Ατομική Διπλωματική Εργασία

Κυριάκος Κυριάκου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2018

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

EXPLOITATION WRONG PATH MISPREDICTIONS FOR IMPROVING PERFORMANCE

Κυριάκος Κυριάκου

Επιβλέπων Καθηγητής
Γιάννος Σαζείδης

Μάιος 2018

Ευχαριστίες

Θα ήθελα καταρχάς να ευχαριστήσω τον επιβλέπων καθηγητή μου Δρ. Γιαννο Σαζείδη, για την βοήθεια και την στήριξη που μου έδειξε όλους αυτούς τους μήνες. Με έβαλε στο ορθό τρόπο σκέψης και τον τρόπο που πρέπει να αντιμετωπίζω τα θέματα σε επαγγελματικό επίπεδο. Τέλος θα ήθελα να τον ευχαριστίσω για το πολύ ενδιαφέρον θέμα που μου ανάθεσε, το οποίο με βοήθησε να μάθω πολλά περισσότερα στον τομέα της Αρχιτεκτονικής Υπολογιστών.

Επίσης θα ήθελα να ευχαριστίσω όλόκληρη την ερευνητική ομάδα του xi lab για την βοήθεια που μου πρόσφεραν για να κατανοήσω όλα τα απαραίτητα εργαλεία τα οποία χρησιμοποίησα.

Περίληψη

Οι επεξεργαστές σήμερα είναι ένα από τα πιο σημαντικά συστατικά ενός μοντερνοϋπολογιστικού συστήματος. Η επίδοση των επεξεργαστών είναι ένας βαρυσήμαντος παράγοντας όσον αφορά τον σχεδιασμό και στην αγορά των υπολογιστικών συστημάτων.

Οι μηχανισμοί πρόβλεψης (predictors) αποτελούν αναπόσπαστο μέρος των σημερινών επεξεργαστών. Η συνεχής ύπαρξη πολλών εντολών διακλάδωσης στα προγράμματα καθιστούν τους μηχανισμούς πρόβλεψης τεράστιας σημασίας για την απόδοση των σύγχρονων επεξεργαστών. Χωρίς την χρήση τους, η σωστή χρήση του επεξεργαστή καθυστερεί(stalls), περιμένοντας την σωστή ακολουθία εντολών για εκτέλεση. Αντίθετα όμως με την χρήση των predictors το CPU προχωρά στην εκτέλεση του προκαθορισμένου μονοπατιού (από την πρόβλεψη του predictor) αποφεύγοντας την καθυστέρηση. Σε αυτή την φάση της εξέλιξης παρουσιάστηκε ένα άλλο πρόβλημα το οποίο ορίζεται ως mispredicts(λανθασμένες προβλέψεις) . Όσο περίπλοκος και ακριβής μπορεί να σχεδιαστεί ένας predictor δυστυχώς δεν υπάρχει η δυνατότητα για να προβλέψει κάποιος το μέλλον (προς το παρών) το οποίο προϋποθέτει σαν λογικό επακόλουθο την ύπαρξη λανθασμένων προβλέψεων. Η επίδοση μπορεί να βελτιώθει με την αύξηση της ακρίβειας πρόβλεψης ή την μείωση του κόστους των λανθασμένων προβλέψεων.

Στην παρούσα διπλωματική εργασία θα παρουσιαστεί μία τεχνική που θα μπορούσε να συνεισφέρει στην αύξηση της επίδοσης των μηχανισμών πρόβλεψης που εκμεταλευέται την διαφορά στην στατιστική συμπεριφορά μιας εντολής διακλάδωσης την ώρα που εκλείται σε σχέση με την ώρα που ειναι γνώστο οτι ανήκει στη σωστή ροή ενος προγράμματος.

Περιεχόμενα

Κεφάλαιο 1 - Εισαγωγή

1.1 Κίνητρο Διπλωματικής Εργασίας

Στις μέρες μας, η ποιότητα και ο τρόπος ζωής των περισσότερων ανθρώπων εξαρτάται σε ένα πλήθος υπολογιστικών συστημάτων, είτε αυτό είναι μια κινητή συσκευή είτε ένας ηλεκτρονικός υπολογιστής είτε οποιαδήποτε συσκευή που παρέχει κάποια υπολογιστική δύναμη. Βασικό συστατικό κάποιου μοντερνοϋπολογιστικού συστήματος είναι οι επεξεργαστές (CPU) οι οποίοι υπάρχουν και χρησιμοποιούνται παντού. Στόχος όλων όσων ασχολούνται με την αρχιτεκτονική υπολογιστών είναι η βελτίωση είτε της επίδοσης είτε της ενέργειας που χρειάζονται οι επεξεργαστές για να εκτελούν τις διάφορες εφαρμογές που υπάρχουν.

Οι μηχανισμοί πρόβλεψης (predictors) αποτελούν αναπόσπαστο μέρος των σημερινών επεξεργαστών. Στα πλείστα προγράμματα που εκτελούνται υπάρχει ένα μεγάλο ποσοστό από εντολές οι οποίες είναι εντολές διακλάδωσης. Με την χρήση των predictors μπορούμε να προβλέψουμε κατά πόσο η συνθήκη είναι TRUE ή FALSE πριν ακόμα ο επεξεργαστής την υπολογίσει. Αναπόφευκτη συνέπεια της χρήσης των predictor είναι ότι υπάρχουν λάθος προβλέψεις συνεπώς και παρουσιάζονται και εκτελούνται εντολές οι οποίες δεν βρίσκονται στην ορθή ροή του προγράμματος. Αυτό το φαινόμενο ονομάζεται mispredict και wrong path execution. Εκτελώντας αχρείαστες εντολές στο λάθος μονοπάτι συνεπάγεται κόστος στην απόδοση του επεξεργαστή.

1.2 Ιδέα της Διπλωματικής Εργασίας

Στη συγκεκριμένη μελέτη παρουσιάζεται μια ιδέα- τεχνική με την οποία θα μπορούσα να βελτιώσω την απόδοση των επεξεργαστών εκμεταλλευόμενος μια συγκεκριμένη συμπεριφορά των εντολών διακλάδωσης κατά την εκτέλεση τους.

Συγκεκριμένα ο στόχος της πτυχιακήςμου είναι να μπορέσω να προσδιορίσω κατα ποσο αν μια εντολή διακλάδωσης την στιγμή που εκτελείται βρίσκεται στο λάθος μονοπάτι, πριν ανακαλύψω τηνεντολή διακλάδωσης που ειναι στην σωστλη ροή ελέγχου αλλά προκάλεσε το λάθος μονοπάτι εξαιτίας λάθος πρόβλεψης της. Πιο συγκεκριμένα για να μπορέσω να κάνω αυτή την υπόθεση, χρησιμοποιώ τα Mispredicts σαν Wrong Path Event [5]. Ουσιαστικά υπόθέτω ότι υπάρχουν περιπτώσεις εντολών διακλάδωσης ππου οταν ανήκουν στο σωστό μονοπάτι –δηλαδή στο τέλος θαγίνουν commit - προκαλούν πολυ πιο λίγα mispredicts, σε σχέση με οταν βρισκονταί στο λάθος μονοπάτι. Αυτότο Wrong Path Event, το ονομάσαμε Wrong Path Mispredicts (WPM). Έτσι θα μπορούσα να ξέρω ότι όταν εντοπίσω κάποιο WPM οτιβρίσκομαι στο λάθος μονοπάτι, το οποίο προκάλεσε μία προηγούμενη εντολή (λογο του οτι ητάν mispredicted) και η οποία ακόμα δεν ολοκλήρωσε την εκτέλεση της. Με αυτό το τρόπο θα μπορούσα να σταματήσω άμεσα την εκτέλεση εντολών που βρίσκονται στο λάθος μονοπάτι και διορθονώντας την λάθος προβλέψη της εντολής που δημιουργήσε το λαθος μονοπάτι. Ετσί θα μειώσουμε το κοστος των mispredicts και θα βελτίωθεί η απόδοση. Τα κυριότερα συμπεράσματα της μελέτης αυτής είναι ότι παρόλο που ο αριθμός των εντολών που παρουσιάζουν αυτή την συμπεριφορά είναι περιορισμένος, σε benchmarks τα οποία έχουν τέτοιες εντολές παρατηρούμε άυξηση στην απόδοση του επεξεργαστή μέχρι και 1%.

1.3 Δομή Διπλωματικής Εργασίας

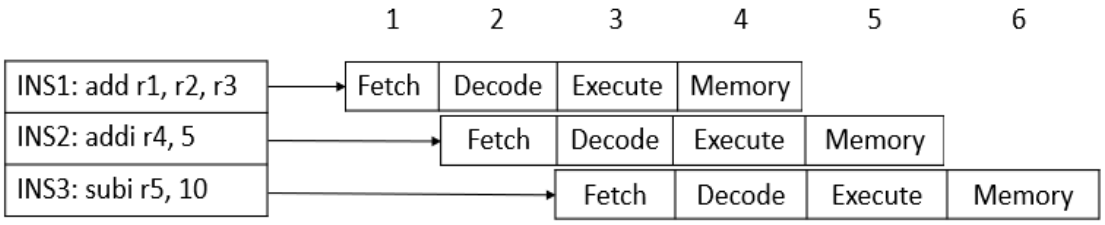
Στην συνέχεια της διπλωματικής μου (Κεφ. 2) θα σας παρουσιάσω και θα εξηγήσω πιο αναλυτικά βασικές έννοιες οι οποίες είναι απαραίτητες για την κατανοήση και την εξαγωγή των συμπερασμάτων μου.

Στο Κεφάλαιο 3 γίνεται μία σύντομη περιγραφή για το τι ειναι τα Wrong Path Event (WPE). Στο Κεφάλαίο 4 περιγράφεται πιο αναλυτικά το φαινόμενο WPM, δηλαδή την χρήση των mispredicts σαν (WPE). Στη συνέχεια, στο Κεφάλαίο 5 θα παρουσιαστεί μια ανάλυση για την συχνότητα των εντολών οι οποιες παρουσιάζουν την WPM συμπεριφορά. Επίσης γινεται ανάλυση με την χρήση ενος αναλυτικού μοντέλου το πόσο θα μπορούσε αυτή η ιδέα να βελτιώσει την απόδοση του επεξεργαστή. Στο τελευταίο κεφάλαιο γινεται συντομή ανασκόπησηση της μελέτης και δίνεται κατεύθυνση για μελλοντική δουλειά.

Κεφάλαιο 2 – Background

2.1 Pipeline

Στους πρώτους επεξεργαστές που δημιουργηθήκαν οι εντολές εκτελούνταν σειριακά. Δηλαδή για να ξεκινήσει την διαδικασία εκτέλεσης της κάποια εντολή θα έπρεπε να περιμένει όλες τις εντολές οι οποίες είναι πιο ‘παλιές’ να ολοκληρώσουν την διαδικασία εκτέλεσης τους για να μπορέσει να χρησιμοποιήσει η εντολή αυτή με τη σειρά της, τους πόρους του επεξεργαστή. Εξαιτίας αυτού παρουσιάζονταν πάρα πολλές καθυστερήσεις και εκτέλεση καθυστερούσε. Για αυτό το λόγο, οι πιο σύγχρονοι επεξεργαστές για να μπορούν να εκτελούν τις διάφορες εντολές με αποδοτικότερο τρόπο χρησιμοποιούν τον όρο της Διασωλήνωσης (Pipeline). Η χρήση της διασωλήνωσης στους επεξεργαστές βασίζεται στην ιδέα ότι θα μπορούσαν να εκτελούνται πολλές εντολές παράλληλα αφού δεν ήταν αποδοτικό κάθε εντολή να περιμένει την προηγούμενη εντολή να ολοκληρώσει την εκτέλεση της, για να μπορέσει να εκτελεστεί. Με αυτή την τεχνική η επεξεργασία κάποιας εντολής χωρίζεται σε διάφορα στάδια, (Σχ. 2.1) και όταν μια πιο παλιά εντολή ολοκλήρωσει την εκτέλεση της στο πρώτο στάδιο της επεξεργασίας και προχωρίσει στο δεύτερο, η επόμενη εντολή χρησιμοποιεί τους πόρους του πρώτου σταδίου επεξεργασίας.



Σχ. 2.1 Φαίνεται η δομή της απλής μορφής μίας διασωλήνωσης με τα 4 βασικά στάδια.

Με αυτή την τεχνική επιτυγχάνεται μεγάλη βελτίωση στην επίδοση του CPU, καθώς σε 1 κύκλο ρολογιού θα μπορούσε να προχωρήσουν στην εκτέλεση τους περισσότερες από μία εντολή.

2.2 Data and Control Hazards

Καταλαμβαίνουμε ότι με την χρήση του όρου pipeline που περιγράψαμε πετυγχάνεται σημαντική αύξηση στην επίδοση του επεξεργαστή εξαιτίας της αύξησης της παραλληλία στην εκτέλεση των εντολών. Αυξάνοντας όμως την παράλληλη εκτέλεση εντολών προέκυψαν κάποια προβλήματα [1]. Οι πλείστες εντολές σε ένα πρόγραμμα αλληλοεπιδρούν μεταξύ τους, δηλαδή υπάρχουν εξαρτήσεις μεταξύ των εντολών αυτών. Για να εντιμετωπιστούν οι εξαρτήσεις αυτές, παρουσιάζεται μια καθυστέρηση στην ροή της διασωλήνωσης (γνωστά σαν Stalls) κατά την οποία το pipeline παραμένει εντελώς αδρανές περιμένωντας μια εντολή να ολοκληρώσει την εκτελέση της έτσι ώστε οι εντολές που εξαρτώνται από αυτή να έχουν τα δεδομένα για να μπορούν να συνεχίσουν και αυτές την εκτέλεση τους. Θα δούμε δύο κατηγορίες των προβλημάτων – κυνδίνων που μπορούν να προκύψουν με την χρήση του pipeline [1].

2.2.1 Data Hazards

Data Hazards είναι ουσιαστικά αυτό που περιγράψαμε πιο πάνω, δηλαδή μια εντολή είναι εξαρτημένη από τα αποτελέσματα κάποιας πιο παλιάς εντολής. Σε αυτή την περίπτωση μπορούν να παρουσιάστούν τρία είδη Data Hazards.

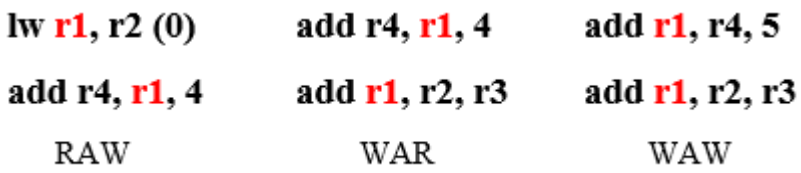
- RAW (Read After Write)
- WAR (Write After Read)
- WAW (Write After Write)

Με λίγα λόγια η περίπτωση του RAW είναι η αυτή που εμφανίζεται πιο συχνά στην διασωλήνωση. Δηλάδη κάποια εντολή A θέλει να διαβάσει κάτι από κάποιο καταχωρητή, και κάποια εντολή B παλαιότερη της A να γράψει στο καταχωρητή αυτό. Αυτό σημαίνει ότι η εντολή A δεν μπορεί να εκτελεστεί αν δεν ολοκληρώσει την εκτέλεση της η B, και ενημερώσει τον συγκεκριμένο καταχωρητή.

Η περίπτωση του WAR είναι αυτή κατά την οποία κάποια εντολη A θέλει να ενημερώσει κάποιο καταχωρητή, αλλά μια παλαιότερη εντολή B θέλει να διαβάσει από τον καταψωρητή αυτό. Αν η A ενημερώσει τον καταχωρητή πριν την εκτέλεση την B, αυτό

σημαίνεί ότι όταν η Β διαβάσει την τιμή του καταχώρητη θα διαβάσει λάθος τιμή, επηρεάζοντας την ορθότητα του προγράμματος.

Τέλος στην περίπτωση του WAW μια εντολή Α θέλεί να γράψει σε κάποιο καταχωρητή αλλά μία πιο παλιά εντολή Β θέλει και αυτή να γράψει στον καταχωρητή αυτό. Αν εκτελέστεί η Α πριν τη Β πάλι θα έχουμε αλλοίωση των δεδομένων με συνέπεια πάλι την ορθότητα του προγράμματος. Στο (Σχ. 2.2) φαίνονται παραδείγματα και για τις τρεις περιπτώσεις Data Hazards.



Σχ. 2.2 Φαίνονται τα τρία ήδη των data hazards
και πως οι συγκεκριμένες εντολές επηρεάζουν
τον καταχωρητή r1 σε κάθε περίπτωση.

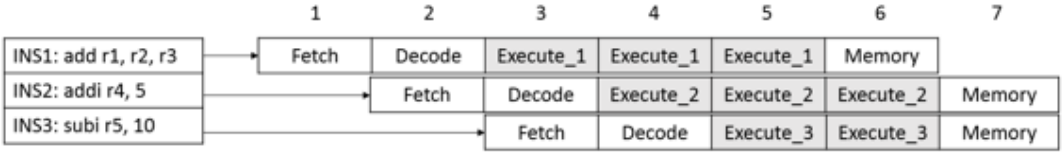
2.2.2 Control Hazards

Το άλλο είδος προβλήματος που παρουσιάζεται με την χρήση της διασωλήνωσης το οποίο έχει και περισσότερη σχέση με την διπλωματική μου εργασία είναι τα Control Hazards. Καταλαβαίνουμε ότι η δομή των πλείστων προγραμμάτων περιέχει πάρα πολλές εντολές διακλάδωσης. Όταν μία εντολή εισέλθει στο pipeline, υπάρχουν δύο επιλογές, είτε η εντολή αυτή είναι taken – δηλαδή η επόμενη εντολή που θα εκτελεστεί θα είναι στο target της εντολής διακλάδωσης - είτε θα είναι no-taken – δηλαδή η επόμενη εντολή θα είναι το PC counter της εντολής διακλάδωσης + 4 (ανάλογα με την αρχιτεκτονική του επεξεργαστή). Το pipeline όμως δεν θα μπορούσε να έχει πιο νέες εντολές από την συγκεκριμένη εντολή διακλάδωσης για το λόγο ότι δεν μπορεί να γνωρίζει για ποια από τις δύο κατευθύνσεις θα πρέπει να φορτώσει τις εντολές. Αυτό σημαίνει ότι το pipeline βρίσκεται σε αδράνεια όσους κύκλους χρειάζεται για να ολοκληρώσει την εκτέλεσή της κάποια εντολή διακλάδωσης. Εμείς θα θέλαμε με κάποιο τρόπο αυτούς τους κύκλους ρολογιού στους οποίους υπάρχουν stalls στο pipeline και οι πόροι του είναι ελεύθεροι να μπορούσαμε να τους αξιοποιήσουμε. Παρακάτω εξηγείτε ο όρος Branch Predictor με τον οποίο καταφέραμε να αντιμετωπίσουμε αυτή την καθυστέρηση.

2.3 Out of Order Pipeline - Superscalar Processor

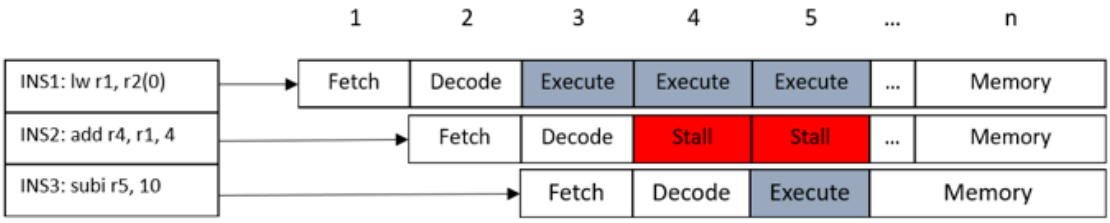
Στην συνέχεια εξαιτίας της αύξησης στην πολυπλοκότητα και των κίνδυνων δεδομένων που αναφέραμε πιο πάνω, οι οποίοι επηρέαζαν την επίδοση των επεξεργαστών υπήρχε ανάγκη για ακόμα μεγαλύτερη αύξηση στην επίδοση. Προσπάθησαν να αυξήσουν την παραλληλία στη εκτέλεση των εντολών που βρίσκονταν στο pipeline χρησιμοποιώντας περισσότερους πόρους εκτέλεσης. Έτσι στο στάδιο της εκτέλεσης – το οποίο χρειαζόταν και το μεγαλύτερο αριθμό κύκλων για να ολοκληρωθεί αλλά προκαλούσε και την μεγαλύτερη καθυστέρηση λόγω των εξαρτήσεων που υπήρχαν μεταξύ των εντολών – αυξήθηκαν οι υπολογιστικές μονάδες και διαχωρίστηκαν σε σχέση με το είδος της πράξης που χρειαζόταν μια εντολή για να εκτελεστεί. Για παράδειγμα αν κάποιες εντολές οι οποίες ήταν ανεξάρτητες από κάποιες άλλες θα μπορούσαν να εκτελεστούν πριν από αυτές αν υπήρχαν οι διαθέσιμοι πόροι. Δηλαδή αλλοιωνόταν η σειρά εκτέλεσης αλλά με την ολοκλήρωση της εκτέλεσης του προγράμματος, οι εντολές φαίνονται στη διαπροσωπία του χρήστη ότι εκτελέστηκαν με την σωστή σειρά.

Η τεχνική αυτή ονομάστηκε Out of Order Pipeline (Σχ. 2.3). Αυτή είναι μια ιδέα με την οποία κατάφεραν σε πολύ μεγάλο βαθμό να μειώσουν τον αριθμό των stalls και την



Σχ. 2.3 Παρατηρούμε ότι οι εντολές INS1, INS2 και INS3 βρίσκονται ταυτόχρονα στο στάδιο της εκτέλεσης, εφόσον ο επεξεργαστής περιέχει τουλάχιστον τρεις υπολογιστικές μονάδες πρόσθεσης.

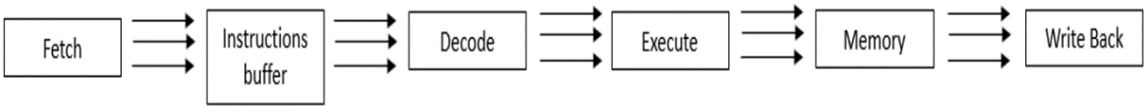
καθυστέρηση που πρόκυπτε από κάποιο data hazard (Σχ2.4). Από το σχήμα φαίνεται διαδικασία εκτέλεση για κάποιο RAW hazard. Η εντολή INS2 καθυστερεί αφού υπάρχει εξάρτηση και περιμένει την εντολή INS1 να ολοκληρωθεί, αλλά βλέπουμε ότι η εντολή INS3 είδη είναι στην φάση της εκτέλεσης αφού ο επεξεργαστής της παρέχει τους απαραίτητους υπολογιστικούς πόρους για την εκτέλεση της και είναι ανεξάρτητη σε σχέση με τις υπόλοιπες εντολές που βρίσκονται εκείνη την στιγμή στην διασωλήνωση.



Σχ2.4 Η εντολής INS2 εξαρτάται από την εντολή INS1 και δεν μπορεί να συνεχίσει την εκτέλεση της πριν την ολοκλήρωση της INS1. Η INS3 όμως μπορεί να συνεχίσει την διαδικασία και να ολοκληρώσει την εκτέλεση της αφού είναι ανεξάρτητη και υπάρχουν διαθέσιμοι πόροι.

Η χρήση του Out of Order pipeline βρίσκεται σε όλους τους σύγχρονους επεξεργαστές αφού κατάφερε να αντιμετωπίσει τα data hazards και να αυξήσει σημαντικά την απόδοση τον επεξεργαστών αυξάνοντας την παραλληλία της εκτέλεσης των διαφόρων εντολών κάποιου προγράμματος.

Τελευταίο στάδιο στην διαμόρφωση της δομής διασωλήνωσης ήταν οι Superscalar επεξεργαστές [1]. Στη συγκεκριμένη έκδοση μας δίνεται η δυνατότητα να φορτώνουμε περισσότερες από μία εντολή στο pipeline του επεξεργαστή στον ίδιο κύκλο. Δηλαδή κατά την επεξεργασία των εντολών στα στάδια του Fetch και Decode υπήρχαν περισσότερες από μία εντολή (Σχ. 2.5). Αυξάνουν το επίπεδο παραλληλίας στην Κεντρική Μονάδα Επεξεργασίας σε σχέση με τις προηγούμενες περιπτώσεις στις οποίες μπορούσαν να φορτώσουν μόνο μία εντολή στο pipeline κάθε κύκλο. Αυτό έχει σαν αποτέλεσμα την εκ νέου βελτίωση της απόδοσης των επεξεργαστών. Αυτή η δομή στη διασωλήνωση είναι παρόμοια με την δομή που έχουν ο σύγχρονοι επεξεργαστές που βρίσκονται στην αγορά.



Σχ. 2.5 Φορτώνονται περισσότερες από μία εντολές σε ένα κύκλο στο pipeline

2.4 Branch Predictor and Speculative execution

Με τον όρο Branch Predictor (Μηχανισμός Πρόβλεψης) [2] [3] επιστημονικά στον τομέα της αρχιτεκτονικής υπολογιστών εννοούμε την όλη διαδικασία που κρύβεται πίσω από τον υπολογισμό του αποτελέσματος μιας πράξης, η οποία μας καθορίζει την ροή του προγράμματος.

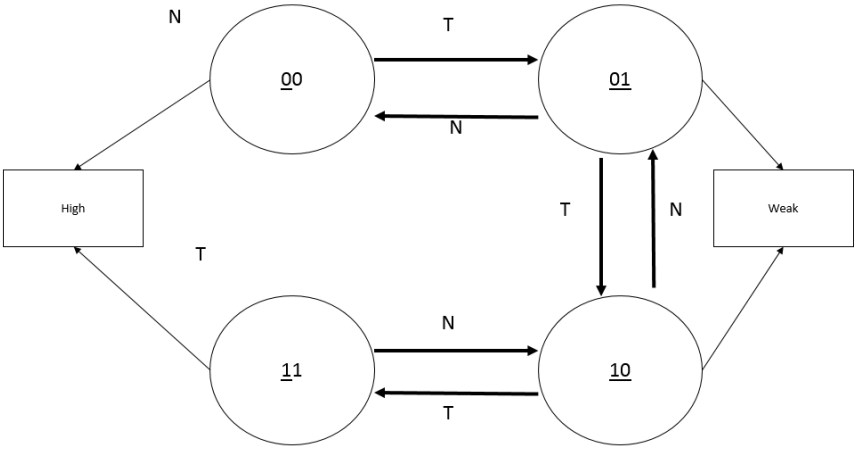
Σκοπός του πιο πάνω είναι να κερδίζουμε χρόνο στην εκτέλεση εντολών αφού με τους Superscalar επεξεργαστές που υπάρχουν σήμερα οι εντολές εκτελούνται σε διαφορετική σειρά από την σειρά τους στο εκτελέσιμο αρχείο και έχουμε πολύ ψηλά επίπεδα παράλληλης επεξεργασίας εντολών. Με το branch prediction θέλουμε να υπολογίσουμε την ροή του προγράμματος με όσο το δυνατό περισσότερη ευστοχία έτσι ώστε να μπορούμε να συνεχίσουμε την εκτέλεση εντολών ακόμα και αν δεν γνωριζούμε την σωστή κατάσταση της ροής του προγράμματος. Η εκτέλεση κάποιου αριθμού εντολών οι οποίες εκτελούνται βασισμένες σε μια υπόθεση ονομάζεται Speculative execution. Αυτό βοηθά την βελτείωση του επεξεργαστή καθώς μπορούμε να εκμεταλλευτούμε τους πόρους του επεξεργαστή στο μέγιστο δυνατό και όσο περισσότερο εκμεταλλευόμαστε αυτούς τους πόρους μειώνοντας τις καθυστερήσεις και τα stalls, τόσο πιο γρήγορα εκτελούνται οι εντολές που φθάνουν στον επεξεργαστή, και ως αποτέλεσμα αυτού αυξάνουμε την συνολική απόδοση του επεξεργαστή. Αυτή η ιδέα προτάθηκε για να μπορέσουν να αντιμετωπιστούν τα Control Hazard που περιγράψαμε πιο πάνω. Εάν στην διασωλήνωση κάποιου επεξεργαστή όποτε εμφανιζόταν μία εντολή branch περίμενε την εκτέλεση του για να ξέρει ποιες εντολές θα εκτελέσει μετά, τότε τα προγράμματα μας σήμερα θα ήταν πολύ πιο αργά καθώς οι πόροι του επεξεργαστή δεν θα χρησιμοποιούνταν από καμιά εντολή και η διασωλήνωση θα ήταν σε αδράνεια. Ο τομέας των branch predictors είναι ένας τομέας που τα τελευταία χρόνια απασχολεί ιδιαίτερα τον τομέα της αρχιτεκτονικής υπολογιστών και γενικότερα της πληροφορικής αφού είναι ένας σημαντικός παράγοντας για την βελτίωση της απόδοσης των επεξεργαστών.

Η βασική ιδέα πίσω από τους predictors είναι ότι κρατάμε διάφορους μετρητές για κάθε εντολή διακλάδωσης. Ανάλογα με την τεχνική που είναι υλοποιημένος ο predictor μπορεί κάποιοι μετρητές να αφορούν και ένα υποσύνολο από εντολές. Έτσι κάποια εντολή αντιστοιχείται σε κάποιο μετρητή, αν το πιο αριστερό bit του μετρητή (most significant bit) είναι 0 τότε η πρόβλεψη είναι no taken διαφορετικά είναι taken. Στις περισσότερες περιπτώσεις για να γίνει μια πρόβλεψη λαμβάνεται υπόψη κάποιο ιστορικό με την συμπεριφορά της ίδιας της εντολής ή ακόμα και την συμπεριφορά κάποιου συνόλου από τις εντολές του προγράμματος η οποία γίνεται HASH με το PC της εντολής και αντιστοιχείται με αυτό τον τρόπο σε κάποιο μετρητή. Η ενημέρωση του ιστορικού αυτού γίνεται αμέσως μετά που γίνεται η πρόβλεψη. Σε περίπτωση που αποδειχτεί στην

συνέχεια ότι έγινε λάθος πρόβλεψη, γίνεται recover όλη η ιστορική πληροφορία που έχει ο predictor έτσι ώστε να αυξήσει το επίπεδο ακρίβειας στην συνέχεια της εκτέλεσης του προγράμματος.

Ένας άλλος παράγοντας που μπορεί να διαφέρει στις διάφορες τεχνικές είναι το σύνολο το μέγεθος (bits) του κάθε μετρητή. Τα πιο συνηθισμένα είναι τα 2bit και τα 3bit. Συγκεκριμένα εάν μια εντολή διακλάδωσης μετά από τον υπολογισμό του αποτελέσματος της είναι ορθή (taken - δηλαδή TRUE), τότε ο αντίστοιχος μετρητής που αντιστοιχεί στην εντολή θα αυξηθεί κατά ένα. Σε αντίθετη περίπτωση ο μετρητής μειώνεται. Έτσι η τιμή του κάθε μετρητή θα βοηθήσει τον predictor να υπολογίσει με όσο το δυνατό μεγαλύτερη ακρίβεια το αποτέλεσμα αυτής της εντολής για τις επόμενες εκτελέσεις της.

Στο (Σχ. 2.6) μπορούμε να δούμε την γραφική απεικόνιση της όλης ιδέας που κρύβεται πίσω από τον όρο του μετρητή σε ένα branch prediction. Αν ο μετρητής κατηγοριοποιείτε ως high συνήθων η ακρίβεια είναι μεγαλύτερη.

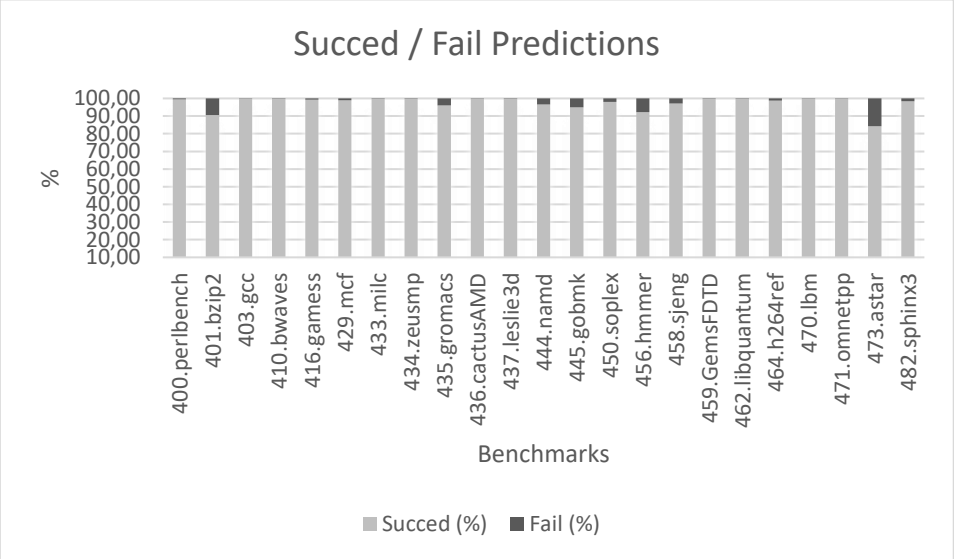


Σχ.2.6 Φαίνεται ένα δείγμα της λογικής που υπάρχει πίσω από την διαδικάσία για να εξαχθεί μία πρόβλεψη. Αν το most significant bit είναι 0 η πρόβλεψη έιναι no taken διαφορετικά είναι taken. Ανάλογα τελικά με την τελική συμπεριφορά της εντολής ενημερώνονται οι μετρητές.

Κεφάλαιο 3 – Wrong Path Events (WPE)

3.1 Theoretical Background

Όπως προαναφέραμε οι μηχανισμοί πρόβλεψης που υπάρχουν στους σύγχρονους επεξεργαστές είναι ένας από τους κύριους παράγοντες που επιτρέπει στους επεξεργαστές να έχουν ψηλή απόδοση ακόμα και σε προγράμματα και εφαρμογές τα οποία είναι δύσκολα δομημένα. Πάρα πολλοί μελετητές οι οποίοι ασχολούνται με τους μηχανισμούς αυτούς και γενικότερα με την Αρχιτεκτονική Υπολογιστών προσπαθούν συνεχώς να βελτιώσουν την ακρίβεια των προβλέψεων που γίνονται διαφοροποιώντας την δομή των predictors και τον τρόπο με τον οποίο γίνονται προβλέψεις. Με την πάροδο του χρόνου οι predictors έχουν εξελιχτεί και το ποσοστό ακρίβειας έχει αυξηθεί σε μεγάλο βαθμό. Στο (Σχ. 3.1) φαίνονται τα ποσοστά επιτυχίας στις προβλέψεις από ένα state of the art predictor - L-TAGE - μετά από κάποιες μετρήσεις που κάναμε για τα SPEC 2006.



Σχ.3.1 Φαίνονται τα ποσοστά επιτυχίας και αποτυχίας αντίστοιχα για κάθε benchmark.

Παρόλο που τα ποσοστά επιτυχίας έχουν φτάσει σε αυτά τα επίπεδα υπάρχει μεγάλος αριθμός λάθος προβλέψεων. Αυτό έχει σαν αποτέλεσμα ο επεξεργαστής να πληρώνει κάποιους κύκλους ως κόστος. Το συνολικό κόστος που πληρώνεται είναι οι κύκλοι που χρειάστηκαν από την στιγμή που η εντολή εισήλθε pipeline μέχρι την στιγμή που εντοπίστηκε η λάθος πρόβλεψη συν το χρόνο που χρειάζεται για να αδειάσει το pipeline και να φορτώσει τις εντολές που κανονικά θα φόρτωνε αν η πρόβλεψη ήταν ορθή. Σκοπός αυτής της εργασίας δεν είναι να βελτιώσουμε την ακρίβεια των προβλέψεων αλλά να μειώσουμε το κόστος που απαιτείται να πληρώσει ο επεξεργαστής σε περίπτωση που προκύψει μια λάθος πρόβλεψη. Αν μπορέσουμε με κάποιο τρόπο να καταλάβουμε ότι μια εντολή που εκτελείται βρίσκεται στο λάθος μονοπάτι πριν ακόμα η εντολή διακλάδωσης που προκάλεσε το λάθος μονοπάτι ολοκληρώσει την εκτέλεση της σημαίνει ότι το κόστος που θα πρέπει να πληρώσουμε μειώνεται και θα είναι οι κύκλοι από την ώρα που η εντολή από την οποία προήλθε το λάθος μονοπάτι μέχρι την ώρα που διαπιστώσαμε ότι εκτελούνται εντολές και βρίσκονται στο λάθος μονοπάτι συν το χρόνο που χρειάζεται για να αδειάσει το pipeline. Με αυτή την τεχνική ο επεξεργαστής γλιτώνει αρκετούς κύκλους που πληρώνει σαν κόστος.

3.2 Wrong Path Execution

Κατά την διάρκεια της εκτέλεσης κάποιας εφαρμογής υπάρχουν διάφορες εντολές που περνούν από το στάδιο της εκτέλεσης της διασωλήνωσης του επεξεργαστή, οι πλείστες από αυτές στο τέλος θα γίνουν commit αλλά αρκετές εντολές από αυτές βρίσκονται στο λάθος μονοπάτι που εκτελέστηκαν εξαιτίας κάποιας λάθος πρόβλεψης που έγινε για μια εντολή διακλάδωσης. Μια συγκεκριμένη συμπεριφορά για κάποια παράμετρο η οποία θα μπορούσε να μας δείξει ότι μια εντολή βρίσκεται στο λάθος μονοπάτι, δηλαδή η συγκεκριμένη συμπεριφορά είναι πολύ σπάνια στο ορθό μονοπάτι και αρκετά συχνή στο λάθος μονοπάτι ονομάζεται **Wrong Path Event** (WPE). Υπάρχουν δύο είδη WPEs [5]. Τα Hard WPEs, όπου όταν παρατηρήσουμε μία συγκεκριμένη συμπεριφορά την οποία θεωρούμε WPE υπάρχει μεγάλη πιθανότητα όντως η εντολή να βρίσκεται λάθος μονοπάτι αλλά υπάρχει και μια πιθανότητα να προκληθεί από εντολή που βρίσκεται στο ορθό μονοπάτι. Το δεύτερο είδος WPEs είναι τα Soft WPEs, όπου παρατηρούμε κάτι και συμπαιρένουμε αμέσως ότι η εντολή βρίσκεται στο λάθος μονοπάτι. Δηλάδη είναι συμπεριφορά που ξέρουμε ότι η πιθανότητα να εμφανιστεί στο ορθό μονοπάτι είναι αρκετά σπάνια.

3.3 Related Work

Αυτό το φάσμα της στατιστικής ανάβλυσης που μπορεί να γίνει για να αποφασιστεί ποια παράμετρος η πιο στατιστικό μπορεί όντως να προκαλέσει WPE, και μας δίνει με κάποιο ποσοστό επιτυχίας ποια εντολή εκτελείται στο σωστό ή λάθος μονοπάτι είναι πολύ μεγάλο και ωθεί πολλούς ερευνητές να το μελετήσουν.

Υπάρχουν διάφοροι ερευνητές που μελέτησαν και εντόπισαν μερικά φαίνόμενα που παρουσιάζονται ως επι το πλείστον στην εκτέλεση μιας εντολής όταν αυτή βρίσκεται στο λάθος μονοπάτι. Πιο συγκεκριμένα ασχολήθηκαν και ανάλυσαν συμπεριφορές διάφορων τύπων εντολών [5].

Memory Instructions: Παρατηρήθηκε ότι μπορεί να υπάρχουν δείκτες (pointers) οι οποίοι έχουν την τιμή μηδέν. Στην περίπτωση αυτή ο επεξεργαστής αρχικοποιεί τους pointers αυτούς σε NULL. Αν εντοπιστεί κάποιος δείχτης ο οποίος είναι NULL σημαίνει ότι βρισκόμαστε σε wrong path. Επίσης παρατηρήθηκε ότι σε περιπτώσεις που μια εντολή βρίσκεται σε λάθος μονοπάτι μπορεί να χρησιμοποιήσει για δίαβασμα ή γράψημό κάποιο μέρος μνήμης το όποίο δεν επιτρέπεται. Επίσης κατάλαβαν ότι αν προκαλούνται πολλά διαδοχικά TLB (Translation Lookaside Buffer) misses υπάρχει μεγάλη πιθανότητα οι εντολές που τα προκαλέσαν να βρίσκονται στο λάθος μονοπατι.

Control Flow Instructions: Για τις εντολές διακλαδωσης παρατηρήθηκε ότι αν προκύψουν τρια διαδοχικα mispredicts από τρεις εντολές και υπάρχει στο reorder buffer εντολή διακλάδωσης η οποία ακόμα να ολοκληρώσει την εκτέλεση της, και είναι πιο πάλια από τις εντολές που προκαλέσαν τα mispredicts υπάρχει πάλι μεγάλη πιθανότητα να βρίσκομαι στο λάθος μονοπάτι. Αυτό το φαινόμενο ονομάστηκε ως “branch under branch”

Arithmetic Instructions: Τέλος παρατηρήθηκε ότι πολλές μεταβλητές που χρησιμοποιούνται για εντολές οι οποίες είναι στο λάθος μονοπάτι δεν αρχικοποιούνται, και προκύπτουν αδύνατες πράξείς όπως για παράδειγμα διαιρέσεις με το 0 ή τετραγωνικές ρίζες για αρνητικούς αριθμους.

Κατάληξαν στο συμπέρασμα οτι εντοπίζοντας αυτά τα φαινόμενα και αποκαθαστόντας άμεσα το λάθος μονοπάτι είχαν κάποια αξιοσημείωτη αυξηση στο IPC του επεξεργαστη για μερικά benchmarks. Στην περίπτωση μας επιλέξαμε κάποια άλλη συμπεριφορά και προσπάθησα να την αναλύσω για να δω αν όντως η ιδέα μας θα μπορούσε να είχε κάποια θετικά αποτελέσματα στον εντοπισμό των εντολών αυτών και πιο ουσιαστικά στην βελτίωση της επίδοσης των επεξεργαστών. Εμείς θα ασχοληθούμε όπως προανάφερα με εντολές διακλάδωσης Control Flow Instructions και το WPE που παρατηρούμρ ανήκει στην κατηγορία Hard WPEs.

Κεφάλαιο 4 – Mispredicts as Wrong Path Events

4.1 Main Idea

Η γενική ιδέα έτσι ώστε να μπορέσουμε να προβλέψουμε αν μια εντολή που εκτελείται είναι στο λάθος μονοπάτι η όχι είναι να παρατηρήσουμε τη συμπεριφορά των εντολών που εκτελούνται, τόσο στο σωστό μονοπάτι όσο και στο λάθος μονοπάτι. Με αυτό το τρόπο μπορούμε να συγκρίνουμε τα στατιστικά του επεξεργαστή και στις δύο περιπτώσεις με σκοπό να μπορέσουμε να διαπιστώσουμε αν κάποια εντολή προκαλεί κάποια ανωμαλία κατά την εκτέλεση της, για να μπορέσουμε να κάνουμε την πρόβλεψη ότι η συγκεκριμένη εντολή υπάρχει πιθανότητα να βρίσκεται στο λάθος μονοπάτι. Βέλτιστο σενάριο θα ήταν να μπορούσαμε να προβλέπουμε με απόλυτη ακρίβεια αν μια συγκεκριμένη εντολή είναι στο λάθος μονοπάτι ή όχι, αλλά αυτό θεωρητικά είναι κάτι αδύνατο, έτσι προσπαθούμε να εκμεταλλευτούμε και να κατανοήσουμε την επίδραση που έχει μια ‘κακή’ εντολή στη λειτουργεία του επεξεργαστή και στις παραμέτρους που χρησιμοποιεί η ίδια η εντολή για την εκτέλεση της για να έχουμε όσο το δυνατό μεγαλύτερη πιθανότητα η εντολή αυτή να είναι όντως στο λάθος μονοπάτι. Για αυτό το λόγο κεντρική ιδέα σχετικά με το τι θα μπορούσαμε να χρησιμοποιήσουμε εμείς σαν WPE είναι τα mispredicts που προκαλούν οι εντολές διακλάδωσης (branch).

Μια ιδανική υπόθεση που μπορούμε να κάνουμε είναι ότι οι εντολές διακλάδωσης που βρίσκονται στο ορθό μονοπάτι εκτέλεσης δεν προκαλούν ποτέ κάποιο mispredict και άρα υποθέτουμε ότι όποτε προκληθεί κάποιο mipsredict ξέρουμε ότι η συγκεκριμένη εντολή που το προκάλεσε βρίσκεται στο λάθος μονοπάτι. Όπως γνωρίζουμε όμως, αυτή δεν θα ήταν μια ρεαλιστική υπόθεση καθώς ξέρουμε ότι για κάθε εντολή branch που γίνεται commit υπάρχει η πιθανότητα να προκύψει κάποιο mispredict ενώ η εντολή αυτή βρίσκεται στο ορθό μονοπάτι. Ξέρουμε ότι για όλες τις εντολές διακλάδωσης που

βρίσκονται στη διασωλήνωση κάποιου επεξεργαστή γίνεται χρήση του εκάστοτε μηχανισμού πρόβλεψης για να ξέρει ποια ακολουθία εντολών θα εκτελεστεί μετά. Οπόταν εμείς σκεφτήκαμε να αναλύσουμε κατά πόσο οι εντολές που είναι στο ορθό μονοπάτι και θα γίνουν commit στο τέλος της εκτέλεσης τους, έχουν κάποια διαφορετική συμπεριφορά στην ακρίβεια των προβλέψεων τους σε σχέση με τις εντολές που εκτελούνται και βρίσκονται στο λάθος μονοπάτι. Με τη προϋπόθεση ότι στις δύο κατηγορίες εντολών υπάρχει μέγαλη διαφορά στο ποσοστό ακρίβειας όταν εκτελείται και όταν γίνεται commit μας δίνεται η εξής ευκαιρία.

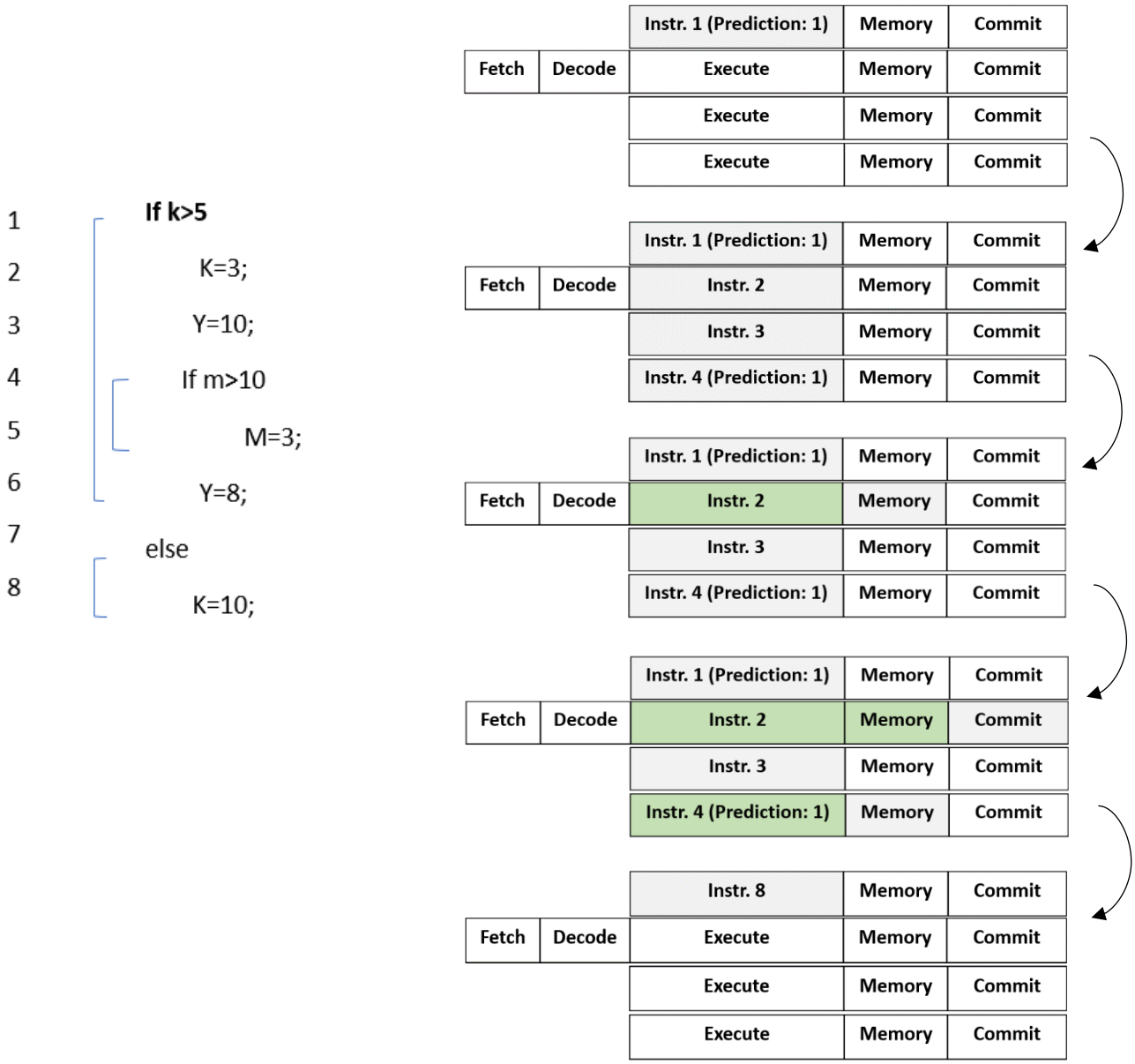
Παράδειγμα η εντολή με PC: 4832404312 απο το benchmark 400.perlbench ξέρουμε ότι έχει 99% πιθανότητα σωστής πρόβλεψης όταν γίνεται commit και 53% κατά την φάση της εκτέλεσης. Στην περίπτωση που αυτή η εντολή εκτελείται και προκαλέσει misprediction ξέρουμε ότι υπάρχει πολύ μεγάλη πιθανότητα η συγκεκριμένη εντολή να βρίσκεται στο λάθος μονοπάτι. Αν μπορέσει ο επεξεργαστής να παρατηρεί δυναμικά την στατιστική συμπεριφορά των εντολών αυτών και να μπορεί να διορθώνει το λάθος μονοπάτι πριν η εντολή που το προκάλεσε ολοκληρώσει την εκτέλεση του τότε θα μπορούσε να υπάρξει μια αξιοσημείωτη βελτίωση στη απόδοση του επεξεργαστή. Το ερώτημα που προκύπτει είναι πόσες εντολές σε κάποιο πρόγραμμα μπορεί να έχουν αυτή την θετική (προς εμάς) συμπεριφορά, τι ποσοστό των συνολικών mispredictions του προγράμματος προκαλούνται από τις συγκεκριμένες εντολές και τέλος πόσο θα μπορούσε να βελτειωθεί η απόδοση του επεξεργαστή για το συγκεκριμένο πρόγραμμα.

Στο (Σχ. 4.1) βλέπουμε ένα παράδειγμα κάποιου προγράμματος με το οποίο εξηγουμε πως θα ηταν χρήσιμη η εφαρμογη μιας τετοιας τεχνική έτσι ώστε να εξοικονομήσουμε κάποιους κύκλους απο το mispredicitn penalty. Ξέρουμε ότι η εντολή 4 του παραδείγματος είναι η εντολή στην οποία μπορούμε να εφαρμόσουμε την τεχνική αυτή. Εμείς θεωρούμε ότι όποτε η εντολή 4 προκαλέσει mispredict βρίσκεται στο λάθος μονοπάτι αλλά όπως εξηγήσαμε και πριν πάντα υπάρχει η πιθανότητα να προβλέψουμε ότι κάποια εντολή βρίσκεται στο λάθος μονοπάτι αλλά στην πραγματικότητα βρίσκεται στην ορθή ροή του προγράμματος. Έτσι όσες φορές κάποια εντολή προβλέπεται ορθά ότι είναι στο λάθος μονοπάτι και όντως είναι, κερδίζουμε κάποιους κύκλους ενώ αν προβλέψουμε ότι βρίσκεται στο λάθος αλλά τελικά βρισκόμαστε σε σωστό μονοπάτι πληρώνουμε κάποιο κόστος.

Benefit: Οι κύκλοι που θα είχαν σπαταληθεί στο λάθος μονοπάτι μέχρι την ολοκλήρωση της εντολής 1 (αυτής που προκάλεσε το λάθος μονοπάτι) – τους κύκλους που ήδη σπαταλήθηκαν στο λάθος μονοπάτι.

Cost: Οι κύκλοι που χρειάζονται για να γίνει αποκατάσταση του pipeline λόγο του 'αχρείαστου' pipeline flush + οι κύκλοι που ήδη σπαταλήθηκαν στο σωστό μονοπάτι εκτέλεσης και θα πρέπει να ξανά εκτελέσουμε τις ίδιες εντολές για δεύτερη φορά.

Στις πλείστες περιπτώσεις το κόστος που πληρώνεται είναι πιο μεγάλο σε σχέση με το κέρδος που έχουμε από την συγκεκριμένη τεχνική. Άρα δεν μπορεί να εφαρμοστεί σε όλες τις εντολές αλλά μόνο στις εντολές που η πιθανότητα επιτυχίας των προβλέψεων ότι βρίσκεται στο λάθος μονοπάτι θα είναι σε υψηλά επίπεδα. Για να γίνει η επιλογή των εντολών που όντως θα μας βοηθούσαν στην συγκεκριμένη ιδέα χρησιμοποιείται μια νέα παράμετρος που εξηγείται στην συνέχεια.



Σχ.4.1 Φαίνεται η θεωρητική διαδικασία εκτέλεσης μιας εντολής η οποία ιδανικά θα είχε διαφορετική ακρίβεια όταν γίνεται commit. Γίνεται πρόβλεψη για την πρώτη εντολή ότι θα είναι taken και φορτώνονται στο pipeline οι εντολές 2,3 και 4. Η εντολή 4 ολοκληρώνεται πριν την εκτέλεση της εντολής 1 και προκαλεί misprediction. Λόγω του ότι η εντολή 4 ξέρουμε ότι υπάρχει διαφορά στην ακρίβεια για τις δύο περιπτώσεις καταλαβαίνουμε ότι βρίσκεται στο λάθος μονοπάτι. Έτσι με αυτό το τρόπο γίνεται άμεση επαναφορά στο pipeline και ξεκινά η εκτέλεση της εντολής 8, η οποία θα έπρεπε να εκτελεστεί εξαρχής.

Benefit: Ο χρόνος που θα σπαταλόταν ακόμα για εκτέλεση άλλων εντολών στο Wrong Path. (3 και ολοκλήρωση 1) μέχρι να ανακαλυφθεί ότι υπήρξε mispredict

Cost: (Σε περίπτωση που βρίσκεται στο σωστό μονοπάτι τελικά η εντολή 4. Ο χρόνος που απαιτήται για να γίνει flush το pipeline και ο χρόνος που χρειάζεται για να ξαναεκτελεστεί οι εντολές 2 και 3

Σε περίπτωση που δεν θα υπήρχε η τεχνική αυτή θα έπρεπε να ολοκληρώσει η εντολή 1 και μετά να εντοπιστεί mispredict για να εκτελεστεί η εντολή 8. Άρα αφού υποθέτουμε ότι τις περισσότερες φορές η εντολή 4 όταν προκαλεί mispredict είναι στο λάθος μονοπάτι άρα τις περισσότερες φορές έχουμε κέρδος από την συγκεκριμένη εντολή.

4.2 Opportunity Ratio Parameter

Για να μπορέσουμε να επιλέξουμε για ποιες εντολές κάποιου benchmark θα ήταν κερδοφόρο να εφαρμόσουμε την τεχνική που εξηγήσαμε στις προηγούμενες σελίδες χρησιμοποιήσαμε ένα νέο στατιστικό στο οποίο δώσαμε την ονομασία **Opportunity Ratio**. Ουσιαστικά είναι η αναλογία των executed mispredicts / committed mispredicts για κάποια συγκεκριμένη εντολή.

Executed Mispredicts: Ο αριθμός των mispredicts που έγινα όταν εκτελούταν η συγκεκριμένη εντολή.
Committed Mispredicts: Ο αριθμός των mispredicts που έγιναν όταν εκτελούταν η εντολή και στο τέλος έγινε commit. Δηλαδή τις φορές που βρίσκεται μια εντολή στο ορθό μονοπάτι.

Ουσιαστικά το Opportunity Ratio μας δηλώνει πόσο συχνά σε μια εντολή διακλάδωσης, όταν εντοπιστεί ότι προκάλεσε λάθος πρόβλεψη η εντολή βρίσκεται στο σωστό μονοπάτι. Για παράδειγμα αν για μια εντολή το Opportunity Ratio είναι ίσο με 1000, αυτό μας δηλώνει ότι μια στις 1000 φορές που η εντολή αυτή προκαλεί κάποιο mispredict θα είναι στο σωστό μονοπάτι. Άρα καταλαβαίνουμε ότι οι εντολές που ψάχνουμε για να εφαρμόσουμε την τεχνική αυτή πρέπει να έχουν όσο πιο ψηλό Opportunity Ratio. Όσο πιο ψηλό είναι το Opportunity Ratio κάποιας εντολής τόσο πιο λίγο θα είναι το συνολικό κόστος που θα καλούμε να πληρώσω όταν η εντολή βρίσκεται στο ορθό μονοπάτι και έγω υποθέτω ότι είναι σε wrong path.

Για να καταλάβουμε όμως αν μια εντολή θα μπορούσε να βελτιώσει την απόδοση του επεξεργαστή δεν είναι αρκετό μόνο να έχει ψηλό opportunity ratio, αλλά θα πρέπει να λάβουμε υπόψη και τις παραμέτρους του κόστους που πληρώνει όταν βρίσκεται στο ορθό μονοπάτι αλλά και το κέρδος που θα είχαμε αν όντως βρίσκεται στο λάθος μονοπάτι. Στις επόμενες σελίδες παρουσιάζεται ένα μαθηματικό μοντέλο για να καταλάβούμε πόσο θα μπορούσαν να επηρεάσουν την συνολική απόδοση οι εντολές αυτές.

Κεφάλαιο 5 – Result and Analysis

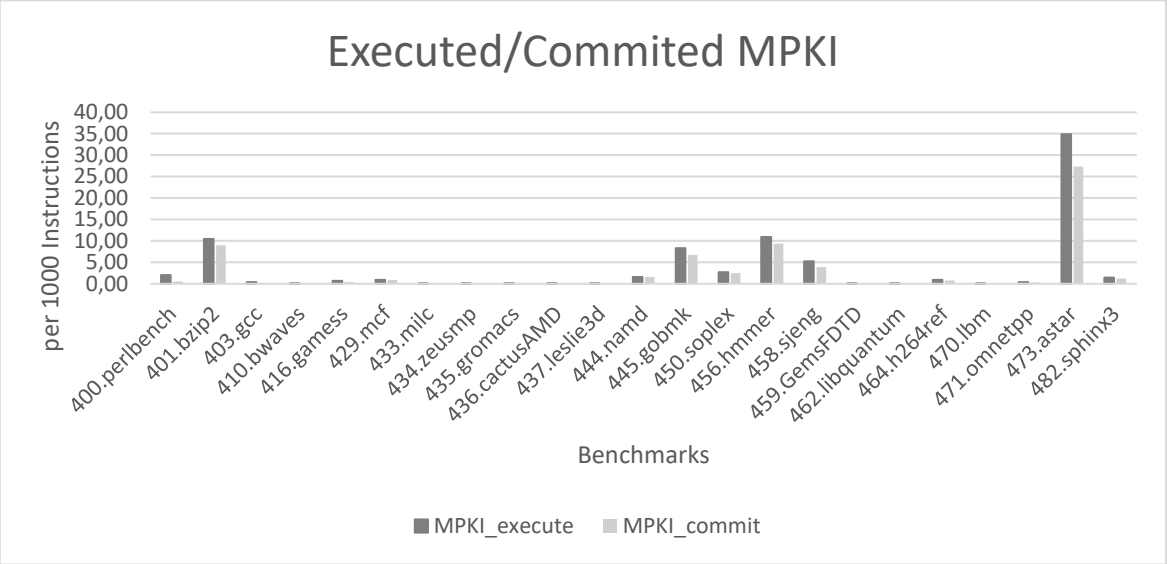
5.1 Methodology

Για να μπορέσουμε να εξάγουμε τα αποτελέσματα και να μπορούμε να κάνουμε την ανάλυση μας αν όντως μπορούμε να διακρίνουμε αν μια εντολή βρίσκεται στο λάθος μονοπάτι με βάση το ποσοστό επιτυχίας στις προβλέψεις του μηχανισμού προβλέψεων χρησιμοποιήσαμε τον προσομοιωτή sim-alpha. Ο sim-alpha είναι ένας superscalar επεξεργαστής και η διασωλήνωση του διαχωρίζετε σε 6 στάδια. Fetch – Slot – Map – Issue – Write back – Commit. L1 cache: 64 KB, L2 cache: 4096 KB, Instructions Cache: 64 KB. Ως μηχανισμό πρόβλεψης χρησιμοποίησα τον predictor L-TAGE-SC-L που προτάθηκε από τον André Seznec και κέρδισε το διαγωνισμό CBP 2016 [4] και είναι το state of the art των μηχανισμών πρόβλεψης. Το μέγεθος του predictor που χρησιμοποίησα για τα αποτελέσματα μου, είναι 64KB. Χρησιμοποίησα και ανάλυσα 23 ξεχωριστά benchmarks για τις μετρήσεις μου, τα SPEC 2006. Ο simulator τρέχει για κάθε benchmark μέχρι να γίνουν commit συνολικά 100 000 000 εντολές. Αυτό σημαίνει ότι εξετάζουμε ένα μέρος των benchmarks και τα στατιστικά που εξάγουμε αφορούν αυτές τις εντολές.

5.2 Experimental Result and Opportunity Ratio

Το πιο σημαντικό κίνητρο που χρειαζόμαστε για να μπορέσουμε να ολοκληρώσουμε την ιδέα που προαναφέραμε, έτσι ώστε να εντοπίζουμε το λάθος μονοπάτι εκμεταλλευόμενοι το γεγονός ότι κάποια εντολή διακλάδωσης προκάλεσε mispredict θα έπρεπε να δούμε ότι όντως στα benchmarks που χρησιμοποιούμε υπάρχει ένας αξιόλογος αριθμός λάθος προβλέψεων. Έτσι αρχικά προσαρμόζοντας τον predictor L-TAGE-SC-L στον προσομοιωτή sim-alpha, υπολογίσαμε τα συνολικά MPKI (Mis predicts per kilo instructions) για το κάθε benchmark ξεχωριστά. Από το (Σχ. 5.1) μπορούμε να

αντιληφθούμε ότι υπάρχουν μερικά προγράμματα στα οποία δεν θα μπορούσαμε να εφαρμόσουμε την ιδέα που είχαμε για το λόγο ότι δεν προκύπτει ικανοποιητικός αριθμός mispredictions. Επίσης μπορούμε μόνο από την γραφική να καταλάβουμε ότι τα executed mispredictions (δηλαδή τα συνολικά mispredictions που προκλήθηκαν – ανεξαρτήτως αν αυτά θα γίνονταν στην πορεία commit) είναι περισσότερα σε σχέση με τα mispredictions που προκάλεσε το κάθε πρόγραμμα και έγιναν commit. Η διαφορά αυτών των δύο στατιστικών είναι και ο βέλτιστός αριθμός mispredictions που θα μπορούσαμε να εκμεταλλευτούμε με αυτή την τεχνική, δηλαδή ο αριθμός των mispredicts που προκαλούνται όταν η εντολή που εκτελείται και βρίσκεται στο λάθος μονοπάτι.



Σχ.5.1 Στην συγκεκριμένη γραφική παράσταση φαίνονται τα executed MPKI (δηλαδή δεν έχουν γίνει απαραίτητα commit και τα commit MPKI τα οποία εκτελέστηκαν και ήταν στο σωστό μονοπάτι.

Στόχος αυτής της διπλωματικής εργασίας είναι να δούμε αν μπορούμε να βελτιώσουμε την απόδοση του επεξεργαστή σε ένα βαθμό εκμεταλλευόμενη την διαφορετική ακρίβεια των προβλέψεων σε εντολές που βρίσκονται στο λάθος μονοπάτι μεταξύ των εντολών που βρίσκονται στο λάθος μονοπάτι. Έτσι στην συνέχεια της ανάλυσης μας θέλαμε να εξάγουμε αν όντως υπάρχει διαφορά μεταξύ των δύο ειδών εντολών στην ακρίβεια των εντολών για ολόκληρη την εκτέλεση της εφαρμογής (Πιν 5.2). Από τον πίνακα μπορούμε να παρατηρήσουμε ότι στα περισσότερα benchmark υπάρχει ελάχιστη διαφορά προς την

συνολική ακρίβεια στις προβλέψεις. Μπορεί να παρατηρούμε ότι στις πλείστες περιπτώσεις όντως στο commit στάδιο οι συνολικές

Benchmarks	Commit (%)	Execute (%)
400.perlbench	99.42	98.11
401.bzip2	90.53	89.86
403.gcc	99.91	99.73
410.bwaves	99.95	99.95
416.gamess	99.35	98.85
429.mcf	98.99	98.99
433.milc	99.99	99.99
434.zeusmp	99.99	99.99
435.gromacs	96.00	96.24
436.cactusAMD	99.92	99.68
437.leslie3d	99.98	99.98
444.namd	96.51	96.81
445.gobmk	94.95	94.13
450.soplex	98.00	97.98
456.hmmer	92.31	92.11
458.sjeng	97.15	96.38
459.GemsFDTD	99.99	99.99
462.libquantum	99.99	99.99
464.h264ref	98.78	98.55
470.lbm	99.91	99.91
471.omnetpp	99.88	99.84
473.astar	84.34	83.60
482.sphinx3	98.36	98.21

Πιν 5.2 Φαίνεται η ακρίβεια για τις συνολικές εντολές κάθε benchmark, τόσο για τις εντολές που βρίσκονται στο λάθος μονοπάτι αλλά και αυτές που στο τέλος γίνονται commit.

εντολές έχουν κάποια ελαφρά αύξηση στην ακρίβεια σε σύγκριση με την φάση του execute. Τα ποσοστά των επιτυχών προβλέψεων όμως είναι σε πολύ ψηλά επίπεδα και για τις δύο περιπτώσεις οπότε καταλαβαίνουμε ότι δεν θα μπορούσε να εφαρμοστεί η τεχνική αυτή σε όλο το σύνολο των εντολών διακλάδωσης κάποιας εφαρμογής – προγράμματος. Άρα αν θα μπορούσε να ισχύει η ιδέα μας θα ίσχυε μόνο σε επίπεδο εντολών όπως αναφέραμε και πιο πριν.

Έτσι συνεχίσαμε την ανάλυση τώρα, πιο συγκεκριμένα καταφέρνοντας να εξάγουμε από τον προσομοιωτή και για όλα τα benchmarks των ακριβή αριθμό των φορών που εκτελέστηκε κάποια εντολή, των ακριβή αριθμό των mispredictions που προκάλεσε και επίσης τον ακριβή αριθμό των φορών που μια εντολή έγινε commit και αντίστοιχα τον ακριβή αριθμό των mispredictions που προκλήθηκαν. Στο (Πιν.5.3) φαίνεται ένα δείγμα των αποτελεσμάτων μου για τα benchmarks 473.astar, 401.bzip2 και 458.sjeng. Στους πίνακες αυτούς φαίνονται οι δέκα πρώτες εντολές των συγκεκριμένων προγραμμάτων οι οποίες προκαλούν το μεγαλύτερο ποσοστό των συνολικών mispredictions που προκαλεί το πρόγραμμα. Για τις συγκεκριμένες εντολές τα αποτελέσματα δεν είναι τόσο θετικά έτσι ώστε να πούμε ότι θα μπορούσαμε να εφαρμόσουμε την τεχνική μας για αυτές έχοντας την επιθυμητή βελτίωση. Φαίνεται ότι η ακρίβεια στις προβλέψεις δεν έχουν τις αναμενόμενες διαφορές μεταξύ τους. Αυτό έχει σαν αποτέλεσμα να μην μας δίνει την

δυνατότητα να προβλέψουμε ότι κάποια εντολή βρίσκεται στο λάθος μονοπάτι με μεγάλη ακρίβεια. Έτσι αφού ξέρουμε ότι το κόστος που πληρώνουμε κάθε φορά είναι πιο μεγάλο σε σ χέση με το κέρδος που θα είχαμε αν όντως ήταν στο λάθος μονοπάτι χρειαζόμαστε μεγάλη ακρίβεια στη πιθανότητα η εντολή να βρίσκεται στο λάθος μονοπάτι για να εφαρμόσουμε την ιδέα μας.

Παράδειγμα:

Benchmark: 401. Bzip2 **Instruction PC**: 4831950952

Executed: 1502555 **Executed Mispredicts**: 427050

Committed: 1298219 **Committed Mispredicts**: 352022

Chance to be in Wrong Path when occur misprediction: 18%

Chance to be in Correct Path when occur misprediction: 82%

473.astar

PCs	Committed	Mis predicted (Commit)	Commit (%)	Executed	Mis predicted (Execute)	Execute (%)
4831950952	1298219	352022	72.8842	1502555	427050	71.5784
4831950792	1298220	286454	77.9349	1461791	339938	76.7451
4831951032	1298220	283092	78.1938	1509205	351685	76.6973
4831951052	736268	212574	71.1282	841157	256998	69.4471
4831950476	1298221	211560	83.7039	1490950	251629	83.1229
4831950632	1298220	208704	83.9238	1493103	256698	82.8078
4831950896	742042	201912	72.7897	813342	232354	71.4322
4831950876	1298220	197440	84.7915	1403215	222834	84.1198
4831950552	1298220	142709	89.0073	1605220	238014	85.1725
4831950652	587359	132736	77.4012	657055	158642	75.8556
4831950712	1298220	120205	90.7408	1406100	140981	89.9736

401.Bzip2

PCs	Committed	Mispredicted (Commit)	Commit (%)	Executed	Mispredicted (Execute)	Execute (%)
4831883648	503665	76092	84.8923	503835	76126	84.8907
4831883704	400762	47031	88.2646	400882	47044	88.2649
4831883748	340136	28517	91.616	340403	28661	91.5803
4831883792	306158	24263	92.075	306367	24400	92.0357
4831883836	278215	22948	91.7517	278420	23076	91.7118
4831883880	249305	22934	90.8008	249510	23084	90.7483
4831883924	218906	16207	92.5964	219132	16394	92.5187
4831883968	201970	14059	93.0391	202185	14225	92.9644
4831884012	185978	16223	91.2769	186175	16378	91.2029
4831884056	164902	14107	91.4452	165092	14241	91.3739
4831884100	149794	10134	93.2347	149966	10250	93.1651

Πιν. 5.3 Στους πιο πάνω πίνακες φαίνονται αναλυτικά η συμπεριφορά που έχουν οι εντολές με τα περισσότερα mispredicts σε ένα δείγμα 2 benchmarks

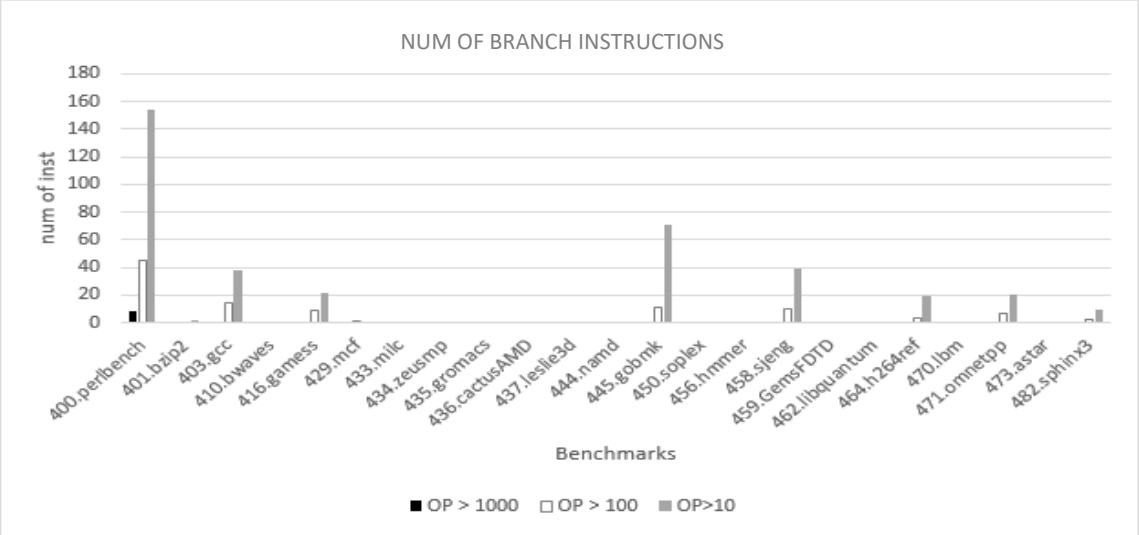
Από τις προηγούμενους πίνακες μπορούμε να εξάγουμε το συμπέρασμα ότι η τεχνική αυτή που θα χρησιμοποιήσουμε δεν ασχολείται με τις εντολές οι οποίες προσφέρουν το μεγαλύτερο ποσοστό mispredictions σε κάποιο πρόγραμμα. Οι συγκεκριμένες εντολές έχουν παρόμοια ποσοστα επιτυχίας τόσο στο λάθος όσο και στο σωστό μονοπάτι εξαιτίας της πολυπλοκότητας τους. Οπως αναφέραμε και στο προηγούμενο κεφάλαιο, για την επιλογή των ‘καλών’ εντολών χρησιμοποιούμε την παράμετρο Opportunity Ration.

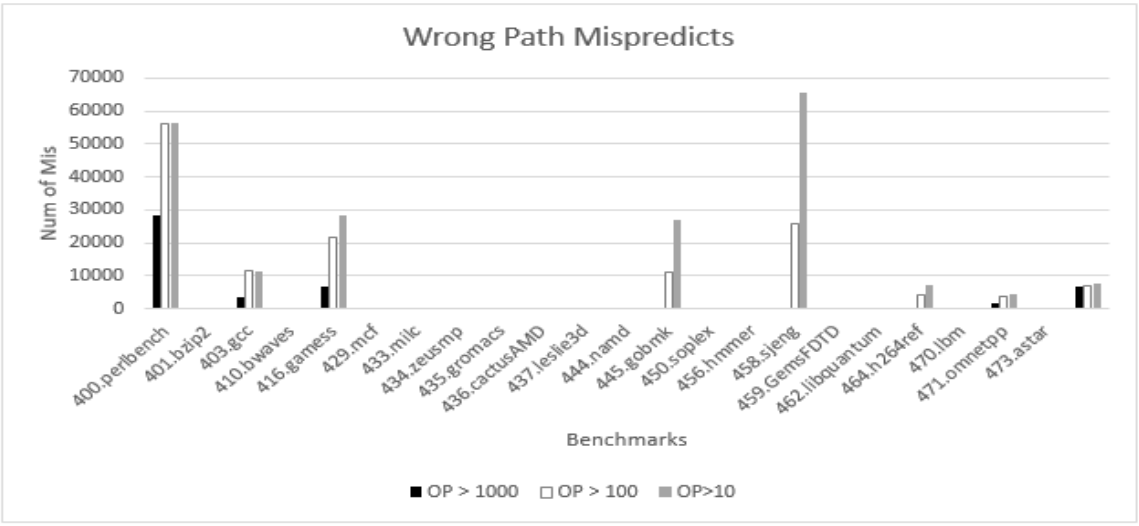
Ψάχνουμε εντολές οι οποίες θα έχουν ψηλό Opportunity Ratio, για το λόγο ότι όσο πιο ψηλό είναι τόσο λιγότερες πιθανότητες θα έχει το mispredict που προκλήθηκε να έιναι στο ορθό μονοπάτι. Το πρόβλημα όμως που προκύπτει χρησιμοποιώντας τις εντολές με πολύ υψηλό Opportunity Ratio είναι ότι δεν υπάρχουν πολλές εντολές (Σχ. 5.4) που να ικανοποιούν αυτή την συνθήκη και αυτόματα δεν υπάρχει περιθώριο αξιοσημείωτης βελτίωσης στην επίδοση. Με πολύ χαμηλό όμως Opportunity Ratio σημαίνει ότι τόσο περισσότερο penalty θα πληρώνει ο επεξεργαστής αφού τόσο πιο συχνά θα υποθέτουμε ότι η εντολή βρίσκεται στο wrong path αλλά ήταν στο σωστό μονοπάτι.

Επίσης από τις γραφικές καταλαβαίνουμε ότι δεν μπορούμε να εξετάσουμε και τα είκοσι τρία benchmarks για το κατά πόσο υπάρχει βελτίωση στην επίδοση του επεξεργαστή. Όπως διαπιστώσαμε πριν από το (Σχ. 5.1) υπάρχουν είδη benchmarks τα οποία δεν προκαλούν ικανοποιητικό αριθμό λάθος προβλέψεων για να μπορέσουμε να τα αναλύσουμε. Από τα benchmarks που έχουν ένα ικανοποιητικό αριθμό λάθος προβλέψεων παρατηρούμε ότι υπάρχουν ορισμένα τα οποία δεν έχουν εντολές με ικανοποιητικό Opportunity Ratio έτσι ώστε να μπορέσουμε να τις εκμεταλλευτούμε.

Τα benchmarks με τα οποία θα συνεχίσουμε την ανάλυση μας είναι αυτά που έχουν τουλάχιστον ένα σεβαστό αριθμό εντολών με Opportunity Ratio > 10 :

- 400.perlbench
- 403.gcc
- 416.games
- 445.gobmk
- 458.sjeng
- 464.h264ref
- 471.omnetpp
- 482.sphinx3





Σχ.5.4 Στις γραφικές φαίνεται ο αριθμός των εντολών σε κάθε benchmark που ικανοποιούν το Opportunity Ratio σε κάθε περίπτωση. Στη δεύτερη γραφική παρατηρούμε τον αριθμό των wrong path mispredictions που προκαλούν οι συγκεκριμένες εντολές.

Στο (Σχ. 5.5) παρουσιάζεται ένα δείγμα από τις δέκα πρώτες εντολές διακλάδωσης των benchmarks (400.perlbench, 445.gobmk, 458.sjeng) που επιλέξαμε οι οποίες έχουν Opportunity Ratio τέτοιο ώστε να μας επιτρέπει να κάνουμε αρκετά υψηλής ακρίβειας υπόθεση αν προκαλέσουν mispredict ότι βρίσκονται στο λάθος μονοπάτι. Φαίνεται ότι όντως υπάρχουν εντολές διακλάδωσης με αρκετά ψηλο opportunity ratio οι οποίες προκαλούν παραπολλά wrong path mispredicts. Πόσο όμως θα βελτείωνε την επίδοση του επεξέργαστη οι εντολές διακλάδωσης αυτές για τα συγκεκριμένα benchmarks?

445.gobmk

PCs	Committed	Mispredicted (Commit)	Commit (%)	Executed	Mispredicted (Execute)	Execute (%)	Wrong Path Mispredicts	Opportunity
4832998580	1601	1	99.9375	2180	537	75.367	536	537
4832994912	2631	3	99.886	3985	989	75.1819	986	329.6
4832982040	1684	1	99.9406	2240	206	90.8036	205	206
4832762024	11260	5	99.9556	12764	987	92.2673	982	197.4
4832734956	7271	4	99.945	8082	755	90.6583	751	188.7
4832898876	3289	4	99.8784	3849	556	85.5547	552	139
4832754928	14232	31	99.7822	19496	4254	78.1801	4223	137.2
4832997068	714	1	99.8599	914	136	85.1204	135	136
4832838876	2473	1	99.9596	3049	128	95.8019	127	128
4832758672	11260	25	99.778	14709	2698	81.6575	2673	107.9
4832983140	1789	1	99.9441	2049	103	94.9732	102	537

458.sjeng

PCs	Committed	Mispredicted (Commit)	Commit (%)	Executed	Mispredicted (Execute)	Execute (%)	Wrong Path Mispredicts	Opportunity
4831955032	365	1	99.726	1035	646	37.5845	645	646
4831991024	423	1	99.7636	1099	642	41.5833	641	642
4831939280	6455	8	99.8761	33426	3467	89.6278	3459	433.3
4831956656	17484	23	99.8685	33918	8729	74.2644	8706	379.5
4831958172	10032	12	99.8804	14056	2164	84.6044	2152	180.3
4831959552	12128	20	99.8351	20487	3432	83.2479	3412	171.6
4831954544	322	1	99.6894	604	170	71.8543	169	170
4832023932	1053	1	99.905	1254	166	86.7624	165	166
4831953104	413	4	99.0315	1055	421	60.0948	417	105.2
4831991924	7477	58	99.2243	17028	6019	64.6523	5961	103.7
4831993076	423	7	98.3452	1081	628	41.9056	621	89.7

400.pernbanch

PCs	Committed	Mispredicted (Commit)	Commit (%)	Executed	Mispredicted (Execute)	Execute (%)	Wrong Path Mispredicts	Opportunity
4832404312	15022	2	99.9867	27912	12872	53.8836	12870	6436
4832876016	8723	2	99.9771	16045	7048	56.0735	7046	3524
4832118528	29655	1	99.9966	31557	1866	94.0869	1865	1866
4833304928	2287	1	99.9563	3778	1477	60.9052	1476	1477
4832537556	2195	1	99.9544	3590	1396	61.1142	1395	1396
4832552688	13989	1	99.9929	15427	1247	91.9168	1246	1247
4832298500	2104	1	99.9525	3334	1231	63.0774	1230	1231
4832609776	1017	1	99.9017	2032	1016	50	1015	1016
4832397156	1865	1	99.9464	2969	972	67.2617	971	972
4833316448	653	1	99.8469	1789	875	51.09	874	875
4832563064	22429	6	99.9732	27822	4243	84.7495	4237	707.1

Σχ. 5.5 Παρουσιάζονται αναλυτικά για 3 από τα 8 benchmarks που επιλέξαμε οι 10 πρώτες εντολές με το μεγαλύτερο Opportunity Ratio και φαίνεται ότι όντως υπάρχουν εντολές με τέτοια συμπεριφοράς.

5.3 Mathematical Model

Από τις προηγούμενες σελίδες παρατηρήσαμε ότι όντως υπάρχουν εντολές διακλάδωσης τις οποίες θα μπορούσαμε να εκμεταλλευτούμε για να εξοικονομήσουμε κάποιους κύκλους από την συνολική εκτέλεση κάποιου προγράμματος εξετάζοντας το Opportunity Ratio τους. Το πιο σημαντικό θέμα που ενδιαφέρει τους ερευνητές στον τομέα της Αρχιτεκτονικής Υπολογιστών είναι ανέκαθεν η βελτίωση της απόδοσης. Αυτό που μας ενδιαφέρει λοιπόν είναι σε πιο βαθμό θα μπορούσαμε να βελτιώσουμε την απόδοση στην συγκεκριμένη περίπτωση και αν η ιδέα μας αξίζει να μελετηθεί περεταίρω. Για να μπορέσουμε να αποφασίσουμε για αυτό το ζήτημα δημιουργήσαμε ένα Μαθηματικό Μοντέλο.

Το μαθηματικό μοντέλο αυτό αποτελείται από 3 διαφορετικές παραμέτρους τις οποίες έπρεπε να εξετάσουμε:

- Opportunity Ratio
- Κόστος
- Κέρδος

Opportunity Ratio: Είναι το αποδεκτό Opportunity Ratio που θεωρούμε ότι χρειάζεται για να έχουμε κάποιο αποδεκτή βελτίωση στην επίδοση. Έστω ότι ορίζουμε το Opportunity Ratio = 100, θα χρησιμοποιήσουμε όλες τις εντολές διακλάδωσης με Opportunity Ratio > 100.

Κόστος: Το μέσο κόστος που μπορεί να αποφέρει μία εντολή κάποιου benchmark που επιλέξαμε σε περίπτωση που υποθέτουμε ότι η εντολή είναι στο λάθος μονοπάτι ενώ τελικά βρίσκεται στο ορθό μονοπάτι.

Κέρδος: Το μέσο κέρδος που θα μπορούσε να προσφέρει μια εντολή κάποιου benchmark η οποία θεωρήσαμε ορθά ότι είναι στο λάθος μονοπάτι.

** Σε κάθε benchmark το μέσο κόστος και κέρδος μπορεί να είναι διαφορετικό εξαιτίας της δομής των προγραμμάτων.

Το μαθηματικό μοντέλο:

$$Tnew = T - \left[\min\left[TCM, \sum_{k=1}^{n} (Ne - Nc)\right] * B\right] + \left[\sum_{k=1}^{n} (Nc) * C\right]$$

Παράμετροι στην εξίσωση:

Tnew → Είναι ο νέος αριθμός του χρόνου εκτέλεσης κάποιου προγράμματος σε κύκλους. Με την εφαρμογή της εξίσωσης σε κάποιο benchmark θα δημιουργηθεί ένας νέος χρόνος εκτέλεσης μετρήσιμος σε κύκλους. Αν ο νέος χρόνος είναι μικρότερος από τον αρχικό το αποτέλεσμα της εξίσωσης θα είναι θετικό, διαφορετικά το αποτέλεσμα της εξίσωσης θα είναι αρνητικό και η ιδέα μας σημαίνει δεν μπορεί να εφαρμοστεί στο συγκεκριμένο benchmark

T → Ο αρχικός χρόνος εκτέλεσης σε κύκλους κάποιου benchmark πριν την εφαρμογή του μηχανισμού που θα δημιουργήσουμε.

n → Ο αριθμός των εντολών κάποιου benchmark που ικανοποιούν το κατώτατο όριο του Opportunity Ratio που ορίσαμε.

Ne → Ο συνολικός αριθμός των mispredictions που προκλήθηκαν καθώς εκτελούταν μια εντολή.

Nc → Ο συνολικός αριθμός των mispredictions που προκλήθηκαν καθώς εκτελούταν μια εντολή και στο τέλος έγινε commit. Δηλαδή ο συνολικός αριθμός mispredictions που προκλήθηκα και στο τέλος η εντολή που το προκάλεσε έγινε commit.

Ne-Nc → Η διαφορά των mispredicts που εκτελέστηκαν και που έγιναν commit. Δηλαδή ο αριθμός των mispredicts που προκλήθηκαν στο λάθος μονοπάτι – ο βέλτιστος αριθμός των φορών που μπορούμε να εφαρμόσουμε τον μηχανισμό μας στο συγκεκριμένο benchmark.

B → Όπως εξηγείται και στις προηγούμενα σελίδες όταν εφαρμοστεί η ιδέα έχω κέρδος κάποιους κύκλους αν όντως η εντολή που προκάλεσε το mispredict βρίσκεται στο λάθος μονοπάτι. Αυτό είναι η μέση τιμή κέρδους σε κύκλους που θα μπορούσαμε να κερδίσουμε από κάποιο benchmark.

C → Στην άλλη περίπτωση που η εντολή που προκάλεσε το mispredict βρίσκεται στο σωστό μονοπάτι και εγώ υποθέσω ότι βρίσκεται στο λάθος έχω κάποιο penalty. Αυτό είναι η μέση τιμή κόστους σε κύκλους που μπορώ να ζημιώσω από κάποιο benchmark σε περίπτωση λάθος πρόβλεψης

TCM → Total Committed Mispredicts. Ο συνολικός αριθμός mispredicts που έγιναν commit για κάποιο benchmark.

min[TCM, $\sum_{k=1}^{n}$ (Ne-Nc)] → Αυτό προέκυψε από κάποια παρατήρηση που κάναμε για μερικά προγράμματα. Σε μερικές περιπτώσεις ο συνολικός αριθμός των wrong path mispredicts όλων των εντολών που ικανοποιούν την παράμετρο Opportunity Ratio ξεπερνούν τον συνολικό αριθμό των mispredictions που έγιναν commit για όλοκληρο το πρόγραμμα. Σε αυτή την περίπτωση δεν μπορούμε να εφαρμόσουμε την τεχνική που σκευτήκαμε σε όλα wrong path mispredictions. Η ιδέα είναι ότι υποθέτουμε ότι κάποια εντολή βρίσκεται σε λάθος μονοπάτι και προσπαθούμε να το διορθώσουμε πριν ακόμη ολοκληρωθει η εκτέλεση της εντολής που προκάλεσε το wrong path. Αυτή η εντολή όμως στο τέλος θα γίνει commit, αφου γνωρίζουμε ότι βρίσκεται στο ορθο μονοπάτι. Αυτομάτος όταν η εντολή αυτή γίνει commit αυξάνονται τα committed mispredicts καθώς πριν την ολοκλήρωση της εντοπίσαμε wrong path εξαιτίας της. Άρα δεν είναι λογικό να μπορέσω να εφαρμόσω την τεχνική αυτή περισσότερες φορές από ότι είναι τα committed misprediction κάποιου προγράμματος. Για αυτό το λόγο χρησιμοποιούμε την συνάρτηση min(). Σε περίπτωση λοιπόν που τα wrong path mispredictions είναι περισσότερα η βέλτιστη περίπτωση δεν είναι (Ne – Nc) * B αλλά το TCM * B καθώς υποθέτουμε ότι για κάθε committed mispredict εφαρμόστικέ το πολύ μια φορά η τεχνική για κάποια εντολή διακλάδωσης η οποία βρίσκετε στο wrong path που προκάλεσε η εντολή αυτή. Επίσης από αυτό μπορούμε να συμπερένουμε ότι σε κάποια wrong path υπάρχουν πάρα πολλες εντολές διακλάδωσης οι οποίες έκαναν την πρόβλεψη τους και ολοκληρώσαν την εκτέλεση τους πριν την ολοκλήρωση της εκτέλεσης της αρχικής εντολής που προκάλεσε το wrong path.

Καταλαμβαίνουμε ότι η λογική του μοντέλου είναι ότι αφαιρούμε από τον αρχικό χρόνο εκτέλεσης το βέλτιστο συνολικό κέρδος που είχαμε, δηλαδή όσες φορές προβλέψαμε το ότι βρισκόμαστε σε λάθος και όντως είμασταν (min[TCM, $\sum_{k=1}^{n}$ (Ne-Nc)]) και ταυτόχρονα προσθέτουμε το συνολικό κόστος που είχαμε, δηλαδή όσες φορές κάναμε λάθος πρόβλεψη (Nc). Έτσι θα έχουμε τον νέο χρόνο εκτέλεσης σε κύκλους κάθε benchmark και αναλογα κρίνουμε αν υπάρχει βελτείωση η όχι.

5.4 Mathematical Model Results

Θέλουμε να μπορέσουμε να κατανοήσουμε αν με την τεχνική που θα προτείνουμε στην συνέχεια υπάρχει βελτίωση στο χρόνο που χρειάζεται το CPU.

CPUtime = I * CPI * Clock Cycle Time.

Στόχος μας είναι να μειώσουμε το CPU time του προγράμματος. Τις δυναμικές εντολές (I) δεν μπορούμε να τις μοιώσουμε καθώς στο προσομοιωτή τον οποίο χρησιμοποιούμε οι δυναμικές εντολές είναι σταθερές (100000000). Επίσης δεν θα μπορούσαμε να επηρεάσουμε το Clock Cycle Time καθώς και αυτό εξαρτάται από την κατασκευή του επεξεργαστή. Στόχος μας να διαφοροποιήσουμε το CPI (Cycles Per Instruction).

Ξέρουμε ότι CPI = 1/IPC (Instruction per Cycle) και IPC = Instructions /Cycles. Άρα εμείς μπορούμε να μειώσουμε τους κύκλους εκτέλεσης κάποιου προγράμματος σύμφωνα με το μαθηματικό μοντέλο που περιγράφεται πιο πάνω για να αυξήσουμε το IPC το οποίο συνεπάγεται ότι θα μειωθεί το CPUtime.

Για να μπορέσουμε να παρουσιάσουμε και να κατανοήσουμε κατά πόσο μπορούμε να έχουμε κέρδος στα συγκεκριμένα benchmarks εξάγαμε από τον προσωμειωτή τον μέσο χρόνο σε κύκλους που καλείται να πληρώσει κάθε εντολή σε περίπτωση που προκαλέσει mispredict (Πιν. 5.6), Μαυρισμένα είναι τα benchmarks τα οποία μας ενδιαφέρουν έτσι ώστε να δούμε με πιο μέσο κέρδος θα μπορούσαμε να βελτειώσουμε το IPC σε ένα επίπεδο το οποίο θεωρούμε ενδιαφέρον. Ετσί εφαρμόσαμε τον Μαθηματικό Μοντέλο που εξηγήσαμε στις προηγούμενες σελίδες με σταθερή μεταβλητή το κόστος.

Η επιθυμητή αυξηση στο IPC με την οποία θα λέγαμε ότι όντως η ιδέα μας θα άξιζε να υλοποιηθεί είναι της τάξης του +0.5%. Μπορεί να παρατηρήρουμε και μια αυξηση σε πιο χαμηλά επίπεδα αλλά θεωρούμε ότι δεν θα άξιζε να υλοποιηθει η ιδέα μας για τόσο μικρή βελτείωση στην επίδοση του επεξεργαστή.

Benchmarks	Average Mispredict Penalty
400.perlbench	32.08
401.bzip2	33.14
403.gcc	42.79
410.bwaves	40.23
416.gamess	47.24
429.mcf	692.82
433.milc	131.63
434.zeusmp	148.82
435.gromacs	47.72
436.cactusAMD	52.37
437.leslie3d	50.38
444.namd	46.75
445.gobmk	32.96
450.soplex	108.94
456.hmmer	39.46
458.sjeng	28.67
459.GemsFDTD	103.25
462.libquantum	107.4
464.h264ref	33.55
470.lbm	799.44
471.omnetpp	34.15
473.astar	36.93
482.sphinx3	78.17

Πιν. 5.6 Φαίνεται το μέσο κόστος που προκαλεί ένα mispredict σε κάθε benchmark, μέχρι την αποκατάσταση του.

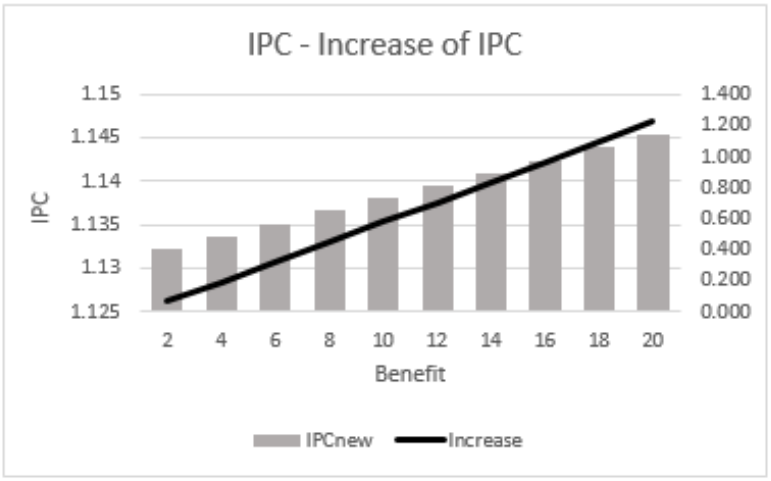
Χρησιμοποιόντας το μοντέλο και με σταθερή παράμετρο πλέον το κόστος το οποίο υπολόγισα υπολογίσα το νέο χρόνο εκτέλεσης για το κάθε benchmarks για δύο τιμές του Opportunity Ratio και για διάφορες τιμές κέρδους. Πιο κάτω φαίνονται οι γραφικές με τη βελτείωση του IPC για κάθε benchmark με βάση το Μαθηματικό Μοντέλο που δημιουργίσαμε.

Opportunity Ratio: Οι τιμές του Opportunity Ratio που χρησιμοποίησα για να δω την αύξηση του IPC είναι 100 και 10. Δεν μπορώ να χρησιμοποιήσω περισσότερο μεγάλες για το λόγο ότι δεν υπάρχουν πολλές εντολές διακλάδωσης με opporunity ration πολύ μεγάλο.

Κέρδός: Οι δοκιμαστικές τιμές που θα χρησιμοποιήσω για το κέρδος είναι μέχρι να δω μια αύξηση στο IPC της τάξης του 0.5% και κατά πόσο το κέρδος που απόφερε αυτή την άυξηση είναι ένας ρεαλιστικός αριθμός

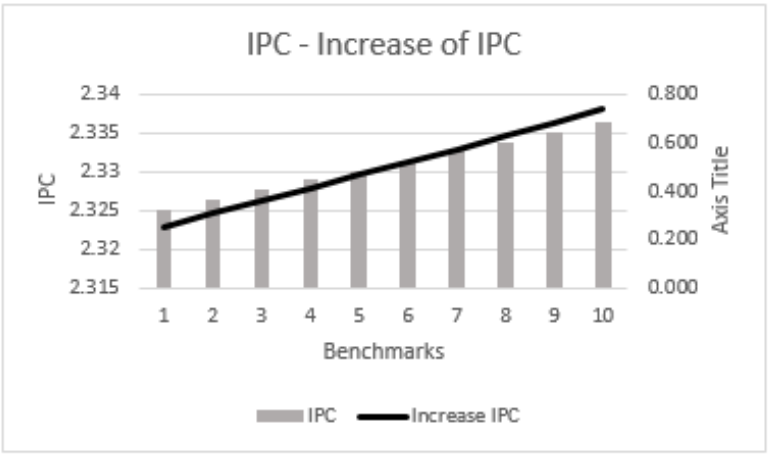
400.perlbench

Benefit (Cycles)	IPC	Increase IPC
2	1.1322371	0.065
4	1.1336816	0.193
6	1.1351297	0.321
8	1.1365816	0.449
10	1.1380371	0.578
12	1.1394965	0.707
14	1.1409595	0.836
16	1.1424263	0.966
18	1.1438969	1.096
20	1.1453713	1.226



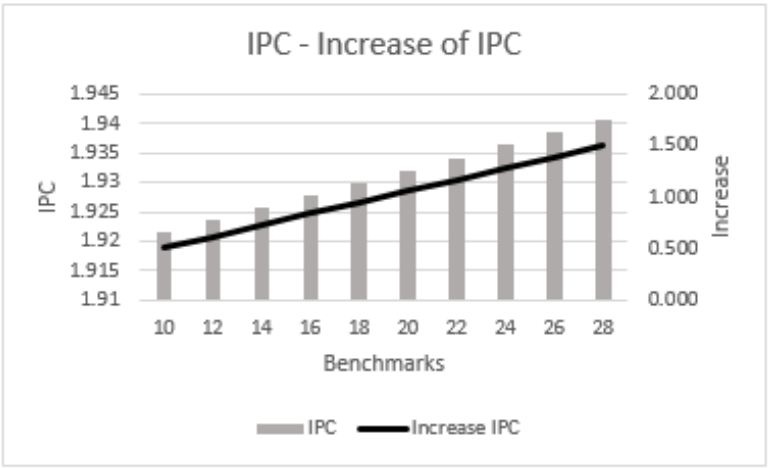
403.gcc

Benefit (Cycles)	IPC	Increase IPC
10	2.325147559	0.252
12	2.326387996	0.306
14	2.327629757	0.359
16	2.328872844	0.413
18	2.33011726	0.466
20	2.331363006	0.520
22	2.332610085	0.574
24	2.333858499	0.628
26	2.33510825	0.682
28	2.33635934	0.736



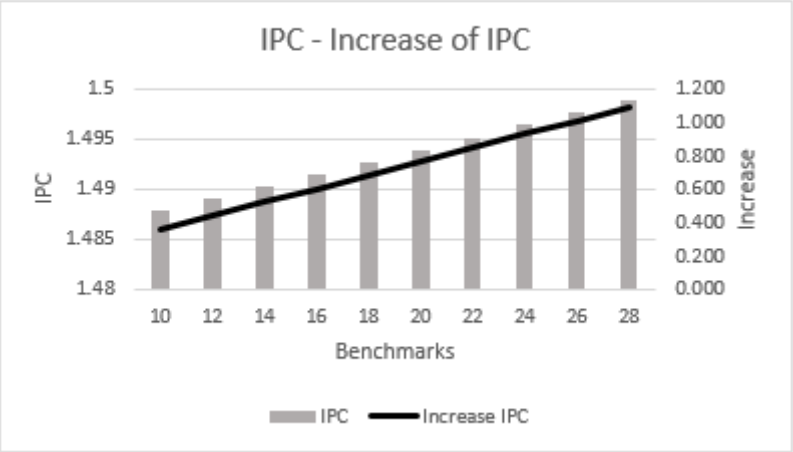
416.games

Benefit (Cycles)	IPC	Increase IPC
10	1.921564093	0.511
12	1.923654052	0.620
14	1.925748561	0.730
16	1.927847637	0.839
18	1.929951294	0.949
20	1.932059546	1.060
22	1.93417241	1.170
24	1.9362899	1.281
26	1.938412032	1.392
28	1.94053882	1.503



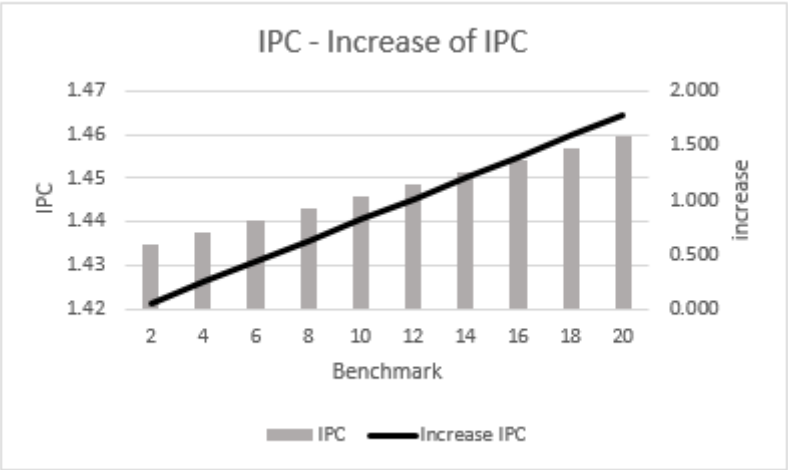
445.gobmk

Benefit (Cycles)	IPC	Increase IPC
10	1.488144838	0.374
12	1.489344821	0.455
14	1.490546739	0.536
16	1.4917506	0.617
18	1.492956406	0.699
20	1.494164164	0.780
22	1.495373877	0.862
24	1.496585551	0.943
26	1.497799189	1.025
28	1.499014798	1.107



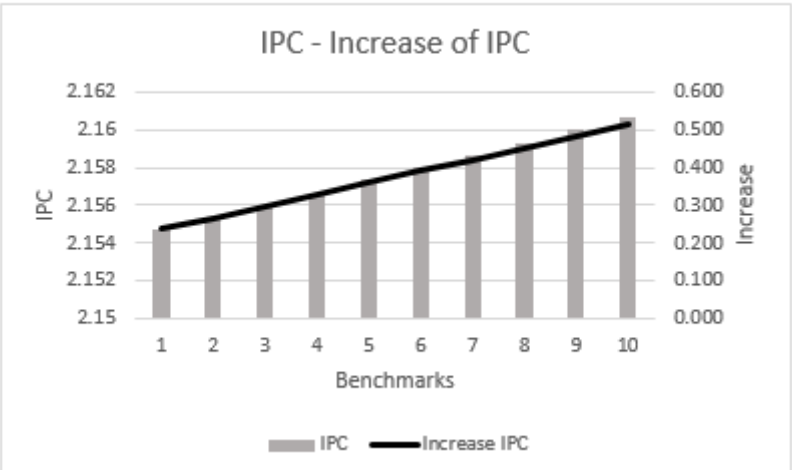
458.sjeng

Benefit (Cycles)	IPC	Increase IPC
2	1.434698752	0.056
4	1.437401645	0.244
6	1.440114742	0.433
8	1.4428381	0.623
10	1.445571777	0.814
12	1.448315834	1.005
14	1.451070327	1.197
16	1.453835318	1.390
18	1.456610867	1.584
20	1.459397033	1.778



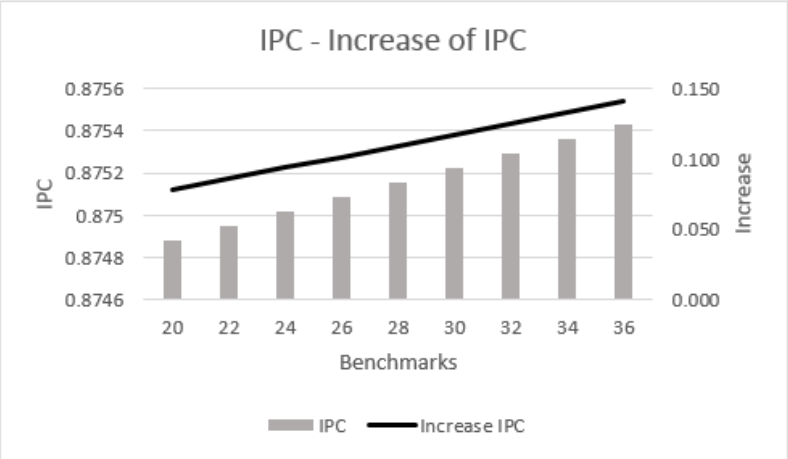
464.h264ref

Benefit (Cycles)	IPC	Increase IPC
16	2.154677	0.236
18	2.155336	0.267
20	2.155996	0.298
22	2.156656	0.328
24	2.157316	0.359
26	2.157977	0.390
28	2.158638	0.420
30	2.1593	0.451
32	2.159962	0.482
34	2.160625	0.513



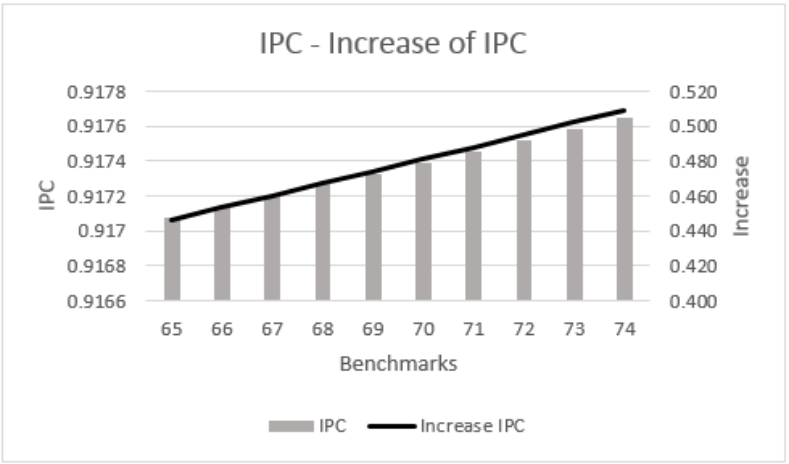
471.omnetpp

Benefit (Cycles)	IPC	Increase IPC
20	0.874884	0.078
22	0.874952	0.086
24	0.87502	0.094
26	0.875088	0.102
28	0.875156	0.109
30	0.875224	0.117
32	0.875292	0.125
34	0.87536	0.133
36	0.875428	0.140



482.sphinx3

Benefit (Cycles)	IPC	Increase IPC
65	0.917075	0.446
66	0.917138	0.453
67	0.917202	0.460
68	0.917266	0.467
69	0.917329	0.474
70	0.917393	0.481
71	0.917457	0.488
72	0.917521	0.495
73	0.917584	0.502
74	0.917648	0.509



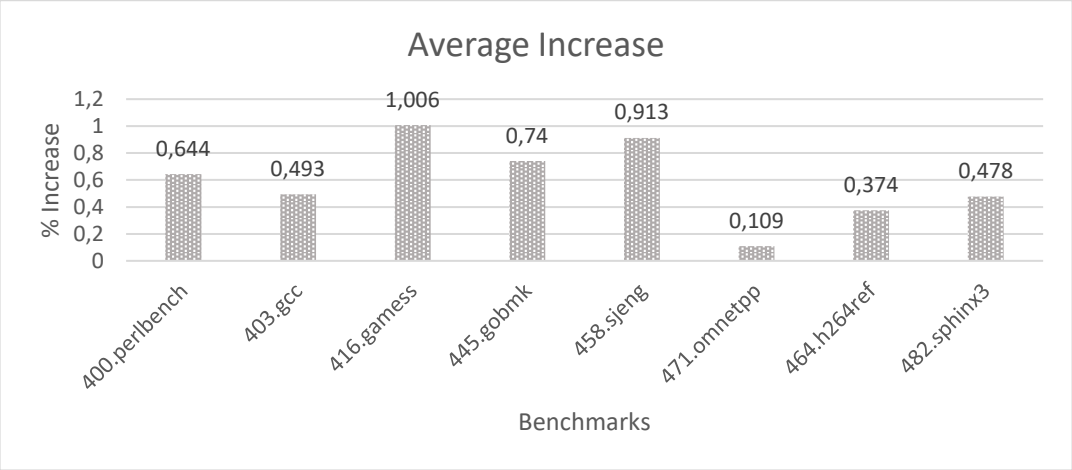
Οι πιο πάνω γραφικές δείχνουν την επίδραση που έχει η τεχνική που σκεφτήκαμε να χρησιμοποιήσουμε στο IPC του κάθε benchmarks και κατά συνέπεια την συνολική απόδοση του επεξεργαστή. Οι γραφικές αυτές πάρθηκαν με σταθερή τιμή του κόστους όπως εμείς το υπολογίσαμε για το κάθε benchmark ξεχωριστά. Χρησιμοποιούμε όλες τις εντολές διακλάδωσης των προγραμμάτων που παρουσιάζουν Opportunity Ratio μεγαλύτερο από 10 και προσπαθούμε με μια αυξομείωση στο κέρδος που θα μπορούσαμε να έχουμε, να δούμε κατά πόσο μπορώ να έχω μια αύξηση γύρω στο 5% σε σχέση με το αρχικό IPC.

Οι γραφικές παραστάσεις παρουσιάζουν το νέο IPC που μπορεί να προκύψει σε κάθε περίπτωση, καθώς επίσης και την αύξηση του σε σχέση με το αρχικό. Στον πίνακα των τιμών για κάθε γραφική φαίνονται με μαυρισμένο χρώμα οι περιπτώσεις όπου παρατηρούμε μια αύξηση >5%, ποσοστό και το οποίο ότι είναι μια αξιοσημείωτη αύξηση. Από τις γραφικες και τους πίνακες των τιμών φαίνεται ότι στα περισσότερα από

τα benchmarks που επιλέξαμε υπάρχει καποιο αξεισιμίοτο ποσοστό αύξησης στο IPC του επεξεργαστή. Πιο αναλυτικά φαίνεται στον (Σχ. 5.7) όπου φαίνεται η μέση άυξηση του IPC για τα συγκεκριμένα benchmarks με βάση τις τίμες που υποθέσαμε ως κέρδος σε κάθε περίπτωση. Οι τιμές αυτές είναι αυτές που απαιτούνται περίπου για να μπορέσουμε να επιτύχουμε μια αύξηση κοντά στο 0.5% που επιθυμούμε.

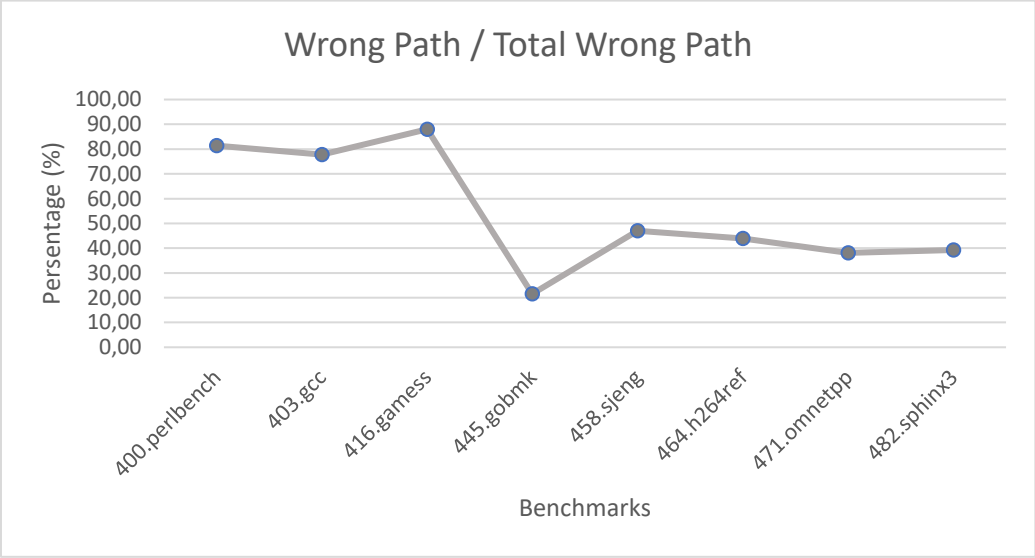
Μπορούμε να πούμε ότι πιο θετικά αποτελέσματα από όλα τα benchmarks βλέπουμε στο 416.games καθώς σχεδόν για όλες τις τιμές που θέσαμε ως κέρδος παρατηρούμε μια αύξηση μεγαλύτερη του 0.5%. Παρόλα αυτά πολύ θετικά συμπεράσματα μπορούμε να βγάλουμε και στα 400.perlbench, 403.gcc, 445.gobmk και 458.sjeng. Φαίνεται η τεχνική ότι έχει θετική επίδραση αφού βλέπουμε ότι για τις περισσότερες τιμές εμφανίζεται αρκετά ικανοποιητική βελτείωση στην απόδοση.

Στην άλλη πλευρά όμως έχουμε και τρια benchmarks των οποίων τα αποτελέσματα δεν ήταν αυτό που αναμέναμε. Αναφερόμαστε στα 464.h264ref, 471.omnetpp και 482.sphinx3. Πιο συγκεκριμένα στο 471.omnetpp και 482.sphinx3 οι τιμές που θέσαμε ως κέρδος δεν είναι απόλυτα ρεαλιστικές αφού για να μπορέσουμε να πετύχουμε μια άυξηση στο επίπεδο που θέλουμε θέσαμε το κέρδος πολύ κοντά στο επίπεδο που κιμαίνεται το μέσο κόστος ανα εντολή για τα συγκεκριμένα benchmarks. Ξέρουμε ότι αυτό δεν θα μπορούσε να ισχύει από το θεωριτικό υπόβαθρο που περιγράψαμε στις προηγούμενες σελίδες.



Σχ. 5.7 Στην πιο πάνω γραφική παράσταση
φαίνεται η μέση αύξηση στο IPC των benchmarks,
με opportunity ratio >10

Σε αυτά τα αποτελέσματα μπορούμε να πούμε ότι ευθύνεται ο μικρός αριθμός των εντολών που ικανοποιούν την συγκεκριμένη τίμη για το opportunity ratio που θέσαμε, έτσι ώστε η εκμετάλλευση αυτών των εντολων να μην έχει σοβαρή επίδραση στην συνολική επίδοση των συγκεκριμένων benchmarks. Η πιο κάτω γραφική (Σχ. 5.8) δείχνει αναλυτικά τι ποσοστο των συνολικών εντολών διακλάδωσης που εκτελούνται και προκαλούν wrong path mispredict, είναι τα wrong path mispredicts που προκαλούν οι εντολές διακλάδωσης που επιλέξαμε για κάθε benchmark.



Σχ. 5.8 Στην γραφική φαίνεται το ποσοστό των συνολικών wrong path mispredicts του συγκεκριμένου benchmark, που καλύπτουν οι εντολές που επιλέξαμε. (OP > 10)

Από την γραφική μπορούμε να δούμε ότι τα benchmarks για τα οποία είδαμε ότι είναι μικρή η βελτειώση, το ποσοστό των wrong path mispredicts για τις εντολές διακλάδωσης που επιλέξαμε, κυμαίνεται γύρω στο 40%. Αυτό είναι χαμηλό ποσοστό αφού σε σύγκριση με τα benchmarks που παρουσιάζουν ικανοποιητική βελτείωση το ποσοστό είναι χαμηλότερο, όπως και περιμέναμε. Το αξιοσημείωτο που παρατηρούμε στην συγκεκριμένη γραφική παράσταση είναι ότι για το 445.gobmk το ποσοστό είναι ακόμα πιο χαμηλό σε σχέση με τα benchmarks που με βάση την πιο πάνω ανάλυση φαίνεται να μην έχουν ικανοποιητικά αποτελέσματα.

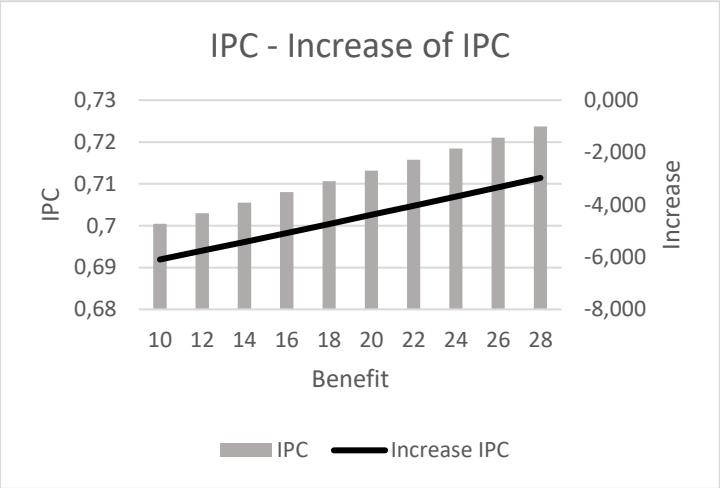
Επίσης θα θέλαμε να δείξουμε ότι η συγκεκριμένη τεχνική όντως δεν έχει ικανοποιητικά αποτελέσματα για κάποιο benchmark το οποίο δεν έχει εντολές διακλάδωσης με υψηλό opportunity ratio. Στο πιο κάτω πίνακα φαίνεται παρόμοια ανάλυση για τις εντολές

διακλάδωσης του 473.astar για τις εντολές διακλάδωσης οι οποίες έχουν opportunity ratio μεγαλύτερο από 1.5. Καταλαμβαίνουμε ότι οι προβλέψεις μας ότι οι συγκεκριμένες εντολές διακλάδωσης βρίσκονται στο λάθος μονοπάτι θα είναι περισσότερες φορές λανθασμένες παρά σωστές.

473.astar

Αρχικό IPC: 0.7459
Average Cost: 36.93

Benefit (Cycles)	IPC	Increase IPC
10	0.700445	-6.094
12	0.702955	-5.758
14	0.705483	-5.419
16	0.708029	-5.077
18	0.710594	-4.733
20	0.713178	-4.387
22	0.71578	-4.038
24	0.718401	-3.687
26	0.721042	-3.333
28	0.723702	-2.976

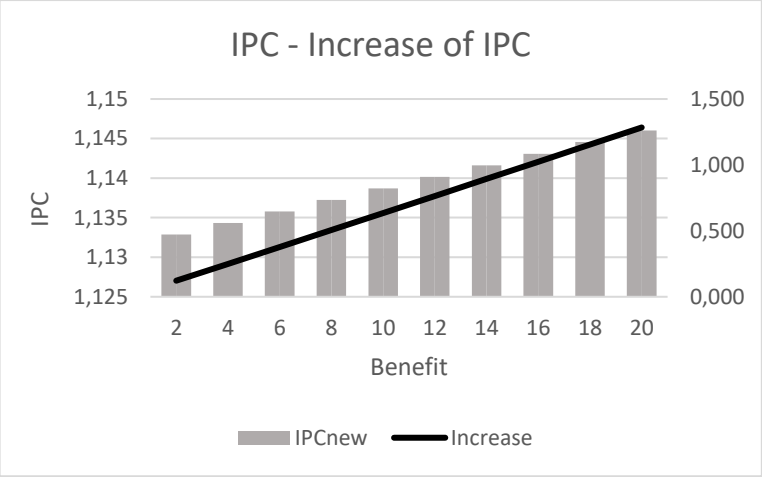


Φαίνεται ξεκάθαρα από τον πίνακα τιμών ότι για τις συγκεκριμένες τιμές opportunity ratio το penalty που καλούμαστε να πληρώσουμε είναι πολύ μεγαλύτερο σε σχέση με το κόστος που θα μπορούσαμε να έχουμε.

Πιο κάτω φαίνονται τα αποτελέσματα που πείραμε χρησιμοποιώντας opportunity ratio>100. Ουσιαστικά αυξάνοντας το κατώφλι για το opportunity ratio μειώνουμε τις εντολές διακλάδωσης στις οποίες θα μπορούσαμε να εφαρμόσουμε την ιδέα μας. Με αυτό τον τρόπο μειώνουμε τις περιπτώσεις κατά τις οποίες το mispredict που προκύπτει είναι στο ορθό μονοπάτι και εμείς υποθέτουμε ότι βρίσκεται στο λάθος, οπού και ο επεξεργαστής πληρώνει κάποιο κόστος, αλλά παράλληλα μειώνουμε και τον αριθμό των φορών που όντως θα βρίσκεται κάποιο mispredict στο λάθος μονοπάτι, και αυτό προκάλεί κέρδος στο επεξεργαστή. Παρουσιάζονται οι γραφικές μόνο για τα πέντε benchmarks τα οποία είχαν θετικά απότελέσματα στην προηγούμενη περίπτωση. Τα υπολοιπά τα αγνοούμε κάθως φαίνεται ότι οι φορές που εφαρμόζουμε την ιδέα δεν είναι αρκετές έτσι ώστε να επηρεάσουν σε ικανοποιητίκο επίπεδο την απόδοση του επεξεργαστή

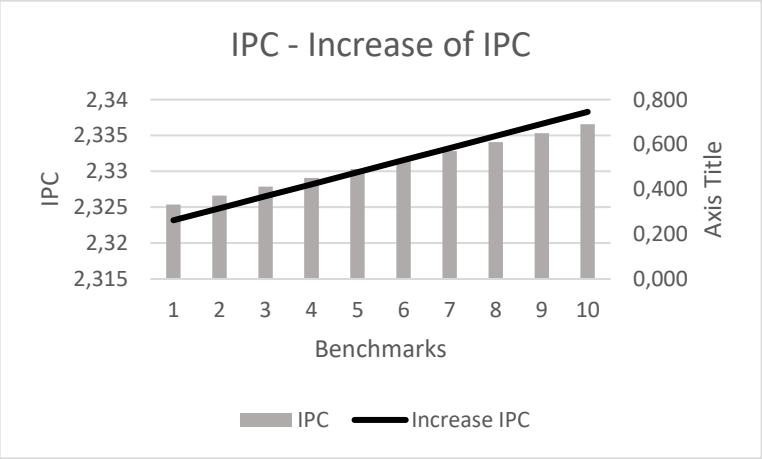
400.perlbench

Benefit (Cycles)	IPC	Increase IPC
2	1.1328799	0.122
4	1.134326	0.250
6	1.1357758	0.378
8	1.1372293	0.506
10	1.1386865	0.635
12	1.1401475	0.764
14	1.1416122	0.894
16	1.1430807	1.023
18	1.144553	1.154
20	1.146029	1.284



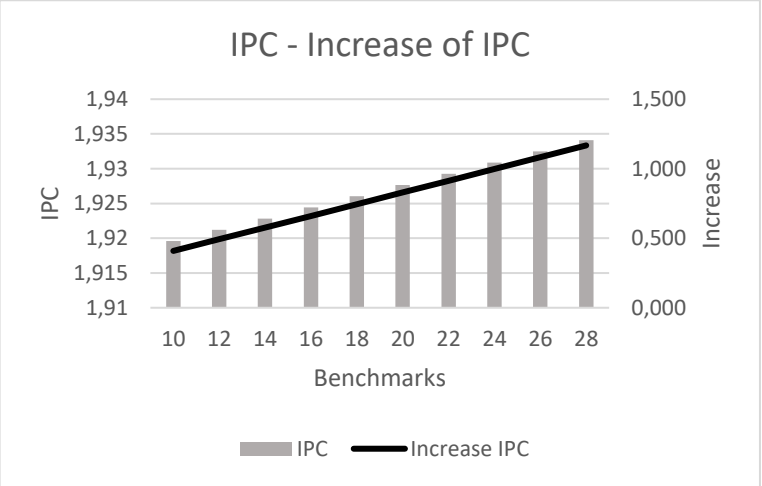
403.gcc

Benefit (Cycles)	IPC	Increase IPC
10	2.325371977	0.262
12	2.326612653	0.315
14	2.327854654	0.369
16	2.329097981	0.422
18	2.330342638	0.476
20	2.331588625	0.530
22	2.332835945	0.584
24	2.334084601	0.637
26	2.335334594	0.691
28	2.336585927	0.745



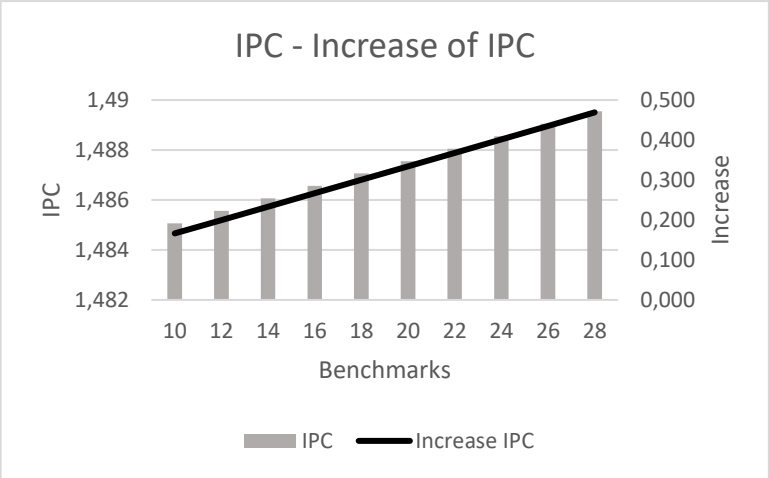
416.games

Benefit (Cycles)	IPC	Increase IPC
10	1.919617353	0.409
12	1.921217799	0.493
14	1.922820916	0.576
16	1.92442671	0.660
18	1.926035189	0.745
20	1.927646358	0.829
22	1.929260226	0.913
24	1.930876798	0.998
26	1.932496081	1.083
28	1.934118083	1.167



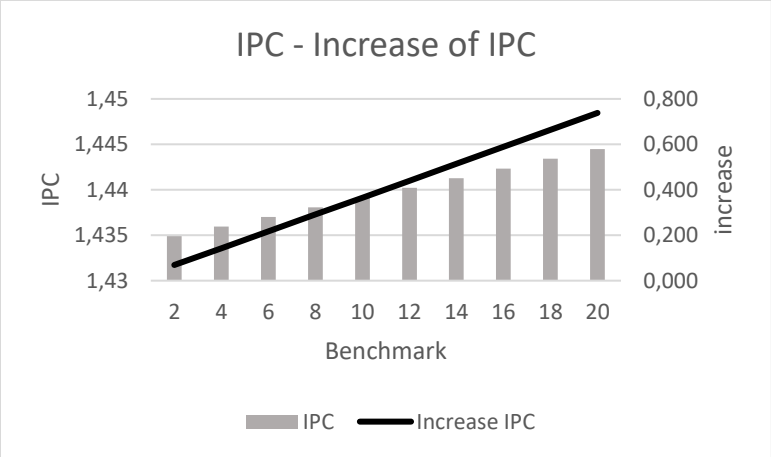
445.gobmk

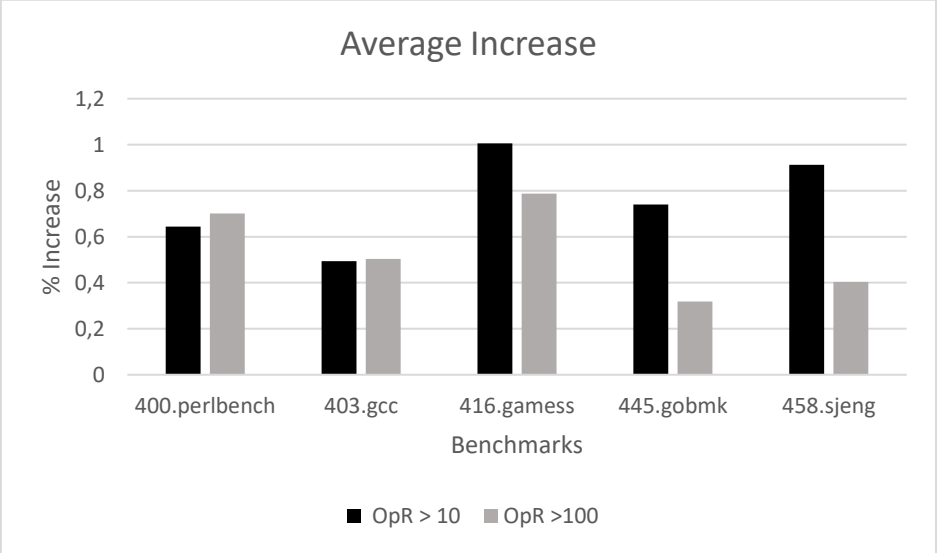
Benefit (Cycles)	IPC	Increase IPC
10	1.485067007	0.166
12	1.485564364	0.200
14	1.486062055	0.234
16	1.486560079	0.267
18	1.487058437	0.301
20	1.487557129	0.334
22	1.488056156	0.368
24	1.488555518	0.402
26	1.489055215	0.435
28	1.489555248	0.469



458.sjeng

Benefit (Cycles)	IPC	Increase IPC
2	1.434889484	0.069
4	1.435949657	0.143
6	1.437011398	0.217
8	1.43807471	0.291
10	1.439139597	0.365
12	1.440206063	0.440
14	1.44127411	0.514
16	1.442343742	0.589
18	1.443414963	0.664
20	1.444487777	0.738





Σχ. 5.9 Στην πιο πάνω γραφική παράσταση συγκρίνετε η μέση αύξηση στο IPC των benchmarks, με opportunity ratio >10 και >100

Στην πιο πάνω γραφική παράσταση (Σχ. 5.9) φαίνεται η διαφορά στην άυξηση του IPC και για τις δύο τιμές που χρησιμοποιήσαμε σαν opportunity ratio. Παρατηρούμε ότι στα benchmark 400.perlbench και 403.gcc με opportunity ratio > 100 υπάρχει μια βελτίοση στην μέση άυξηση του IPC σε σχέση με opportunity ratio > 10. Αυτό γίνεται για το λόγο ότι στα συγκεκριμένα benchmark μειώνονται οι περιπτώσεις που θα προκαλούσαν mispredict στο ορθό μονοπάτι αλλά δεν μειώνονται τα πιθανά wrong path mispredicts καθώς και στις δύο τιμές του opportunity ratio με βάση το μαθηματικό μοντέλο τα πιθανά wrong path mispredicts είναι περισσότερα από τα συνολικά commit mispredicts του προγράμματος, άρα επιλέγονται αυτά σαν το μέγιστο αριθμό που θα εφαρμοστεί η ιδέα και το mispredict θα βρίσκεται όντως στο ορθό μονοπατι. Για τα υπόλοιπα benchmark παρατηρούμε ότι η βελτείωση στο >10 είναι πιο μεγάλη. Αυτό γίνεται για το λόγο τα μειώνοντια πολύ τα πιθανα wrong path mispredicts, γιατί δεν υπάρχουν πολλές εντολές με opportunity ration > 100.

Άρα μπορούμε να καταλήψουμε στο συμπέρασμα ότι με opportunity ratio > 10 τα αποτελέσματα είναι καλύτερα στις πλείστες των περιπτώσεων και αυτό έχει να κάνει σχέση με τον αριθμό των εντολών οι οποίες ικανοποιούν το κατώφλι που θέτουμε σε κάθε περίπτωση. Αν το κατώφλι είναι πολύ ψηλό οι εντολές διακλάδωσης που θα μπορούσαν να το ικανοποιήσουν είναι περιορισμένες. Αντίθετα αν είναι πολύ χαμηλό τόσο περισσότερο κόστος καλείται να πληρώσει ο επεξεργαστής.

Κεφάλαιο 6 – Conclusions

Συνοψίζοντας καταλαμβαίνουμε ότι η τεχνική που προτείναμε στη συγκεκριμένη δούλεια θεωριτικά θα μπορούσε να αποφέρει βελτίωση στην επίδοση του επεξεργαστή, που είναι το πιο σημαντικό σημείο της λειτουργείας κάποιού σύγχρονου επεξεργαστή. Φαίνεται ότι τουλάχιστον για πέντε benchmarks από την συγκεκριμένη τεχνική κερδίζουμε ένα ψηλό αριθμό σε κύκλους, η άυξηση του IPC μπορεί να αυξηθεί μέχρι και 1%. Μπορεί σε κάποιες περιπτώσεις να είδαμε θετικά αποτελέσματα όμως στο σύνολο των benchmarks παρουσιάζεται κάποιο πρόβλημα. Αυτό είναι ότι ο αριθμός των εντολών διακλάδωσης που θα μπορούσαμε να εκμεταλλευτούμε, και μπορει να παρουσιάζουν WPM στην συμπεριφορά τους είναι περιορισμένος. Χρησιμοποιήσαμε στο σύνολο 23 benchmarks για να τρέξουμε τα πειραματα μας και να εξάγουμε κάποια συμπεράσματα. Μόνο για πέντε από τα 23 benchmarks παρουσιάζεται ένας αριθμός εντολών που φαίνεται ότι επηρεάζουν την συνολική επίδοση του επεξεργαστή. Αν πούμε ότι φτάνουμε στο συμπέρασμα ότι από της συνολικές εφαρμογές που εκτελεί ένας επεξεργαστής η μία στις πέντε εμφανίζει την συγκεκριμένη συμπεριφορά σε ένα ικανοποιητικό αριθμό εντολών αυτό είναι ένα κίνητρο να υλοποιηθεί η ιδέα ή θα μπορούσαμε να πούμε ότι τελικά δεν αξίζει αφού ανταποκρίνεται μόνο σε ένα μικρό ποσοστό εντολών.

Επίσης στο θεωρητικό επίπεδο που αναλύσαμε λαμβάνουμε το καλύτερο σενάριο στο οποίο σε όλα τα mispredicts του wrong path εντοπίζουμε ότι βρισκόμαστε σε wrong path και το διορθώνουμε πριν κανονικά το εντοπίσουμε. Αυτό στο πρακτικό κομμάτι δεν ξέρουμε κατά πόσο μπορεί να ισχύει για το λόγο ότι δυο wrong path mispredict μπορούν να προκλήθηκαν από το ίδιο commit mispredict. Άρα αυτόματα διορθώνουμε μόνο ένα. Αυτό φαίνεται καλύτερα στο 400.perlbench όπου τα συνολικά commit mispredicts του benchmark είναι πιο λίγα από τα wrong path mispredicts που εντοπίζουμε. Στην βέλτιστη περίπτωση την οποία και χρησιμοποιήσαμε στην ανάλυση πιο πάνω, υποθέτουμε ότι το κάθε commit mispredict προκαλεί ακριβώς ένα wrong path mispredict το οποίο και μας βοηθά να συμπεραίνουμε ότι βρισκόμαστε σε λάθος μονοπάτι. Αυτό όμως δεν ξέρουμε κατά πόσο ισχύει αυτό κάθε αυτό και στο πρακτικό κομμάτι.

Κεφάλαιο 7 – Future Work

Στόχος μας είναι να συνεχίσουμε την εξέλιξη της ιδέας μας έτσι ώστε να μπορέσουμε να καταλάβουμε κατά πόσο τα θετικά συμπεράσματα που παρατηρήσαμε για κάποια benchmarks ισχυούν και σε πρακτικό επίπεδο. Αρχικά θα πρέπει να αναλύσω τις εντολές διακλάδωσης οι οποίες παρουσιάζουν την συγκεκριμένη συμπεριφορά για να μπορέσουμε να καταλάβουμε σε μορφή assembly η ακόμα και σε επίπεδο source code τι δομή του κώδικα που μπορεί να προκαλέσει αυτή την συμπεριφορά.

Επείτα θα τροποποιήσουμε τον προσωμειωτή με τον οποίο τρέξαμε τα πειράματα μας έτσι ώστε να μπορέσουμε να προσθέσουμε το μηχανισμό μας και να αξιολογήσουμε όντως τα αποτελέσματα μας και σε πρακτικό επίπεδο. Αν παρατηρήσουμε θετική συμπεριφορά στα προγράμματα γιατι όχι να μην υλοποιηθεί η ιδέα μας και να γίνει μέρος των σύγχρονων επεξεργαστών.

Επίσης καταλμαβαίνουμε ότι στην στατιστική ανάλυση των εντολών που εκτελούνται στο ορθό μονοπάτι και αυτές που εκτελούνται στο λάθος μονοπάτι υπάρχουν ακόμα πολλές συμπερίφορες που θα μπορούσαμε να χαρακτηρίσουμε σαν Wrong Path Event. Απώτερος σκοπός είναι να μπορούμε είτε με την τεχνική που προτείναμε η με κάποια άλλη τεχνική να εντοπίζουμε όσο πιο γρήγορα μπορούμε, ότι εκτελόνται εντολές που είναι στο λάθος μονοπάτι και να επαναφέρω την ορθή κατάσταση στο pipeline του επεξεργαστή όσο το δυνατό πιο γρήγορα.

Στο μέλλον επιβάλλεται η εξέταση εναλλακτικών τεχνικών για την επίτευξη αυτού του στόχου. Κάποιο παράδειγμα με το οποίο θα μπορούσαμε να ασχοληθούμε στο μέλλον είναι για τις εντολές με μικρό opportunity ratio. Μπορούμε να συμπαράνουμε ότι μια εντολή με μικρό opportunity ratio δεν βρίσκεται μετά από κάποιο mispredict κάποιας άλλης εντολής (Δηλαδή δεν είναι σε wrong path). Άρα αν παρουσιάσει μια τέτοια εντολή mispredict, σημαίνει δεν πρέπει να υπάρχει μια εντολή branch πιο παλιά απο αυτή που ακόμα να εκτελείται. Αν υπάρχει θα μπορούσαμε να υποθέσουμε ότι η εντολή αυτή βρίσκεται σε wrong path;

Βιβλιογραφία

[1] J. Smith and G. Sohi, Superscalar Processors Microarchitecture, TR UW 1995

[2] S. McFarling. Combining branch predictors. Technical Report TN-36, Digital Western Research Laboratory, June 1993

[3] A. Seznec, The L-Tage Branch Predictor, JILP 2007

[4] A. Seznec, TAGE-SC-L Branch Predictors Again, CBP2016

[5] David N. Armstrong Hyesoon Kim Onur Mutlu Yale N. Patt, Wrong Path Events: Exploiting Unusual and Illegal Program Behavior for Early Misprediction Detection and Recovery