Ατομική Διπλωματική Εργασία

ΜΕΛΕΤΗ ΚΑΙ ΑΝΑΛΥΣΗ ΤΕΧΝΙΚΩΝ ΠΡΟΒΛΕΨΗΣ

Νικόλας Λαδάς

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2008

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μελέτη και Ανάλυση Τεχνικών Πρόβλεψης

Νικόλας Λαδάς

Επιβλέπων Καθηγητής
Γιαννάκης Σαζεΐδης

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

Μάιος 2008

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον κ. Σαζεΐδη για την συνεχή βοήθεια και καθοδήγηση που μου πρόσφερε σε όλη τη διάρκεια της εργασίας αυτής.

Επίσης θα ήθελα να ευχαριστήσω την αρραβωνιαστικιά μου Έλενα για τη βοήθεια και συμπαράσταση της.

Περίληψη

Στην εργασία αυτή έγινε μια μελέτη των διαφόρων τεχνικών πρόβλεψης. Μελετήθηκαν διάφοροι μηχανισμοί που προτάθηκαν κατά καιρούς από ερευνητές. Στις τεχνικές που μελετήθηκαν περιλαμβάνονται οι: bimodal, local history, global history, gshare, combining, perceptron, GEHL και L-TAGE predictors. Για τον κάθε ένα από αυτούς τους μηχανισμούς πρόβλεψης παρουσιάζουμε τη λειτουργία του καθώς και τυχόν ιδιαιτερότητες, προτερήματα και μειονεκτήματα που μπορεί να παρουσιάζει. Ταυτόχρονα παρουσιάζουμε κάποια από τα χαρακτηριστικά που πρέπει να έχει μια εντολή διακλάδωσης για να είναι εύκολα προβλέψιμη από ένα μηχανισμό πρόβλεψης.

Στη συνέχεια παρουσιάζουμε μια ανάλυση που έγινε με σκοπό να διαπιστωθούν ελλείψεις και αδυναμίες σε ένα από τους πιο σύγχρονους και ακριβής μηχανισμούς πρόβλεψης, τον L-TAGE. Για την ανάλυση αυτή ορίζουμε ένα δικό μας στατιστικό, το entropy, το οποίο αποτελεί μια ένδειξη για το ποσοστό πληροφορίας που υπάρχει σε διάφορα μεγέθη ιστορίας και μπορεί να χρησιμοποιηθεί για πρόβλεψη. Στη συνέχεια συγκρίνουμε το entropy με την ακρίβεια του predictor για διάφορες εντολές που ο predictor έχει χαμηλή ακρίβεια πρόβλεψης.

Από την ανάλυση αυτή καταλήξαμε στο εξής συμπέρασμα: Για τις δύσκολες εντολές διακλάδωσης υπάρχει κάποιο περιθώριο βελτίωσης από την αναγνώριση συνεχόμενων μοτίβων ιστορίας, αλλά αυτό το περιθώριο είναι σχετικά μικρό. Για τις εντολές αυτές, τα μεγάλα μεγέθη ιστορίας δεν επιφέρουν αύξηση στην ακρίβεια πρόβλεψης.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

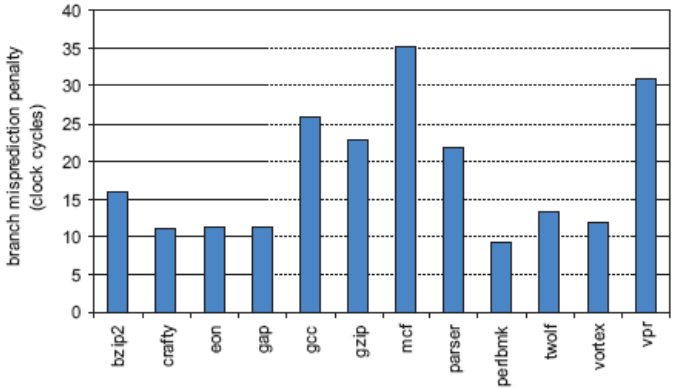
1.1 Ρόλος της πρόβλεψης στους σημερινούς επεξεργαστές

Οι σημερινοί επεξεργαστές με στόχο την αύξηση της επίδοσης χρησιμοποιούν διασωλήνωση μεγάλου βάθους. Οι εξαρτήσεις έλεγχου που προκαλούνται από εντολές διακλάδωσης αποτελούν εμπόδιο στη συνεχή λειτουργία ενός διασωληνωμένου επεξεργαστή. Αυτό γίνεται γιατί οι εντολές διακλάδωσης πρέπει να εκτελεστούν ώστε να γίνει γνωστό ποια είναι η επόμενη εντολή προς εκτέλεση. Αυτό έχει ως αποτέλεσμα να προκαλείται καθυστέρηση, ανάλογη του βάθους της διασωλήνωσης, κάθε φορά που παρουσιάζεται μια τέτοια εντολή.

Ένας τρόπος για βελτίωση της επίδοσης είναι με τη χρήση πρόβλεψης. Όταν εμφανιστεί μια εντολή διακλάδωσης αποφασίζουμε με τη χρήση κάποιου μηχανισμού ποιο θα είναι το αποτέλεσμα της πριν αυτή η εντολή εκτελεστεί. Έτσι μπορεί να συνεχιστεί η εκτέλεση των εντολών που ακολουθούν χωρίς καθυστέρηση. Όταν το αποτέλεσμα της εντολής διακλάδωσης γίνει γνωστό, ελέγχουμε κατά πόσο η πρόβλεψη που έγινε ήταν σωστή. Σε περίπτωση σωστής πρόβλεψης η εκτέλεση συνεχίζει κανονικά. Σε αντίθετη περίπτωση όλες οι εντολές που μπήκαν στον επεξεργαστή μετά από την εντολή διακλάδωσης πρέπει να αναιρεθούν.

Στους σημερινούς επεξεργαστές η διασωλήνωση μπορεί να αποτελείται από πολλά στάδια. Για παράδειγμα ο Pentium 4 της Intel έχει 20 στάδια διασωλήνωσης [3]. Το αποτέλεσμα

μιας εντολής διακλάδωσης είναι γνωστό προς το τέλος της διασωλήνωσης. Αυτό σημαίνει πως σε περίπτωση λανθασμένης πρόβλεψης ένα μεγάλο μέρος του επεξεργαστή περιέχει άχρηστες εντολές οι οποίες πρέπει να αναιρεθούν για να συνεχίσει η εκτέλεση. Σε ένα τέτοιο επεξεργαστή οι λανθασμένες προβλέψεις έχουν μεγάλο κόστος στην επίδοση [1] όπως βλέπουμε στο σχήμα 1.1. Ως αποτέλεσμα, η ακρίβεια του μηχανισμού πρόβλεψης έχει άμεση επίπτωση στην συνολική επίδοση του επεξεργαστή.



Σχήμα 1.1. Επιπτώσεις λανθασμένων προβλέψεων[1].

1.2 Στόχος της εργασίας

Οι στόχος της εργασίας αυτής είναι η μελέτη διαφόρων τεχνικών πρόβλεψης. Γίνεται μια προσπάθεια περιγραφής της λειτουργίας κάποιων από τους πιο αντιπροσωπευτικούς μηχανισμούς πρόβλεψης που παρουσιάστηκαν τα τελευταία χρόνια, ξεκινώντας από τους πιο απλούς μηχανισμούς και συνεχίζοντας μέχρι τους σημερινούς πιο εξεζητημένους.

Γίνεται μια πειραματική μελέτη ενός από τους τελευταίους μηχανισμούς πρόβλεψης που έχουν προταθεί, του L-TAGE. Η μελέτη μας επικεντρώθηκε στην ανάλυση διαφόρων μεγεθών ιστορίας. Για τα διάφορα μεγέθη ιστορίας προσπαθούμε να αναγνωρίσουμε κατά πόσο υπάρχει πληροφορία που δεν εκμεταλλεύεται ο μηχανισμός πρόβλεψης, την οποία θα μπορούσε να εκμεταλλευτεί για να γίνει πιο ακριβής. Για τις περιπτώσεις στις οποίες ο μηχανισμός πρόβλεψης όντως υπολειτουργεί επιχειρούμε να αναγνωρίσουμε την αιτία και να προτείνουμε κάποιες λύσεις.

1.3 Συνεισφορές

Στην εργασία αυτή δείξαμε, με τη χρήσης ενός μηχανισμού ο οποίος υπολογίζει την προδιάθεση ενός pattern ιστορίας προς μια κατεύθυνση, πως για λίγες, δύσκολες εντολές διακλάδωσης υπάρχει μικρό περιθώριο βελτίωσης της ακρίβειας πρόβλεψης με τη χρήση συνεχόμενων bits από τον history register. Για τις περισσότερες εντολές διακλάδωσης που εξετάσαμε, δείξαμε πως δεν υπάρχει βελτίωση από την χρήση συνεχόμενων bits της ιστορίας, πράγμα που υποδηλώνει πως πρέπει να εξεταστούν άλλοι μέθοδοι για βελτίωση των μηχανισμών πρόβλεψης όπως η αναγνώριση συσχετίσεων σε μη συνεχόμενα bits ιστορίας [6].

Επίσης δείξαμε πως η μεγάλη αύξηση του δικού μας υπολογισμού - entropy - γίνεται σε σχετικά μικρά μεγέθη ιστορίας οπότε τα μεγέθη αυτά είναι πιο σημαντικά γα την πρόβλεψη. Αυτό υποστηρίζει τη χρήση μικρού μεγέθους ιστορίας για δύσκολα προγράμματα, πράγμα που γίνεται σε δύο από τους τελευταίους μηχανισμούς πρόβλεψης που προτάθηκαν [7] [8].

1.4 Δομή εργασίας

Η δομή της εργασίας που ακολουθεί έχει ως εξής. Στο δεύτερο κεφάλαιο παρουσιάζουμε κάποια χαρακτηριστικά των προγραμμάτων που καθιστούν τις εντολές διακλάδωσης εύκολα προβλέψιμες. Στη συνέχεια παρουσιάζουμε κάποιους από τους πιο αντιπροσωπευτικούς μηχανισμούς πρόβλεψης που έχουν προταθεί. Προσπάθεια μας ήταν να κρατήσουμε ιστορική σειρά στην παρουσίαση των μηχανισμών αυτών ώστε να δείξουμε και την πορεία εξέλιξής τους. Οι μηχανισμοί που παρουσιάζονται είναι ο bimodal, local history, global history, gshare, combining, perceptron, GEHL και L-TAGE predictor. Για τους μηχανισμούς αυτούς παρουσιάζουμε τη λειτουργία τους, τα θετικά και τα αρνητικά τους στοιχεία.

Στο τρίτο κεφάλαιο παρουσιάζουμε την πειραματική ανάλυση που κάναμε. Αρχικά αναφέρονται τα εργαλεία που χρησιμοποιήθηκαν και αποτελούν μέρος προηγούμενης

εργασίας. Στη συνέχεια παρουσιάζουμε τη διάταξη του πειράματος. Ακολουθεί ο τρόπος που εκτελέσαμε την ανάλυση και η διαδικασία εξαγωγής στατιστικών.

Στο τέταρτο κεφάλαιο παρουσιάζουμε τα αποτελέσματα της ανάλυσης μας καθώς και κάποιες παρατηρήσεις που εξάγονται από αυτά. Στο πέμπτο κεφάλαιο παραθέτουμε τα συμπεράσματα μας από την εργασία αυτή καθώς και κάποιες κατευθύνσεις για μελλοντική εργασία.

Κεφαλαίο 2

Τεχνικές Πρόβλεψης

2.1 Σύνοψη κεφαλαίου

Στο κεφάλαιο αυτό γίνεται μια ανασκόπηση διαφόρων σημαντικών τεχνικών πρόβλεψης. Οι μηχανισμοί που παρουσιάζονται είναι ο bimodal, local history, global history, gshare, combining, perceptron, GHEL και L-TAGE predictor. Για τους μηχανισμούς αυτούς παρουσιάζουμε τη λειτουργία τους, τα θετικά και τα αρνητικά τους στοιχεία. Επίσης παρουσιάζουμε κάποια χαρακτηριστικά των προγραμμάτων που καθιστούν τις εντολές διακλάδωσης εύκολα προβλέψιμες

2.2 Βασικές Αρχές

Οι τεχνικές πρόβλεψης βασίζονται στο γεγονός ότι οι πολλές εντολές διακλάδωσης παρουσιάζουν προβλέψιμη συμπεριφορά. Ένα τυπικό παράδειγμα φαίνεται πιο κάτω:

```
for  (i=0 ; i<1000 ; i++)
{
//code
}
```

Στον πιο πάνω κώδικα έχουμε μια εντολή διακλάδωσης η οποία παρουσιάζει ιδιαίτερα προβλέψιμη συμπεριφορά, έχει 1000 φορές αρνητικό αποτέλεσμα και μία φορά θετικό. Για τέτοιες εντολές χρειαζόμαστε μόνο τα αποτελέσματα από προηγούμενες εκτελέσεις της ίδιας εντολής (το ιστορικό της εντολής) για να έχουμε ακριβής πρόβλεψη. Οι τεχνικές που εκμεταλλεύονται αυτού του είδους συμπεριφορά λέγονται τοπικές (local history branch prediction schemes).

Κάποιες εντολές διακλάδωσης είναι πιο εύκολα προβλέψιμες αν γνωρίζουμε το αποτέλεσμα προηγούμενων εντολών διακλάδωσης του προγράμματος. Στο πιο κάτω παράδειγμα βλέπουμε μια τέτοια εντολή:

```
if ( x > 2)
{
        //code
}
…
if (x < 0)
{
        //code
}
```

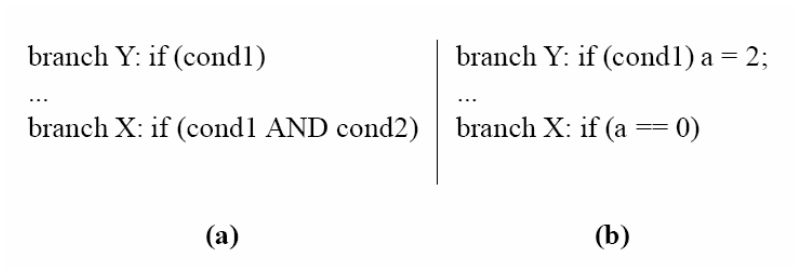
Στο παράδειγμα αυτό μπορεί να είναι εύκολο να προβλέψουμε το δεύτερο if αν γνωρίζουμε το αποτέλεσμα του πρώτου. Οι τεχνικές που χρησιμοποιούν αυτού του είδους πληροφορία ονομάζονται ολικές (global history branch prediction schemes).

Στα υποκεφάλαια που ακολουθούν θα εξετάσουμε διάφορες τεχνικές που χρησιμοποιούν τοπική και ολική ιστορία καθώς και διάφορες μεθόδους που συνδυάζουν τις δύο τεχνικές αυτές για καλύτερα αποτελέσματα. Θα δούμε διάφορες μελέτες που αναγνωρίζουν τα χαρακτηριστικά των προγραμμάτων που καθιστούν προβλέψιμες κάποιες εντολές διακλάδωσης. Επίσης θα εξετάσουμε διάφορες στατικές μεθόδους πρόβλεψης καθώς και κάποιους από τους τελευταίους (και πιο εξελιγμένους) μηχανισμούς πρόβλεψης που έχουν προταθεί από ερευνητές.

2.3 Προβλεψιμότητα και συσχετίσεις των εντολών διακλάδωσης

Όπως περιγράψαμε πιο πάνω, οι εντολές διακλάδωσης παρουσιάζουν συχνά προβλέψιμη συμπεριφορά. Στο υποκεφάλαιο αυτό θα εξετάσουμε διάφορες μελέτες που προσπαθούν να αναγνωρίσουν τα χαρακτηριστικά που καθιστούν τις εντολές διακλάδωσης προβλέψιμες . Στις μελέτες αυτές, η κύρια κατεύθυνση που ακολουθείται είναι η αναγνώριση διαφόρων συσχετίσεων μεταξύ εντολών διακλάδωσης και ο τρόπος που οι συσχετίσεις αυτές βοηθούν τη πρόβλεψη.

Στη μελέτη [2] αναγνωρίστηκαν οι εξής συσχετίσεις μεταξύ εντολών διακλάδωσης:



Σχήμα2.1 Δύο είδη συσχετίσεων [2].

Οι συσχετίσεις στο σχήμα 2.1 ονομάζονται συσχετίσεις κατεύθυνσης (direction correlations). Στο (a) παρατηρούμε πως η εντολή διακλάδωσης X έχει ως μέρος του ελέγχου της την εντολή Y. Αυτό σημαίνει πως γνώση για το αποτέλεσμα του Y πιθανό να είναι χρήσιμη για την πρόβλεψη του X. Στο (b) αν η εντολή διακλάδωσης Y έχει αρνητικό

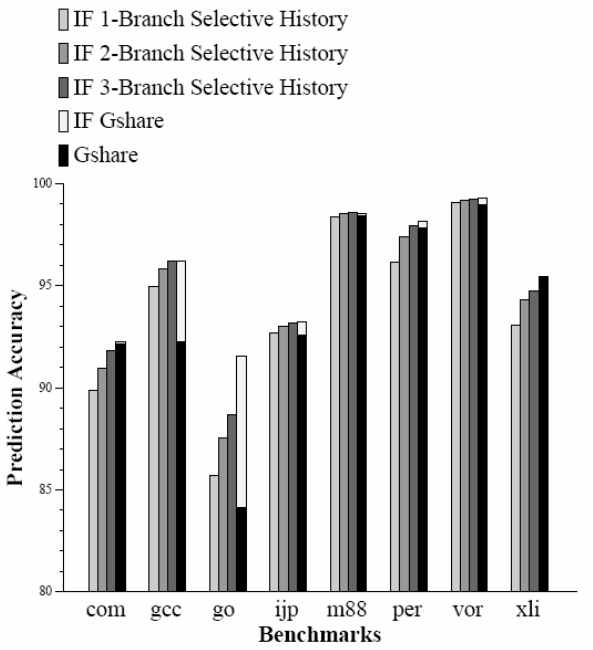
αποτέλεσμα (not taken) τότε επηρεάζει την μεταβλητή που ελέγχει η εντολή X. Πάλι είναι εμφανές πως για την πρόβλεψη του X θα μας ήταν χρήσιμο να γνωρίζουμε το αποτέλεσμα της εντολής διακλάδωσης Y.

Μια άλλη μορφή συσχέτισης που αναγνωρίστηκε από αυτή τη μελέτη φαίνεται πιο κάτω:

Branch Y: if (NOT(cond1))…
Branch Z: else if (NOT(cond2))…
Branch V: else if (cond3)…
…
Branch X: if (cond1 AND cond2)

Στο πιο πάνω παράδειγμα η εντολή V δεν συσχετίζεται με την εντολή X με κάποιο από τους δύο τρόπους που περιγράψαμε πιο πάνω. Παρόλα αυτά η εκτέλεση της εντολής V είναι μια καλή ένδειξη για τη πρόβλεψη του αποτελέσματος της X, αφού η εκτέλεσή της συνεπάγει πως εκτελέστηκαν οι εντολές Y και Z. Η εντολή V μπορεί να είναι ιδιαίτερα χρήσιμη για ένα global predictor αν οι εντολές Y και Z έχουν σημαντική απόσταση από την V, ώστε να μην χωρούν στον global history register τη στιγμή που θα προβλεφθεί η εντολή X.

Στη μελέτη αυτή έγινε μια προσπάθεια να αναγνωριστεί ποιες από τις εντολές διακλάδωσης που βρίσκονται στον global history register είναι πραγματικά χρήσιμες στην πρόβλεψη. Για τον σκοπό αυτό έγινε προσομοίωση του gshare predictor. Στην προσομοίωση που έγινε χρησιμοποιήθηκε ένας υποθετικός predictor του οποίου ο global history register περιέχει μόνο τις μια, δύο ή τρεις προηγούμενες εντολές διακλάδωσης που είναι σημαντικές για την πρόβλεψη. Τα αποτελέσματα της προσομοίωσης φαίνονται στο σχήμα 2.9.



Σχήμα 2.2. Σύγκριση υποθετικού predictor και gshare [2].

Από το σχήμα 2.2 μπορούμε να δούμε πως ο θεωρητικός predictor μόνο με τρία bits μπορεί να πετύχει ψηλότερο ποσοστό ακρίβειας από έναν κανονικό gshare predictor, ο οποίος χρησιμοποιεί 16 bits global history. Αυτό μας οδηγεί στο συμπέρασμα ότι με λίγες προσεκτικά επιλεγμένες εντολές διακλάδωσης μπορούμε να επιτύχουμε ψηλά ποσοστά ακρίβειας. Αυτό επίσης υπονοεί πως οι μηχανισμοί πρόβλεψης που δουλεύουν με παρόμοιο τρόπο, όπως ο gshare, περιέχουν στον global history register τους, πληροφορία που είναι αχρείαστη.

Στη μελέτη [9] αναγνωρίζονται οι συσχετίσεις όπως στο σχήμα 2.1(b) τις οποίες και ονομάζει affectors. Για την αναγνώριση των εντολών διακλάδωσης που αποτελούν affector για κάποια άλλη εντολή χρησιμοποιείται ένας γράφος εξάρτησης (data dependence graph). Οι affector εντολές διακλάδωσης χρησιμοποιούνται σε ένα μηχανισμό πρόβλεψης δευτέρου επιπέδου ο οποίος διορθώνει τα αποτελέσματα ενός πιο απλού αρχικού μηχανισμού, αν αυτά είναι λανθασμένα. Τα αποτελέσματα της μελέτης αυτής δείχνουν πως οι πληροφορίες για affector εντολές μπορεί να είναι χρήσιμες στην πρόβλεψη.

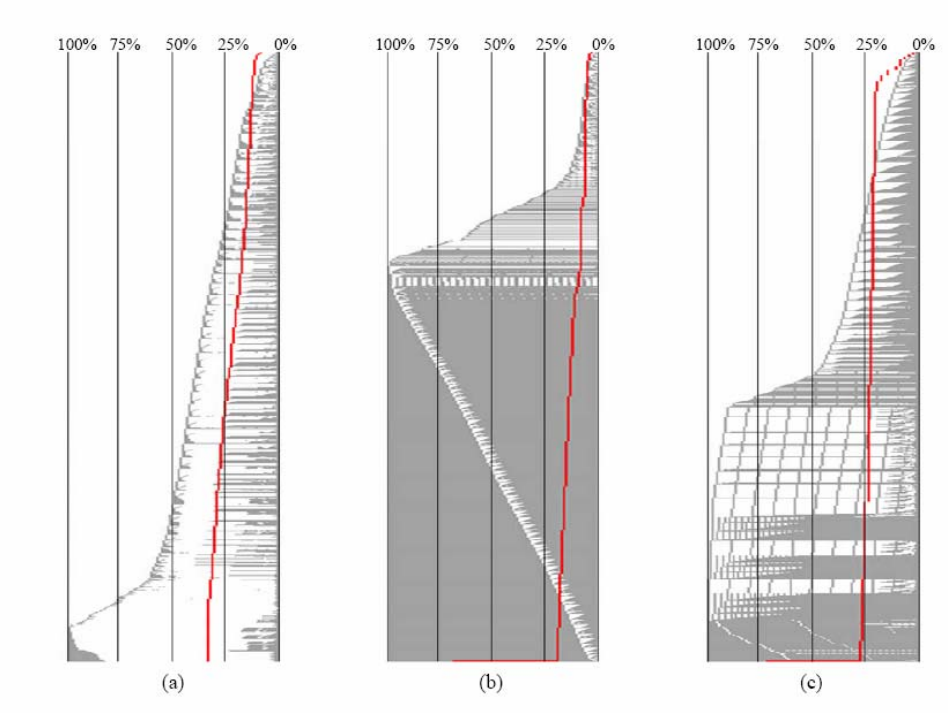
Η μελέτη [6] επιχειρεί να αναλύσει σε μεγαλύτερο βάθος τις συσχετίσεις μεταξύ των εντολών διακλάδωσης. Δίνεται ένας πιο τυπικός ορισμός για τις συσχετίσεις κατεύθυνσης (direction correlations) που είδαμε στο σχήμα 2.1. Πιο συγκεκριμένα δίνονται οι εξής ορισμοί:

- Μια εντολή διακλάδωσης A είναι *affector* μιας άλλης εντολής διακλάδωσης B αν το αποτέλεσμα (taken ή not taken) της A επηρεάζει δεδομένα που χρησιμοποιεί η B.
- Μια εντολή διακλάδωσης A είναι *affectee* μιας άλλης εντολής διακλάδωσης B αν η A ελέγχει το αποτέλεσμα μιας εντολής Γ που εξαρτάται από μια εντολή Δ που ανήκει στο διάγραμμα ροής δεδομένων της B.

Στη μελέτη αυτή χρησιμοποιείται ένας μηχανισμός που καθορίζει τις affector και τις affectee εντολές διακλάδωσης. Ο μηχανισμός αυτός χρησιμοποιεί όπως και στο [9] ένα γράφο εξάρτησης (dataflow graph) για να αναγνωρίσει τους affectors και affectees. Οι πληροφορίες για τις affector και τις affectee εντολές διακλάδωσης τροφοδοτούνται σε δύο διαφορετικούς μηχανισμού πρόβλεψης: GTL και L-TAGE.

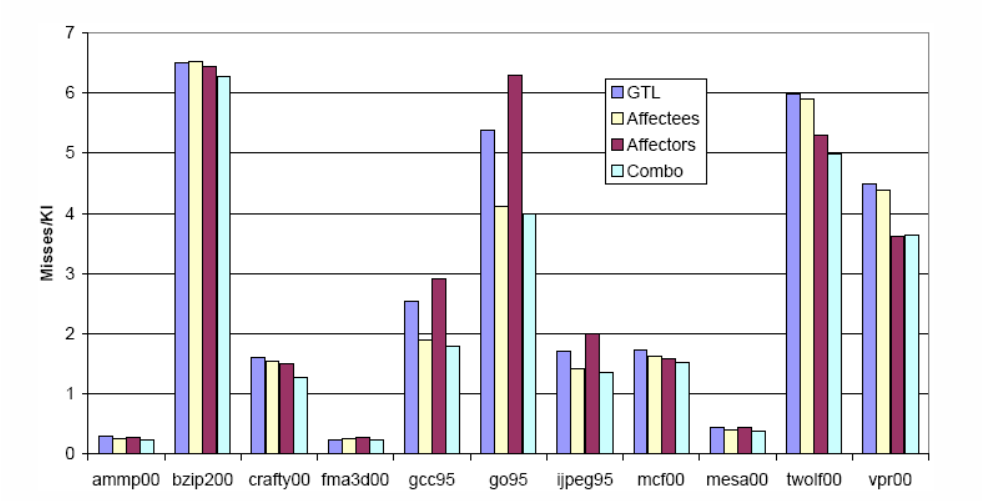
Τα αποτελέσματα του πειράματος αυτού οδήγησαν στις εξής παρατηρήσεις:

- Η πλειονότητα των εντολών διακλάδωσης έχει σημαντικά περισσότερους affectees παρά affectors.
- Για τα περισσότερα προγράμματα που προσομοιώθηκαν 80% των εντολών διακλάδωσης έχουν λιγότερους από 30 affectors και 50% έχουν περισσότερους από 30 affectees. Για κάποια προγράμματα οι affectees ξεπερνούν τους 300.
- Οι εντολές που έχουν συσχέτιση affector ή affectee με κάποια εντολή διακλάδωσης δεν βρίσκονται πάντοτε σε συνεχείς θέσεις στον global history register. Αντίθετα σε πολλές περιπτώσεις παρατηρούνται «τρύπες» στα bits της ιστορίας που έχουν συσχέτιση (σχήμα 2.3).

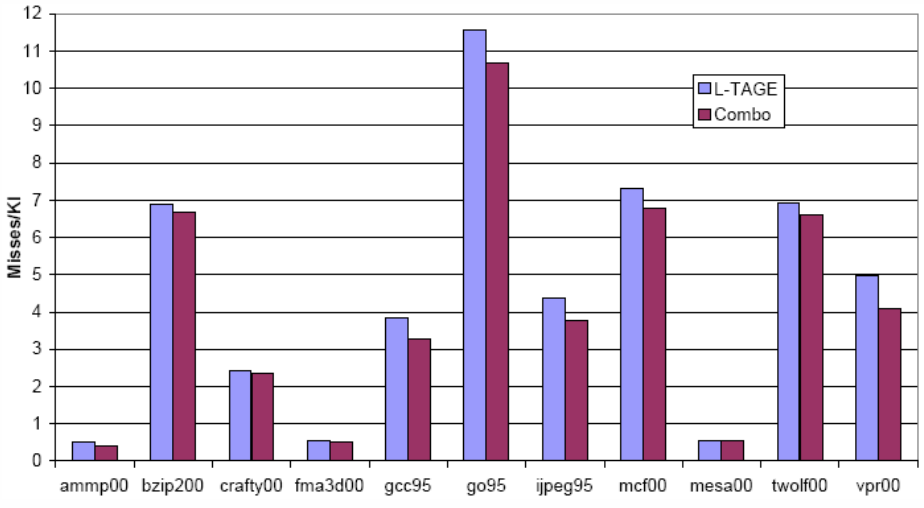


Σχήμα 2.3. Κατανομή των συσχετιζόμενων εντολών [6].

Τα αποτελέσματα από την προσομοίωση των δύο predictors GTL και L-TAGE καθώς και τα αποτελέσματα των predictor που χρησιμοποιούν τις πληροφορίες για affectors και affectees, φαίνονται στα σχήματα 2.4 και 2.5.



Σχήμα 2.4. Σύγκριση affectors, affectees, combo και GTL [6].



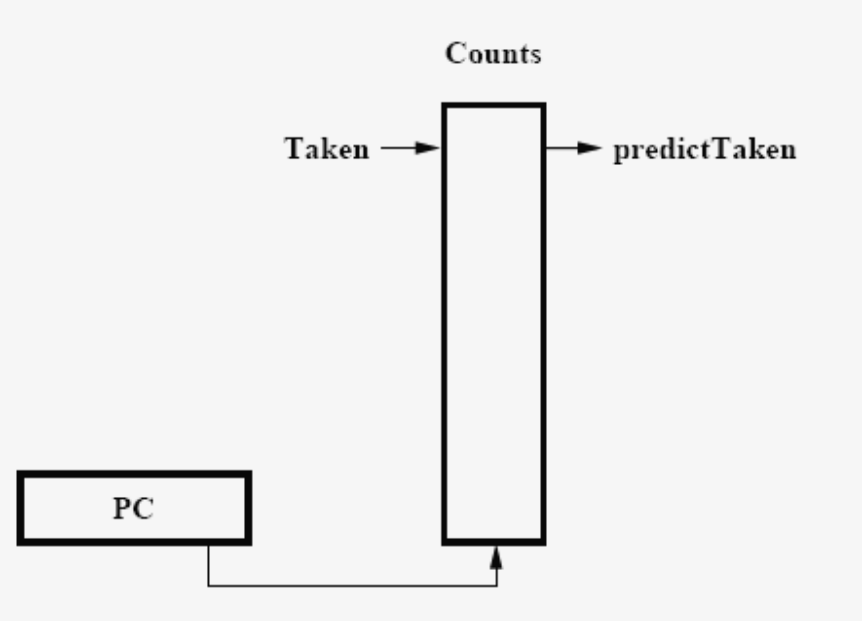
Σχήμα 2.5. Σύγκριση combo - L-TAGE [6].

Από το σχήμα 2.11 και 2.12 παρατηρούμε πως ο συνδυασμός affectors και affectees έχει σχεδόν πάντοτε τα καλύτερα αποτελέσματα Αυτό οδηγεί στο συμπέρασμα πως η χρήση των συσχετίσεων αυτών μπορεί να οδηγήσει σε ψηλότερα ποσοστά ακρίβειας.

2.4 Τοπική (local) και ολική (global) πρόβλεψη

Το υποκεφάλαιο αυτό βασίζεται στο [5].

Η πλειονότητα των εντολών διακλάδωσης έχει συνήθως το ίδιο αποτέλεσμα όταν εκτελεστεί. Ένας απλός μηχανισμός πρόβλεψης που εκμεταλλεύεται το γεγονός αυτό είναι ο Bimodal branch predictor τον οποίο βλέπουμε στο σχήμα 2.6.



Σχήμα 2.6. Ο bimodal predictor [5].

Ο μηχανισμός αυτός δουλεύει ως εξής:

- Κάθε θέση στον πίνακα Counts έχει δύο bits.
- Η κάθε εντολή αντιστοιχείται σε μια θέση στον πίνακα με βάση την διεύθυνσή της (Program Counter).
- Όταν μια εντολή διακλάδωσης εκτελεστεί και το αποτέλεσμα της είναι γνωστό, ενημερώνεται η κατάλληλη θέση στον πίνακα counts. Αν η εντολή διακλάδωσης είχε θετικό αποτέλεσμα (taken) τότε η τιμή στη θέση αυτή αυξάνεται κατά 1. Αντίθετα αν το αποτέλεσμα ήταν λανθασμένο (not taken) τότε η τιμή μειώνεται κατά 1. Οι τιμές στον πίνακα Counts είναι saturated, δηλαδή αν η τιμή στον πίνακα είναι 11 και εκτελεστεί μια εντολή της οποίας το αποτέλεσμα είναι taken, τότε η τιμή του πίνακα θα παραμείνει 11. Το ίδιο συμβαίνει για not taken εντολές.
- Όταν μια εντολή πρόβλεψης εισέλθει στο pipeline, αυτή προβλέπεται με βάση τον πίνακα Counts. Το αριστερό (most significant) bit δίνει την πρόβλεψη, δηλ για τις τιμές 01 και 00 η συγκεκριμένη εντολή διακλάδωσης προβλέπεται ως λανθασμένη (not taken-N) και για τις τιμές 10 και 11 προβλέπεται ως ορθή (taken).

Ο λόγος που χρησιμοποιείται saturated counter είναι για αποφυγή αχρείαστων λανθασμένων προβλέψεων όπως στο πιο κάτω παράδειγμα:

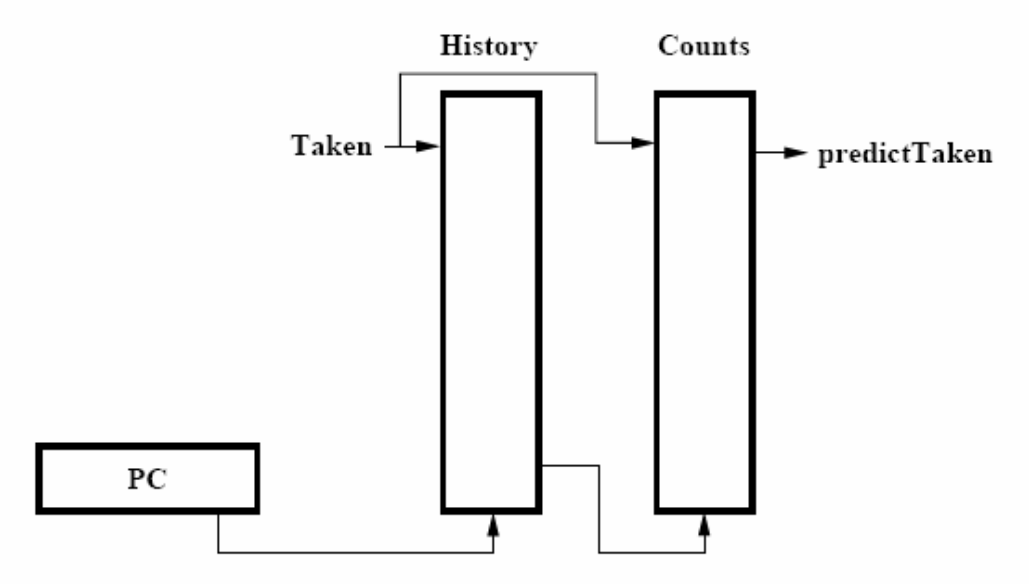
```
i = 0;
while (i<4) { //code; i++; }
```

Αν υποθέσουμε πως το συγκεκριμένο κομμάτι κώδικα εκτελείτε τρεις φορές, το ιστορικό της εντολής διακλάδωσης του πιο πάνω while θα έχει την εξής μορφή (1 = taken, 0 = not taken):

000010000100001

Υποθέτουμε πως ο πίνακας counts είναι αρχικοποιημένος σε 00. Χρησιμοποιώντας saturated counter θα έχουμε συνολικά τρεις αποτυχημένες προβλέψεις, μια για κάθε φορά που ο έλεγχος i<4 είναι θετικός. Χρησιμοποιώντας 1-bit counter θα είχαμε συνολικά 5 αποτυχημένες προβλέψεις, μία για 1<4 true και μία για τον επόμενο έλεγχο.

Ο bimodal predictor έχει καλή ακρίβεια για εντολές διακλάδωσης που έχουν ιδιαίτερα ψηλό λόγο taken/not taken ή not taken/taken. Ο local branch predictor είναι ένας μηχανισμός που κάνει χρήση του ιστορικού μιας εντολής διακλάδωσης για να επιτύχει ακόμα πιο ακριβής προβλέψεις. Ο μηχανισμός αυτός φαίνεται στο σχήμα 2.7.



Σχήμα 2.7. Local branch predictor [5].

Ο μηχανισμός αυτός λειτουργεί ως εξής:

- Κάθε θέση στον πίνακα counts έχει δύο bits και όπως και στον bimodal predictor είναι saturated.
- Η κάθε εντολή αντιστοιχείται σε μια θέση στον πίνακα History με βάση την διεύθυνσή της (Program Counter).
- Κάθε θέση του πίνακα History καταγράφει τα αποτελέσματα των τελευταίων n εκτελέσεων των εντολών που αντιστοιχούν στη θέση αυτή.
- Η τιμή σε μια θέση του πίνακα History δίνει τη θέση στον πίνακα Counts στην οποία υπάρχει η πρόβλεψη.
- Για να προβλέψουμε μια εντολή διακλάδωσης χρησιμοποιούμε την διεύθυνσή της για να κάνουμε πρόσβαση στον πίνακα History. Η τιμή του πίνακα History στη θέση αυτή μας δίνει τη θέση στον πίνακα Counts από όπου παίρνουμε την πρόβλεψη (χρησιμοποιούμε το most significant bit όπως και στον bimodal predictor).

Δεδομένου ότι οι πίνακες History και Counts έχουν ικανοποιητικό μέγεθος, ο local branch predictor έχει σημαντικά καλύτερη επίδοση από τον bimodal. Αυτό συμβαίνει γιατί μετά από μια περίοδο αρχικοποίησης μπορεί να αναγνωρίζει με ακρίβεια εντολές διακλάδωσης που ακολουθούν συχνά τα ίδια μοτίβα. Αυτό φαίνεται στο πιο κάτω παράδειγμα:

```
i = 0;
while (i<4) { //code; i++; }
```

Για τον κώδικα αυτό ο bimodal predictor θα έχει ακρίβεια 1/5 γιατί όπως είδαμε και πιο πάνω θα προβλέπει λανθασμένα, κάθε φορά που ο έλεγχος του while θα είναι θετικός. Αντίθετα, ο local branch predictor μετά από μια περίοδο αρχικοποίησης θα έχει, για κάθε συνδυασμό του ιστορικού της εντολής διακλάδωσης, μια καταχώρηση στον πίνακα Counts με τη σωστή πρόβλεψη όπως φαίνεται στον πιο κάτω πίνακα.

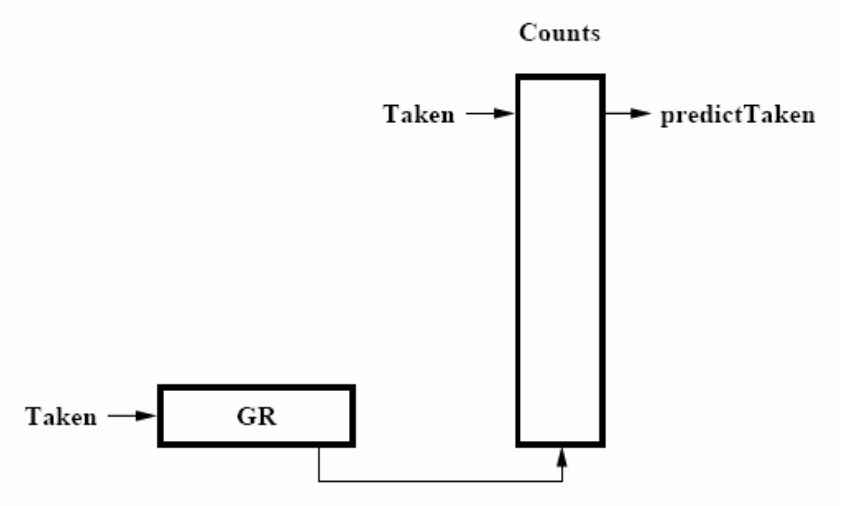
History (4 bits)	Counts(MSB)
0000	1
0001	0
0010	0
0100	0
1000	0

Πίνακας 2.1. Τιμές του πίνακα Counts για τις αντίστοιχες τιμές του πίνακα History.

Με αυτό τον τρόπο επιτυγχάνεται πολύ ψηλότερο ποσοστό ακρίβειας.

Οι μηχανισμοί πρόβλεψης που είδαμε μέχρι τώρα χρησιμοποιούν τοπικό ιστορικό, δηλαδή βασίζουν τις προβλέψεις τους σε προηγούμενα αποτελέσματα της ίδιας εντολής που πρόκειται να προβλέψουν. Όπως είδαμε και στην εισαγωγή του κεφαλαίου, η πληροφορία από το αποτέλεσμα διαφορετικών εντολών διακλάδωσης μπορεί να είναι χρήσιμη στην πρόβλεψη.

Ο Global history predictor (σχήμα 2.8) είναι ένας μηχανισμός πρόβλεψης που κάνει χρήση της πληροφορίας ενός αριθμού από προηγούμενες εντολές διακλάδωσης.



Σχήμα 2.8.Global history predictor [5].

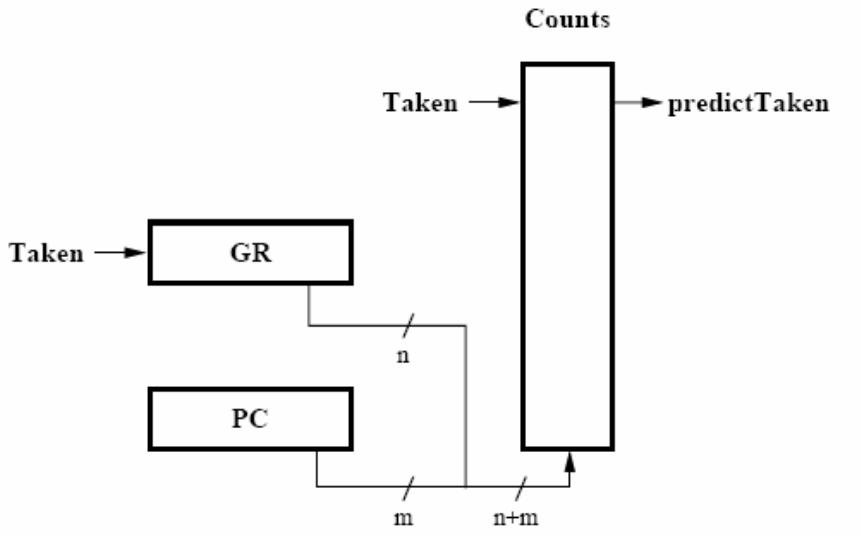
Ο μηχανισμός αυτός λειτουργεί ως εξής:

Ο global register (GR) είναι ένας shift καταχωρητής μεγέθους n στον οποίο καταγράφονται τα αποτελέσματα των n προηγούμενων εντολών διακλάδωσης που εκτελέστηκαν.
Ο global register χρησιμοποιείται ως δείκτης στον πίνακα Counts ο οποίος είναι ο ίδιος με τον πίνακα που χρησιμοποιεί ο bimodal predictor (2-bit saturated counter). Όπως και στον bimodal predictor το most significant bit του πίνακα Counts δίνει την πρόβλεψη.
Όταν μια εντολή εκτελεστεί, το αποτέλεσμα της καταγράφεται στον global register και η θέση στον πίνακα Counts που αντιστοιχεί στο περιεχόμενο του global register, ενημερώνεται αναλόγως.

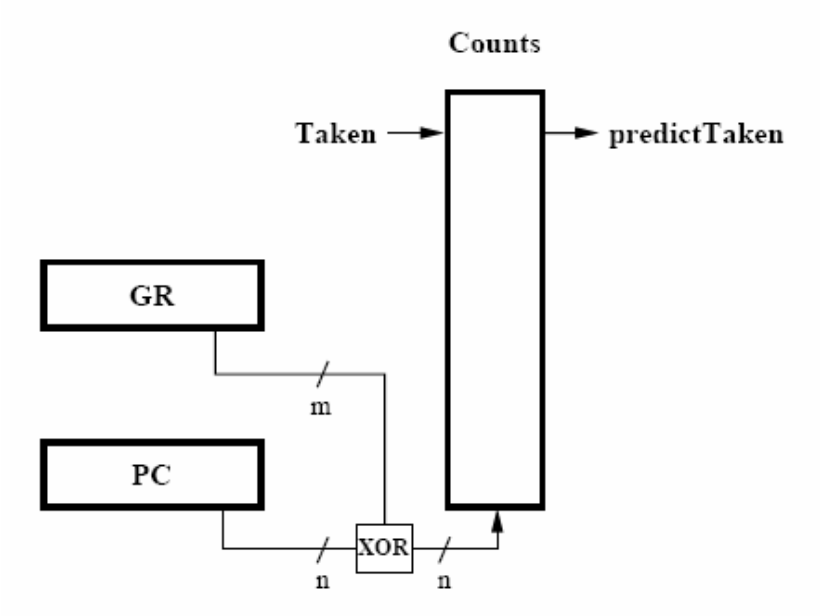
Ο global history predictor μπορεί να επιτυγχάνει ψηλά ποσοστά ορθών προβλέψεων γιατί μπορεί να αναγνωρίζει συσχετίσεις μεταξύ διαφορετικών εντολών διακλάδωσης. Για παράδειγμα οι εντολές: if(x>0){//code} if(x>10){… έχουν ιδιαίτερα ψηλή συσχέτιση. Έτσι με το να γνωρίζουμε το αποτέλεσμα της πρώτης εντολής διακλάδωσης, μπορούμε να προβλέψουμε με ακρίβεια το αποτέλεσμα της δεύτερης.

Ένας άλλος λόγος που ο global history predictor επιτυγχάνει μεγάλη ακρίβεια είναι ότι, δεδομένου ικανοποιητικά μεγάλου μεγέθους history register, μπορεί να μιμηθεί τη λειτουργία του local predictor.

Δύο παραλλαγές του global history predictor είναι ο gselect (σχήμα 2.9) και ο gshare (σχήμα 2.10). Οι δύο αυτοί μηχανισμοί πρόβλεψης χρησιμοποιούν ένα συνδυασμό της ολικής και της τοπικής ιστορίας για να επιτύχουν μεγαλύτερη ακρίβεια. Ο συνδυασμός αυτός πετυχαίνει καλύτερα αποτελέσματα γιατί η χρήση της τοπικής ιστορίας βοηθά να διαχωρίζονται καλύτερα οι διάφορες εντολές διακλάδωσης. Όταν υπάρχουν αρκετά bits τοπικής ιστορίας, ώστε να αναγνωρίζονται οι περισσότερες εντολές διακλάδωσης, τότε προσθέτονται bits από την ολική ιστορία τα οποία μπορεί να περιέχουν πληροφορία που να βοηθά στην πρόβλεψη.



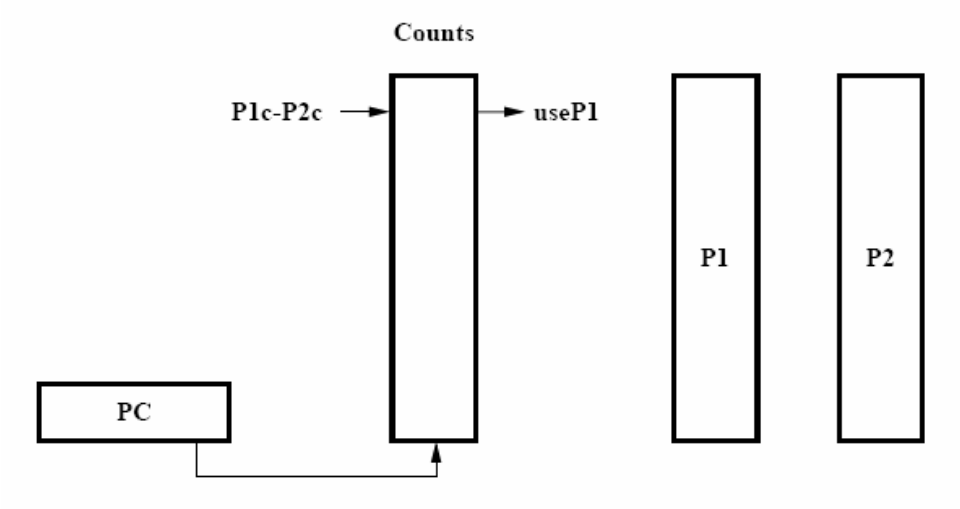
Σχήμα 2.9 Gselect predictor [5].



Σχήμα 2.10. Gshare predictor [5].

Οι μηχανισμοί που είδαμε μέχρι τώρα έχουν ο καθένας κάποιο σημείο που υπερτερούν έναντι των άλλων. Ένα φυσικό ερώτημα που προκύπτει είναι κατά πόσο είναι δυνατό να συνδυαστούν οι μηχανισμοί αυτοί, με σκοπό την αύξηση στην ακρίβεια πρόβλεψης.

Ένας τέτοιος μηχανισμός φαίνεται στο σχήμα 2.11. Χρησιμοποιείται ένας 2-bit saturated πίνακας, ο ίδιος με αυτόν που χρησιμοποιεί ο bimodal predictor, για να επιλέγει μεταξύ δύο διαφορετικών μηχανισμών πρόβλεψης. Ο πίνακας αυτός ενημερώνεται με το αποτέλεσμα της πρόβλεψης των δύο διαφορετικών predictors, έτσι ώστε να επιλέγεται ο καταλληλότερος predictor για κάθε εντολή διακλάδωσης. Το most significant bit του πίνακα καθορίζει τον μηχανισμό πρόβλεψης που πρέπει να χρησιμοποιηθεί. Ο τρόπος με τον οποίο ενημερώνεται ο πίνακας αυτός φαίνεται στον πίνακα 2.2.



Σχήμα 2.11. Combined predictor [5].

Predictor 1 correct	Predictor 2 Correct	P1correct-P2correct	Counts
0	0	0	-
0	1	-1	-1
1	0	1	+1
1	1	0	-

Πίνακας 2.2. Ο τρόπος με τον οποίο ενημερώνεται ο πίνακας Counts με βάση τα αποτελέσματα των δύο Predictors.

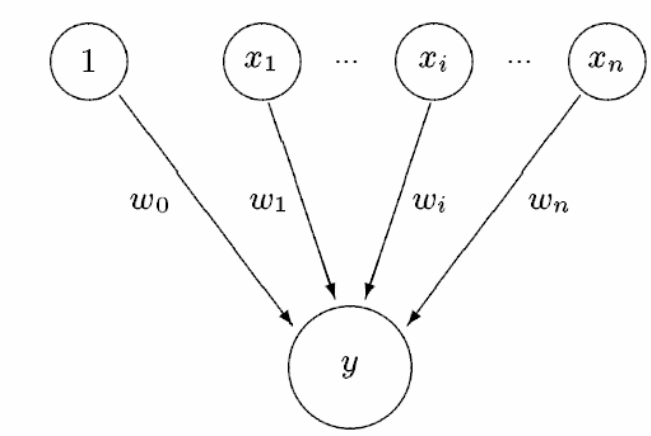
Ένας χρήσιμος συνδυασμός για τους δύο μηχανισμούς πρόβλεψης είναι ο bimodal/gshare. Ο συνδυασμός αυτός έχει τη δυνατότητα να χρησιμοποιεί τη γενική ιστορία όταν αυτή είναι χρήσιμη, διαφορετικά χρησιμοποιείται ο bimodal predictor.

2.5 Πρόβλεψη με Χρήση Perceptrons

Το υποκεφάλαιο αυτό βασίζεται στο [4].

Ο perceptron predictor είναι ένας ενδιαφέρον μηχανισμός πρόβλεψης ο οποίος χρησιμοποιεί πολύ απλά νευρωνικά δίκτυα –τους perceptrons- για να πετύχει ψηλά ποσοστά ακρίβειας πρόβλεψης.

Ο perceptron είναι ένας απλός μηχανισμός ο οποίος δέχεται σαν είσοδο ένα διάνυσμα και με βάση αυτό, μαθαίνει το αποτέλεσμα μιας συνάρτησης με είσοδο αυτό το διάνυσμα. Στη περίπτωση του perceptron predictor η είσοδος είναι ο καταχωρητής ολικής ιστορίας (global history register) και η έξοδος του perceptron είναι μια πρόβλεψη για την κατεύθυνση της εντολής διακλάδωσης που βρίσκεται στον επεξεργαστή. Ο perceptron predictor φαίνεται στο σχήμα 2.12.



Σχήμα 2.12. Percepron predictor [4].

Η πρόβλεψη λαμβάνεται ως εξής. Τα δεδομένα εισόδου x1,x2,…,xn (δηλ. τα περιεχόμενα του global history register) πολλαπλασιάζονται με τα αντίστοιχα βάρη w1,w2,…,wn. Τα μηδενικά στο global history register εκλαμβάνονται ως -1. Στη συνέχεια τα γινόμενα x1.w1, x2.w2, …, xn.wn προσθέτονται μεταξύ τους. Αν το αποτέλεσμα είναι θετικό τότε η εντολή προβλέπεται ως ορθή (taken), διαφορετικά προβλέπεται ως λανθασμένη (not taken).

Όταν η εντολή διακλάδωσης εκτελεστεί, γίνεται η ενημέρωση του predictor. Αυτό γίνεται με το να ενημερωθούν τα κατάλληλα βάρη w στον perceptron με τη χρήση του πιο κάτω αλγορίθμου:

```
if (sign(y_out) != t or |y_out|<=θ ){
        for (i=0 to n)
                wi = wi + t * xi ;
}
```

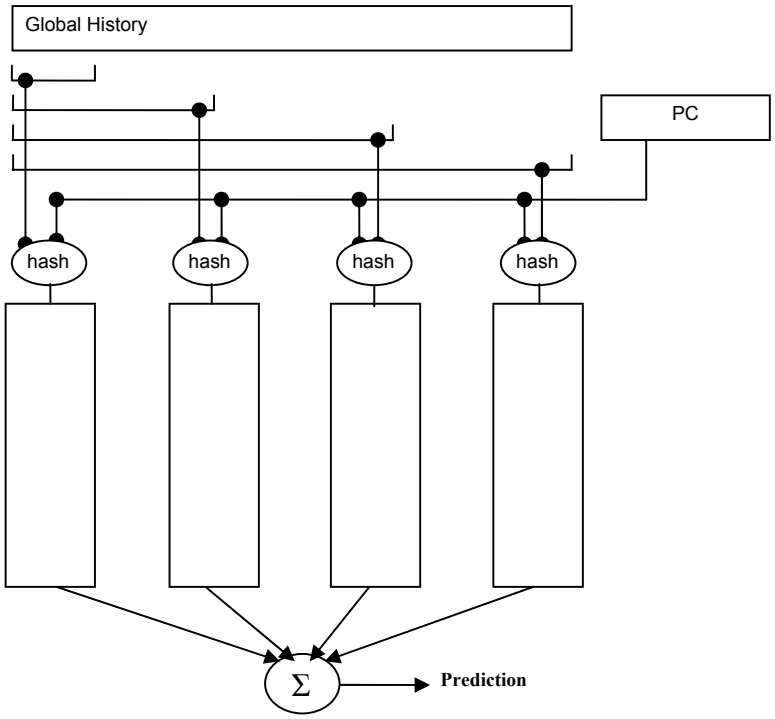
Όπου το θ είναι ένα όριο για το πότε πρέπει να σταματήσει να ενημερώνεται ο perceptron και t είναι το αποτέλεσμα της εντολής διακλάδωσης με τιμές 1 για taken και -1 για not taken.

Με τον αλγόριθμο αυτό, τα bits του global history που έχουν κάποια συσχέτιση με το αποτέλεσμα της εντολής διακλάδωσης, θα έχουν μεγάλη τιμή (θετική ή αρνητική). Αντίθετα τα bits που δεν συσχετίζονται με το αποτέλεσμα της εντολής διακλάδωσης θα έχουν βάρη κοντά στο μηδέν. Έτσι στο αποτέλεσμα του predictor συμβάλλουν μόνο τα bits του global history που έχουν συσχέτιση με την εντολή που πρόκειται να προβλεφθεί.

2.6 Προχωρημένοι μηχανισμοί πρόβλεψης: O-GEHL

Το υποκεφάλαιο αυτό βασίζεται στο [7].

Ο O-Geometric History Length predictor είναι ένας μηχανισμός πρόβλεψης που χρησιμοποιεί μια σειρά από πίνακες για να παράγει μια πρόβλεψη. Ο δείκτης για πρόσβαση στον κάθε πίνακα υπολογίζεται ως η σύμπτυξη (hash) της διεύθυνσης της εντολής που πρόκειται να προβλεφθεί και ενός μέρους της ολικής ιστορίας (global history). Για τον κάθε πίνακα χρησιμοποιείται γεωμετρικά μεγαλύτερο μέγεθος ολικής ιστορίας. Ο O-GEHL predictor φαίνεται στο σχήμα 2.13.



Σχήμα 2.13. O-GHEL Predictor.

Ο O-GEHL predictor ενημερώνεται με παρόμοιο τρόπο όπως και ο perceptron predictor, δηλ. ενημερώνεται μόνο σε περίπτωση λανθασμένης πρόβλεψης ή όταν το αποτέλεσμα της πρόβλεψης (Σ) είναι μικρότερο από κάποια τιμή. Αυτός ο τρόπος ενημέρωσης εξυπηρετεί δύο σκοπούς:

- Δεν έχουμε αχρείαστες ενημερώσεις των πινάκων. Όταν μια εντολή διακλάδωσης προβλέπεται ορθά, οι θέσεις στους πίνακες στις οποίες αντιστοιχεί παραμένουν αναλλοίωτες.
- Όταν η τιμή που παράγει ο predictor είναι χαμηλότερη από μια δεδομένη τιμή (threshold) ενημερώνουμε τις κατάλληλες θέσεις στους πίνακες έστω και αν η εντολή διακλάδωσης έχει προβλεφθεί ορθά. Αυτό βοηθά στο να αναγνωρίζονται καλύτερα τα bits στο ιστορικό που έχουν μεγάλη συσχέτιση με την εντολή (παρομοίως με τον perceptron predictor).

Ένα ενδιαφέρον χαρακτηριστικό του GEHL predictor είναι πως μπορεί να αναγνωρίζει κατά πόσο πρέπει να χρησιμοποιήσει μεγάλο ή μικρό μέγεθος global history. Ο τρόπος που το επιτυγχάνει αυτό είναι ο εξής: Ένας αριθμός από τους συνολικούς πίνακες του predictor έχει τη δυνατότητα να χρησιμοποιεί μεταβαλλόμενο αριθμό bits ιστορίας (π.χ. αν ο συνολικός αριθμός πινάκων είναι 8, τότε τρεις από τους πίνακες έχουν αυτή τη δυνατότητα). Ένας από τους πίνακες που δεν χρησιμοποιεί μεταβαλλόμενη ιστορία, χρησιμοποιείται για να αναγνωρίζεται πότε πρέπει να αλλάξουν οι πίνακες από μεγάλη ιστορία σε μικρή και αντίθετα. Ο πίνακας αυτός έχει σε κάποιες από τις θέσεις του ένα Tag bit το οποίο χρησιμοποιείται για να αναγνωρίζεται ταυτοποίηση (aliasing) στον πίνακα. Αυτό γίνεται με τον εξής μηχανισμό:

```
if ((p!out)) & (|S|<=θ){
if ((PC & 1)==Tag[indexT[7]]) AC++; else AC-=4;
if (AC == SaturatedPositive) Use Long Histories
if (AC == SaturatedNegative) Use Short histories
Tag[indexT7]]=(PC &1);
}
```

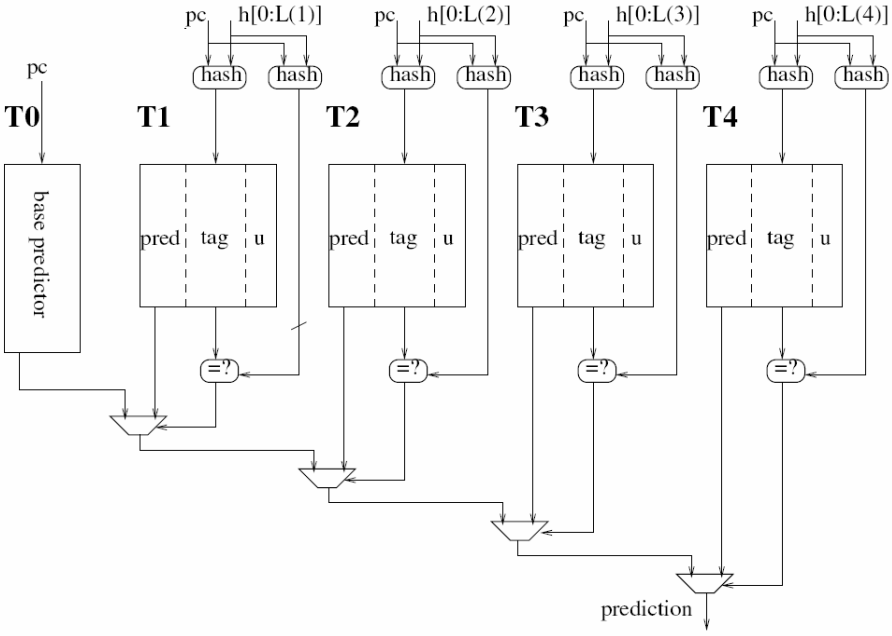
Ουσιαστικά ο αλγόριθμος αυτός μειώνει το AC όταν έχουμε συχνά προσβάσεις στην ίδια θέση του πίνακα από διαφορετικές εντολές. Αυτό υποδηλώνει ταυτοποίηση (aliasing) οπότε δίνεται το σήμα να χρησιμοποιηθούν μικρότερα μεγέθη ιστορίας. Διαφορετικά χρησιμοποιείται μεγάλη ιστορία.

Ο O-GEHL predictor επιτυγχάνει ψηλά ποσοστά ακρίβειας γιατί έχει τη δυνατότητα, για δύσκολα προγράμματα, να χρησιμοποιεί πολλούς πίνακες στους οποίους γίνεται πρόσβαση με μικρό μέγεθος ιστορίας. Με αυτό τον τρόπο αποφεύγονται φαινόμενα ταυτοποίησης. Για πιο εύκολα στην πρόβλεψη προγράμματα ο O-GEHL predictor έχει τη δυνατότητα να χρησιμοποιήσει μεγάλα μεγέθη ιστορίας, της τάξης των 200bits, για να αυξήσει το ποσοστό επιτυχημένων προβλέψεων.

2.7 Προχωρημένοι μηχανισμοί πρόβλεψης: L-TAGE

Το υποκεφάλαιο αυτό βασίζεται στο [8].

Ο L-TAGE είναι ένας μηχανισμός πρόβλεψης ο οποίος, όπως και ο O-GEHL, χρησιμοποιεί μια σειρά από πίνακες για την πρόβλεψη. Οι πίνακες αυτοί ενημερώνονται και χρησιμοποιούνται με χρήση ενός συνδυασμού της διεύθυνσης της εντολής που πρόκειται να προβλεφθεί και της γενικής ιστορίας (global history). Όπως και ο O-GEHL, ο L-TAGE χρησιμοποιεί γεωμετρικά μεγέθη γενικής ιστορίας. Επίσης χρησιμοποιείται ένας απλός bimodal predictor σαν βάση. Η οργάνωση του L-TAGE φαίνεται στο σχήμα 2.14.



Σχήμα 2.14. L-TAGE Predictor [8].

Η πρόβλεψη για μια εντολή διακλάδωσης επιλέγεται από τον πίνακα με το μεγαλύτερο ιστορικό στον οποίο έχουμε ταύτιση του tag. Σε περίπτωση που κανένας πίνακας δεν περιέχει καταχώρηση που να αντιστοιχεί στην εντολή που πρόκειται να προβλεφθεί, τότε χρησιμοποιείται ο bimodal πίνακας. Αν η πρόβλεψη στον πίνακα με το μεγαλύτερο history είναι ασθενής, δηλαδή η τιμή του pred είναι μικρή, τότε χρησιμοποιείται ο αμέσως μικρότερος πίνακας(alpred) στον οποίο υπάρχει ταύτιση (match).

Ο useful counter (u) ενημερώνεται όταν υπάρχει διαφορά μεταξύ της πρόβλεψης και της εναλλακτικής πρόβλεψης, δηλ. της πρόβλεψης που προέρχεται από το αμέσως μικρότερο history length/tag match. Σε περίπτωση λανθασμένης πρόβλεψης ο u χρησιμοποιείται για να επιλεχθεί ποια θέση στους πίνακες πρέπει να ενημερωθεί. Επιλέγεται η θέση στον πίνακα με το μικρότερο u. Έτσι γίνεται μόνο μια ενημέρωση σε κάθε λανθασμένη πρόβλεψη.

Παρατηρήσεις:

- Η χρήση του alpred επιτρέπει στον L-TAGE να χρησιμοποιεί μικρά μεγέθη global history για τις εντολές που δεν επωφελούνται από μεγαλύτερα ιστορικά.
- Μια εντολή διακλάδωσης θα χρησιμοποιεί για την πρόβλεψη της ολοένα και μεγαλύτερο ιστορικό, μέχρι να φτάσει στο σημείο όπου η αύξηση του ιστορικού δεν επιφέρει αποτελέσματα.
- Ο μηχανισμός pred/alpred επιτρέπει να χρησιμοποιούνται μεγάλα ιστορικά για τις εντολές που τα χρειάζονται και πιο μικρά ιστορικά για τις άλλες εντολές μέσα στο ίδιο πρόγραμμα.
- Η χρήση του useful counter εμποδίζει τον predictor να ενημερώνει θέσεις στους πίνακες που είναι χρήσιμες για την πρόβλεψη.

2.7 Τάσεις στις τεχνικές πρόβλεψης

Από την μελέτη της βιβλιογραφίας που έγινε παρατηρήσαμε την πορεία εξέλιξης των μηχανισμών πρόβλεψης. Σ’ αυτό βοήθησε το γεγονός ότι η μελετήσαμε τα διάφορα ερευνητικά άρθρα ξεκινώντας από το παλαιότερο ιστορικά και προχωρώντας προς τα πιο πρόσφατα. Στη μελέτη αυτή είδαμε πως η αρχικοί μηχανισμοί πρόβλεψης ήταν πολύ απλοί και περιορίζονταν σε πολύ μεγάλο βαθμό από τις απαιτήσεις σε υλικό (hardware budget).

Καθώς οι επεξεργαστές εξελίσσονται, το βάθος της διασωλήνωσης και το issue width αυξάνονται. Ως αποτέλεσμα ο αριθμός των δυναμικών εντολών που μπορούν να βρίσκονται ταυτόχρονα μέσα στον επεξεργαστή αυξάνεται, με μεγάλο μέρος του αριθμού αυτού να είναι speculative εντολές. Οι εντολές αυτές, σε περίπτωση λανθασμένης πρόβλεψης χρειάζεται να αναιρεθούν. Έτσι στους επεξεργαστές αυτούς η ακρίβεια της πρόβλεψης είναι πιο σημαντική και γ' αυτό είναι λογικό να χρησιμοποιείται μεγάλο μέρος από το hardware budget για τον μηχανισμό πρόβλεψης.

Οι νεότεροι μηχανισμοί πρόβλεψης χρησιμοποιούν τα μεγάλα hardware budgets για να πετυχαίνουν μεγαλύτερη ακρίβεια. Αυτό επιτυγχάνεται με τη χρήση συνδυασμού διαφορετικών μηχανισμών πρόβλεψης οι οποίοι αλληλοσυμπληρώνουν ο ένας τις ελλείψεις του άλλου.

Οι τελευταίοι μηχανισμοί πρόβλεψης που έχουν προταθεί χρησιμοποιούν πολλαπλούς πίνακες με στόχο να εκμεταλλευτούν πληροφορίες από διάφορα μεγέθη ιστορίας. Συγκεκριμένα η τάση που ακολουθείται είναι να χρησιμοποιείται μικρό μέγεθος ιστορίας για τα δύσκολα στην πρόβλεψη προγράμματα και μεγαλύτερη ιστορία για τα υπόλοιπα.

Κεφάλαιο 3

Ανάλυση Μεταβλητού Μεγέθους Ιστορίας

3.1 Σύνοψη κεφαλαίου

Στο κεφάλαιο αυτό παρουσιάζουμε την πειραματική ανάλυση που κάναμε, στόχος της οποίας είναι να αναγνωριστούν τα επίπεδα της ιστορίας που είναι σημαντικά στην πρόβλεψη και να ελεγχθεί αν ο predictor που χρησιμοποιούμε αξιοποιεί τα επίπεδα αυτά. Αρχικά αναφέρονται τα εργαλεία που χρησιμοποιήθηκαν και αποτελούν μέρος προηγούμενης εργασίας. Στη συνέχεια παρουσιάζουμε τη διάταξη του πειράματος. Ακολουθεί ο τρόπος που εκτελέσαμε την ανάλυση και η διαδικασία εξαγωγής στατιστικών

3.2 Στόχος και μεθοδολογία

Με στόχο την καλύτερη κατανόηση της λειτουργίας των σημερινών μηχανισμών πρόβλεψης αλλά και για να αναγνωρίσουμε τυχόν αδυναμίες που μπορεί να παρουσιάζουν, πραγματοποιήσαμε μια ανάλυση σε δεδομένα από προσομοιώσεις του L-TAGE predictor, την λειτουργία του οποίου περιγράψαμε στο κεφάλαιο 2.6. Οι παράμετροι του L-TAGE είναι οι ίδιοι όπως περιγράφονται στο [8]. Το μέγεθος του predictor είναι 256 Kbits και το

μέγεθος της ιστορίας που χρησιμοποιεί είναι 400 bits. Ο predictor αυτός κέρδισε τον 2^{ο} διαγωνισμό μηχανισμών πρόβλεψης (CBP-2) στην κατηγορία ρεαλιστικών μηχανισμών πρόβλεψης οπότε μπορεί να θεωρηθεί ότι αντιπροσωπεύει τις τελευταίες εξελίξεις στον τομέα της πρόβλεψης.

Από τις προσομοιώσεις του L-TAGE είχαμε τη δυνατότητα να εξάγουμε λεπτομερείς πληροφορίες για κάθε εντολή διακλάδωσης που εκτελέστηκε σε ένα πρόγραμμα. Πιο συγκεκριμένα εμάς μας ενδιαφέρει η κατεύθυνση της εντολής (taken ή not taken) και η πρόβλεψη που έκανε ο predictor για την εντολή αυτή.

Η έμφαση της ανάλυσής μας ήταν στο να αναγνωριστούν τα διάφορα μεγέθη ιστορίας που είναι χρήσιμα για την πρόβλεψη. Συγκεκριμένα θέλαμε να δούμε αν υπάρχουν μεγέθη της ολικής ιστορίας (global history) στα οποία υπάρχει χρήσιμη πληροφορία αλλά για κάποιο λόγο δεν την εκμεταλλεύεται ο predictor.

3.3 Προηγούμενη εργασία

Εκτός από τα ερευνητικά άρθρα που μελετήθηκαν και τα οποία παρουσιάσαμε στο προηγούμενο κεφάλαιο, για τους σκοπούς της εργασίας χρησιμοποιήσαμε κάποια εργαλεία:

- Οι προσομοιώσεις του L-TAGE έγιναν από τον επιβλέπον καθηγητή της εργασίας κ. Σαζείδη.
- Χρησιμοποιήθηκε parser ο οποίος διαβάζει τα αποτελέσματα των προσομοιώσεων και παράγει στατιστικά. Ο parser αυτός γράφτηκε από τον κ. Σαζείδη (στο υπόλοιπο της εργασίας θα αναφερόμαστε στον parser αυτό σαν process-ladas).
- Χρησιμοποιήθηκε το εργαλείο αναπαράστασης γράφων Walrus [10] για σκοπούς debugging (Παράρτημα Α).

3.4 Διαδικασία ανάλυσης προσομοιώσεων

Η ανάλυση που κάναμε ξεκινά με την ανάλυση των αποτελεσμάτων των προσομοιώσεων. Αναλύσαμε τα αποτελέσματα προσομοιώσεων 4 προγραμμάτων: vpr00, bzip200, mcf00 και crafty00. Τα προγράμματα αυτά προέρχονται από τον integer component του SPEC CPU2000 benchmark. Μια περιγραφή του κάθε benchmark φαίνεται στον πίνακα 3.1.

vpr00	FPGA Circuit Placement and Routing
bzip200	Compression
mcf00	Combinatorial Optimization
crafty00	Game Playing: Chess

Πίνακας 3.1. Τα benchmarks που χρησιμοποιήθηκαν.

Τα αρχεία αποτελεσμάτων των προσομοιώσεων έχουν καταχωρήσεις της εξής μορφής:

[PC ADDRESS] [DIRECTION] [PREDICTION]

Στα αρχεία αυτά περιέχονται μόνο εντολές διακλάδωσης. Η σειρά με την οποία εμφανίζονται οι εντολές στο αρχείο είναι και η σειρά με την οποία εκτελέστηκαν στην προσομοίωση. Σε ένα αρχείο, κάποιο PC μπορεί να εμφανίζεται πολλαπλές φορές, μία για κάθε δυναμικό στιγμιότυπο εκτέλεσης της ίδιας εντολής. Από το αρχείο αυτό μπορούμε να γνωρίζουμε για κάθε εντολή διακλάδωσης που εκτελέστηκε στην προσομοίωση ποια ήταν η κατεύθυνσή της και ποια ήταν η πρόβλεψη του predictor. Σημείωση: Το direction και το

prediction φυλάγονται σε διαφορετικά αρχεία με το κάθε αρχείο να περιέχει το PC. Αυτό γίνεται για σκοπούς debugging και ευρωστίας.

Από τα αρχεία αυτά μπορούμε εύκολα να εξάγουμε κάποιες σημαντικές πληροφορίες. Για κάθε εντολή αν το [DIRECTION] είναι διαφορετικό από το[PREDICTION] τότε έχουμε λανθασμένη πρόβλεψη. Μπορούμε με τη χρήση του parser process-ladas να μετρήσουμε πόσες ήταν οι συνολικά οι ορθές προβλέψεις και πόσες ήταν οι λανθασμένες. Ο τρόπος που γίνεται αυτό είναι σχετικά απλός και φαίνεται σε ψευδοκώδικα πιο κάτω:

```
for (each record in the simulation file){
        if(direction == prediction) correct_predictions++;
        else incorrect_predictions++;
}
```

Αυτό μπορεί να μας δώσει την ακρίβεια του predictor σαν απόλυτο αριθμό. Accuracy=correct_predictions / (correct_predictions+incorrect_predictions). Με μια απλή τροποποίηση του πιο πάνω αλγορίθμου μπορούμε να εξάγουμε αποτελέσματα για κάθε εντολή διακλάδωσης που εκτελέστηκε.

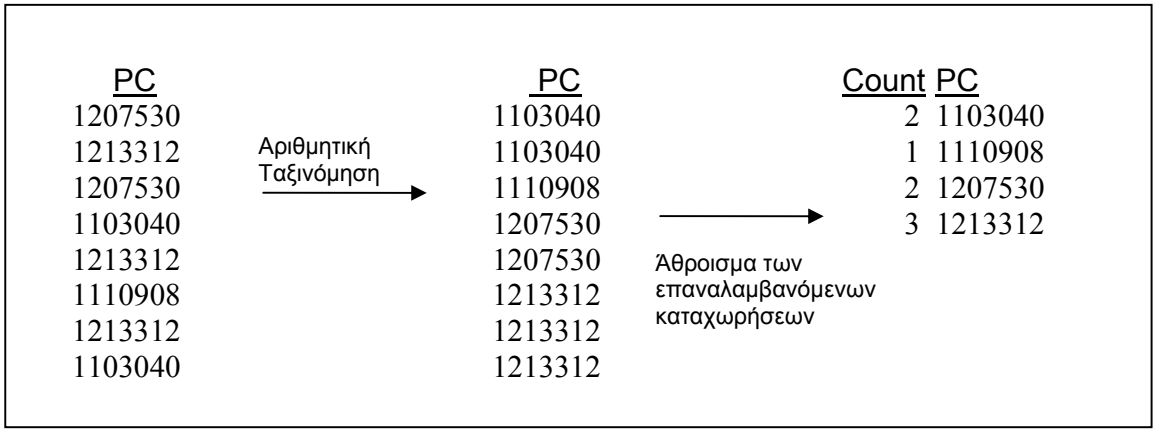
```
for (each record in the simulation file){
        if(direction == prediction) print(file1, “%d”, current_record.PC);
        else print(file2, “%d”, current_record.PC);
}
```

Το αποτέλεσμα του πιο πάνω αλγόριθμου είναι δύο αρχεία. Στο ένα αρχείο περιέχονται τα PCs όλων των εντολών που προβλέφθηκαν ορθά και στο άλλο αρχείο περιέχονται τα PCs των εντολών που προβλέφθηκαν λανθασμένα. Και στα δύο αρχεία για κάθε στιγμιότυπο της εντολής υπάρχει και μία καταχώρηση. Για παράδειγμα αν η εντολή με PC 12077889 εκτελέστηκε 5 φορές και είχε προβλεφθεί ορθά δύο φορές τότε τα αρχεία θα έχουν την εξής μορφή:

File 1	File 2
12077889	12077889
12077889	12077889
	12077889

Πίνακας 3.2. Παράδειγμα εκτέλεσης.

Στα αρχεία αυτά ο αριθμός των φορών που παρουσιάζεται ένα PC δίνει και τον αριθμό των ορθών και λανθασμένων προβλέψεων για το PC αυτό (το file1 δίνει τις ορθές προβλέψεις και το file2 τις λανθασμένες). Τα PCs στα αρχεία αυτά πιθανόν να μην είναι σε συνεχόμενες θέσεις. Οπότε για να εξάγουμε τα στατιστικά αυτά πρέπει να ταξινομήσουμε τις καταχωρίσεις των δύο αρχείων και να μετρήσουμε τις συνεχόμενες καταχωρίσεις με το ίδιο PC. Η λειτουργία αυτή φαίνεται στο σχήμα 3.1.



Σχήμα 3.1. Ορθές προβλέψεις ανά PC.

Στη συνέχεια για το κάθε benchmark θέλαμε να βρούμε τις εντολές διακλάδωσης με τις περισσότερες λανθασμένες προβλέψεις. Αυτό μπορεί να γίνει με επέκταση του πιο πάνω αλγορίθμου:

- Ταξινομούμε αριθμητικά τις καταχωρίσεις στο file2 (misspredictions).
- Για κάθε PC μετρούμε τις συνεχόμενες καταχωρίσεις του.

- Ταξινομούμε τα PCs ανάλογα με το ποιο έχει τις περισσότερες συνεχόμενες καταχωρίσεις.

Εκτελέσαμε τη διαδικασία αυτή για το κάθε benchmark. Τα αποτελέσματα φαίνονται στον πίνακα 3.3. Παρατηρήστε ότι τα αποτελέσματα δεν δείχνουν τις εντολές με το χαμηλότερο accuracy αλλά τις εντολές με τα περισσότερα misspredictions. Η ιδέα πίσω από αυτό είναι πως οι εντολές με λίγα misspredictions, έστω και αν έχουν χαμηλό accuracy, δεν συμβάλλουν ουσιαστικά στη ολική χαμηλή επίδοση του predictor.

Benchmark	**PC**	**Correct**	**Incorrect**	**%Correct (accuracy)**
vpr00	1207997943	449390	129133	77
	1207997053	163994	106952	60
	1207997856	252437	90902	73
bzip200	1207973322	377861	67253	84
	1207970298	2840599	67172	97
	1207973312	204185	58998	77
mcf00	1207969938	1383370	222707	86
	1207967657	1547275	174497	89
	1207969744	1272902	117002	91
crafty00	1207979539	110401	13651	88
	1207993858	149200	11698	92
	1207993850	256840	11342	95

Πίνακας 3.3. Οι τρεις εντολές με τα περισσότερα misspredictions ανά benchmark.

Για σκοπούς σύγκρισης επαναλάβαμε τη διαδικασία αλλά με κριτήριο τις εντολές που έχουν τις περισσότερες ορθές προβλέψεις. Τα αποτελέσματα φαίνονται στον πίνακα 3.4. Από σύμπτωση για το crafty00 η εντολή με PC 1207993850 έχει ταυτόχρονα τα περισσότερα misspredictions και τα περισσότερα correct predictions.

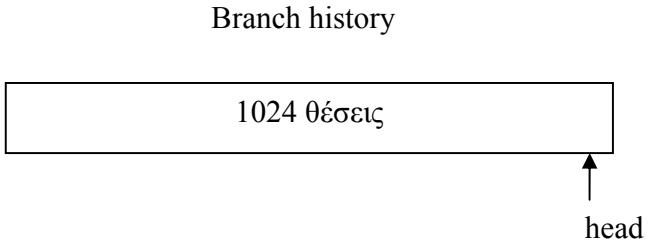
Benchmark	PC	Correct	Incorrect	%Correct (accuracy)
vpr00	1207998239	1742184	28990	98
bzip200	1207970304	2767012	53828	98
mcf00	1207967638	1680189	41583	97
crafty00	1207993850	256840	11342	95

Πίνακας 3.4. Οι εντολές με τα περισσότερα correct predictions.

3.5 Ανάλυση μεταβλητού μεγέθους ιστορίας

Έχοντας εις γνώση ποιες είναι οι εντολές με τις περισσότερες λανθασμένες προβλέψεις για κάθε benchmark, προχωρήσαμε σε λεπτομερή ανάλυση για την κάθε μια από τις εντολές αυτές. Ο σκοπός της ανάλυσης ήταν να μελετηθεί η συμπεριφορά των εντολών αυτών σε σχέση με τον αριθμό global history bits που μπορούμε να δούμε σε κάθε στιγμιότυπο της εντολής.

Για την ανάλυση αυτή χρειαστήκαμε τη δυνατότητα να βλέπουμε το ιστορικό κάθε δυναμικής εντολής που εκτελέστηκε. Για να το πετύχουμε αυτό τροποποιήσαμε τον parser process-ladas ως εξής. Δημιουργήσαμε μια δομή για να κρατά το branch history. Η δομή αυτή υλοποιήθηκε ως ένας κυκλικός πίνακας μεγέθους 1024 bits (σχήμα 3.2).



Σχήμα 3.2. Δομή για φύλαξη του history.

Η λειτουργία του τροποποιημένου parser περιγράφεται σε ψευδοκώδικα ως εξής:

```
for (each record in the simulation file){
        add_to_history(record.direction);
        if (record.PC == inputPC){
                print_history();
                println (record.direction, record.prediction); }
}
```

Κάθε φορά που ο parser διαβάζει μια καταχώρηση με το PC address που θέλουμε τυπώνει τα περιεχόμενα του πίνακα br_history ως εξής:

- Τυπώνει τα περιεχόμενα από τη θέση head μέχρι τη θέση 1023
- Τυπώνει τα περιεχόμενα από τη θέση 0 μέχρι τη θέση head-1

Στη συνέχεια γράφει το [DIRECTION] μέρος της καταχώρησης στην θέση head του πίνακα και ενημερώνει το head με τον εξής τρόπο: head = (head+1) modulo 1024. Αφού γίνει αυτό τυπώνεται το [DIRECTION] και το [PREDICTION] όπως αυτό διαβάστηκε από το αρχείο εισόδου.

Οι τροποποιήσεις αυτές επιτρέπουν στον parser παράγει για μια συγκεκριμένη εντολή διακλάδωσης ένα αρχείο με εγγραφές της μορφής:

[BRANCH HISTORY] [DIRECTION] [PREDICTION]

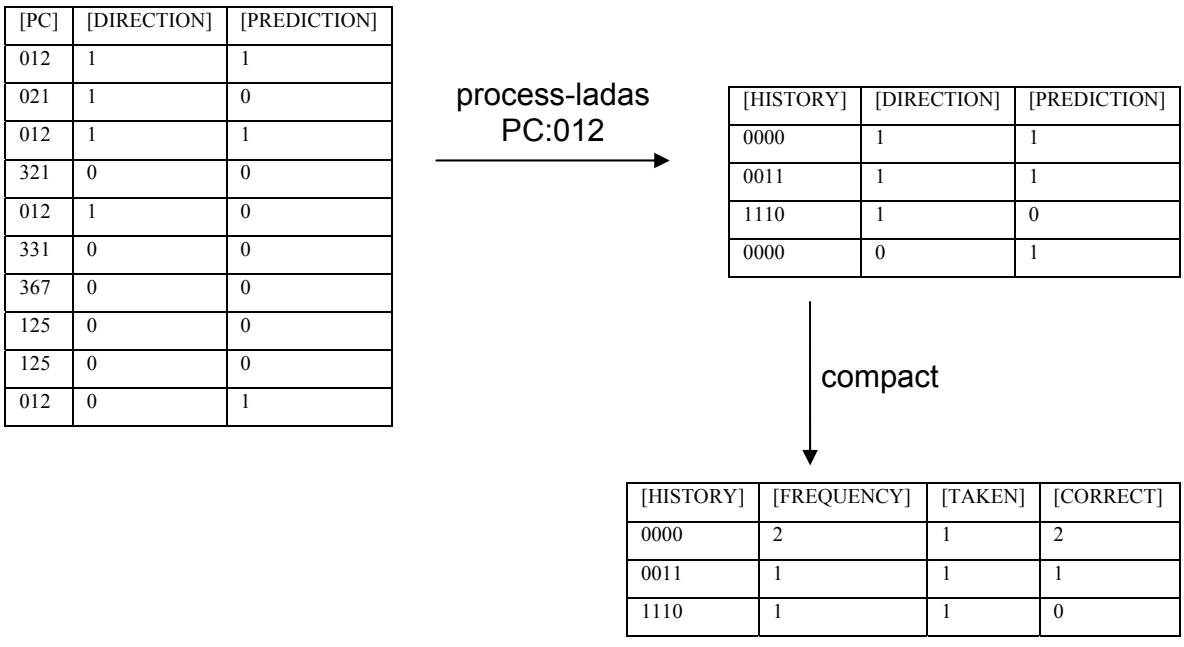
Ο parser τυπώνει τα bits της ιστορίας ανά 4 σε δεκαεξαδική μορφή, δηλ η ιστορία 101000011111 γίνεται a1f. Αυτό είναι χρήσιμο στο να μειώνει το μέγεθος του παραγόμενου αρχείου. Στο αρχείο που παράγεται μπορεί να υπάρχουν πολλές καταχωρίσεις με το ίδιο [BRANCH HISTORY] μέρος. Για τον λόγο αυτό συμπτύσσουμε το αρχείο αυτό ώστε να έχει την εξής μορφή:

[BRANCH HISTORY] [FREQUENCY] [#TAKEN] [#CORRECT]

Σε ψευδοκώδικα η λειτουργία που κάνει τη σύμπτυξη φαίνεται πιο κάτω (στο παράρτημα Β δείχνουμε πως ο ψευδοκώδικας αυτός υλοποιείται πολύ εύκολα σαν awk script).

```
for (each record in the brhistory-direction-prediction file){
        table [record.BRANCH HISTORY].frequency++;
        table [record.BRANCH HISTORY].taken += record.DIRECTION;
        table [record.BRANCH HISTORY].tcorrect += record.CORRECT;
}
for ( every entry in table[] ){
        println (entry, frequency, taken, correct);
}
```

Μια σύνοψη των λειτουργιών αυτών φαίνεται στο σχήμα 3.3:



Σχήμα 3.3 Σύνοψη της διαδικασίας παραγωγής του history file.

Το αρχείο που παράγεται με τη σύμπτυξη (cmp-file) χρησιμοποιείται για να κάνουμε την ανάλυση μεταβλητού μήκους ιστορίας. Η βασική ιδέα της ανάλυσης αυτής είναι να αναγνωρίσουμε διάφορα μήκη ιστορίας (history patterns) σε όλα τα [BRANCH HISTORY] του αρχείου cmp-file και να εξάγουμε κάποια στατιστικά. Αυτό που θέλουμε να δούμε είναι κατά πόσο υπάρχει ανεκμετάλλευτη πληροφορία στο branch history την οποία ο predictor δεν αναγνωρίζει.

Ο τρόπος που αναγνωρίζονται τα διάφορα history patterns είναι ο εξής:

- Έχουμε ένα πίνακα (patterns[]) του οποίου η κάθε θέση περιέχει 4 πεδία. Αυτά είναι τα: frequency, taken, not-taken, correct.
- Διαβάζουμε όλες τις καταχωρίσεις στο cmp-file.
- Για κάθε καταχώρηση αναγνωρίζουμε 11 διαφορετικά patterns. Το κάθε pattern ξεκινά από τα δεξιά του [BRANCH HISTORY] και έχει γεωμετρικά μεγαλύτερο μέγεθος (1,2,4,8,16,...1024). Ο λόγος που ξεκινούμε από δεξιά είναι γιατί το δεξιότερο bit του branch history είναι και το τελευταίο που εισήχθηκε.
- Το κάθε pattern που αναγνωρίζουμε αντιστοιχείται σε μία θέση στον πίνακα patterns[]. Για τη θέση στην οποία αντιστοιχείται το pattern ενημερώνουμε τα 4 πεδία:
    - frequency = frequency + record.freuency
    - taken = taken + record.taken
    - not-taken = not-taken + (record.frequency-record.taken)
    - correct = correct + record.correct
- Όταν διαβαστεί όλο το cmp-file τυπώνουμε τα περιεχόμενα του patterns[].

Σε ψευδοκώδικα ο αλγόριθμος είναι ο εξής:

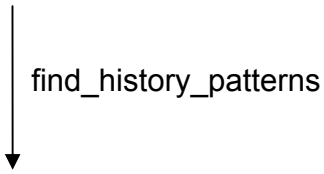
```
for (each record in cmp-file){
        for (i=1 to 1024){
                pattern = last i bits of record.branch_history;
```

```
                table[pattern].frequency += record.frequency;
                table[pattern].taken += record.taken;
                table[pattern].nottaken += record.frequency - record.taken;
                table[pattern].correct += record.correct;
        }
}

for (every pattern in table[]){
        entropy = max(taken, nottaken)/frequency;
        println ( length(pattern), pattern,  frequency, taken, nottaken, correct /frequency,
        entropy );
}
```

Ένα παράδειγμα της χρήσης του αλγορίθμου αυτού φαίνεται στο σχήμα 3.4.

history	freq	taken	correct
03fa	6	2	3
0632	9	1	7
353a	3	1	2
5602	5	2	2
6070	11	6	9
3041	1	1	0



pattern	freq	taken	not taken	predictor accuracy	entropy
0	11	6	5	81%	54%
1	1	1	0	0%	100%
2	14	3	11	64%	78%
a	9	3	6	55%	66%
70	11	6	5	81%	54%
41	1	1	0	0%	100%
02	5	2	3	40%	40%
32	9	1	8	77%	77%
3a	3	1	2	66%	66%
fa	6	2	4	50%	50%
6070	6	6	0	81%	54%
3041	9	1	8	0%	100%
5602	3	2	1	40%	40%
0632	5	1	4	77%	77%
353a	11	1	10	66%	66%
03fa	1	2	1	50%	50%

Σχήμα 3.4. Παραγωγή μοτίβων μεταβλητού μεγέθους.

Ένα ενδιαφέρον σημείο του αλγορίθμου αυτού είναι ο υπολογισμός του entropy. Το entropy ορίζεται ως εξής:

Entropy = max (taken, not_taken) / (taken + not_taken)

Η τιμή αυτή είναι μια ένδειξη για το πόσο προβλέψιμη είναι η συμπεριφορά μια εντολής διακλάδωσης. Όταν το entropy είναι ψηλό η εντολή διακλάδωσης κλίνει πως μία κατεύθυνση και γι αυτό είναι εύκολο να προβλεφθεί. Το αντίθετο όμως δεν ισχύει. Μια εντολή μπορεί να έχει την εξής συμπεριφορά (T = taken, N = not taken):

N N N N N N N T T T T T T T

Για αυτή την εντολή το entropy θα είναι 0.5 που είναι και η χαμηλότερη τιμή που μπορεί να πάρει βάση του ορισμού του. Παρόλα αυτά η ακρίβεια ενός bimodal predictor για τη συγκεκριμένη εντολή θα ήταν 0.92 (13/14 ορθές προβλέψεις). Έτσι το entropy μπορεί να χαρακτηρίσει μία εντολή διακλάδωσης ως εύκολα προβλέψιμη αλλά όχι ως μη προβλέψιμη.

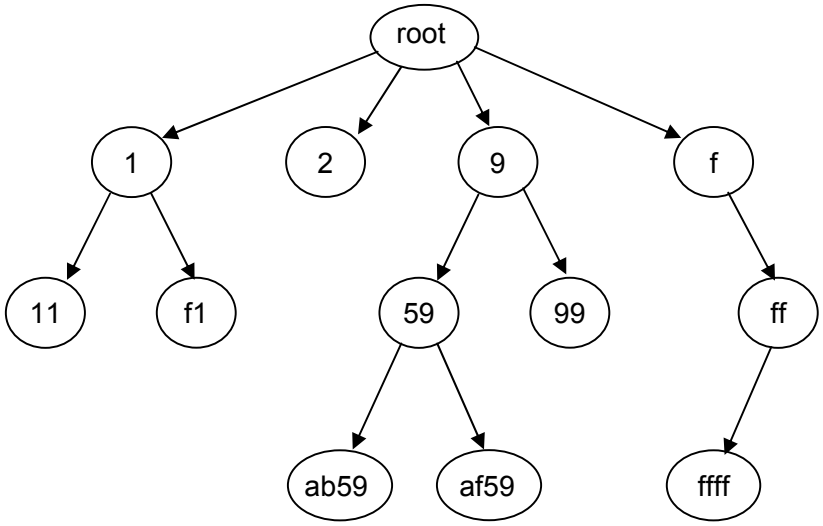
Το παράδειγμα του σχήματος 3.4, αν και υποθετικό, δείχνει κάποια από τα σημαντικά χαρακτηριστικά της ανάλυσης που κάναμε. Συγκεκριμένα παρατηρούμε ότι τα στατιστικά για τα patterns μήκους 4 έχουν τα ίδια στατιστικά με τα patterns που έχουν μήκος δύο. Αυτό υποδεικνύει ότι πιθανόν να μην χρειάζονται όλα τα patterns που αναγνωρίζουμε. Επίσης βλέπουμε πως κάποια patterns εμφανίζονται μόνο μία φορά. Για τέτοια patterns κανένας predictor δεν μπορεί να πετύχει ψηλά ποσοστά πρόβλεψης αφού δεν του δίνεται η δυνατότητα να μάθει τη συμπεριφορά αυτή. Αυτό μας οδήγησε στο συμπέρασμα ότι πρέπει να φιλτράρουμε τα αποτελέσματα που παράγονται.

Για το φιλτράρισμα χρησιμοποιήσαμε δύο κριτήρια:

- Αν ένα pattern στο αρχείο που παράγεται (pattern-file) έχει frequency μικρότερο του 10, τότε διαγράφουμε όλα τα παιδιά του.

- Αν ένα pattern στο pattern-file έχει accuracy μεγαλύτερο του 99.5%, τότε διαγράφουμε τα παιδιά του.
- Αν τα παιδία ενός pattern έχουν συνολικό entropy μικρότερο από αυτό του πατέρα τότε τα διαγράφουμε και ο πατέρας γίνεται leaf.

Τα παιδιά ενός pattern A ορίζονται ως τα patterns που έχουν στο δεξί μισό μέρος τους το A. Επειδή ακολουθούμε γεωμετρική με βάση 2, τά τα παιδιά του A θα έχουν διπλάσιο μέγεθος από ότι ο A. Έτσι ο A μπορεί να έχει το πολύ $16^{length(A)}$ παιδιά επειδή χρησιμοποιούμε δεκαεξαδική βάση για τα patterns μας. Από τον ορισμό αυτό κατανοούμε πως τα patterns που βρίσκουμε μπορούν να αναπαρασταθούν σε μορφή δέντρου. Ένα παράδειγμα φαίνεται στο σχήμα 3.5.

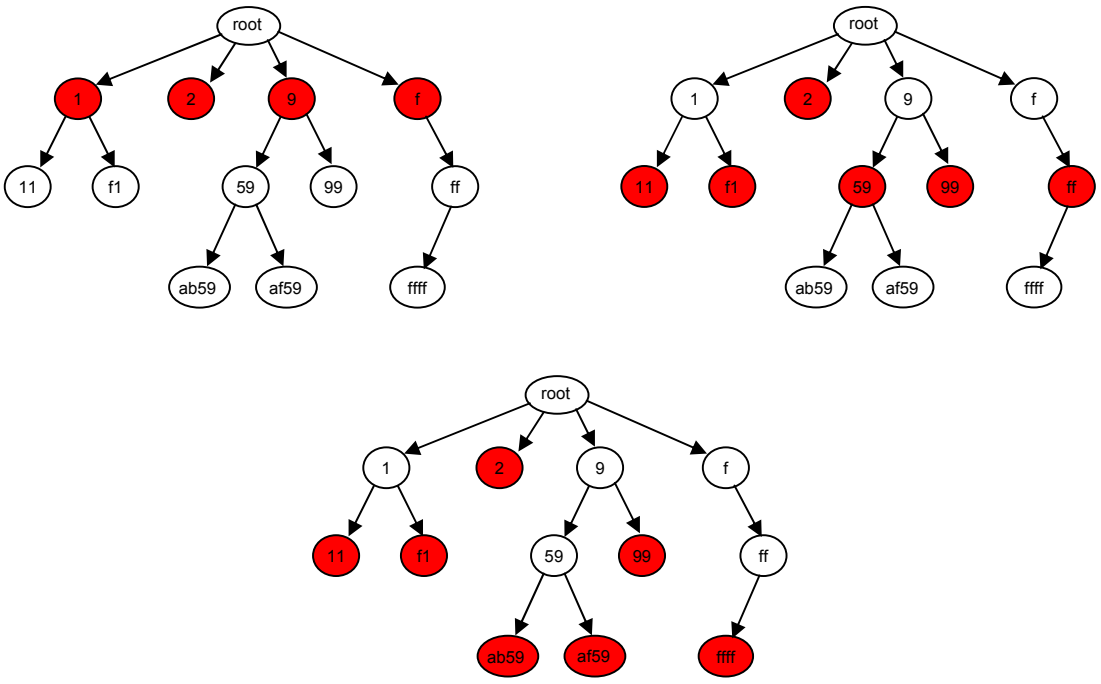


Σχήμα 3.5. Pattern tree.

Με το φιλτράρισμα αυτό επιχειρούμε να αναγνωρίσουμε τα patterns που έχουν σημασία. Με το να φιλτράρουμε τα patterns που δεν εμφανίζονται πολλές φορές εξασφαλίζουμε ότι στην ανάλυση που κάνουμε συμπεριλαμβάνονται μόνο patterns που
ο predictor έχει περιθώριο να μάθει. Αφήνοντας έξω τα patterns που έχουν μεγάλη ακρίβεια εξασφαλίζουμε ότι βλέπουμε τα μικρότερα δυνατά patterns στον υπολογισμό μας.

Στη συνέχεια για να αποφασίσουμε ποια μεγέθη ιστορίας είναι σημαντικά χρησιμοποιούμε την εξής μέθοδο:

- Κατασκευάζουμε ένα δέντρο με τα φιλτραρισμένα patterns παρομοίως με το σχήμα 3.5
- Για κάθε επίπεδο του δέντρου βρίσκουμε τα φύλλα του δέντρου μέχρι το επίπεδο αυτό (σχήμα 3.6).
- Για όλα τα φύλλα που βρήκαμε:
    - Βρίσκουμε το συνολικό frequency τους προσθέτοντας το frequency του κάθε leaf.
    - Βρίσκουμε το συνολικό accuracy προσθέτοντας το frequency*accuracy του κάθε leaf και διαιρώντας δια το συνολικό frequency.
    - Βρίσκουμε το συνολικό entropy προσθέτοντας το frequency*entropy του κάθε leaf και διαιρώντας δια το συνολικό frequency.



Σχήμα 3.6. Leaf nodes για τα επίπεδα ένα, δύο και τρία.

3.6 Περίληψη κεφαλαίου

Στο κεφάλαιο αυτό παρουσιάσαμε τη μεθοδολογία με την οποία αναλύουμε τα δεδομένα εκτέλεσης διαφόρων benchmarks. Χρησιμοποιούμε αποτελέσματα εκτέλεσης των benchmarks vpr00, bzip200, mcf00 και crafty00 από το SPEC2000 benchmark suite τα οποία τρέχουν σε εξομοίωση του L-TAGE predictor. Η μεθοδολογία ξεκινά με το να αναγνωρίσουμε τις εντολές που συνεισφέρουν σε μεγάλο βαθμό στην χαμηλή επίδοση του L-TAGE. Εντοπίζουμε τις εντολές αυτές μετρώντας τον συνολικό αριθμό φορών που έγιναν miss predict.

Στη συνέχεια επιλέξαμε τρεις εντολές από κάθε benchmark που έχουν τον μεγαλύτερο αριθμό misspredictions καθώς και μία που να έχει πολύ ψηλό αριθμό ορθών predictions. Για τις εντολές αυτές δημιουργούμε για κάθε δυναμικό στιγμιότυπο της εντολής, τα 1024 bits ιστορίας που προηγήθηκαν. Στη συνέχεια αναλύουμε την ιστορία αυτή σε patterns μεγέθους 2^i (i=0,1,2,3..11). Αφού το κάνουμε αυτό χτίζουμε ένα δέντρο όπου το κάθε pattern έχει για πατέρα του το pattern που αποτελεί το δεξί μισό τους μέρος. Κάθε pattern δηλαδή περιέχει στο δεξί του μισό μέλος τον πατέρα του.

Με βάση το δέντρο αυτό μετρούμε την ποσότητα entropy για κάθε επίπεδο του δέντρου αφού εφαρμόσουμε κάποια κριτήρια αποκοπής. Το entropy το ορίσαμε ως:

Entropy = max (number_of_times_taken, number_of_times_not_taken) / number_of_times_executed

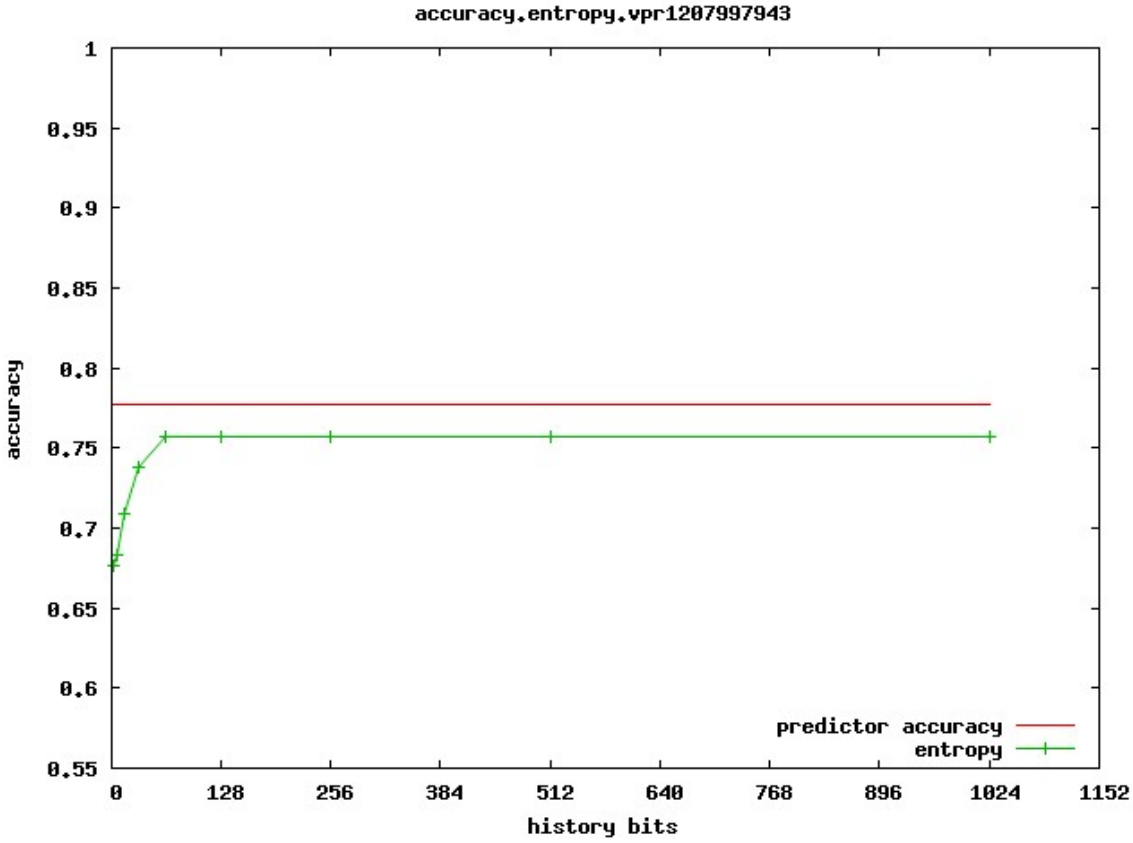
Στη συνέχεια παράγουμε γραφικές παραστάσεις για να συγκρίνουμε το entropy που υπολογίσαμε με την ακρίβεια (accuracy) του predictor.

Κεφάλαιο 4

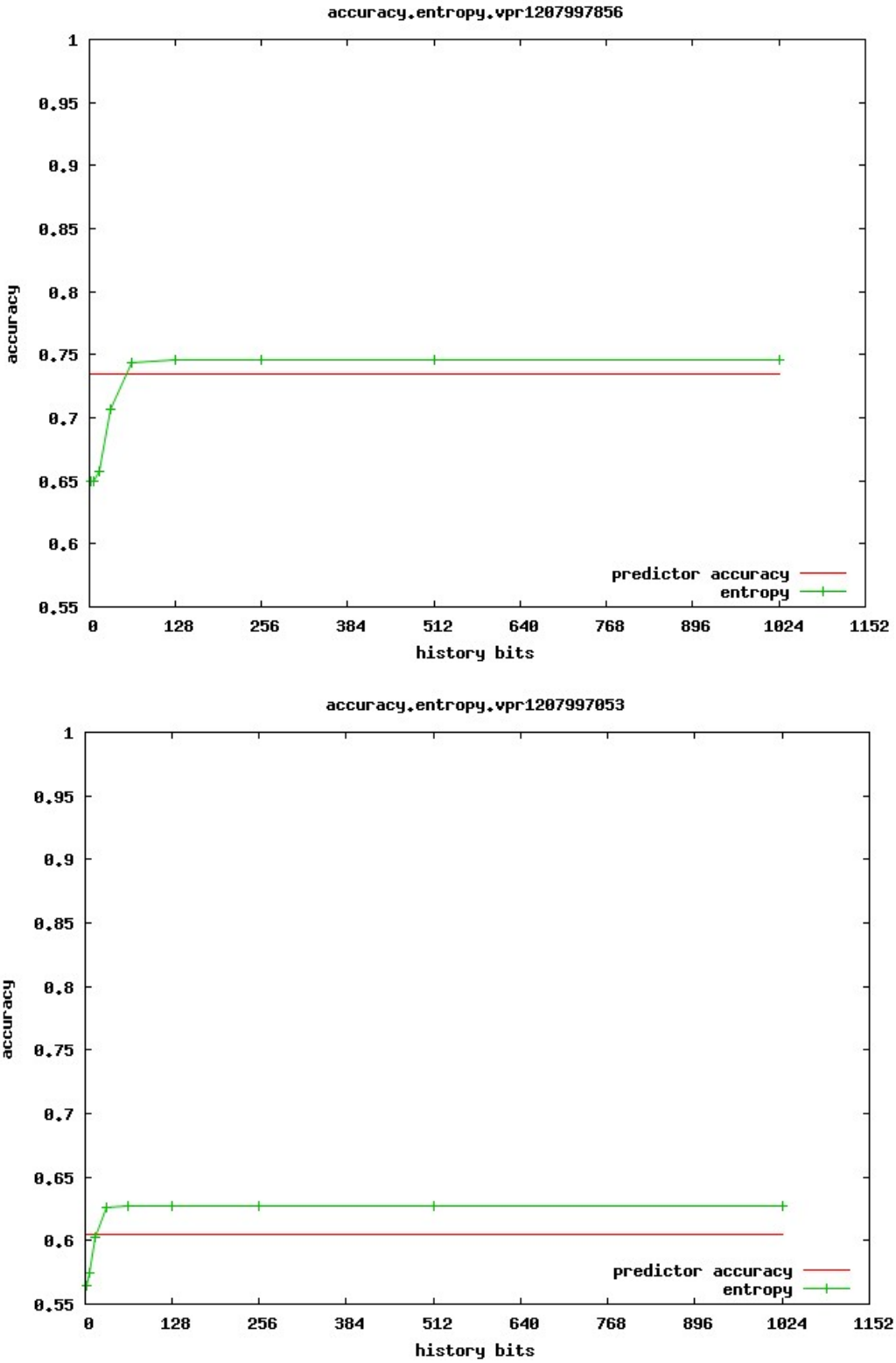
Αποτελέσματα

4.1 Αποτελέσματα entropy

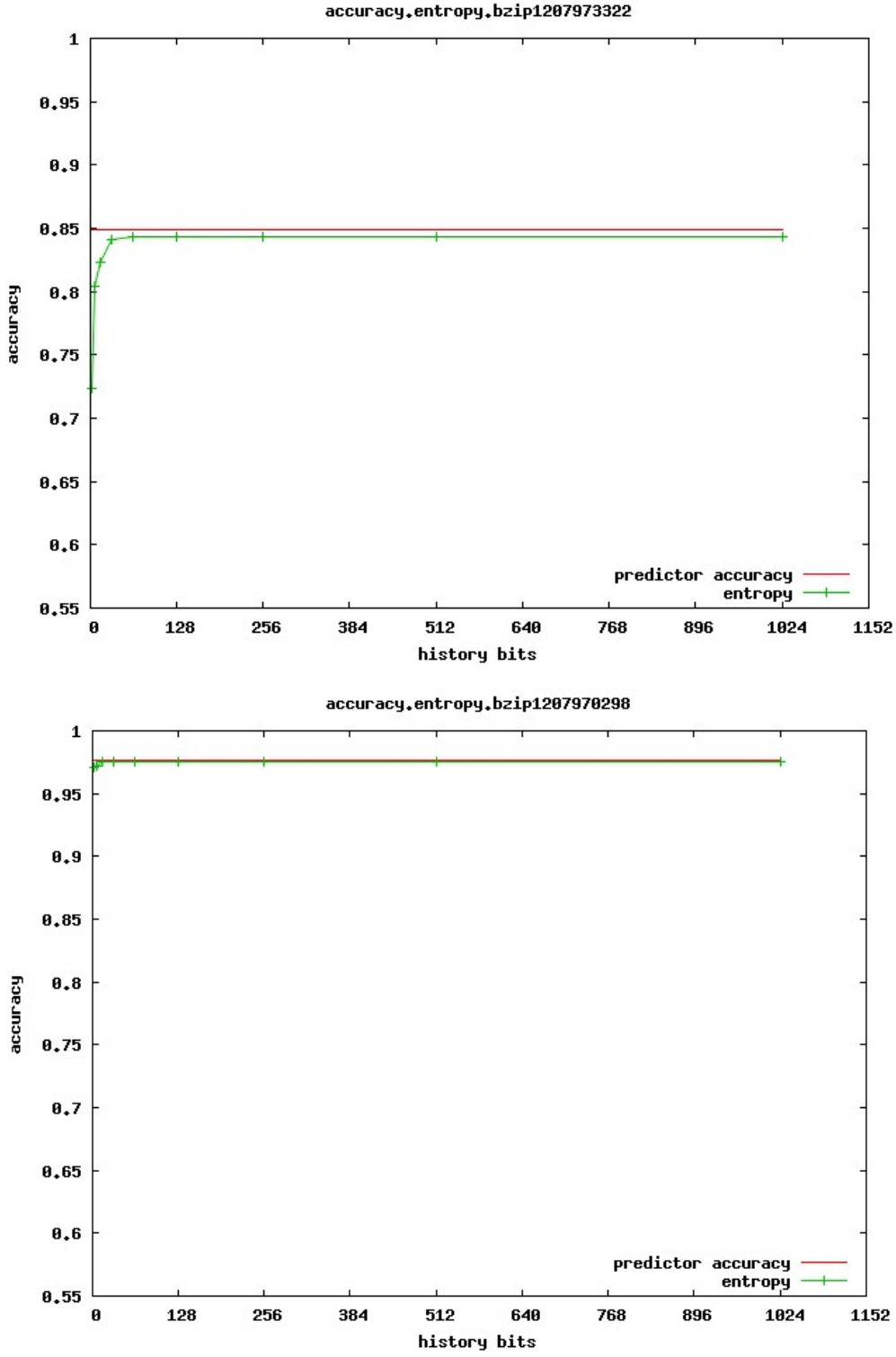
Στο υποκεφάλαιο αυτό παραθέτουμε τα αποτελέσματα της ανάλυσης που έγινε. Παρουσιάζονται τα αποτελέσματα για τις 12 «δύσκολες» εντολές του πίνακα 3.3 και ακολουθούν τα αποτελέσματα των «εύκολων» εντολών του πίνακα 3.4. Στις γραφικές που ακολουθούν δείχνουμε το entropy που υπολογίσαμε και την ακρίβεια του L-TAGE για σύγκριση.



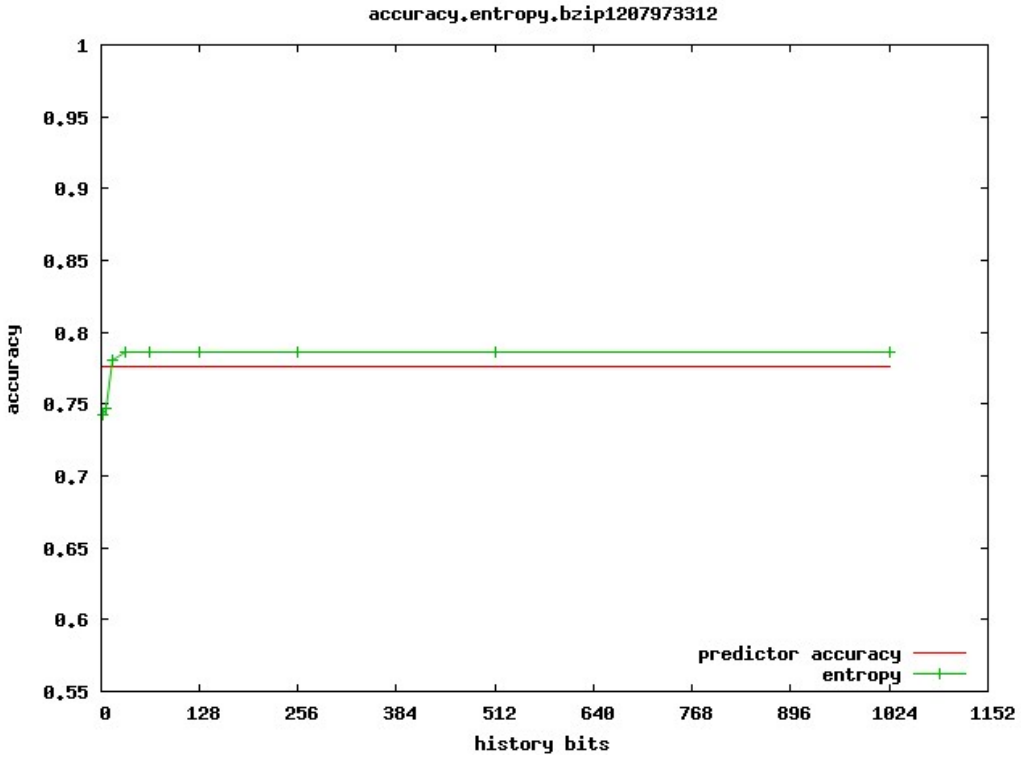
Σχήμα 4.1. Αποτέλεσμα vpr00.

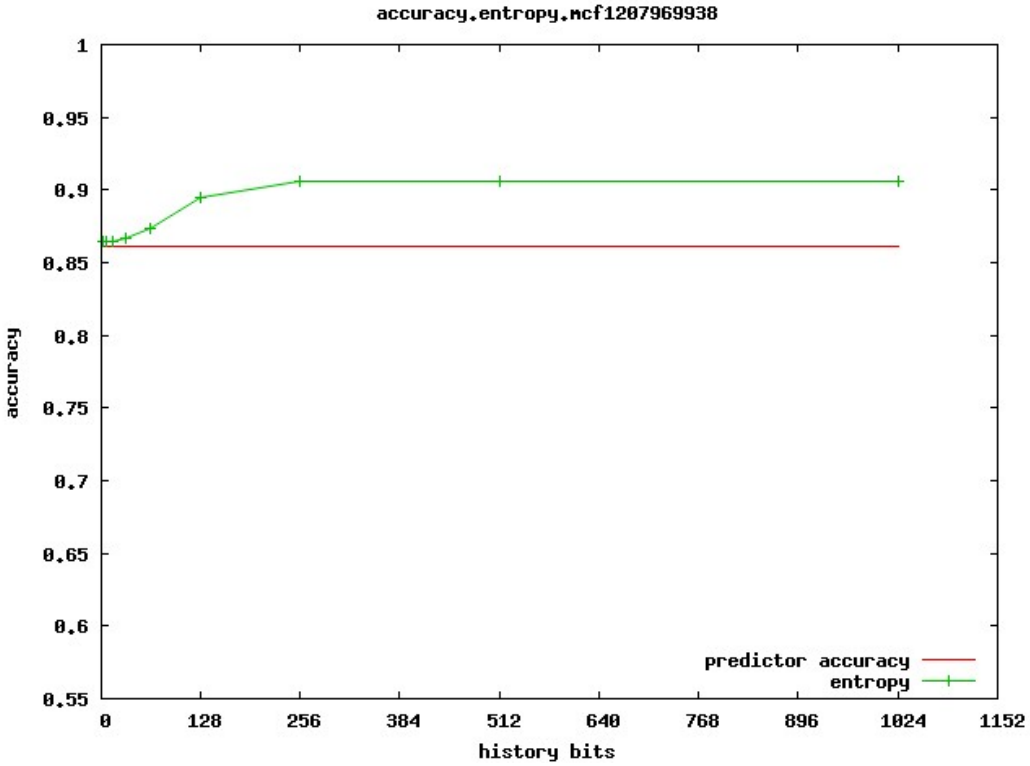


Σχήμα 4.2. Αποτελέσματα vpr00.

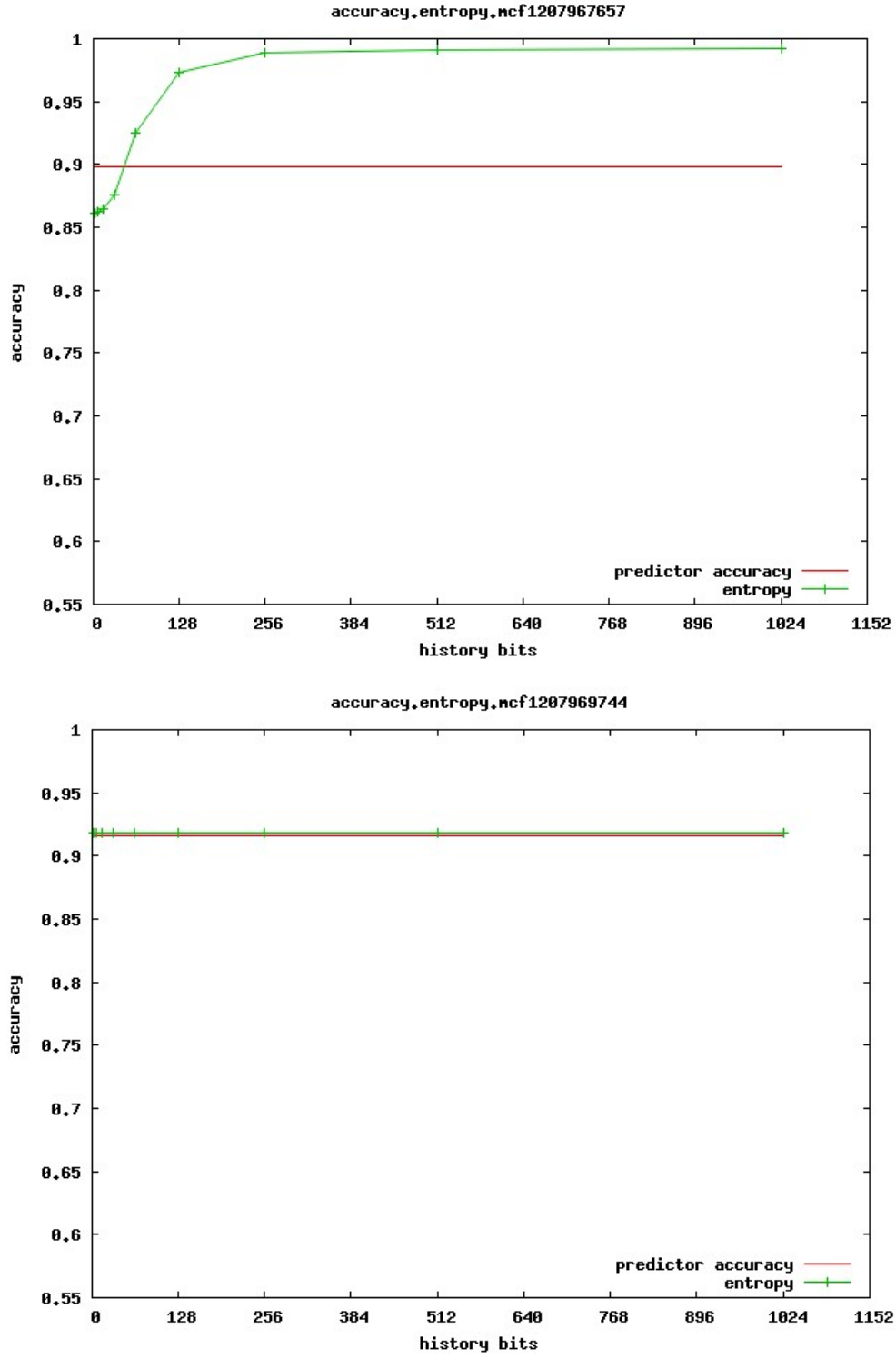


Σχήμα 4.3. Αποτελέσματα bzip00.

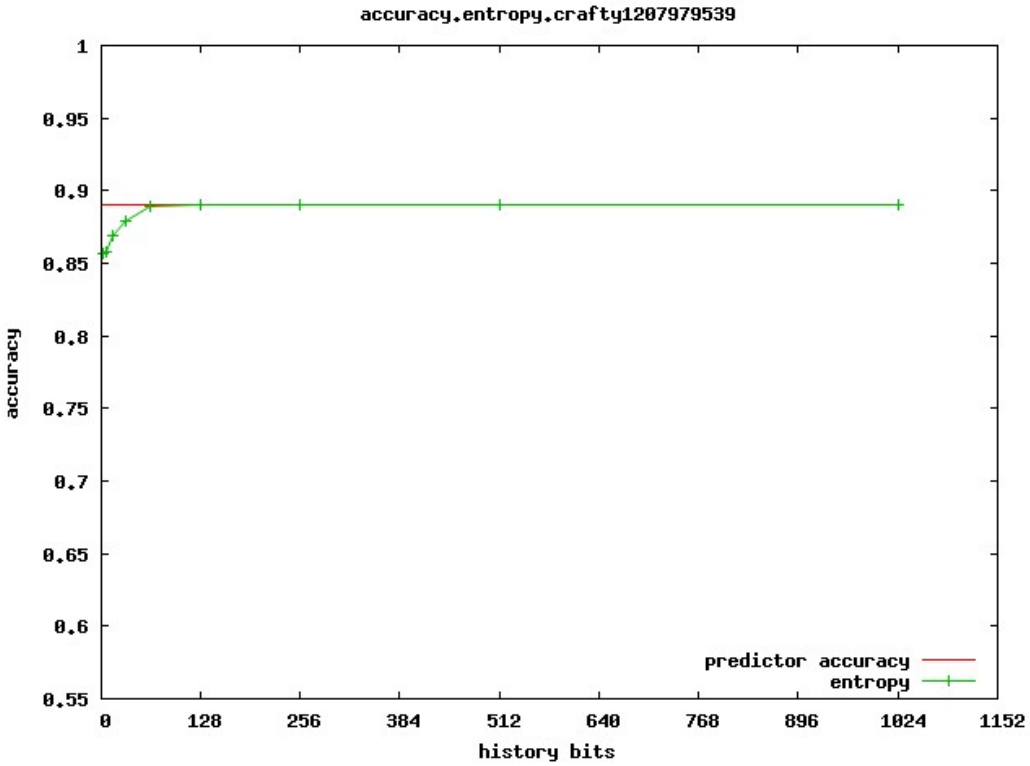


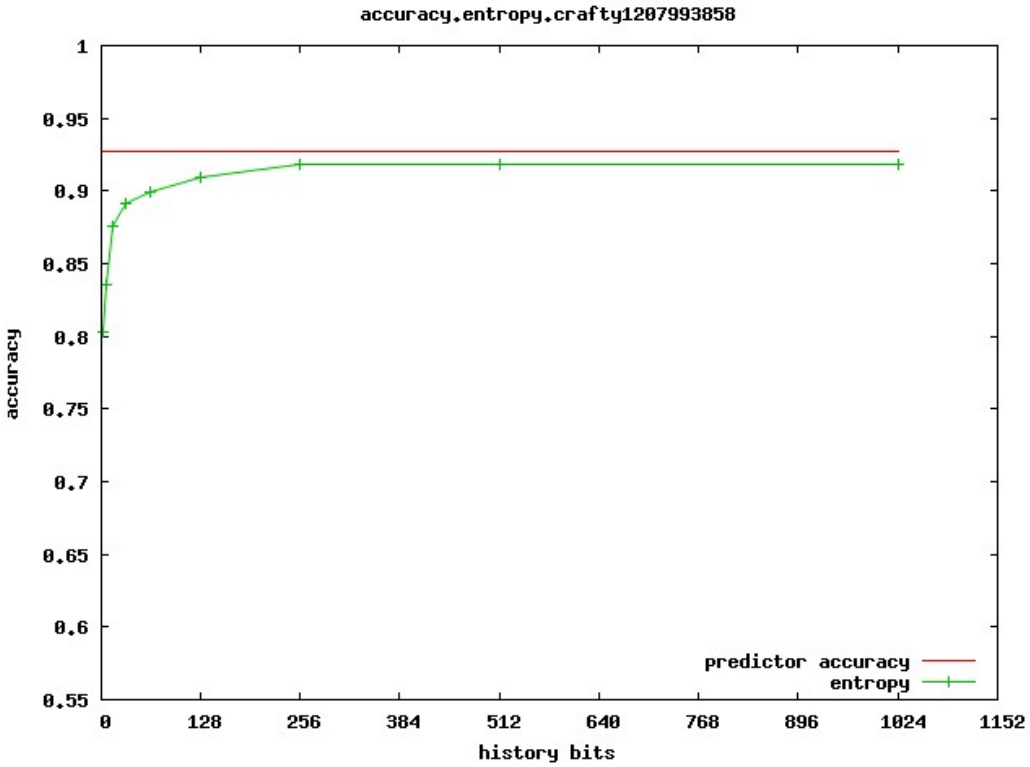


Σχήμα 4.4. Αποτελέσματα bzip00 / mcf00.

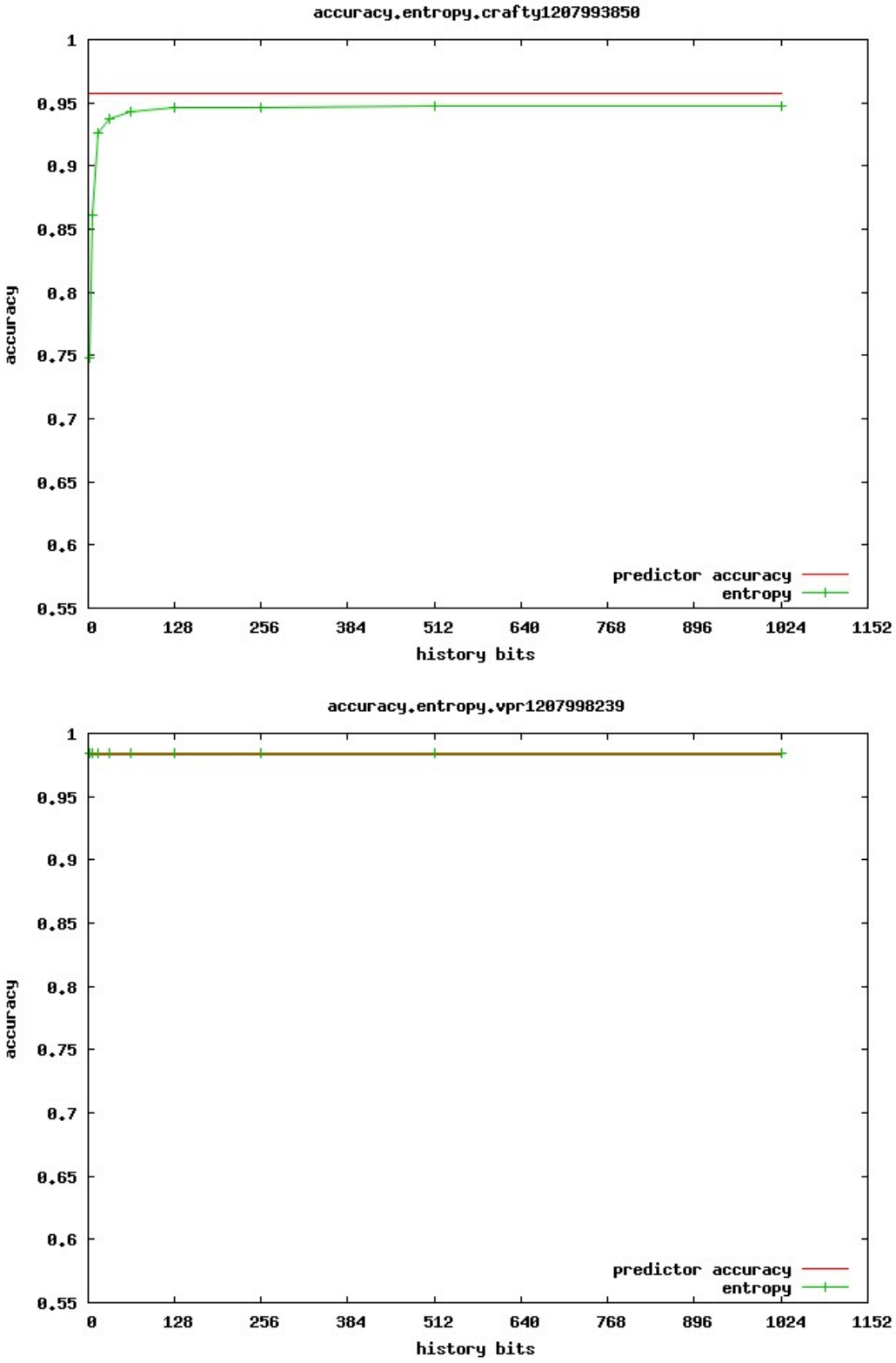


Σχήμα 4.5. Αποτελέσματα mcf00.

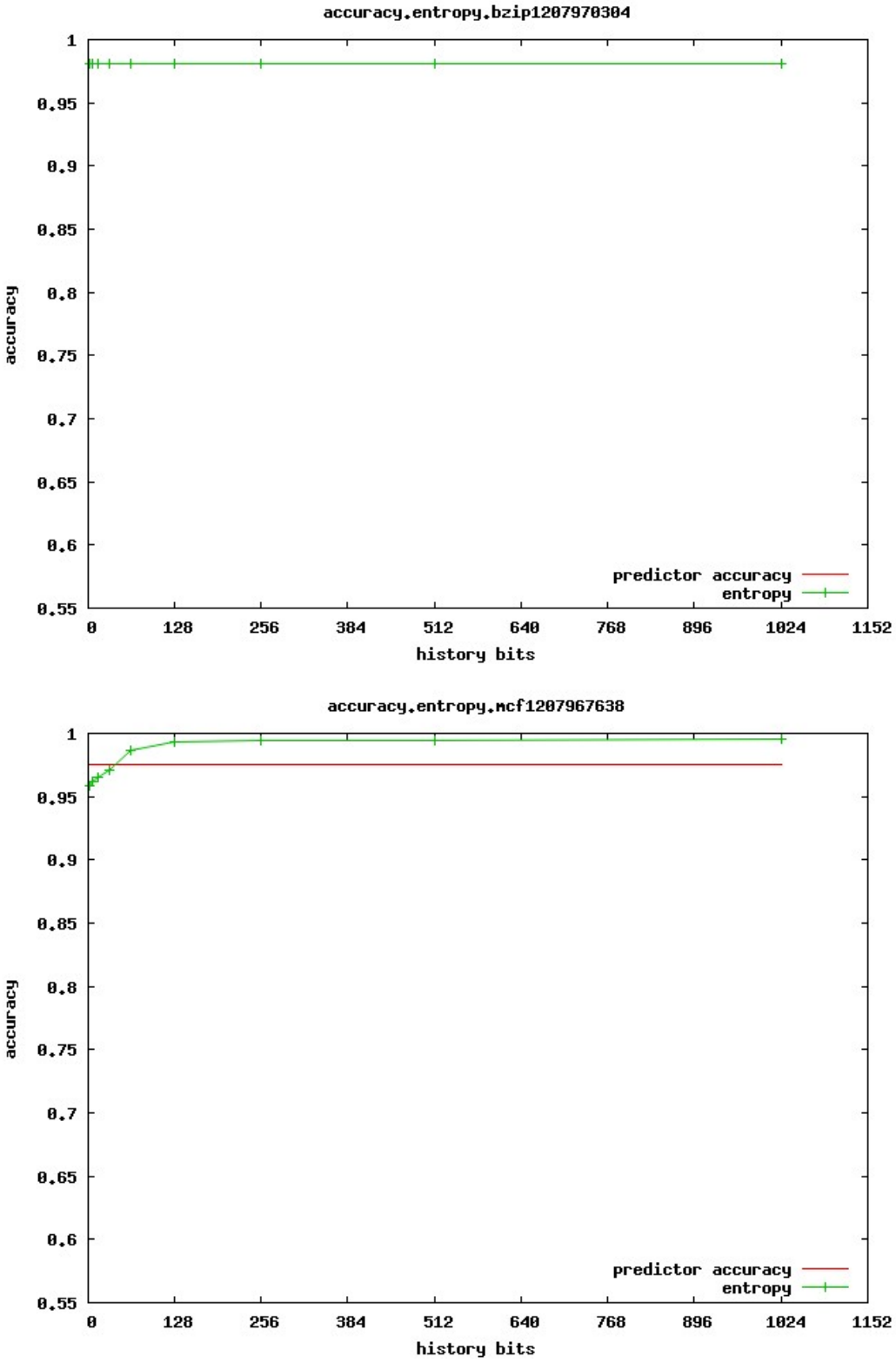




Σχήμα 4.6. Αποτελέσματα crafty00.



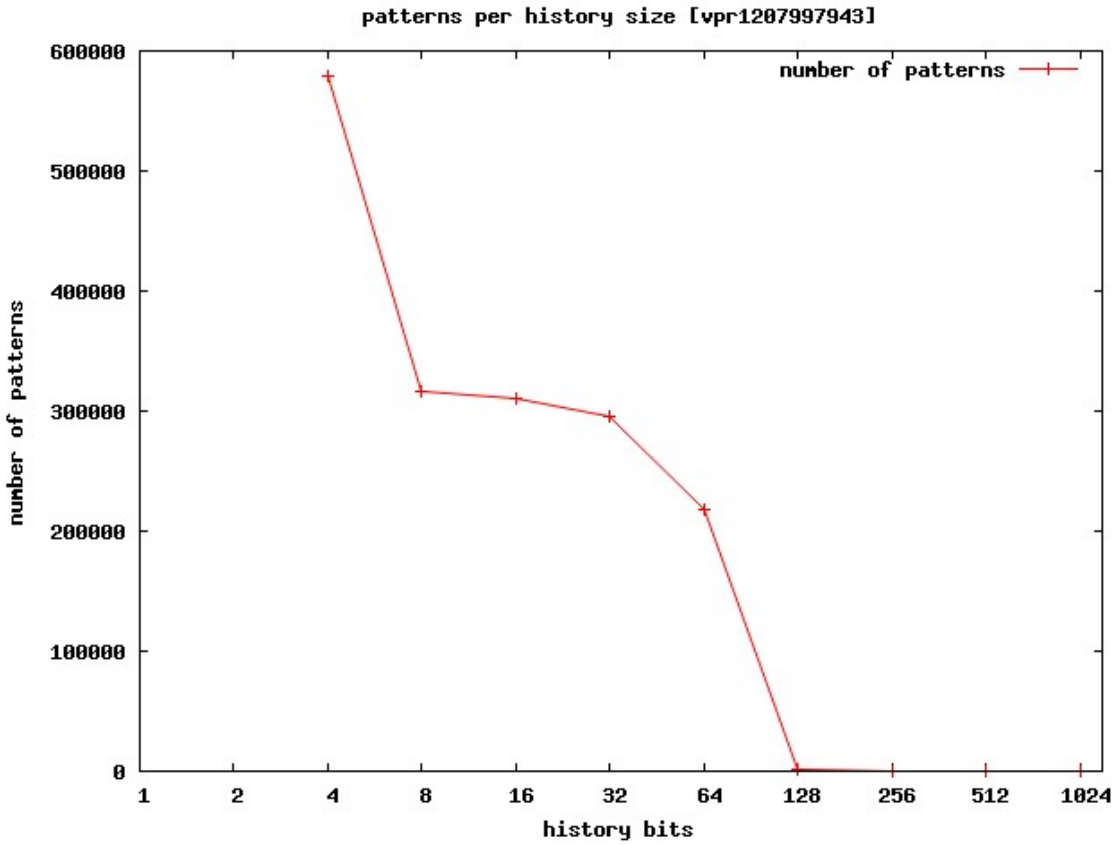
Σχήμα 4.7. Αποτελέσματα crafty00 / vpr00.



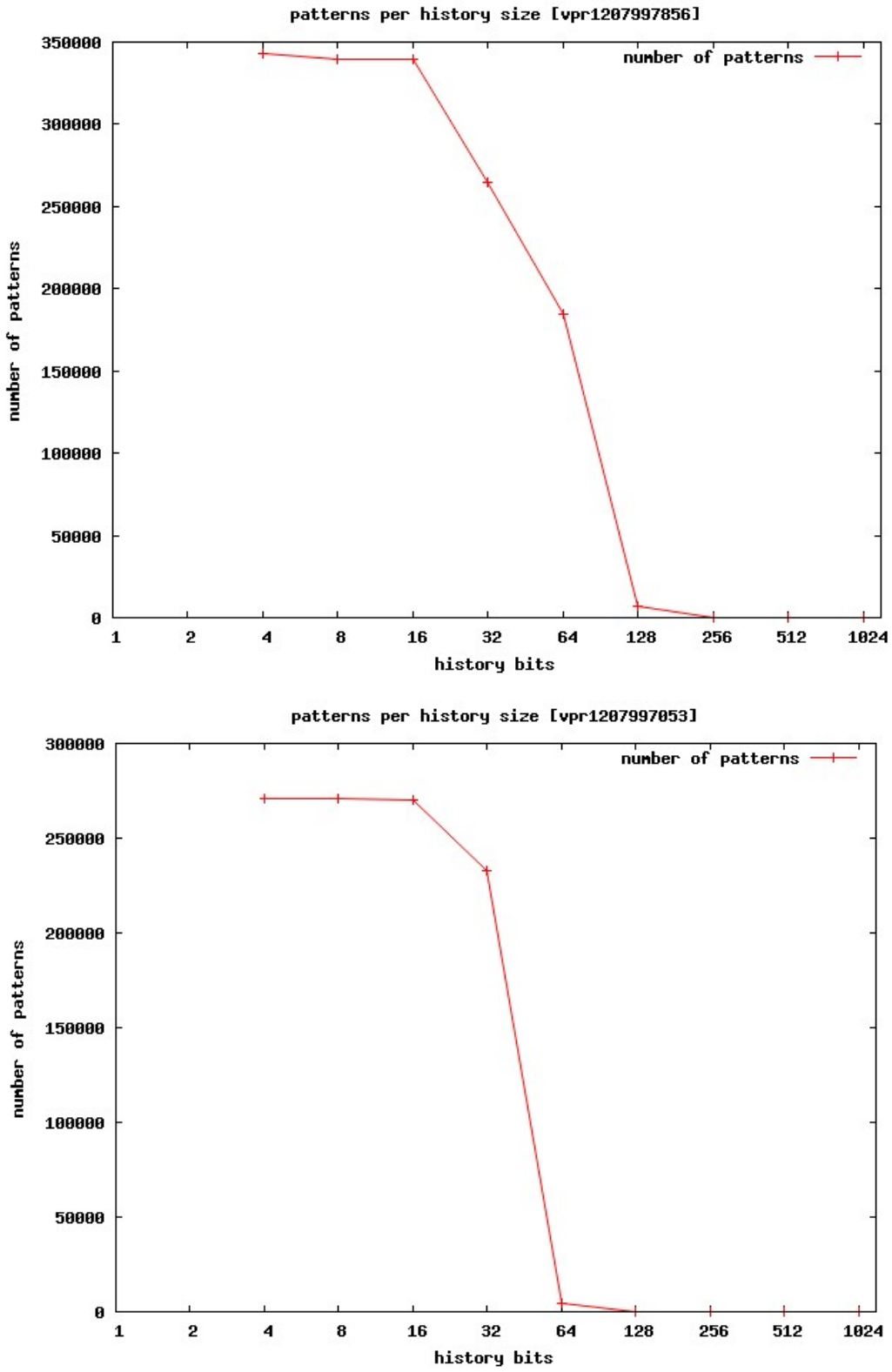
Σχήμα 4.8. Αποτελέσματα bzip00 / mcf00.

4.2 Αριθμός patterns ανά επίπεδο ιστορίας

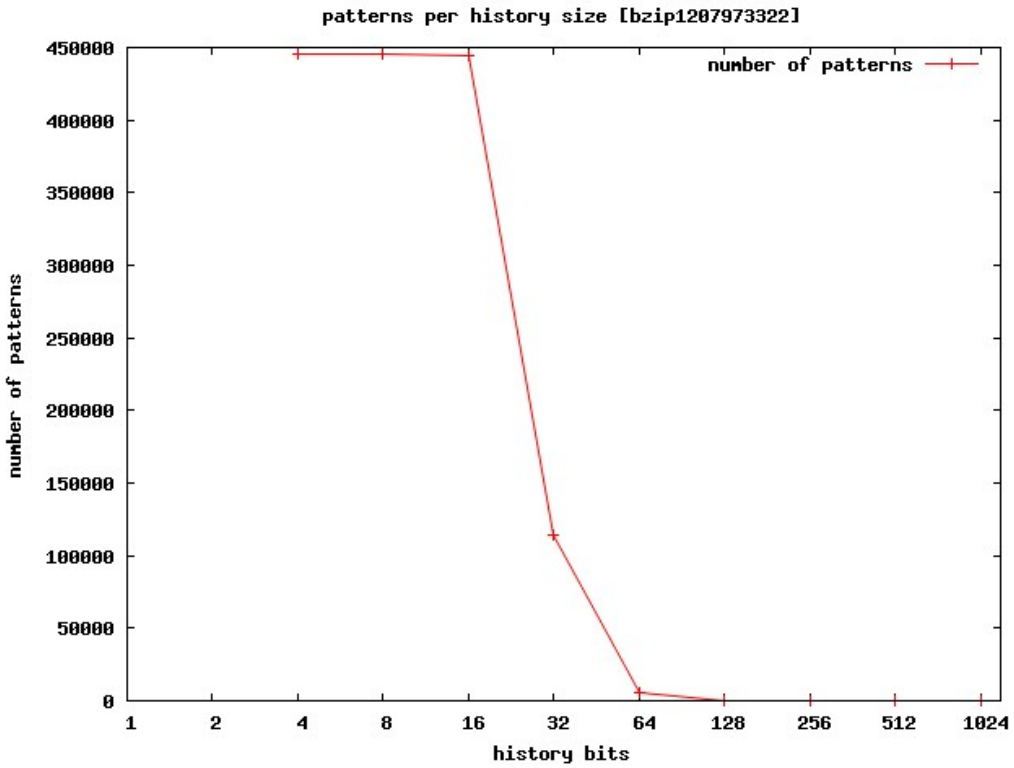
Στο υποκεφάλαιο αυτό παρουσιάζουμε, για όλες τις εντολές που εξετάσαμε, τον αριθμό των patterns που παρουσιάζονται σε κάθε επίπεδο ιστορίας.

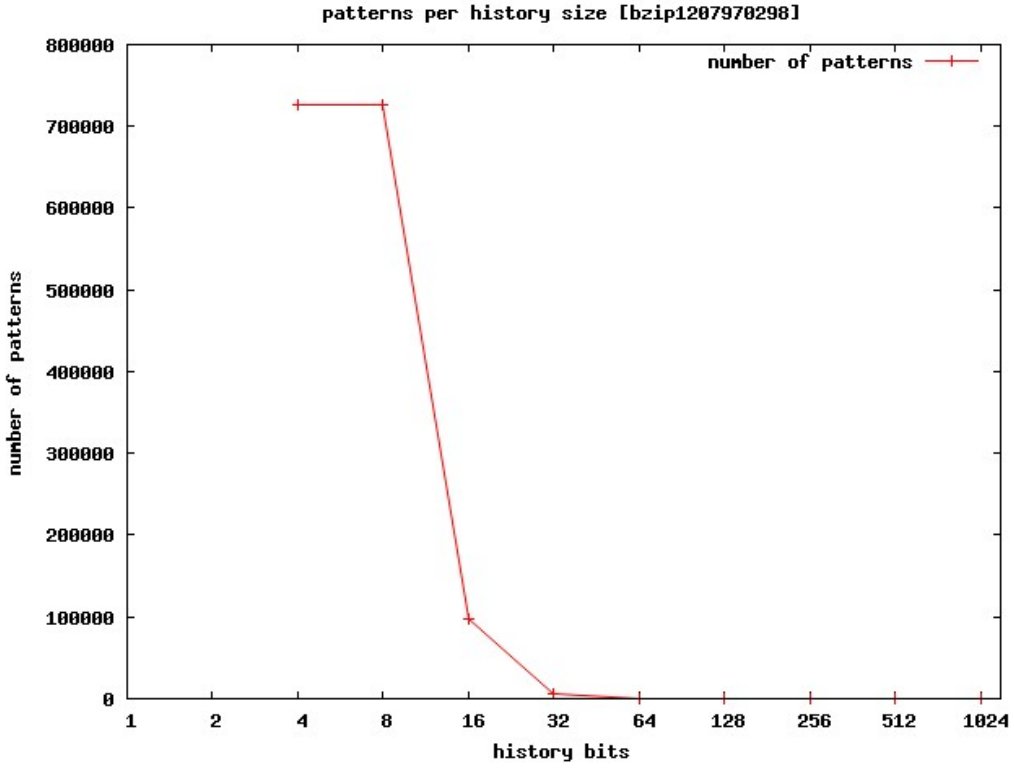


Σχήμα 4.9. Αποτέλεσμα vpr00.

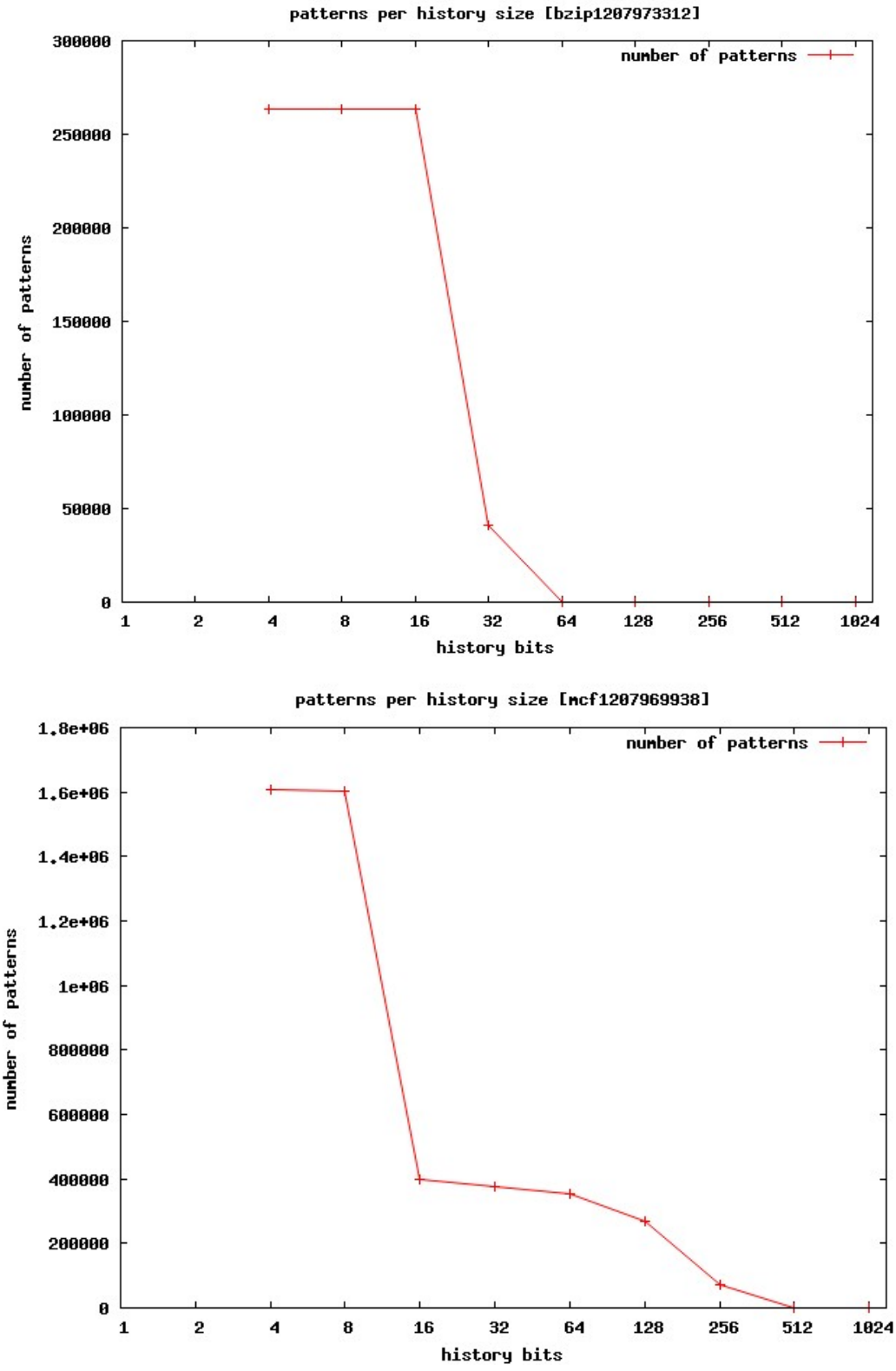


Σχήμα 4.10. Αποτελέσματα vpr00.

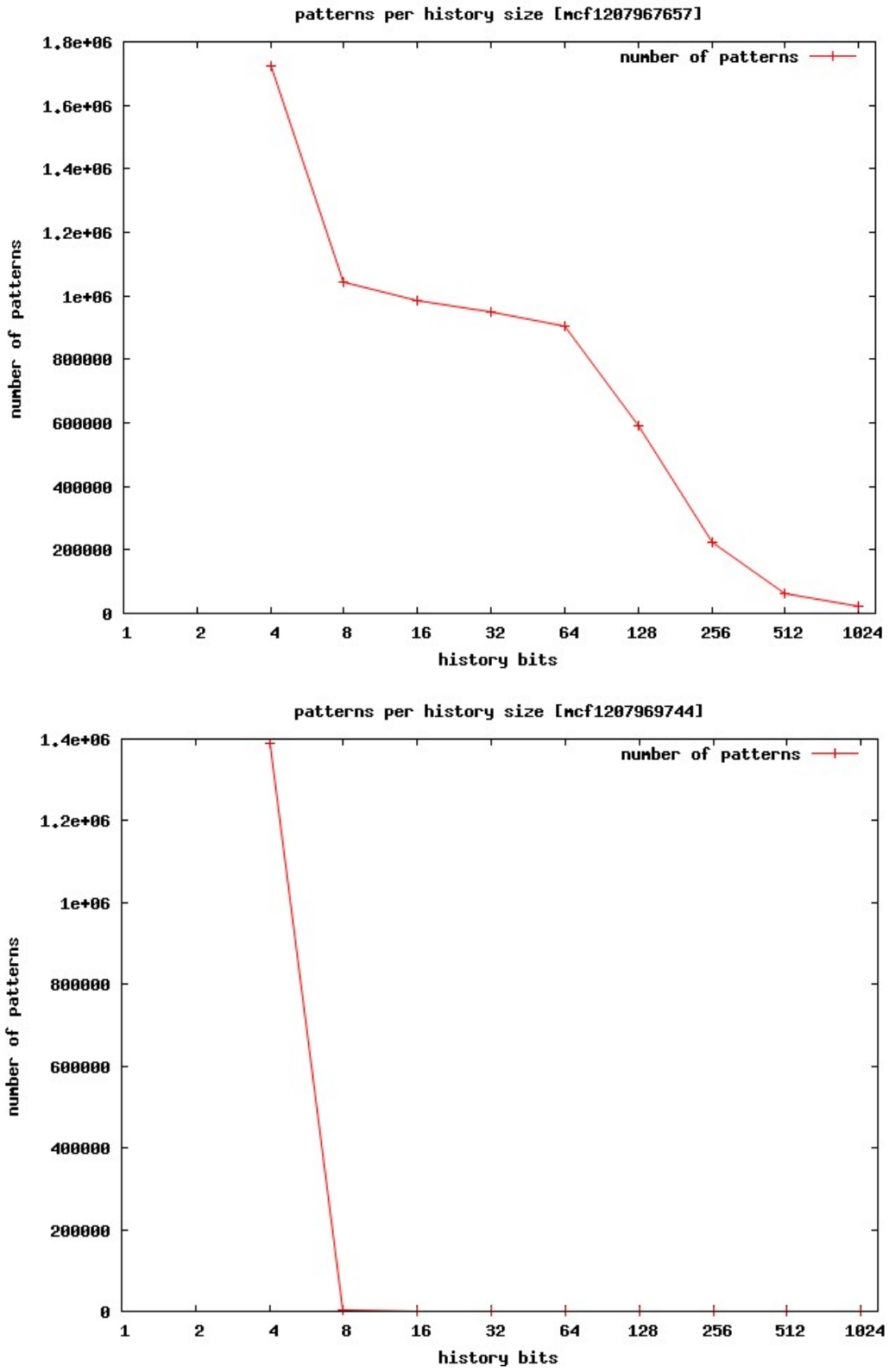




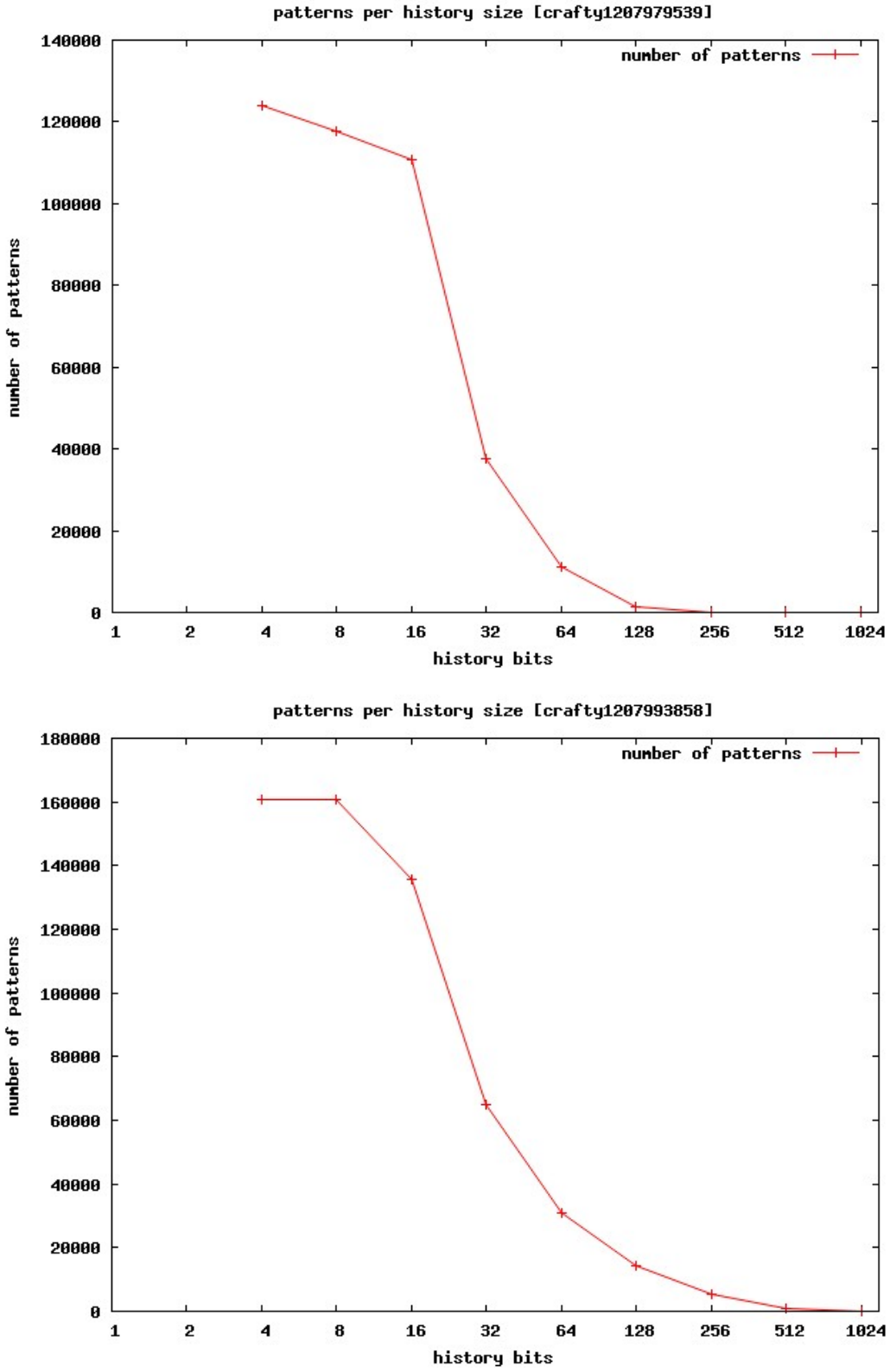
Σχήμα 4.11. Αποτελέσματα bzip00.



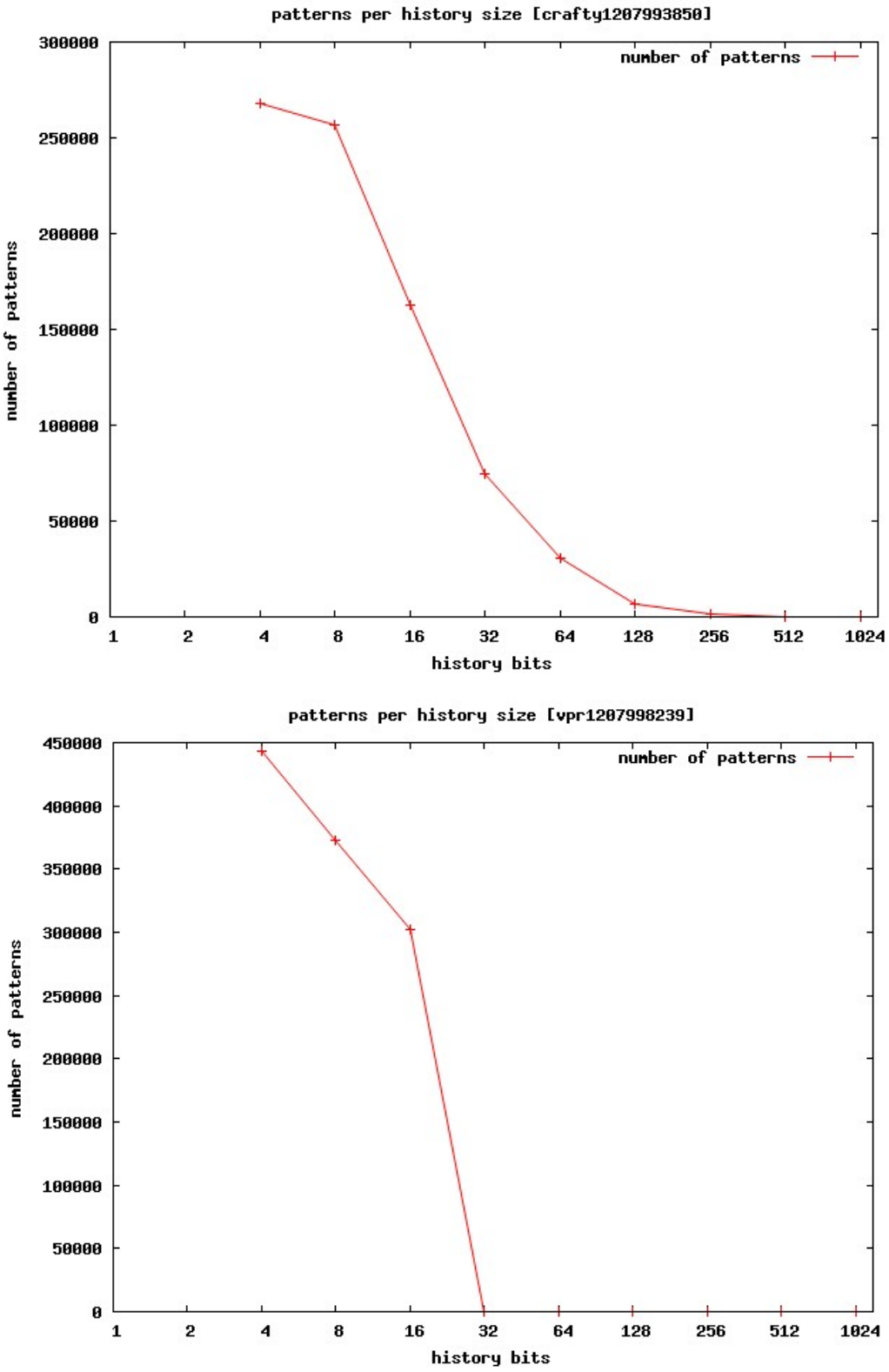
Σχήμα 4.12. Αποτελέσματα bzip00 / mcf00.



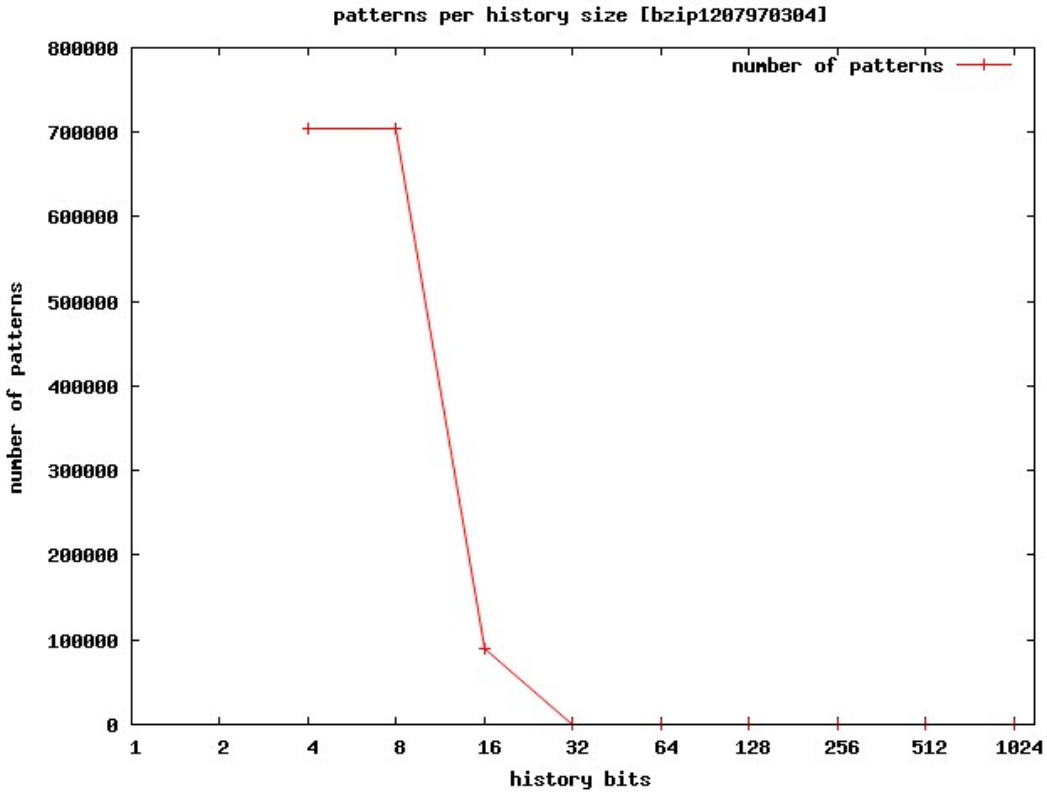
Σχήμα 4.13. Αποτελέσματα mcf00.

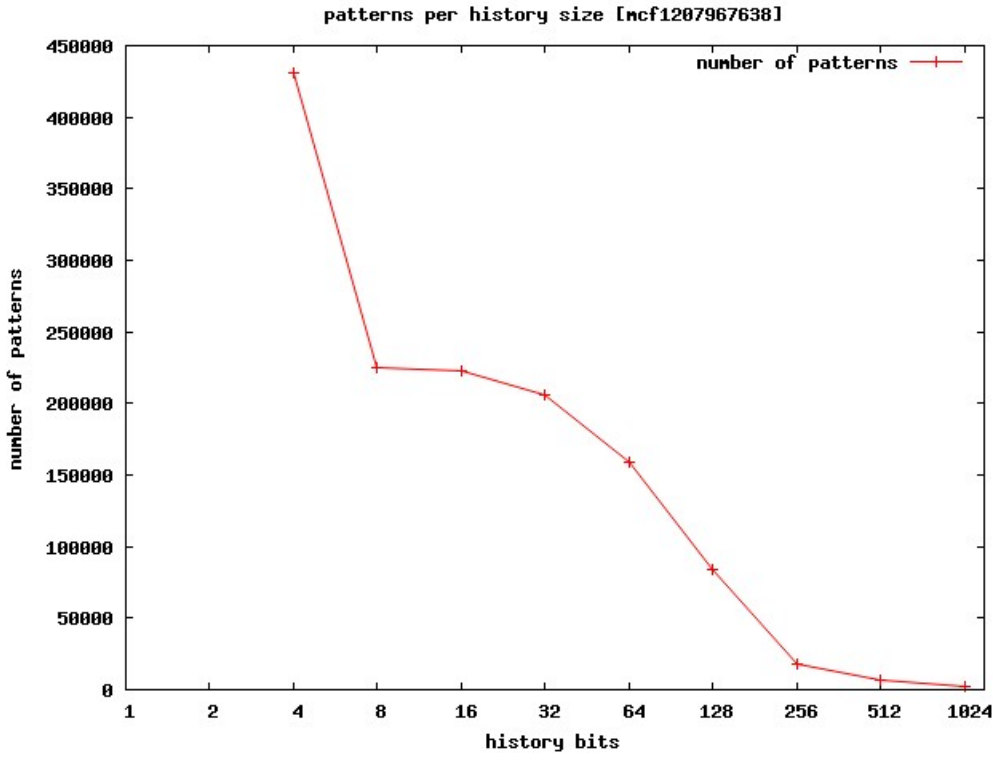


Σχήμα 4.14. Αποτελέσματα crafty00.



Σχήμα 4.15. Αποτελέσματα crafty00 / vpr00.



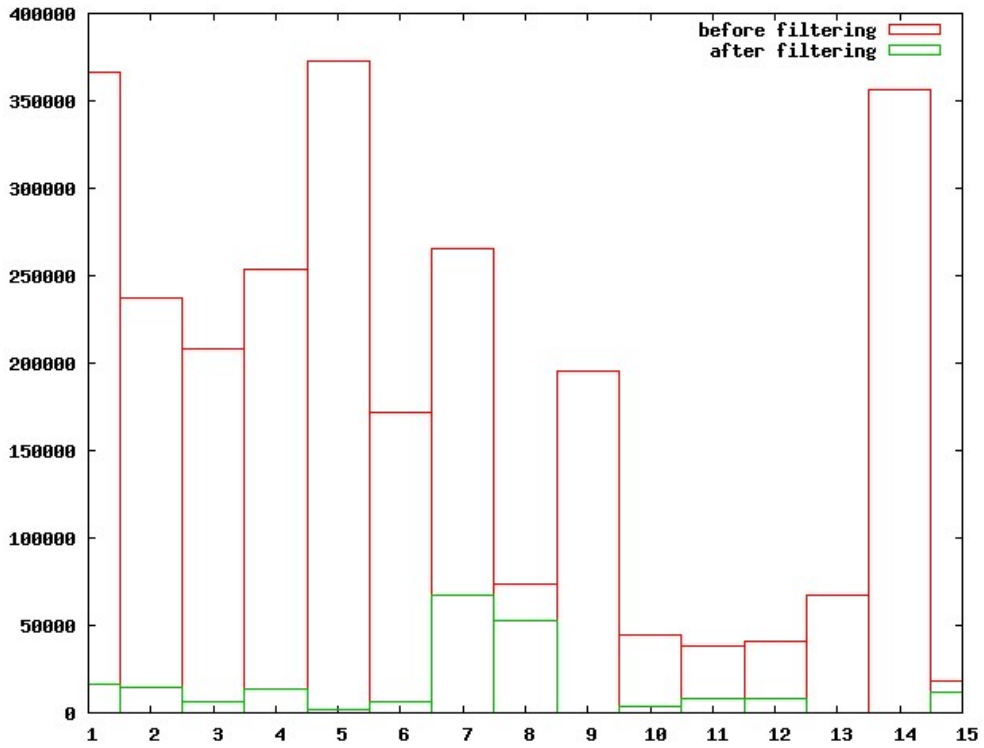


Σχήμα 4.16. Αποτελέσματα bzip00 / mcf00.

Πιο κάτω παρουσιάζουμε τον αριθμών leafs που υπάρχουν πριν και μετά την εφαρμογή της λειτουργίας αποκοπής για τα φύλλα του δέντρου μοτίβων. Η αντιστοίχηση για τον άξονα χ φαίνεται στον πίνακα 4.2:

1	vpr.1207997943
2	vpr.1207997856
3	vpr.1207997053
4	bzip.1207973322
5	bzip.1207970298
6	bzip.1207973312
7	mcf.1207969938
8	mcf.1207967657
9	mcf.1207969744
10	crafty.1207979539
11	crafty.1207993858
12	crafty.1207993850
13	vpr.1207998239
14	bzip.1207970304
15	mcf.1207967638

Πίνακας 4.2. Αντιστοίχηση στήλης και εντολής.



Σχήμα 4.9. Διαφορά leaf nodes πριν και μετά την αποκοπή για όλα τα benchmarks.

4.3 Σχολιασμός

Από τις πιο πάνω γραφικές παραστάσεις παρατηρούμε πως στις περισσότερες περιπτώσεις η διαφορά entropy – accuracy είναι πολύ μικρή (~1-3%) και πως σε καμία περίπτωση το accuracy δεν είναι σημαντικά μεγαλύτερο από το entropy. Αντίθετα σε δύο περιπτώσεις το entropy είναι σημαντικά μεγαλύτερο του accuracy (σχήματα 4.4 και 4.5).

Στις περισσότερες περιπτώσεις έχουμε αύξηση του entropy μέχρι τα 128 bits ενώ σε τρεις περιπτώσεις η αύξηση του entropy συνεχίζει μέχρι τα 256 bits. Σε καμία περίπτωση δεν έχουμε αύξηση του entropy για history bit length μεγαλύτερο του 256.

Αν παρατηρήσουμε τα σχήματα που δείχνουν τον αριθμό των patterns ανά επίπεδο ιστορίας βλέπουμε πως ο αριθμός των patterns πλησιάζει το μηδέν στα επίπεδα ιστορίας 128-256 bits, στα ίδια επίπεδα που παρατηρείται και η σταθεροποίηση της τιμής του entropy. Αυτή η παρατήρηση μας οδηγεί στο συμπέρασμα ότι το entropy σταματά να αυξάνεται γιατί ο αριθμός των patterns που παρουσιάζονται στα επίπεδα ιστορίας μεγαλύτερα των 256 bits είναι πολύ μικρός. Αυτό συμβαίνει γιατί στα μεγάλα επίπεδα ιστορίας έχουμε μεγάλο αριθμό από patterns να παρουσιάζεται λίγες φορές και γι αυτό δεν τα συμπεριλαμβάνουμε στην ανάλυση μας (όπως περιγράψαμε στο κεφάλαιο 3, δεν συμπεριλαμβάνουμε patterns που παρουσιάζονται λιγότερο από 10 φορές).

Κεφάλαιο 5

Συμπεράσματα

5.1 Σημασία του μεγέθους ιστορίας

Τα αποτελέσματα της ανάλυσης μας δείχνουν ξεκάθαρα πως η βελτίωση της ακρίβειας entropy είναι μηδαμινή για μεγέθη ιστορίας μεγαλύτερα των 256 bits. Για καμία από τις εντολές που αναλύσαμε δεν υπάρχει αισθητή μεταβολή στην ακρίβεια μετά το μέγεθος αυτό. Αυτό συμβαίνει γιατί με την αύξηση των bits ιστορίας αυξάνεται και ο αριθμός των διαφορετικών συνδυασμών που μπορεί να εμφανιστούν και τους οποίους ο predictor δεν μπορεί να αναγνωρίσει. Με την αύξηση του history size όλο και περισσότεροι συνδυασμοί εμφανίζονται πολύ λίγες φορές και έτσι δεν τους συμπεριλαμβάνουμε στον υπολογισμό μας. Αυτό το κάνουμε γιατί είναι αδύνατο για ένα πραγματικό predictor να μάθει τη συμπεριφορά για patterns που δεν εμφανίζονται συχνά. Στα αποτελέσματά μας αυτό φαίνεται από την μείωση των leaf nodes. Σε κάθε αύξηση του history size έχουμε όλο και περισσότερα patterns να γίνονται leafs γιατί δεν ικανοποιούν το κριτήριο που θέσαμε με αποτέλεσμα το entropy να σταματά να αυξάνεται.

Αυτές οι παρατηρήσεις υποστηρίζουν την απόφαση στα [7] και [8] να χρησιμοποιούνται μικρά μεγέθη ιστορίας για την πλειονότητα των πινάκων των μηχανισμών πρόβλεψης, αφού στα μεγέθη αυτά δείξαμε πως αναγνωρίζονται οι περισσότερες συσχετίσεις.

5.2 Πρόβλεψη δύσκολων εντολών διακλάδωσης

Από την ανάλυση που έγινε παρατηρήσαμε πως για κάποιες εντολές διακλάδωσης η δική μας μέτρηση entropy είναι μεγαλύτερη του accuracy του predictor. Αυτό μας οδηγεί στο συμπέρασμα ότι ακόμα και για δύσκολες εντολές διακλάδωσης μπορεί να υπάρχει κάποιο περιθώριο βελτίωσης της ακρίβειας πρόβλεψης. Στις περισσότερες περιπτώσεις όμως η ακρίβεια της μέτρησης μας ήταν πολύ κοντά ή και χαμηλότερη από αυτήν του predictor. Αυτό μας οδηγεί στο συμπέρασμα πως για την πλειονότητα των εντολών που εξετάσαμε ο predictor χρησιμοποιεί αποδοτικά τις συσχετίσεις που υπάρχουν.

Παρόλα αυτά η ακρίβεια του predictor, αν και πολύ κοντά στη δική μας μέτρηση, για κάποιες εντολές, είναι πολύ χαμηλή. Στην ανάλυση που κάναμε, διαχωρίσαμε τα history patterns σε διάφορα μεγέθη. Κάθε pattern ξεκινά από το πιο πρόσφατο bit ιστορίας και συνεχίζει για κάποιο μέγεθος n. Αυτή η ανάλυση συμβαδίζει με τον τρόπο λειτουργίας του L-TAGE predictor με τον οποίο συγκρίναμε τα αποτελέσματά μας. Το χαμηλό ποσοστό της δικής μας μέτρησης μάς ωθεί στο συμπέρασμα πως μπορεί να υπάρχουν συσχετίσεις που η μέθοδός μας δεν αναγνωρίζει. Αυτό συμβαδίζει με τα αποτελέσματα του [6] όπου διαπιστώθηκε πως υπάρχει σημαντικός αριθμός συσχετίσεων σε μη συνεχείς θέσεις στον history register.

5.3 Μελλοντική εργασία

Μια ενδιαφέρον κατεύθυνση για μελλοντική εργασία είναι να μελετηθεί ο κώδικας των benchmarks που αναλύσαμε. Συγκεκριμένα θα ήταν χρήσιμο να γίνει ανάλυση του κώδικα που προηγείται των εντολών που μελετήθηκαν. Μια τέτοια μελέτη ίσως να ήταν χρήσιμη για να κατανοήσουμε την αιτία που προκαλεί τόσο χαμηλά ποσοστά ακρίβειας για τις εντολές αυτές.

Επίσης θα ήταν ενδιαφέρον να επαναληφθεί το πείραμα για διαφορετικά μεγέθη ιστορίας. Στην δική μας μελέτη η αύξηση γίνεται σε δυνάμεις του δύο. Θα ήταν ενδιαφέρον να

δούμε πως η μέτρηση entropy συμπεριφέρεται για διαφορετικό τρόπο αύξησης του μεγέθους ιστορίας (π.χ. x1.5, x1.6 κλπ).

Επίσης ιδιαίτερα χρήσιμο θα ήταν να γίνει ανάλυση των affectors και affectees όπως περιγράφεται στο [6]. Συγκεκριμένα θα ήταν ενδιαφέρον να γνωρίζουμε κατά πόσο οι εντολές που εξετάσαμε οφείλουν τα ψηλά ποσοστά λανθασμένων προβλέψεων που έχουν, στην έλλειψη συσχετίσεων με άλλες εντολές. Έχουν δηλαδή μικρό αριθμό affectors και affectees. Με την ανάλυση αυτή θα μπορούσαμε να εξερευνήσουμε το ενδεχόμενο να υπάρχουν οι συσχετίσεις, αλλά επειδή βρίσκονται σε μη συνεχείς θέσεις στον history register (υπάρχουν «τρύπες» μεταξύ τους), να μην αναγνωρίζονται από τον predictor. Τέτοιες συσχετίσεις δεν αναγνωρίζονται με τη δική μας μέθοδο, οπότε μια τέτοια ανακάλυψη θα σήμαινε πως το περιθώριο βελτίωσης για τις εντολές που είδαμε είναι μεγαλύτερο.

Τέλος, είναι ενδιαφέρον να συγκριθεί η ακρίβεια της μεθόδου με αυτήν ενός μηχανισμού πρόβλεψης με άπειρους πόρους όπως ο GTL. Μια τέτοια σύγκριση θα φανερώσει αν η διαφορά της δικής μας μεθόδου και του αποτελέσματος του L-TAGE οφείλεται σε έλλειψη πόρων ή σε κακή διαχείριση των πόρων που διαθέτει.

Βιβλιογραφία

[1] Eyerman, S., Smith, J.E., Eeckhout, L.: Characterizing the branch misprediction penalty. In: IEEE International Symposium on Performance Analysis of Systems and Software (March 2006)

[2] Evers, M., Patel, S.J., Chappel, R.S., Patt, Y.N.: An Analysis of Correlation and Predictability: What Makes Two-Level Branch Predictors Work. In: 25th International Symposium on Computer Architecture. (June 1998)

[3] Hinton, G., Sager, D., Upton, M., Boggs, D.,Carmean, D., Kyker, A., Roussel, P.: The Microarchitecture of the Pentium® 4 Processor. In: Intel Technology Journal Q1, 2001.

[4] Jimenez, D.A., Lin, C.: Dynamic Branch Prediction with Perceptrons. In: 7th International Symposium on High Performance Computer Architecture. (Feb. 2001)

[5] McFarling, S.: Combining Branch Predictors. Technical Report DEC WRL TN-36, Digital Western Research Laboratory (June 1993)

[6] Sazeides, Y., Moustakas, A., Constantinides, K., Kleanthous, M.: The Significance of Affectors and Affectees Correlations for Branch Predicion. In: International Conference on High Performance Embedded Architectures and Compilers (HiPEAC), (January, 2008)

[7] Seznec, A.: Analysis of the O-GEometric History Length branch predictor. In: 32nd International Symposium on Computer Architecture. (2005)

[8] Seznec, A.: The L-TAGE Branch Predictor. Journal of Instruction-Level Parallelism 9 (2007)

[9] Thomas, R., Franklin, M., Wilkerson, C., Stark, J.: Improving Branch Prediction by Dynamic Dataflow-based Identification of Correlated Branches from a Large Global History. In: 30th International Symposium on Computer Architecture. (June 2003) 314–323

[10] Walrus Graph Visualization Tool: http://www.caida.org/tools/visualization/walrus/

Άλλη Χρήσιμη Βιβλιογραφία

Εδώ παρουσιάζουμε μια σειρά από άρθρα που μελετήσαμε αλλά δεν αναφέρονται στην εργασία. Τα άρθρα αυτά είναι χρήσιμα στην περαιτέρω κατανόηση των εννοιών της πρόβλεψης καθώς και για μελέτη εναλλακτικών μεθόδων πρόβλεψης όπως η στατική πρόβλεψη.

1) Constantinidis, K., Sazeides, Y.: A Hardware Based Method for Dynamically Detecting Instruction Isomorphism and its Application to Branch Prediction. In: 2nd Value Prediction Workshop. (2004)

2) Chen, L., Dropsho, S., Albonesi, D.H.: Dynamic Data Dependence Tracking and its Application to Branch Prediction. In: 9th International Symposium on High Performance Computer Architecture. (February 2003) 65–76

3) Mahlke, S., Natarajan, B.: Compiler Synthesized Dynamic Branch Prediction. In: 29th International Symposium on Microarchitecture. (December 1996) 153–164

4) Sazeides, Y.: Dependence Based Value Prediction. Technical Report CS-TR-02-00, University of Cyprus (February 2002)

5) Sazeides, Y.: Dependence Based Value Prediction. Technical Report CS-TR-02-00, University of Cyprus, February 2002.

6) Seznec, A., Felix, S., Krishnan, V., and Sazeides, Y.: Design tradeoffs for the alpha EV8 conditional branch predictor, In Proceedings of the 29th International Symposium on Computer Architecture (ISCA-02), New York, 2002, pp. 295--306.

Παράρτημα Α

Για της ανάγκες της εργασίας χρησιμοποιήθηκε το εργαλείο απεικόνισης γράφων walrus. Στο εργαλείο αυτό κάναμε μια σειρά από προσθήκες με στόχο να χρησιμοποιηθεί σε μεταγενέστερη εργασία για ανάλυση γράφων εξαρτήσεων (dependence graphs). Στην εργασία αυτή χρησιμοποιήθηκε για debugging στη φάση της δημιουργίας του δέντρου μοτίβων (pattern tree). Το εργαλείο αυτό δημιουργήθηκε από τον Young Hyun για την εταιρία CAIDA και είναι βασισμένο στην έρευνα της Tamara Munzner. Οι δυνατότητες του εργαλείου περιλαμβάνουν:

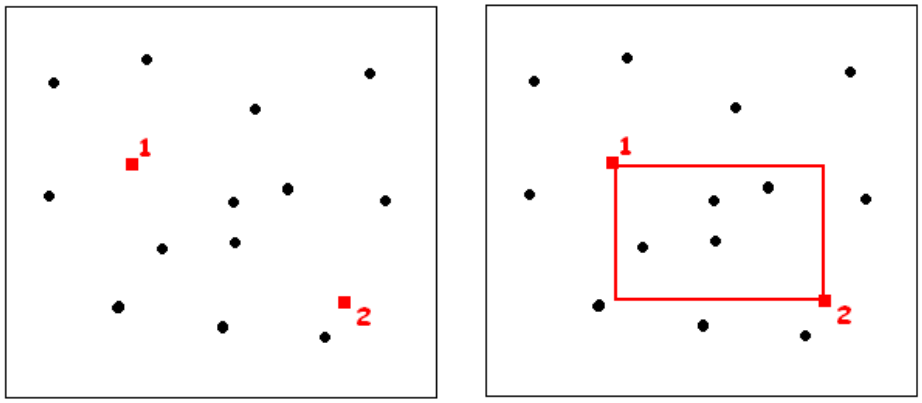
- Τρισδιάστατη απεικόνιση μεγάλων γράφων (ικανότητα απεικόνισης γράφων της τάξης των 10^6 κόμβων).
- Σταθερό frame rate ανεξαρτήτως του μεγέθους του γράφου
- Χρησιμοποιεί fish eye distortion. Αυτό δίνει τη δυνατότητα να απεικονίζεται μεγάλο μέρος του γράφου σε περιορισμένο χώρο(οθόνη). Στο κέντρο της απεικόνισης υπάρχει μεγαλύτερη λεπτομέρεια και όσο απομακρυνόμαστε από το κέντρο η λεπτομέρεια μειώνεται.
- Ικανότητα χρωματισμού των κόμβων.
- Ικανότητα απεικόνισης πληροφοριών(attributes) για κάθε κόμβο.

Περιορισμοί:

- Ο γράφος που απεικονίζεται δεν μπορεί να μεταβάλλεται δυναμικά κατά την εκτέλεση του προγράμματος.
- Δεν υπάρχει δυνατότητα επιλογής πολλών κόμβων.
- Δεν υπάρχει δυνατότητα απεικόνισης του περιεχομένου πολλών κόμβων.
- Υποστηρίζονται μόνο κατευθυνόμενοι γράφοι.
- Για να είναι σε χρήσιμη μορφή η απεικόνιση του γράφου, πρέπει να δοθεί από τον χρήστη κατάλληλο ελάχιστο δέντρο(spanning tree).
- Ανά πάσα στιγμή μόνο ένας γράφος μπορεί να απεικονίζεται

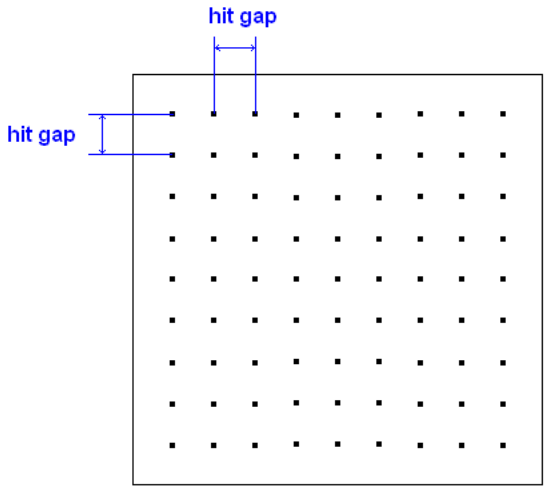
Έγιναν οι ακόλουθες επεκτάσεις – τροποποιήσεις στο εργαλείο:

Προστέθηκε η δυνατότητα για απεικόνιση πληροφοριών (attributes) για πολλούς κόμβους ταυτόχρονα. Ο χρήστης επιλέγει ένα μέρος της εικόνας κάνοντας δύο φορές κλικ με το middle mouse button ορίζοντας έτσι δύο σημεία. Τα δύο σημεία που ορίζει ο χρήστης καθορίζουν ένα ορθογώνιο (Σχήμα Α.1).



ΣχήμαΑ.1. Μαύρο: κόμβοι. Κόκκινο: τα δύο διαδοχικά κλικ του χρήστη και το ορθογώνιο που καθορίζουν.

Τα attributes για όλους τους κόμβους που περιέχονται στο ορθογώνιο αυτό απεικονίζονται. Αυτό γίνεται χρησιμοποιώντας τη λειτουργία που ήδη περιέχει το walrus για επιλογή ενός κόμβου. Μέσα στο ορθογώνιο που καθορίζει ο χρήστης καλείται για κάθε σημείο(pixel) η συνάρτηση picking του walrus και έτσι εμφανίζονται όλα τα attributes. Η λειτουργία αυτή δεν χρειάζεται να εκτελεστεί για κάθε pixel ειδικά όταν οι κόμβοι έχουν αρκετή απόσταση μεταξύ τους. Γ' αυτό τον σκοπό ο χρήστης μπορεί να καθορίσει μια ποσότητα(hit gap) ώστε να μην γίνονται αχρείαστοι υπολογισμοί (σχήμα Α.2).



Σχήμα Α.2. Οι μαύρες τελείες είναι τα σημεία για το οποία θα εκτελεστεί ο αλγόριθμος picking.

Επίσης προστέθηκε η δυνατότητα να υπολογίζονται οι affectees για ένα κόμβο. Αυτό γίνεται ως εξής:

- Ο χρήστης κάνει right click στον κόμβο που επιθυμεί.
- Επιλέγει από το μενού την επιλογή **find older relatives**

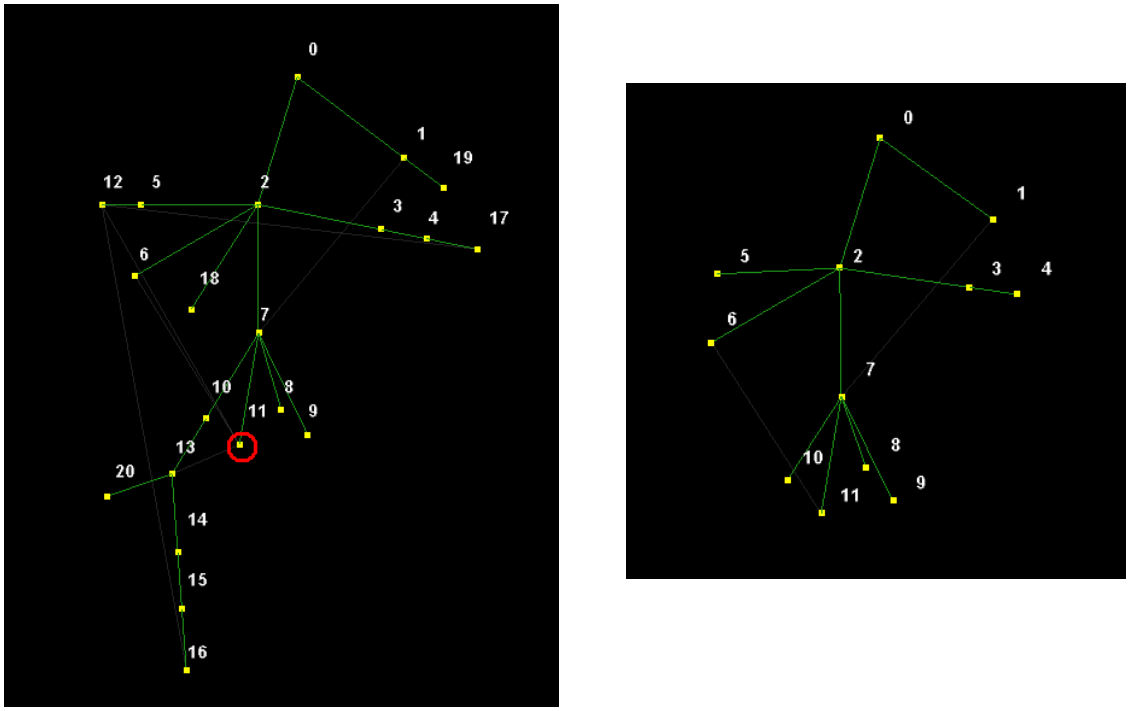
Σε ψευδοκώδικα η λειτουργία έχει ως εξής:

```
global int selected_node = getUsersClick();
hideAllNodesAndLinks();
affectees(selected_node);
```

Όπου:

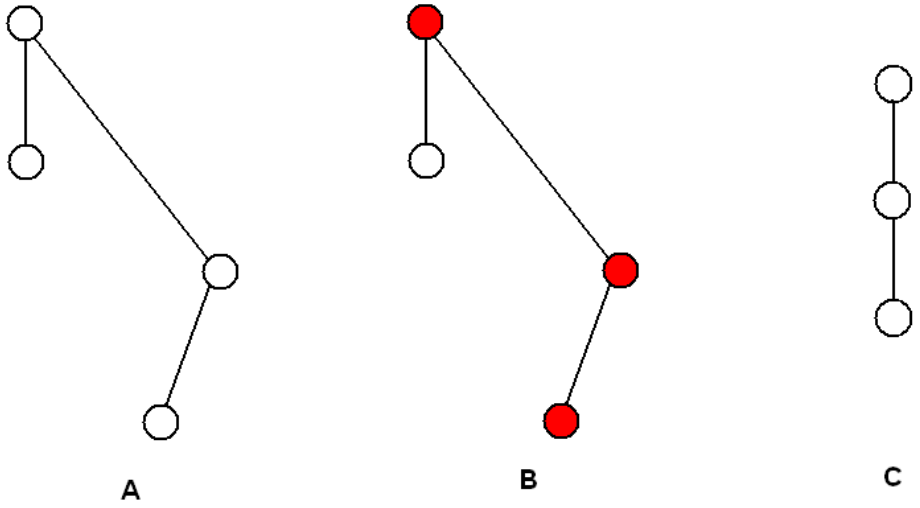
```
affectees(int a){
        if(a.value < selected_node){
                makeNodeVisible(a);
                for(each child i) : affectees(i);
```

```
        }
        affectees(a.parent1); affectees(a.parent2);
}
```



Σχήμα Α.3. Αριστερά: Ο κόκκινος κύκλος δείχνει την επιλογή του χρήστη. Δεξιά: Αποτέλεσμα.

Τέλος προστέθηκε δυνατότητα να επανεμφανίζεται κάποιο υπο-μέρος του γράφου. Αυτό είναι χρήσιμο όταν θέλουμε να εξετάσουμε κάποιο συγκεκριμένο μέρος του γράφου (π.χ τους affectees για κάποιο κόμβο). Η λειτουργία αυτή είναι χρήσιμη για να εξομαλύνει καταστάσεις όπως στο σχήμα Α.4.



Σχήμα Α.4. Α: Αρχικός γράφος. Β: Με κόκκινο οι κόμβοι που επιλέγει ο χρήστης. C: Αποτέλεσμα της λειτουργίας. Αν ο υπό-γράφος που θέλουμε να απεικονίσουμε είναι πολύπλοκος, η λειτουργία αυτή βοηθά στο να μετατρέπει σε πιο κατανοητή μορφή.

Παράρτημα Β

Στο παράρτημα αυτό παρουσιάζουμε την υλοποίηση των λειτουργίες που περιγράψαμε στο κεφάλαιο 3. Γενικά η υλοποίηση σε scripts είναι αρκετά ευκολότερη από ότι η υλοποίηση σε μια γλώσσα γενικού σκοπού όπως η C και η Java, ειδικά για προγράμματα που επικεντρώνονται σε text processing αρχείων. Δυστυχώς όμως οι υλοποιήσεις αυτές είναι πολύ απαιτητικές σε πόρους (ειδικά σε μνήμη) και γι αυτό δεν συστήνονται για επεξεργασία πολύ μεγάλου όγκου δεδομένων.

Β.1 Λειτουργία compact

```
#!/bin/bash

gawk 'BEGIN{
            c=0;
        }
        {
            if($1 in indexx)
            {
                  k=indexx[$1];
            }
            else
            {
                  indexx[$1]=c;
                  k=c;
                  c=c+1;
            }
            pats[k]++;

            if($2==1)taken[k]++;

            if($3==1)correct_pred[k]++;

        }
        END{
            for(pat in indexx)
            {
                  k=indexx[pat];
                  if(taken[k]==NULL)taken[k]=0;
                  if(correct_pred[k]==NULL)correct_pred[k]=0;

                  print pat, pats[k], taken[k], pats[k]-taken[k],
                  correct_pred[k], pats[k]-correct_pred[k];
            }
        }' $1
```

B.2 Λειτουργία find-history-patterns

```
#!/bin/bash
gawk '{
            for(i=1 ; i<=128 ; i=i*2)
            {
                  patern=substr($1,257-i);
                  patterns[patern]=patterns[patern]+$2;
                  correct[patern]=correct[patern]+$5;
                  taken[patern]=taken[patern]+$3;
            }
      }

      END {
                  for(pat in patterns)
                  {
                        if(taken[pat]==NULL) taken[pat]=0;
                        ntaken = patterns[pat]-taken[pat];

                        if(patterns[pat]>0)
                        {
                              if(taken[pat]>ntaken) entropy=taken[pat];
                              else entropy=ntaken;
                              entropy=entropy/patterns[pat];

                              print length(pat), pat, patterns[pat],
                              taken[pat] ,ntaken,
                              correct[pat]/patterns[pat],entropy;
                        }
                  }

            }' $1 | sort -n
```

B.3 Φιλτράρισμα των patterns

```
#!/bin/bash
gawk '{
                  if(length($2)>1)
                  {
                        parent=substr($2,length($2)/2+1);
                        if(parent in banlist)
                        {
                              banlist[$2]=1;
                        }
                        else
                        {
                              if(($6 > 0.995)||($3<10))
                              {
```

```
                                    banlist[$2]=1;
                                    print $1,$2,$3,$4,$5,$6,$7;
                              }
                              else
                              {
                                    print $1,$2,$3,$4,$5,$6,$7;
                              }
                        }
                  }
                  else
                  {
                        if(($6 > 0.995)||($3<10))
                        {
                              banlist[$2]=1;
                              print $1,$2,$3,$4,$5,$6,$7;
                        }
                        else
                        {
                              print $1,$2,$3,$4,$5,$6,$7;
                        }
                  }
        }' $1
```

B.4 Εξαγωγή αποτελεσμάτων από τα history patterns

```
#!/bin/bash
for ((i=1;i<=256;i=i*2)); do
echo -n $i
echo -n " "
gawk -v x=$i '{
                        if($1>x)
                        {
                              exit;
                        }
                        else
                        {
                              print $1,$2,$3,$4,$5,$6,$7;
                        }
                  }' $1 | ./leaf-patterns | ./acc-entr-averages
done

#./leaf-patterns:

#!/bin/bash
gawk 'BEGIN{
            i=0;
       }

       {
            paterns[$2]=i;
            freq[i]=$3;
            tk[i]=$4;
            nt[i]=$5;
```

```
            ac[i]=$6;
            en[i]=$7;

            if(length($2)!=1)
            {
                  parent=substr($2,length($2)/2+1);
                  parents[parent]=parent;
            }
            i++;
      }
      END{

            for(p in paterns)
            {
                  if(p in parents)
                  {
                        #ignore
                  }
                  else
                  {
                        print length(p), p, freq[paterns[p]],
                        tk[paterns[p]], nt[paterns[p]], ac[paterns[p]],
                        en[paterns[p]];
                  }
            }

      }'   $1  | sort -n


#./acc-entr-averages:

#!/bin/bash

gawk 'BEGIN{ enum=1; }
       {
                  num=num+$3;
                  acc=acc+($6 * $3);
                  if($3>1)
                  {
                        ent=ent+($7 * $3);
                        enum=enum+$3;
                  }
       }

       END{
                        print acc/num, ent/enum;
       }' $1
```