                           .
```

Κώδικας ΙΙ-1: Ένα παράδειγμα αρχείου ccov

II.III.I Τα αρχεία ccov

Ο τύπος αρχείου ccov ακολουθεί τη μορφή του τύπου αρχείων .gcov. Παραπάνω (Κώδικας II-1) δίνεται ένα παράδειγμα αρχείου ccov και ακολουθεί η περιγραφή του.

Όπως φαίνεται παραπάνω ένα αρχείο ccov δομείται σε τρείς στήλες οι οποίες διαχωρίζονται με τον χαρακτήρα ‘:’. Στη τρίτη στήλη εμφανίζεται αυτούσιος ο assembly κώδικας της εφαρμογής, στη δεύτερη στήλη εμφανίζονται οι αριθμοί γραμμής του assembly κώδικα και στη πρώτη στήλη εμφανίζεται το πλήθος εκτελέσεων της αντίστοιχης εντολής. Οι πρώτες γραμμές του ccov αρχείου περιέχουν επιπλέον ενημερωτικές πληροφορίες οι οποίες διακρίνονται από τον πηγαίο κώδικα της εφαρμογής μας καθώς «μαρκάρονται» με τον χαρακτήρα ‘–‘ στα πεδία του αριθμού γραμμής και του αριθμού εκτελέσεων.

Με τη παράμετρο `–print-length-info` μπορούμε να ζητήσουμε από το ccov να δημιουργήσει ακόμα μία στήλη (πρώτη από αριστερά) στην έξοδό του η οποία θα περιέχει τα μεγέθη των εντολών σε bytes.

```
    -:    -:Assembly file :test.cross.s
    -:    -:Gcov files :test.cross.c.gcov .
    -:    -:Source files :test.native.c .
    -:    -:Instructions :32
    -:    -:Basic blocks :6
    -:    -:Instructions' size :128
    -:    -:Instructions Executed :7165
    -:    -:Bytes transfered :28660
    5:   21:mov ip, sp @ 61 *arm_movsi_insn/1 [length = 4]
    5:   23:stmfd sp!, {fp, ip, lr, pc} @ 62 *push_multi [length = 4]
    5:   25:sub fp, ip, #4 @ 63 *arm_addsi3/2 [length = 4]
    5:   27:sub sp, sp, #12 @ 64 *arm_addsi3/2 [length = 4]
    5:   31:mov r3, #1 @ 5 *arm_movsi_insn/1 [length = 4]
    5:   32:str r3, [fp, #-20] @ 6 *arm_movsi_insn/5 [length = 4]
    5:   35:mov r3, #1 @ 7 *arm_movsi_insn/1 [length = 4]
    5:   36:str r3, [fp, #-16] @ 8 *arm_movsi_insn/5 [length = 4]
    5:   39:ldr r3, [fp, #-20] @ 9 *arm_movsi_insn/4 [length = 4]
    5:   40:cmp r3, #0 @ 10 *arm_cmpsi_insn/1 [length = 4]
    5:   41:bne .L2 @ 11 *arm_cond_branch [length = 4]
    0:   45:mov r3, #0 @ 13 *arm_movsi_insn/1 [length = 4]
    0:   46:str r3, [fp, #-16] @ 14 *arm_movsi_insn/5 [length = 4]
    5:   51:mov r3, #0 @ 17 *arm_movsi_insn/1 [length = 4]
    5:   52:str r3, [fp, #-24] @ 18 *arm_movsi_insn/5 [length = 4]
    5:   53:b .L3 @ 58 *arm_jump [length = 4]
  505:   58:ldr r2, [fp, #-20] @ 23 *arm_movsi_insn/4 [length = 4]
  505:   59:ldr r3, [fp, #-24] @ 24 *arm_movsi_insn/4 [length = 4]
  505:   60:add r3, r2, r3 @ 25 *arm_addsi3/1 [length = 4]
  505:   61:str r3, [fp, #-20] @ 26 *arm_movsi_insn/5 [length = 4]
  505:   64:ldr r2, [fp, #-16] @ 27 *arm_movsi_insn/4 [length = 4]
  505:   65:ldr r3, [fp, #-20] @ 28 *arm_movsi_insn/4 [length = 4]
  505:   66:add r3, r2, r3 @ 29 *arm_addsi3/1 [length = 4]
  505:   67:str r3, [fp, #-16] @ 30 *arm_movsi_insn/5 [length = 4]
  505:   71:ldr r3, [fp, #-24] @ 33 *arm_movsi_insn/4 [length = 4]
  505:   72:add r3, r3, #1 @ 34 *arm_addsi3/1 [length = 4]
  505:   73:str r3, [fp, #-24] @ 35 *arm_movsi_insn/5 [length = 4]
  510:   78:ldr r3, [fp, #-24] @ 38 *arm_movsi_insn/4 [length = 4]
  510:   79:cmp r3, #100 @ 39 *arm_cmpsi_insn/1 [length = 4]
  510:   80:ble .L4 @ 40 *arm_cond_branch [length = 4]
    5:   85:ldr r3, [fp, #-20] @ 43 *arm_movsi_insn/4 [length = 4]
    5:   88:mov r0, r3 @ 48 *arm_movsi_insn/1 [length = 4]
```

Κώδικας II-2: Παράδειγμα αρχείου ccov με ενεργοποιημένη τη σημαία -skip-assembler-directives

Ένα αρχείο ccov πολλές φορές είναι δυσανάγνωστο και εξαιρετικά μακροσκελές λόγω των πολλών directives που προσθέτει ο compiler. Εάν θέλουμε το αρχείο ccov να μη περιέχει αυτά τα directives, μπορούμε να ενεργοποιήσουμε τη παράμετρο `-skip-assembler-directives`. Έτσι το αρχείο παίρνει τη μορφή που φαίνεται παραπάνω (Κώδικας ΙΙ-2).

Παρατηρείστε ότι η αρίθμηση των γραμμών είναι αυτή του αρχικού αρχείου assembly και δεν είναι συνεχής καθώς λείπουν οι ενδιάμεσες γραμμές που περιέχουν

```
        -:    -:Assembly file :test.cross.s
        -:    -:Gcov files :test.cross.c.gcov .
        -:    -:Source files :test.native.c .
        -:    -:Instructions :32
        -:    -:Basic blocks :6
        -:    -:Instructions' size :128
        -:    -:Instructions Executed :8598
        -:    -:Bytes transfered :34392
        -:    -:Basic Block :2, Executions :6, Instructions :11, Executed
instructions :66, Transfered bytes :264, Length :44
        6:   21:mov ip, sp @ 61 *arm_movsi_insn/1 [length = 4]
        6:   23:stmfd sp!, {fp, ip, lr, pc} @ 62 *push_multi [length = 4]
        6:   25:sub fp, ip, #4 @ 63 *arm_addsi3/2 [length = 4]
        6:   27:sub sp, sp, #12 @ 64 *arm_addsi3/2 [length = 4]
        6:   31:mov r3, #1 @ 5 *arm_movsi_insn/1 [length = 4]
        6:   32:str r3, [fp, #-20] @ 6 *arm_movsi_insn/5 [length = 4]
        6:   35:mov r3, #1 @ 7 *arm_movsi_insn/1 [length = 4]
        6:   36:str r3, [fp, #-16] @ 8 *arm_movsi_insn/5 [length = 4]
        6:   39:ldr r3, [fp, #-20] @ 9 *arm_movsi_insn/4 [length = 4]
        6:   40:cmp r3, #0 @ 10 *arm_cmpsi_insn/1 [length = 4]
        6:   41:bne .L2 @ 11 *arm_cond_branch [length = 4]
        -:    -:Basic Block :3, Executions :0, Instructions :2, Executed
instructions :0, Transfered bytes :0, Length :8
        0:   45:mov r3, #0 @ 13 *arm_movsi_insn/1 [length = 4]
        0:   46:str r3, [fp, #-16] @ 14 *arm_movsi_insn/5 [length = 4]
        -:    -:Basic Block :4, Executions :6, Instructions :3, Executed
instructions :18, Transfered bytes :72, Length :12
        6:   51:mov r3, #0 @ 17 *arm_movsi_insn/1 [length = 4]
        6:   52:str r3, [fp, #-24] @ 18 *arm_movsi_insn/5 [length = 4]
        6:   53:b .L3 @ 58 *arm_jump [length = 4]
        -:    -:Basic Block :5, Executions :606, Instructions :11,
Executed instructions :6666, Transfered bytes :26664, Length :44
      606:   58:ldr r2, [fp, #-20] @ 23 *arm_movsi_insn/4 [length = 4]
      606:   59:ldr r3, [fp, #-24] @ 24 *arm_movsi_insn/4 [length = 4]
      606:   60:add r3, r2, r3 @ 25 *arm_addsi3/1 [length = 4]
      606:   61:str r3, [fp, #-20] @ 26 *arm_movsi_insn/5 [length = 4]
      606:   64:ldr r2, [fp, #-16] @ 27 *arm_movsi_insn/4 [length = 4]
      606:   65:ldr r3, [fp, #-20] @ 28 *arm_movsi_insn/4 [length = 4]
      606:   66:add r3, r2, r3 @ 29 *arm_addsi3/1 [length = 4]
      606:   67:str r3, [fp, #-16] @ 30 *arm_movsi_insn/5 [length = 4]
      606:   71:ldr r3, [fp, #-24] @ 33 *arm_movsi_insn/4 [length = 4]
      606:   72:add r3, r3, #1 @ 34 *arm_addsi3/1 [length = 4]
      606:   73:str r3, [fp, #-24] @ 35 *arm_movsi_insn/5 [length = 4]
        -:    -:Basic Block :6, Executions :612, Instructions :3, Executed
instructions :1836, Transfered bytes :7344, Length :12
      612:   78:ldr r3, [fp, #-24] @ 38 *arm_movsi_insn/4 [length = 4]
      612:   79:cmp r3, #100 @ 39 *arm_cmpsi_insn/1 [length = 4]
      612:   80:ble .L4 @ 40 *arm_cond_branch [length = 4]
        -:    -:Basic Block :7, Executions :6, Instructions :2, Executed
instructions :12, Transfered bytes :48, Length :8
        6:   85:ldr r3, [fp, #-20] @ 43 *arm_movsi_insn/4 [length = 4]
        6:   88:mov r0, r3 @ 48 *arm_movsi_insn/1 [length = 4]
```

Κώδικας ΙΙ-3: Παράδειγμα αρχείου ccov με ενεργοποιημένη τη σημαία print-block-info

τα directives. Άλλες παράμετροι που βοηθούν στην αναγνωσιμότητα του κώδικα είναι οι -skip-comments και -skip-empty-lines με τις οποίες αφαιρούνται από την έξοδο τα σχόλια και οι κενές γραμμές αντίστοιχα.

Για να κάνουμε ακόμα πιο ευανάγνωστη την έξοδο του ccov μπορούμε να ενεργοποιήσουμε τη σημαία `-print-block-info` με την οποία εισάγονται στο αρχείο ccov επιπλέον γραμμές με τις οποίες διαχωρίζονται τα basic blocks του κώδικα και δίνονται συνοπτικές πληροφορίες για κάθε basic block (βλ Κώδικας II-3).

Οι γραμμές συνοπτικής περιγραφής των basic blocks διαφέρουν από τις υπόλοιπες καθώς «μαρκάρονται» με τον χαρακτήρα '–' στα πεδία του αριθμού γραμμής και του αριθμού εκτελέσεων. Αν επιθυμούμε αυτές οι γραμμές μπορούν να περιέχουν και μία λίστα με τα labels (ετικέτες) που αντιστοιχούν στο κάθε basic block. Για να ενεργοποιήσουμε αυτή τη λειτουργία πρέπει να δώσουμε στο ccov τη σημαία `-print-label-info`. Σε αυτή τη περίπτωση μία γραμμή περιγραφής basic block μοιάζει κάπως έτσι.

```
-:    -:Basic Block :2, Executions :7, Instructions :11, Executed
instructions :77, Transfered bytes :308, Length :44 Labels :
main.Ldebug_line0 .Ldebug_info0 .Ldebug_abbrev0 .
```

Κώδικας II-4: Παράδειγμα γραμμής περιγραφής basic block με ενεργοποιημένη τη σημαία -print-label-info

Αν επιθυμούμε μία πολύ συνοπτική περιγραφή της εκτέλεσης του κώδικα μηχανής, μπορούμε να ζητήσουμε από το ccov να παράγει μόνο τις γραμμές περιγραφής basic block με χρήση της παραμέτρου –summary.

Κανονικά το ccov θα παράγει ένα ξεχωριστό αρχείο ccov για κάθε ένα αρχείο assembly που επεξεργάζεται. Αν επιθυμούμε όμως μπορούμε να ζητήσουμε από ccov να ενοποιήσει όλα τα αρχεία ccov σε ένα και να προσθέσει επιπλέον στατιστικά για το σύνολο των αρχείων δίνοντας του τη σημαία `-o SINGLE_OUTPUT_FILE`, όπου `SINGLE_OUTPUT_FILE` είναι το όνομα του αρχείου που θέλουμε να δημιουργηθεί..

II.III.II Τα αρχεία gxml

Εκτός από τα αρχεία ccov που δεν δίνουν καμία πληροφορία για τις ακμές του Γ.Ρ.Ε. του κώδικα μηχανής, το ccov μπορεί να παράγει και πλήρη περιγραφή του Ε.Γ.Ρ.Ε. σε μορφή xml. Για να παραχθεί έναν αρχείο με τη περιγραφή του Ε.Γ.Ρ.Ε. θα πρέπει να ενεργοποιηθούν οι παράμετροι –graphs και -print-xml-graph ενώ θα πρέπει για κάθε αρχείο assembly να δοθεί και το αντίστοιχο αρχείο vcg (βλ. ενότητα II.II, σελ 97). Έτσι, για κάθε αρχείο assembly δημιουργείται και ένα αρχείο με κατάληξη .gxml το οποίο περιέχει την xml αναπαράσταση του Ε.Γ.Ρ.Ε. Παρακάτω (Κώδικας II-5) βλέπουμε ένα παράδειγμα αρχείου gxml και έπειτα ακολουθεί η περιγραφή του.

```
<?xml version="1.0" ?>
<cfg instructions="1433">
    <function id="main" basic_block_num="6">
        <basic_block id="main.BB2" executions="1"/>
        <branch source_id="main.BB2" destination_id="main.BB4" executions="1"/>
        <branch source_id="main.BB2" destination_id="main.BB3" executions="0"/>
        <basic_block id="main.BB3" executions="0"/>
        <branch source_id="main.BB3" destination_id="main.BB4" executions="0"/>
        <basic_block id="main.BB4" executions="1"/>
        <branch source_id="main.BB4" destination_id="main.BB6" executions="1"/>
        <basic_block id="main.BB5" executions="101"/>
        <branch source_id="main.BB5" destination_id="main.BB6" executions="101"/>
        <basic_block id="main.BB6" executions="102"/>
        <branch source_id="main.BB6" destination_id="main.BB7" executions="1"/>
        <branch source_id="main.BB6" destination_id="main.BB5" executions="101"/>
        <basic_block id="main.BB7" executions="1"/>
        <branch source_id="main.BB7" destination_id="main.end" executions="1"/>
        <basic_block id="main.end" executions="0"/>
    </function>
</cfg>
```

Κώδικας II-5: Παράδειγμα αρχείου gxml

Όπως βλέπουμε στο παραπάνω παράδειγμα το στοιχείο-ρίζα του αρχείου gxml είναι ένα στοιχείο τύπου "`cgf`" το οποίο έχει μία ιδιότητα (attribute) που ονομάζεται "`instructions`" που συμβολίζει το πλήθος εντολών που εκτελέστηκαν συνολικά

ΣΤΟΙΧΕΙΟ	ΕΠΕΞΗΓΗΣΗ	ΙΔΙΟΤΗΤΕΣ	ΕΠΕΞΗΓΗΣΗ ΙΔΙΟΤΗΤΩΝ
`cgf`	Το στοιχείο-ρίζα του αρχείου gxml.	`instructions`	Το πλήθος εντολών που εκτελέστηκαν.
`function`	Αναπαριστά τον Ε.Γ.Ρ.Ε. μίας συνάρτησης. Στοιχεία-παιδιά είναι τα "`basic_block`" και "`branch`".	`id`	Το όνομα της συνάρτησης
		`basic_block_num`	Το πλήθος των basic blocks της συνάρτησης.
		`address`	Η διεύθυνση της συνάρτησης.
		`length`	Το μέγεθος της συνάρτησης σε bytes.
`basic_block`	Αναπαριστά το basic block του Ε.Γ.Ρ.Ε..	`id`	Ένα μοναδικό όνομα πού αντιστοιχεί στο basic block.
		`executions`	Το πλήθος εκτελέσεων του basic block.
		`address`	Η διεύθυνση του basic block.

		`length`	Το μέγεθος του basic block σε bytes.
`branch`	Αναπαριστά ένα άλμα του Ε.Γ.Ρ.Ε..	`source_id`	Το όνομα του basic block από το οποίο ξεκινά το άλμα.
		`destination_id`	Το όνομα του basic block στο οποίο καταλήγει το άλμα.
		`executions`	Το πλήθος εκτελέσεων του άλματος.

Πίνακας ΙΙ-1: Τα στοιχεία των αρχείων gxml και οι ιδιότητές τους

από το συγκεκριμένο αρχείο. Παιδιά του στοιχείου-ρίζα είναι οι Ε.Γ.Ρ.Ε. όλων των συναρτήσεων που εμφανίζονται στο αντίστοιχο αρχείο assembly (στο παράδειγμά μας μόνο η συνάρτηση main).

Ο Ε.Γ.Ρ.Ε. μίας συνάρτησης αναπαρίσταται με ένα στοιχείο τύπου "`function`". Το στοιχείο αυτό περιέχει δύο ιδιότητες. Η πρώτη είναι η "`id`" η οποία είναι το όνομα της συνάρτησης και η δεύτερη είναι η "`basic_block_num`" η οποία αναπαριστά τον αριθμό των basic block της συνάρτησης. Τα στοιχεία-παιδιά της συνάρτησης είναι στοιχεία τύπου basic block και στοιχεία τύπου branch τα οποία περιγράφουμε παρακάτω.

Τα στοιχεία τύπου basic block αναπαριστούν τα basic block του Ε.Γ.Ρ.Ε.. Αυτά τα στοιχεία έχουν δύο ιδιότητες. Η πρώτη ("`id`") είναι ένα μοναδικό όνομα που αντιστοιχίζεται στο basic block και η δεύτερη ("`executions`") είναι το πλήθος εκτελέσεων του basic block.

Τα στοιχεία τύπου branch αναπαριστούν τις διακλαδώσεις του Ε.Γ.Ρ.Ε.. Αυτά τα στοιχεία έχουν τρείς ιδιότητες. Η ιδιότητα "`source_id`" αναπαριστά το basic block από το οποίο ξεκινά μία ακμή του Γ.Ρ.Ε. και η ιδιότητα "`destination_id`" αναπαριστά το basic block στο οποίο καταλήγει. Και στης δύο περιπτώσεις η τιμή των ιδιοτήτων είναι τα μοναδικά ids των αντίστοιχων basic blocks. Τέλος η ιδιότητα "`executions`" αναπαριστά τον αριθμό εκτελέσεων του άλματος.

Προαιρετικά, τα αρχεία gxml μπορούν να περιέχουν πληροφορία και για τις διευθύνσεις των συναρτήσεων και των basic blocks. Για να ενεργοποιήσουμε αυτή τη λειτουργία αρκεί να ενεργοποιήσουμε τη σημαία `–with-addresses` στο ccov. Σε αυτή τη περίπτωση, το αρχείο gxml παίρνει τη παρακάτω μορφή .

```
<?xml version="1.0" ?>
<cfg instructions="2866">
    <function id="main" basic_block_num="6" address="0" length="0x80">
        <basic_block id="main.BB2" executions="2" address="0x0" length="0x2c"/>
        <branch source_id="main.BB2" destination_id="main.BB4" executions="2"/>
        <branch source_id="main.BB2" destination_id="main.BB3" executions="0"/>
        <basic_block id="main.BB3" executions="0" address="0x2c" length="0x8"/>
        <branch source_id="main.BB3" destination_id="main.BB4" executions="0"/>
        <basic_block id="main.BB4" executions="2" address="0x34" length="0xc"/>
        <branch source_id="main.BB4" destination_id="main.BB6" executions="2"/>
        <basic_block id="main.BB5" executions="202" address="0x40" length="0x2c"/>
        <branch source_id="main.BB5" destination_id="main.BB6" executions="202"/>
        <basic_block id="main.BB6" executions="204" address="0x6c" length="0xc"/>
        <branch source_id="main.BB6" destination_id="main.BB7" executions="2"/>
        <branch source_id="main.BB6" destination_id="main.BB5" executions="202"/>
        <basic_block id="main.BB7" executions="2" address="0x78" length="0x8"/>
        <branch source_id="main.BB7" destination_id="main.end" executions="2"/>
        <basic_block id="main.end" executions="0" address="80" length="0x0"/>
    </function>
</cfg>
```

Κώδικας II-6: Ένα παράδειγμα αρχείου gxml με επισημειωμένες τις διευθύνσεις των στοιχείων του.

Όπως βλέπουμε στο παραπάνω παράδειγμα, το ccov αρχείο εισάγει τις ιδιότητες "`address`" και "`length`" στα στοιχεία "`function`" και "`basic_block`" στις οποίες δίνονται ως τιμές η διεύθυνση του στοιχείου και το μέγεθός του σε bytes αντίστοιχα (σε δεκαεξαδική μορφή).

Στη σελίδα 103 (Πίνακας II-1) παρουσιάζονται συνοπτικά τα διάφορα στοιχεία των gxml αρχείων, η σημασία τους και οι ιδιότητές τους.

II.III.III Τα αρχεία gdl

Τα αρχεία gdl περιέχουν τις ίδιες πληροφορίες με τα αρχεία gxml αλλά σε μορφή έτοιμη για γραφική απεικόνιση στον χήστη. Για την δημιουργία τους απαιτείται η ενεργοποίηση της παραμέτρου `-print-gdl-graph`.

II.IV Παράμετροι ρύθμισης της ακρίβειας του εργαλείου

Το ccov υποστηρίζει τη χρήση κάποιων παραμέτρων με τις οποίες μπορεί να τροποποιηθεί ο τρόπος που εξάγονται τα αποτελέσματα. Οι παράμετροι αυτές είναι οι:

- `-disable-graph-correction`: Η φύση της διαδικασίας που ακολουθείται για την εξαγωγή του Ε.Γ.Ρ.Ε. από το ccov είναι τέτοια που σε κάποιες περιπτώσεις μπορεί να δημιουργηθούν σφάλματα στους γράφους, όπως κόμβοι για τους οποίους δεν ισχύουν οι εξισώσεις (3.1) και (3.2). Το ccov προσπαθεί, όποτε είναι δυνατό, να διορθώνει αυτά τα σφάλματα. Με τη παράμετρο `-disable-graph-correction` απενεργοποιείται ο μηχανισμός διόρθωσης του γράφου.

- `-const-inst-len-<len>`: Η παράμετρος αυτή γνωστοποιεί στο ccov ότι οι εντολές της αρχιτεκτονικής του target επεξεργαστή έχουν σταθερό μέγεθος και ίσο με <len> (σε bytes). Αυτό μπορεί κάποιες φορές να οδηγήσει σε καλύτερη ακρίβεια του εκτιμώμενου αριθμού εκτελουμένων εντολών.

- -level-asm-<level algorithm>: Κάποιες φορές, λόγω αστοχίας της μεθόδου, μπορεί να βρεθεί ότι εντολές του ίδιου basic block του κώδικα μηχανής έχουν διαφορετικούς αριθμούς εκτελέσεων. Η παράμετρος -level-asm-<level algorithm> καθορίζει τον αλγόριθμο με τον οποίο θα επιδιορθωθεί το παραπάνω σφάλμα.

- -level-gcov-<level algorithm>: Όπως και στα basic block του κώδικα μηχανής, έτσι και στον πηγαίο κώδικα μπορεί να βρεθεί ότι μία γραμμή έχει παραπάνω από έναν αριθμούς εκτελέσεων (πχ. αν δεν γίνει χρήση του csa). Η παράμετρος -level-gcov-<level algorithm> καθορίζει τον αλγόριθμο με τον οποίο θα επιδιορθωθεί το παραπάνω σφάλμα.

Οι διαθέσιμοι αλγόριθμοι -<level algorithm> είναι οι εξής:

- `first`: Ολόκληρο το basic block /γραμμή χαρακτηρίζεται από τον πρώτο αριθμό εκτελέσεων που εμφανίζεται.

- `last`: Το basic block /γραμμή χαρακτηρίζεται από τον τελευταίο αριθμό εκτελέσεων που εμφανίζεται.

- `greater`: Το basic block /γραμμή χαρακτηρίζεται από τον μεγαλύτερο αριθμό εκτελέσεων που εμφανίζεται.

- `smaller`: Το basic block /γραμμή χαρακτηρίζεται από τον μικρότερο αριθμό εκτελέσεων που εμφανίζεται.

- `average`: Το basic block /γραμμή χαρακτηρίζεται από τη μέση τιμή των αριθμών εκτελέσεων.

- `most-frequent`: Το basic block /γραμμή χαρακτηρίζεται από τον πιο συχνά εμφανιζόμενο αριθμό εκτελέσεων.

II.V Παράμετροι για επιπρόσθετες λειτουργίες

Το ccov παρέχει κάποιες επιπλέον λειτουργίες που μπορεί να φανούν χρήσιμες σε ορισμένες περιπτώσεις. Παρακάτω θα αναφερθούμε σε αυτές επιγραμματικά.

- `-symbol-list <SYMBOL_LIST>`: Όπως είδαμε στην ενότητα II.III.II το ccov μπορεί να επισημειώσει τα αρχεία gxml με τις διευθύνσεις των

basic blocks. Αυτές οι διευθύνσεις, ωστόσο, είναι σχετικές με αναφορά την αρχή της κάθε συνάρτησης. Αν θέλουμε να εμφανίζονται οι πραγματικές διευθύνσεις των basic block στο εκτελέσιμο αρχείο αντί των σχετικών διευθύνσεων, θα πρέπει χρησιμοποιήσουμε τη παράμετρο `-symbol-list <SYMBOL_LIST>` όπου `<SYMBOL_LIST>` είναι το όνομα ενός αρχείου που περιέχει την έξοδο της εντολής nm –S για το τελικό εκτελέσιμο που παράγει ο cross compiler.

- `-generate-address-list`: Το ccov μπορεί, επίσης, να παράγει μία συνοπτική λίστα με τις συναρτήσεις τις οποίες επεξεργάστηκε αναφέροντας για κάθε μία τη διεύθυνσή της και το μέγεθός της. Κάτι τέτοιο μπορεί να φανεί χρήσιμο για λόγους debugging. Η παράμετρος με την οποία ενεργοποιείται αυτή η λειτουργία είναι η `-generate-address-list` και το αποτέλεσμα θα αποθηκευτεί σε ένα αρχείο με όνομα “addres_list”.

- `-suppress-warnings`: Απενεργοποιεί τις προειδοποιήσεις (warnings).

II.VI Συνοπτική περιγραφή όλων των παραμέτρων του ccov

Παρακάτω (Πίνακας II-2) δίνονται συνοπτικά οι λειτουργίες κάθε μίας παραμέτρου που αναγνωρίζεται από το ccov.

ΠΑΡΆΜΕΤΡΟΣ	ΛΕΙΤΟΥΡΓΊΑ
`-help`	Εμφάνιση σύντομης βοήθειας προς τον χρήστη.
`-suppress-warnings`	Απενεργοποίηση των προειδοποιήσεων (warnings).
`-print-length-info`	Επισημείωση του αρχείου ccov με τα μεγέθη των εντολών μηχανής.
`-print-block-info`	Δημιουργία γραμμών σύντομης περιγραφής των basic block στα αρχεία ccov.
`-print-label-info`	Επέκταση των γραμμών περιγραφής basic block με τη λίστα ετικετών του basic block (σε συνδυασμό με -print-block-info).

`-summary`	Παραγωγή συνοπτικών αρχείων ccov
`-skip-comments`	Αφαίρεση των σχόλιων από τα αρχεία ccov
`-skip-assembler-directives`	Αφαίρεση των assembler-directives από τα αρχεία ccov
`-skip-empty-lines`	Αφαίρεση των κενών γραμμών από τα αρχεία ccov
`-graphs`	Ενεργοποίηση του μηχανισμού ανάγνωσης Γ.Ρ.Ε. από τα αρχεία vcg
`-symbol-list <symbol_list>`	Ανάγνωση των πραγματικών διευθύνσεων από το αρχείο <symbol_list> (σε συνδυασμό με -graphs)
`-print-gdl-graph`	Αποθήκευση του Ε.Γ.Ρ.Ε. σε μορφή gdl για γραφική απεικόνιση
`-print-xml-graph`	Αποθήκευση του Ε.Γ.Ρ.Ε. σε μορφή xml
`-with-addresses`	Επισημείωση του Ε.Γ.Ρ.Ε. με τις διευθύνσεις των συναρτήσεων και των basic blocks
`-generate-address-list`	Παραγωγή λίστας όλων των συναρτήσεων που επεξεργάστηκαν και των διευθύνσεων τους (σε συνδυασμό με -graphs)
`-disable-graph-correction`	Απενεργοποίηση του μηχανισμού διόρθωσης γράφων
`-const-inst-len-<len>`	Δήλωση σταθερού μεγέθους εντολών
`-level-asm-<level_algorithm>`	Διόρθωση των basic block με πολλαπλούς αριθμούς εκτελέσεων με χρήση του αλγορίθμου <level_algorithm>
`-level-gcov-<level_algorithm>`	Διόρθωση των γραμμών με πολλαπλούς αριθμούς εκτελέσεων με χρήση του αλγορίθμου <level_algorithm>

Πίνακας ΙΙ-2: Συνοπτική περιγραφή των παραμέτρων του ccov

ΠΑΡΑΡΤΗΜΑ ΙΙΙ *ΕΓΧΕΙΡΙΔΙΟ ΧΡΗΣΗΣ ΤΟΥ ΕΡΓΑΛΕΙΟΥ CANAL*

III.I Η βασική εντολή εκτέλεσης του canal

Η βασική εντολή εκτέλεσης του εργαλείου canal είναι η:

```
canal cxml_file [-tthread_id=gxml_file]...
      [option]...
```

- *cxml_file*: Το αρχείο περιγραφής αρχιτεκτονικών κρυφής μνήμης εντολών
- *thread_id*: Αριθμός νήματος εκτέλεσης
- *gxml_file*: xml αναπαράσταση ενός Ε.Γ.Ρ.Ε.
- *option*: Παράμετρος εκτέλεσης του εργαλείου

Με την εντολή αυτή διαβάζεται από το αρχείο *cxml_file* η περιγραφή ενός συνόλου αρχιτεκτονικών κρυφής μνήμης εντολών για τις οποίες στη συνέχεια γίνεται εκτίμηση επίδοσης. Οι εφαρμογές για τις οποίες δοκιμάζονται οι αρχιτεκτονικές περιγράφονται μέσω του Ε.Γ.Ρ.Ε. τους με τη χρήση αρχείων gxml (βλ. σελ. 101 για περιγραφή των αρχείων gxml).

III.II Αρχεία περιγραφής αρχιτεκτονικών κρυφής μνήμης (cxml).

Το εργαλείο canal διαβάζει τις περιγραφές των αρχιτεκτονικών μνήμης από έναν νέο τύπου αρχείου (cxml). Τα αρχεία αυτά χρησιμοποιούν XML για να περιγράψουν ένα σύνολο διαφορετικών αρχιτεκτονικών μνήμης. Το canal κάνει εκτίμηση της επίδοσης όλων των αρχιτεκτονικών που περιγράφονται στο cxml αρχείο ταυτόχρονα και εμφανίζει τα αποτελέσματα στον χρήστη. Παρακάτω (Κώδικας ΙΙΙ-1) δίνεται ένα παράδειγμα αρχείου cxml.

Η έννοια του κάθε στοιχείου είναι αρκετά προφανής, ωστόσο θα ακολουθήσει μία πιο αναλυτική περιγραφή για κάθε έναν τύπο στοιχείου.

architecture_explorer: Το στοιχείο ρίζας ενός αρχείου cxml πρέπει πάντα να είναι ένα στοιχείο τύπου *architecture_explorer*. Τα στοιχεία-παιδιά του στοιχείου ρίζας είναι οι περιγραφές των αρχιτεκτονικών μνήμης.

memory_architecture: Τα στοιχεία *memory_architecture* (παιδιά του στοιχείου ρίζα *architecture_explorer*) αναπαριστούν αρχιτεκτονικές κρυφής μνήμης. Συγκεκριμένα ένα στοιχείο τύπου *memory_architecture* μπορεί να περιέχει ως παιδιά του ένα στοιχείο τύπου *id* και ένα σύνολο στοιχείων

cache. Το στοιχείο τύπου *id* θα πρέπει να περιέχει κείμενο το οποίο θα αποτελέσει ένα μοναδικό όνομα για την αρχιτεκτονική, ενώ τα στοιχεία *cache* αποτελούν τις περιγραφές όλων των μονάδων κρυφής μνήμης που περιέχει η αρχιτεκτονική.

```
<?xml version="1.0" ?>
<architecture_explorer>
    <memory_architecture>
        <id> ARCH1 </id>
        <cache>
            <id> L1 </id>
            <size> 1k </size>
            <line_size> 8 </line_size>
            <associativity> 4 </associativity>
            <miss_penalty>100</miss_penalty>
            <replacement_policy>lru</replacement_policy>
            <bus>thread1</bus>
        </cache>
        <cache>
            <id> L2 </id>
            <size> 512 </size>
            <line_size> 32 </line_size>
            <associativity>4</associativity>
            <miss_penalty>1000</miss_penalty>
            <replacement_policy>lru</replacement_policy>
            <bus>L1</bus>
            <bus>thread2</bus>
        </cache>
    </memory_architecture>
    <memory architecture>
        <id> ARCH2 </id>
        <cache>
            <id> L1 </id>
            <size> 128 </size>
            <line_size> 8 </line_size>
            <associativity> 4 </associativity>
            <miss_penalty>100</miss_penalty>
            <replacement_policy>lru</replacement_policy>
            <bus>thread1</bus>
        </cache>
    </memory_architecture>
```

Κώδικας ΙΙΙ-1: Ένα παράδειγμα αρχείου περιγραφής αρχιτεκτονικών κρυφής μνήμης (cxml).

cache: Κάθε στοιχείο τύπου *cache* περιγράφει μία μονάδα κρυφής μνήμης και τη διασύνδεσή της. Τα στοιχεία τα οποία μπορούν να περιέχονται ως παιδιά σε ένα στοιχείο τύπου *cache* είναι τα παρακάτω:

- ***id***: Σε αυτό το στοιχείο πρέπει να δοθεί το όνομα της μονάδας κρυφής μνήμης. Το όνομα αυτό θα πρέπει να είναι μοναδικό στην αρχιτεκτονική.
- ***size***: Το μέγεθος της μονάδας κρυφής μνήμης. Έγκυρες τιμές αποτελούν θετικοί ακέραιοι αριθμοί οι οποίοι μπορούν να συνοδεύονται από το επίθεμα k (kilobytes) ή M (Megabytes).
- ***line_size***: Το μέγεθος του block μνήμης. Έγκυρες τιμές αποτελούν μόνο οι δυνάμεις του 2. Ο αριθμός μπορεί επίσης να συνοδεύονται από το επίθεμα k (kilobytes) ή M (Megabytes).

- ***associativity***: Ο βαθμός συσχετικότητας της μονάδας μνήμης. Έγκυρες τιμές αποτελούν Έγκυρες τιμές αποτελούν μόνο οι δυνάμεις του 2.

- ***miss_penalty***: Το κόστος αστοχίας μνήμης σε κύκλους ρολογιού. Έγκυρες τιμές αποτελούν οι θετικοί ακέραιοι αριθμοί.

- ***replacement_policy***: Η πολιτική αντικατάστασης τις μνήμης. Έγκυρες τιμές αποτελούν οι λέξεις: "lru" (least recently used), "mru" (least recently used), "rr" (random replacement). Από τις τρείς πολιτικές μόνο η "lru" είναι υλοποιημένη στη παρούσα έκδοση του εργαλείου. Για τις υπόλοιπες εκκρεμεί η υλοποίηση.

- ***bus***: Το στοιχείο *bus* είναι αυτό που περιγράφει τη διασύνδεση της μονάδας κρυφής μνήμης. Ένα στοιχείο τύπου *bus* μπορεί να περιέχει είτε το *id* ενός άλλου στοιχείου *cache* τις αρχιτεκτονικής ή ένα ενδεικτικό νήματος εκτέλεσης. Ένα ενδεικτικό νήματος εκτέλεσης έχει τη μορφή *thread<n>* όπου *<n>* είναι ένας θετικός ακέραιος αριθμός (πχ. *thread1*, *thread289*). Ο αριθμός *<n>* ονομάζεται thread id.

  Αν το στοιχείο *bus* μίας μονάδας κρυφής μνήμης (έστω Α) έχει ως τιμή το *id* μίας άλλης μονάδας κρυφής μνήμης (έστω Β), αυτό σημαίνει ότι η μνήμη Α δρα ως μνήμη επόμενου επιπέδου για τη μνήμη Β. Επομένως σε περίπτωση αστοχίας, η μνήμη Β θα αναζητήσει τα ζητούμενα δεδομένα από τη μνήμη Α.

  Αν όμως η τιμή του στοιχείου bus είναι ένα ενδεικτικό νήματος εκτέλεσης, αυτό σημαίνει ότι η μονάδα κρυφής μνήμης είναι συνδεδεμένη απ' ευθείας σε μία μονάδα επεξεργασίας (έστω Μ). Σε αυτή τη περίπτωση, το thread id του ενδεικτικού νήματος εκτέλεσης θα πρέπει αντιστοιχηθεί σε ένα ή περισσότερα αρχεία τύπου gxml με χρήση της παραμέτρου -t. Με αυτόν τον τρόπο δηλώνουμε ότι ο κώδικας των gxml αρχείων εκτελέσθηκε στη μονάδα Μ.

  Η σύνταξη της παραμέτρου –t είναι η παρακάτω:

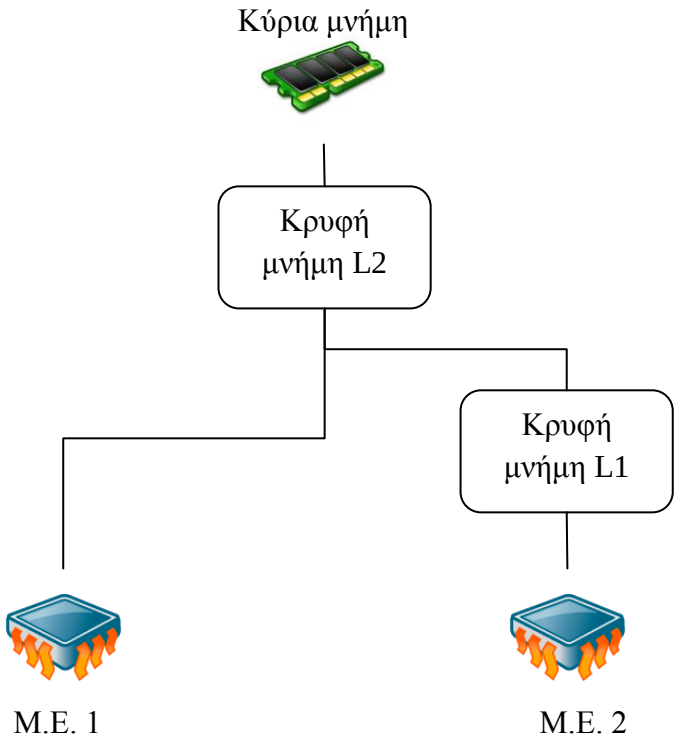
  ```
  -t<thread id>="<gxml_file1> <gxml_file2> ..."
  ```

  Αν πχ. θέλουμε να δηλώσουμε ότι τα αρχεία f1.gxml και f2.gxml περιγράφουν κώδικα που εκτελέστηκε στο νήμα εκτέλεσης 3 θα πρέπει να δώσουμε στο canal τη παράμετρο:

  ```
  -t1="f1.gxml f2.gxml"
  ```

  Ένα στοιχείο τύπου *cache* μπορεί να περιέχει περισσότερα από ένα στοιχεία *bus*. Αυτό σημαίνει ότι μία μονάδα κρυφής μνήμης μπορεί να

είναι συνδεδεμένη σε περισσότερες από μία οντότητες (άλλες μονάδες κρυφής μνήμης ή μονάδες επεξεργασίας). Με αυτόν τον τρόπο μπορούμε να περιγράφουμε σύνθετες αρχιτεκτονικές μνήμης για αρχιτεκτονικές πολλών επεξεργαστών.



Σχήμα ΙΙΙ-1: Γραφική αναπαράσταση της αρχιτεκτονικής μνήμης ARCH1 που περιγράφεται στο παράδειγμα αρχείου cxml (Κώδικας ΙΙΙ.1)

ΙΙΙ.ΙΙΙ Η έξοδος του canal

Η έξοδος του canal προς τον χρήστη χωρίζεται σε δύο ενότητες. Στη πρώτη ενότητα παρουσιάζεται μία σύντομη σύνοψη της παραμετροποίησης του canal και στη δεύτερη ενότητα παρουσιάζονται οι εκτιμήσεις επίδοσης για τις αρχιτεκτονικές κρυφής μνήμης που περιγράφονται στο cxml αρχείο. Για κάθε αρχιτεκτονική και για κάθε μονάδα κρυφής μνήμης παρουσιάζονται διάφορα στατιστικά όπως η ποσότητα δεδομένων που μεταφορτώθηκε, το πλήθος των αστοχιών, ο ρυθμός αστοχίας και η συνολική χρονική επιβάρυνση λόγω των αστοχιών (miss penalty). Αν επιθυμούμε, οι αρχιτεκτονικές μπορούν εμφανιστούν ταξινομημένες με βάση τη μέση τιμή ρυθμού αστοχίας η τη συνολική χρονική επιβάρυνση λόγο αστοχίας. Οι αντίστοιχες παράμετροι είναι οι –sm και –sp.

Αν ενεργοποιηθεί η παράμετρος –ts τότε πριν την ενότητα εκτιμήσεων εισάγεται μία επιπλέον ενότητα στην οποία εμφανίζονται στατιστικά για κάθε ένα αρχείο gxml (πλήθος εντολών). Παρακάτω εμφανίζεται ένα παράδειγμα της εξόδου του canal με ενεργοποιημένη την παράμετρο –ts.

```
Canal: A fast instruction cache analyser.
=========================================

General information
===================

  Word size:          32 bits
  Sort architectures: no

Thread statistics
=================

  Thread 1 instructions:
    test.cross.gxml          : 1433
    -------------------------
    Total thread instructions: 1433

Memory architectures' statistics
================================

  Memory Architecture: A_single_cache
  -----------------------------------
    Total instructions:      1433
    Average cache misses:    5
    Average cache miss rate: 0.35%
    Total penalty:           5

    Cache: L1 (S: 128, LS: 32, A: 1, RP: lru, MP: 1)
    Statistics:
      Bytes transfered: 5784
      Fetch demands:    1443
      Misses:           5
      Miss rate:        0.35%
      Penalty:          5
```

Κώδικας III-2: Ένα παράδειγμα εξόδου του εργαλείου canal

III.IV Παράμετροι λειτουργίας

Το canal υποστηρίζει δύο επιπλέον παραμέτρους:

`-dw`: Απενεργοποίηση των προειδοποιήσεων προς τον χρήστη (warnings)

`-ws` *`word_size`*: Δήλωση του μεγέθους λέξης της αρχιτεκτονικής. Με τη παράμετρο αυτή δηλώνεται ότι το μέγεθος λέξης είναι ίσο με *`word_size`* bytes. Η παράμετρος αυτή χρησιμεύει στον σωστό υπολογισμό των αιτημάτων ανάγνωσης (fetch demands) προς μία μονάδα κρυφής μνήμης.

III.V Συνοπτική περιγραφή όλων των παραμέτρων του canal

Παρακάτω () δίνονται συνοπτικά οι λειτουργίες κάθε μίας παραμέτρου που αναγνωρίζεται από το ccov.

ΠΑΡΆΜΕΤΡΟΣ	ΛΕΙΤΟΥΡΓΊΑ
`-h`	Εμφάνιση σύντομης βοήθειας προς τον χρήστη.
`-dw`	Απενεργοποίηση των προειδοποιήσεων (warnings)
`-sm`	Ταξινόμηση των αρχιτεκτονικών μνήμης με βάση τη μέση τιμή ρυθμού αστοχίας.
`-sp`	Ταξινόμηση των αρχιτεκτονικών μνήμης με βάση τη συνολική χρονική επιβάρυνση λόγω αστοχιών.
`-ts`	Εμφάνιση στατιστικών για τα αρχεία gxml.
`-ws` *`word_size`*	Δήλωση του μεγέθους λέξης της αρχιτεκτονικής.

Πίνακας III-1: Συνοπτική περιγραφή των παραμέτρων του canal

ΠΑΡΑΡΤΗΜΑ IV *ΑΥΤΟΜΑΤΟΠΟΙΗΣΗ ΤΗΣ ΔΙΑΔΙΚΑΣΙΑΣ ΕΚΤΙΜΗΣΗΣ*

Όπως αναφέρθηκε στην ενότητα 4.4, για τη διευκόλυνση του χρήστη έχει υλοποιηθεί ένα Makefile το οποίο αναλαμβάνει να δρομολογήσει τις απαραίτητες διαδικασίες της μεθόδου αυτόματα. Σε αυτό το παράρτημα δίνεται μία συνοπτική περιγραφή του πως μπορούμε προσαρμόσουμε το Makefile σε μία οποιαδήποτε εφαρμογή για να κάνουμε εκτίμηση της επίδοσης της.

Για να μπορεί το Makefile να εντοπίσει τα εργαλεία csa, ccov και canal θα πρέπει αυτά να βρίσκονται σε έναν κατάλογο που έχει δηλωθεί σε μία μεταβλητή περιβάλλοντος με όνομα *NXFICA*. Σον ίδιο κατάλογο μπορεί να εισαχθεί και το ίδιο το Makefile. Εναλλακτικά ο χρήστης μπορεί να δηλώσει άμεσα τις θέσεις των εργαλείων αν επεξεργαστεί τις αρχικοποιήσεις των μεταβλητών CCOV, CSA και CANAL εντός του Makefile.

IV.I Δημιουργία ενός Makefile χρήστη για την εκτίμηση επίδοσης της εφαρμογής μας

Για να προσαρμόσουμε λοιπόν τη διαδικασία εκτίμησης στη δική μας εφαρμογή θα πρέπει να δημιουργήσουμε ένα νέο Makefile χρήστη στο οποίο θα πρέπει να δώσουμε τιμές σε 16 μεταβλητές (όχι υποχρεωτικά σε όλες) και στο τέλος του οποίου θα συμπεριλάβουμε το Makefile που αυτοματοποιεί τη διαδικασία με την εντολή include (βλ. Κώδικας IV-1).

Παρακάτω αναλύεται η λειτουργία της κάθε μίας μεταβλητής:

SOURCE: Η μεταβλητή αυτή πρέπει να αρχικοποιηθεί στα αρχεία πηγαίου κώδικα της εφαρμογής. Όλα τα αρχεία με κατάληξη .c πρέπει να παρατεθούν χωρισμένα με κενό.

COMMENT_SYMBOL: Το σύμβολο που χρησιμοποιεί ο compiler του target επεξεργαστή για να εισάγει σχόλια στον assembly κώδικα. Απαραίτητη πληροφορία για να μπορέσει να λειτουργήσει σωστά το εργαλείο ccov.

CROSS_EXECUTABLE: Το όνομα που θέλουμε να δοθεί στο τελικό εκτελέσιμο αρχείο για την αρχιτεκτονική του target επεξεργαστή.

CROSS_COMPILER: Ο cross compiler που θέλουμε να χρησιμοποιηθεί. Πχ. για την αρχιτεκτονική ARM θα μπορούσε να είναι arm-elf-gcc.

```
SOURCE = main.c dir1/f1.c dir2/f2.c

COMMENT_SYMBOL=@

CROSS_EXECUTABLE = cross_exec
CROSS_COMPILER = arm-elf-gcc
CROSS_COMPILER_FLAGS =
CROSS_LINKER = arm-elf-gcc
                     CROSS_LINKER_FLAGS = -lm

NATIVE_EXECUTABLE = native_exec
NATIVE_COMPILER = gcc
NATIVE_COMPILER_FLAGS =
NATIVE_LINKER = gcc
NATIVE_LINKER_FLAGS = -lm

EXECUTION_COMMAND = ./$(NATIVE_EXECUTABLE)

ACTUAL_ADDRESSES=1
CCOV_FLAGS = -level-asm-greater -level-gcov-greater \
             -print-block-info -summary -print-gdl-graph
CSA_FLAGS = -l -s -p

include $(NXFICA)/Makefile.include
```

Κώδικας IV-1**: Παράδειγμα Makefile για αυτοματοποίηση της διαδικασίας εκτίμησης**

CROSS_COMPILER_FLAGS (προαιρετικό): Οι παράμετροι που θέλουμε να περάσουμε στον CROSS_COMPILER.

CROSS_LINKER: Ο linker που θέλουμε να χρησιμοποιηθεί για τη σύνδεση των κώδικα που προορίζεται για τον target επεξεργαστή.

CROSS_LINKER_FLAGS (προαιρετικό): Οι παράμετροι που θέλουμε να περάσουμε στον CROSS_LINKER.

NATIVE_EXECUTABLE: Το όνομα που θέλουμε να δοθεί στο τελικό εκτελέσιμο αρχείο για την αρχιτεκτονική του host υπολογιστή.

NATIVE_COMPILER: Ο native compiler που θέλουμε να χρησιμοποιηθεί. Συνήθης επιλογή είναι η "gcc".

NATIVE_COMPILER_FLAGS (προαιρετικό): Οι παράμετροι που θέλουμε να περάσουμε στον NATIVE_COMPILER.

NATIVE_LINKER: Ο linker που θέλουμε να χρησιμοποιηθεί για τη σύνδεση των κώδικα που προορίζεται για τον host υπολογιστή.

NATIVE_LINKER_FLAGS (προαιρετικό): Οι παράμετροι που θέλουμε να περάσουμε στον NATIVE_LINKER.

EXECUTION_COMMAND: Η εντολή με την οποία εκτελείται η εφαρμογή στον host υπολογιστή.

AVTUAL_ADDRESSES (προαιρετικό): Αυτή η μεταβλητή μπορεί να αρχικοποιηθεί στο 1 ή στο 0 ανάλογα με αν επιθυμούμε τα αρχεία gxml να περιέχουν απόλυτες οι σχετικές διευθύνσεις (βλ. παράμετρος -symbol-list του ccov).

CCOV_FLAGS (προαιρετικό): Οι επιπλέον παράμετροι που θέλουμε να δοθούν στο ccov. Η απαραίτητες παράμετροι για τη διεξαγωγή της διαδικασίας είναι προρυθμισμένοι. Σε αυτή τη μεταβλητή καθορίζονται μόνο επιπλέον παράμετροι που μπορεί να επιθυμεί ο χρήστης (βλέπε εγχειρίδιο χρήσης ccov).

CSA_FLAGS (προαιρετικό): Οι επιπλέον παράμετροι που θέλουμε να δοθούν στο csa. Η απαραίτητες παράμετροι για τη διεξαγωγή της διαδικασίας είναι προρυθμισμένοι. Σε αυτή τη μεταβλητή καθορίζονται μόνο επιπλέον παράμετροι που μπορεί να επιθυμεί ο χρήστης (βλέπε εγχειρίδιο χρήσης csa). Αν δεν αρχικοποιηθεί αυτή η μεταβλητή το csa θα κάνει απλή στοίχιση και δεν θα εφαρμόζει κανένα συντακτικό μετασχηματισμό.

IV.II Τρόπος χρήσης του Makefile που δημιουργήσαμε

Από τη στιγμή που έχουμε έτοιμο το Makefile μπορούμε να κάνουμε χρήση των παρακάτω «στόχων»:

nxfica: Δημιουργία ενός αρχείου τύπου gxml και ενός αρχείου τύπου ccov για κάθε ένα αρχείο πηγαίου κώδικα της εφαρμογής μας (βλ. II.III.I και II.III.II για περιγραφή των αρχείων ccov και gxml).

cleanstats: Καθαρισμός των στατιστικών εκτέλεσης. Αν εκτελέσουμε πολλές φορές διαδοχικά τον στόχο nxfica τότε τα gxml και ccov αρχεία θα περιέχουν στατιστικά για όλες συνολικά τις εκτελέσεις. Αν θέλουμε να το αποφύγουμε αυτό θα πρέπει πριν την κάθε εκτέλεση του στόχου nxfica να εκτελούμε τον στόχο cleanstats.

clean: Διαγραφή όλων των ενδιάμεσων αρχείων που δημιουργήθηκαν από τη διαδικασία (και των στατιστικών εκτέλεσης).

distclean: Διαγραφή όλων των ενδιάμεσων αρχείων, των στατιστικών εκτέλεσης και των αρχείων gxml και ccov.

Αφού λοιπόν δημιουργήσουμε τα αρχεία τύπου gxml με χρήση του Makefile, στη συνέχεια μπορούμε να προχωρήσουμε σε εκτίμηση της επίδοσης της ιεραρχίας μνήμης σύμφωνα με τις οδηγίες που δίνονται στο εγχειρίδιο χρήσης του canal.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] N. Kroupis and D. Soudris, "High-Level Estimation Methodology for Designing the Instruction Cache Memory of Programmable Embedded Platforms," *The Institution of Engineering and Technology, IET Computer and Digital Techniques*, vol. 3, 2009.

[2] UNIX International, July 1993. [Online]. Available: http://dwarfstd.org/doc/dwarf-2.0.0.pdf.

[3] Free Software Foundation, Inc, [Online]. Available: http://gcc.gnu.org/.

[4] M. Barr, 8 2009. [Online]. Available: http://www.eetimes.com/electronics-blogs/industrial-control-designline-blog/4027479/Real-men-program-in-C.

[5] J. Edler and M. Hill, "Dinero IV Trace-Driven Uniprocessor Cache Simulator," [Online]. Available: http://pages.cs.wisc.edu/~markhill/DineroIV/.

[6] T. Austin, E. Larson and D. Ernst, "SimpleScalar: an infrastructure for computer system modeling computer," *IEEE Comput.*, vol. 2, no. 35, pp. 59-67, 2002.

[7] I. Software. [Online]. Available: http://www.ovpworld.org/.

[8] R. A. Uhlig and M. T. N., "Trace-driven memory simulation: a survey," *ACM Comput. Surv.*, vol. 2, no. 29, p. 128–170, 1997.

[9] A. Agarwal, R. L. Sites and M. Horowitz, "A new technique for capturing address traces using microcode," in *In Proceedings of the 13th International Symposium on Computer Architecture*, Tokyo, 1987.

[10] Wikipedia. [Online]. Available: http://en.wikipedia.org/wiki/Microcode.

[11] L. L. and P. J. K., "Cache Sampling by Sets," *IEEE Trans. VLSI Systems*, vol. 1, pp. 98-105, 6 1993.

[12] A. Borg, R. Kessler and D. Wall, "Generation and analysis of very long address traces," *Int. Symp. Computer Architecture*, pp. 270-279, May 1990.

[13] D. Whalley, "Fast instruction cache performance evaluation using compile-time analysis," in *Conf. Measurement and Modeling of Computer*

Systems, Newport, Rhode Island, 1992.

[14] R. Sugumar and S. Abraham, "Set associative cache simulation using generalized Binomial trees," *ACM Trans. Comput. Syst.*, vol. 1, no. 13, pp. 32-56, 1995.

[15] A. Janapsatya, A. Ignjatovic and S. Parameswaran, "Finding optimal L1 cache configuration for embedded systems," in *Conf. Asia South Pacific Design Automation*, Yokohama, Japan, 2006.

[16] V. P., G.-R. A., B. E. and F. Vahid, "A table based method for single pass cache optimization," *ACM Great Lakes Symp. VLSI (GLSVLSI)*, May 2008.

[17] E. Schnarr and J. Larus, "Fast out-of-order processor simulation using memoization," *Proc. 8th Int. Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOSVIII)*, p. 283–294, 10 1998.

[18] A. Nohl, G. Braun, O. Schilebusch, R. Leupres and H. Meyr, "A universal technique for fast and flexible instruction-set architecture simulation," in *Proc. 39th Conf. Design Automation, DAC 2002*, New Orleans, Louisiana, USA, 2002.

[19] A. Ghosh and T. Givargis, "Cache optimization for embedded processor cores: an analytical approach," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 4, no. 9, p. 419–440, 2004.

[20] M. Oskin, F. Chong and M. Farrens, "HLS: Combining statistical and symbolic simulation to guide microprocessor designs," *Proc. 27th Annual Int. Symp. Computer Architecture (ISCA-27)*, p. 71–82, 6 2000.

[21] L. Eeckhout and K. De Bosschere, "Hybrid analytical–statistical modeling for efficiently exploring architecture and workload design spaces," *Proc. 20th IEEE Int. Performance, Computing and Communication Conf. (IPCCC 2001)*, p. 196–204, 4 2001.

[22] X. Vera and J. Xue, "Let's study whole-program cache behavior analytically," *Proc. 8th Int. Symp. High Performance Computer Architecture (HPCA)*, p. 176–185, 2 2002.

[23] F. Mueller and D. Whalley, "Fast instruction cache analysis via static cache simulation," *Proc. 28th Annual Simulation Symp.*, p. 105–114, 1995.

[24] N. Liveris, N. Zervas, D. Soudris and C. Goutis, "A code transformation-based methodology for improving I-cache performance of DSP

applications," in *Proc. DATE*, Paris, 2002.

[25] Wikipedia, "Basic block," [Online]. Available: http://en.wikipedia.org/wiki/Basic_block.

[26] Wikipedia. [Online]. Available: http://en.wikipedia.org/wiki/Code_coverage.

[27] Free Software Foundation, Inc, "gcov—a Test Coverage Program," [Online]. Available: http://gcc.gnu.org/onlinedocs/gcc/Gcov.html.

[28] Wikipedia, "Spanning tree," [Online]. Available: http://en.wikipedia.org/wiki/Spanning_tree.

[29] B. Holger, G. Frank and U. Jürgen, "Gcov on an embedded system," in *Workshop: GCC for Research in Embedded and Parallel Systems*, Brasov, 2007.

[30] Tool Interface Standards (TIS) Committee, [Online]. Available: http://pdos.csail.mit.edu/6.828/2011/readings/elf.pdf.

[31] Free Software Foundation, Inc, [Online]. Available: http://www.gnu.org/software/binutils/.

[32] Richard M. Stallman and the GCC Developer Community, [Online]. Available: http://gcc.gnu.org/onlinedocs/gccint.pdf.

[33] AbsInt Angewandte Informatik, [Online]. Available: http://www.aisee.com/gdl/nutshell/.

[34] Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Heuristic.

[35] Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Depth-first_search.

[36] W3C, [Online]. Available: http://www.w3.org/XML/.

[37] Free Software Foundation, Inc., [Online]. Available: http://www.gnu.org/s/bison/.

[38] D. Jutta. [Online]. Available: http://www.quut.com/c/ANSI-C-grammar-y.html.

[39] The Flex Project, [Online]. Available: http://flex.sourceforge.net/.

[40] L. Thomason. [Online]. Available:

http://www.grinninglizard.com/tinyxml/.

[41] Free Software Foundation, Inc., [Online]. Available: http://www.gnu.org/s/make/.

[42] Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Named_pipe.

[43] OpenMP, [Online]. Available: http://openmp.org/wp/.

[44] P. Prieto, V. Puente and J. Gregorio, "Multilevel Cache Modeling for Chip-Multiprocessor Systems," *Computer Architecture Letters,* no. 99, pp. 1-1, 7 2011.

[45] Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Binary_File_Descriptor_library.

[46] D. Brooks, V. Tiwari and M. Martonosi, "Wattch: a framework for architecturallevel power analysis and optimizations," in *Proceedings of the 27th annual international symposium on Computer architecture*, New York, 2000.

[47] Free Software Foundation, Inc, [Online]. Available: http://gcc.gnu.org/onlinedocs/gcc/C-Extensions.html#C-Extensions.

[48] G. Sander. [Online]. Available: http://rw4.cs.uni-sb.de/~sander/html/gsvcg1.html.

[49] AbsInt Angewandte Informatik GmbH, [Online]. Available: http://www.aisee.com/.

[50] Free Software Foundation, Inc, "The "stabs" representation of debugging information," [Online]. Available: http://sourceware.org/gdb/current/onlinedocs/stabs.html.

ΕΥΡΕΤΗΡΙΟ ΟΡΩΝ