Ατομική Διπλωματική Εργασία

PREDICTION OF WRONG PATH EXECUTION AND SPECULATIVE BRANCH RESOLUTION TO REDUCE THE BRANCH MISPREDICTION PENALTY IN AN OUT-OF-ORDER CORE

Κωνσταντίνος Δημητρίου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ιανουάριος 2021

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Prediction of Wrong Path Execution and Speculative Branch Resolution to Reduce the Branch Misprediction Penalty in an Out-of-Order Core

Κωνσταντίνος Δημητρίου

Επιβλέπων Καθηγητής
Δρ. Γιαννάκης Σαζείδης

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

Ιανουάριος 2021

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου, Δρ. Γιαννάκη Σαζεΐδη, για την ευκαιρία που μου έδωσε να εκπονήσω μαζί του την διπλωματική μου εργασία, σε ένα τόσο ενδιαφέρον και σημαντικό θέμα. Επίσης, θα ήθελα να τον ευχαριστήσω για την εμπιστοσύνη που μου έδειξε και με ένταξε στην ομάδα του. Η συνεχής καθοδήγηση και οι άπειρες συμβουλές που μου έδωσε του κατά τη διάρκεια της εργασίας μου ήταν καταλυτικής σημασίας. Ο τρόπος σκέψης και οι τεχνικές που μου έχει μεταλαμπαδεύσει αποτέλεσαν καθοριστικό παράγοντα στην επίτευξη των στόχων μου. Τέλος, θα ήθελα να ευχαριστήσω τα μέλη του ερευνητικού εργαστηρίου Ξ-Computer Architecture για το φιλικό περιβάλλον κατά τη διάρκεια των συναντήσεων μας που με έκανε να νιώθω άνετα και να μπορώ να εκφραστώ ελεύθερα.

Περίληψη

Οι επεξεργαστές σήμερα αποτελούν ένα από τα σημαντικότερα συστατικά ενός υπολογιστικού συστήματος. Η σχεδίαση σύγχρονων επεξεργαστών απαιτεί τη συνεχή αύξηση της επίδοσης και την μείωση της κατανάλωσης ενέργειας. Αυτό απαιτεί από τους σχεδιαστές την συνεχή έρευνα για την ανάπτυξη νέων μηχανισμών βελτιστοποίησης την επίδοσης των επεξεργαστών.

Οι μηχανισμοί πρόβλεψης διακλαδώσεων αποτελούν ένα από τα πιο κρίσιμα μέρη 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 υπάρχουν σημαντικές καθυστερείς, περιμένοντας την σωστή ακολουθία εντολών για εκτέλεση. Δυστυχώς, μερικές φορές οι προβλέψεις των μηχανισμών πρόβλεψης διακλαδώσεων είναι λανθασμένες και έτσι εκτελούνται εντολές οι οποίες βρίσκονται στο λάθος μονοπάτι ροής της εκτέλεσης του προγράμματος. Δηλαδή εκτελούνται αχρείαστες εντολές άρα εκτελούνται περισσότεροι κύκλοι μηχανής κάτι που προσδίδει σημαντικό κόστος στην επίδοση του επεξεργαστή.

Στόχος της παρούσας διπλωματικής εργασίας είναι η διερεύνηση και η εφαρμογή μίας τεχνικής που θα μπορούσε να συνεισφέρει στην αύξηση της επίδοσης ενός μοντέρνου επεξεργαστή. Συγκεκριμένα διερευνήθηκε η εφαρμογή ενός μηχανισμού πρόβλεψης ο οποίος θα προβλέπει εάν κάποια εντολή είχε λανθασμένη πρόβλεψη διακλάδωσης, πριν αυτή ολοκληρώσει την εκτέλεση της. Με αυτόν τον τρόπο εντοπίζεται η εντολή που προκάλεσε το λάθος μονοπάτι εκτέλεσης και σε μια τέτοια περίπτωση σταματούμε την λανθασμένη κερδοσκοπική εκτέλεση σε εκείνο το σημείο, αποφεύγοντας να πληρώσουμε ολόκληρο το κόστος.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Κίνητρο

Στις μέρες μας η ανθρώπινη ζωή εξαρτάται σε μεγάλο βαθμό από τα υπολογιστικά συστήματα. Αναπόσπαστο στοιχείο ενός υπολογιστικού συστήματος είναι ο επεξεργαστής (CPU). Οι επεξεργαστές βρίσκονται παντού, στους ηλεκτρονικούς υπολογιστές, στα κινητά μας τηλέφωνα, στις τηλεοράσεις, στα αυτοκίνητα και σε κάθε είδους ηλεκτρική συσκευή. Οι επεξεργαστές συντονίζουν και εκτελούν όλες εκείνες τις πράξεις οι οποίες δίνουν την δυνατότητα στην κάθε συσκευή να εκτελέσει το έργο της.

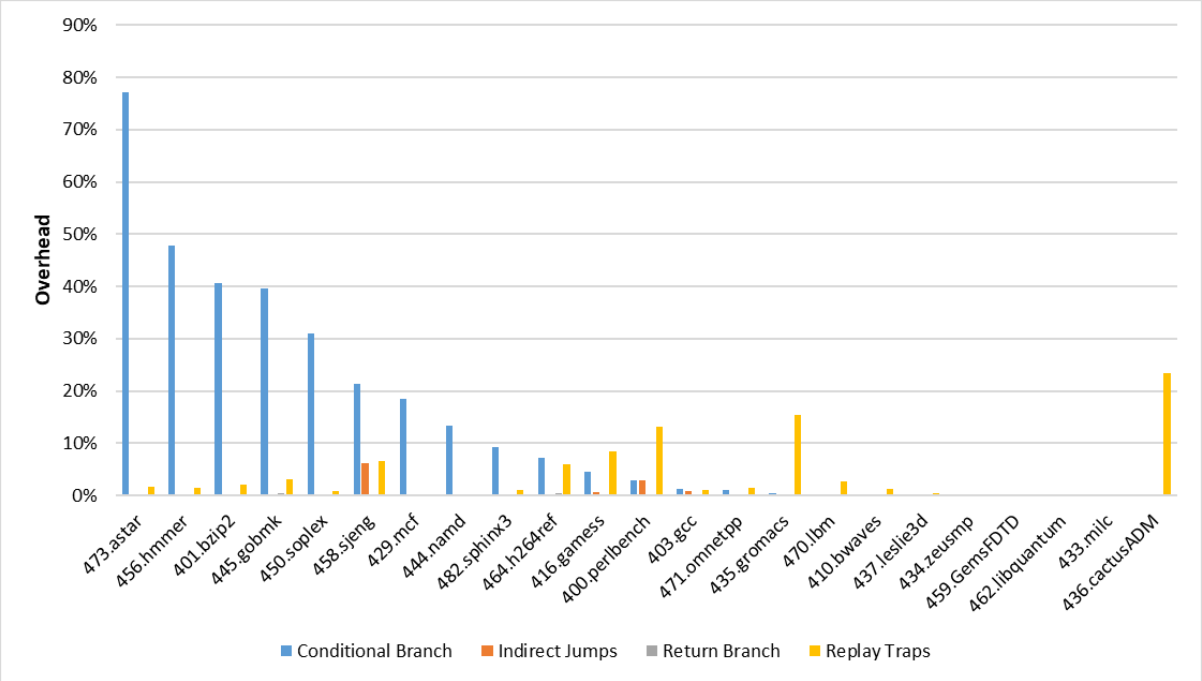
Οι μοντέρνοι επεξεργαστές περιέχουν περισσότερους από έναν πυρήνα (core) επιτρέποντας έτσι στον ίδιο τον επεξεργαστή να εκτελεί περισσότερα από ένα προγράμματα παράλληλα. Κάθε πυρήνας χρησιμοποιεί μηχανισμούς όπως η διασωλήνωση (pipeline) και η εκτέλεση εντολών εκτός σειράς (out-of-order execution) για αύξηση της επίδοσης του επεξεργαστή. Επίσης, χρησιμοποιούν τεχνικές όπως η κερδοσκοπική εκτέλεση (speculative execution) έτσι ώστε να υποστηρίζεται η εκτέλεση εντολών οι οποίες δεν είναι απόλυτα σίγουρος ο επεξεργαστής ότι βρίσκονται στο σωστό μονοπάτι εκτέλεσης του προγράμματος.

Ίσως ο σημαντικότερος μηχανισμός για την επιτυχία τη κερδοσκοπικής εκτέλεσης είναι ο μηχανισμός πρόβλεψης διακλαδώσεων (branch predictor). Σκοπός του branch predictor είναι η πρόβλεψη του αποτελέσματος μιας διακλάδωσης. Αυτό είναι πολύ σημαντικό διότι, όταν εισέλθει μια εντολή διακλάδωσης στον επεξεργαστή δεν μπορούμε να γνωρίζουμε τη διεύθυνση της επόμενης εντολής που θα εκτελεστεί μέχρι να τελειώσει η εκτέλεση της διακλάδωσης. Αυτό μπορεί να κοστίσει αρκετούς κύκλους μηχανής. Με σκοπό την αύξηση της επίδοσης, ο branch

predictor προσπαθεί να προβλέψει ποια θα είναι η επόμενη εντολή έτσι ώστε να επωφεληθεί από τους χαμένους κύκλους.

Δυστυχώς, μερικές φορές οι προβλέψεις του branch predictor είναι λανθασμένες (mispredict) και έτσι εκτελούνται εντολές οι οποίες βρίσκονται στο λάθος μονοπάτι ροής της εκτέλεσης του προγράμματος. Αυτό το φαινόμενο ονομάζεται **wrong path execution**. Δηλαδή εκτελούνται αχρείαστες εντολές άρα εκτελούνται περισσότεροι κύκλοι μηχανής κάτι που προσδίδει σημαντικό κόστος στην επίδοση του επεξεργαστή.

Πέραν από καθυστερήσεις που προκαλούν τα mispredicts στα conditional branch καθυστερήσεις προκαλούν και τα mispredicts των indirect jumps, των return branch και των replay traps (memory dependance prediction [1]). Στην παρούσα εργασία έγινε σύγκριση του κόστους των καθυστερήσεων των πιο πάνω mispredicts. Με βάση τη γραφική παράσταση που απεικονίζεται στο Σχήμα 1.1 μπορούμε να δούμε ότι το μεγαλύτερο κόστος προκαλείται από τα mispredicts των conditional branches. Για παράδειγμα, το benchmark *astar*, χάνει το 77% του χρόνου εκτέλεσης του στην εκτέλεση εντολών στο λάθος μονοπάτι. Ως εκ τούτου στην παρούσα εργασία αποφασίσαμε να επικεντρωθούμε στα conditional branch mispredicts.



Σχήμα 1.1: Σύγκριση του κόστους που προκαλούν τα mispredicts

Για να καταφέρουμε να μειώσουμε τους κύκλους μηχανής που σπαταλούνται στο λάθος μονοπάτι, θα πρέπει είτε να επιτύχουμε βελτίωση της επίδοσης των branch predictors, είτε να μειώσουμε το συνολικό penalty που προκαλούν τα conditional branch mispredicts. Λόγω του ότι έχει γίνει ήδη πολλή έρευνα στην περιοχή των branch predictors, στην παρούσα μελέτη εστιάσαμε στη μείωση του penalty που προκαλούν τα conditional branch mispredicts.

1.2 Ιδέα

Ο κύριος στόχος της διπλωματικής εργασίας είναι η υλοποίηση ενός μηχανισμού πρόβλεψης (**wrong path predictor**) ο οποίος θα προβλέπει εάν κάποια εντολή είχε λανθασμένη πρόβλεψη διακλάδωσης δηλαδή εάν ήταν mispredict, πριν αυτή ολοκληρώσει την εκτέλεση της. Με αυτόν τον τρόπο εντοπίζεται η εντολή που προκάλεσε το λάθος μονοπάτι εκτέλεσης. Σε μια τέτοια περίπτωση σταματούμε το λανθασμένο speculative execution σε εκείνο το σημείο, αποφεύγοντας να πληρώσουμε ολόκληρο το κόστος.

Η βασική ιδέα για επίτευξη του στόχου αυτού, βασίζεται στο ότι κατά την διάρκεια εκτέλεσης του λανθασμένου μονοπατιού, συμβαίνουν γεγονότα μέσα στον επεξεργαστή τα οποία μπορούν να μαρτυρήσουν το γεγονός ότι το speculative βρίσκεται σε λάθος μονοπάτι. Εκτιμούμε ότι αυτά τα γεγονότα πιθανόν να εμφανιστούν μέσω κάποιων features τα οποία θα συλλέξουμε και αφορούν πληροφορίες μετά το renaming stage στο pipeline.

1.3 Συνεισφορά

Στα πλαίσια της διπλωματικής εργασίας υλοποιήθηκε ένας wrong path predictor ο οποίος ενσωματώθηκε σε έναν προσομοιωτή. Επίσης, χρησιμοποιήθηκαν features μετά το renaming stage στο pipeline για την εκπαίδευση του predictor. Αξίζει να σημειωθεί ότι αξιολογήθηκε η επίδοση του επεξεργαστή και η ακρίβεια του predictor μετά την λειτουργία του wrong path predictor. Στα ευρήματα της παρούσας εργασίας μπορούμε να αναφέρουμε ότι το λάθος μονοπάτι είναι προβλέψιμο με ακρίβεια 52%-83% ανάλογα με το benchmark που εξετάζεται και ότι υπάρχει γραμμική συσχέτιση ανάμεσα στην ακρίβεια και την επιτάχυνση. Δεν καταφέραμε να πετύχουμε επιτάχυνση για τις παραμέτρους που εξετάσαμε αλλά εκτιμούμε ότι με περαιτέρω διερεύνηση του προβλήματος θα πετύχουμε αύξηση της επίδοσης.

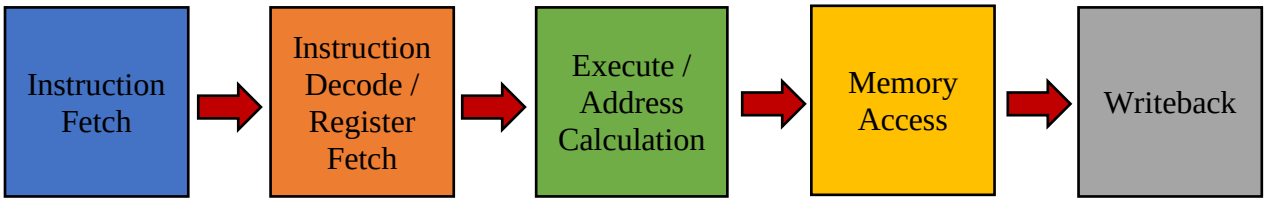
Κεφάλαιο 2

Θεωρητικό υπόβαθρο

2.1 Pipeline

Στους πρώτους επεξεργαστές που δημιουργηθήκαν οι εντολές εκτελούνταν σειριακά. Δηλαδή για να ξεκινήσει η εκτέλεση κάποιας εντολή θα έπρεπε να περιμένει όλες τις πιο παλιές εντολές να ολοκληρώσουν την διαδικασία εκτέλεσης τους για να μπορέσει να χρησιμοποιήσει τους πόρους του επεξεργαστή, η εντολή αυτή. Με αυτόν τον τρόπο δημιουργούνταν σημαντικές καθυστερήσεις στην επίδοση του επεξεργαστή.

Οι μοντέρνοι επεξεργαστές για να μπορούν να εκτελούν τις διάφορες εντολές με αποδοτικότερο τρόπο χρησιμοποιούν τη διασωλήνωση (**pipeline**) [2]. Το pipeline επιτρέπει την παράλληλη εκτέλεση εντολών οι οποίες βρίσκονται σε διαφορετικά στάδια. Ουσιαστικά η επεξεργασία κάποιας εντολής χωρίζεται σε διάφορα στάδια και όταν μια πιο παλιά εντολή ολοκληρώσει την εκτέλεση της στο πρώτο στάδιο της επεξεργασίας και προχωρήσει στο δεύτερο, η επόμενη εντολή χρησιμοποιεί τους πόρους του πρώτου σταδίου επεξεργασίας. Το Σχήμα 2.1 παρουσιάζει τα στάδια ενός pipeline.



Σχήμα 2.1 Μοντέλο Pipeline

Ένα μοντέλο pipeline αποτελείται από τα εξής στάδια εκτέλεσης μιας εντολής:

- **Instruction Fetch**: Φόρτωση της επόμενης εντολής η οποία πρόκειται να εκτελεστεί.
- **Instruction Decode / Register Fetch**: Αποκωδικοποίηση της εντολής και φόρτωση των τιμών από τους απαιτούμενους καταχωρητές.
- **Execute / Address Calculation**: Εκτέλεση της εντολής ή υπολογισμός διεύθυνσης.
- **Memory Access**: Προσπέλαση και ανάγνωση μνήμης.
- **Writeback**: Αποθήκευση του αποτελέσματος στον καταχωρητή.

2.2 Pipeline Hazards

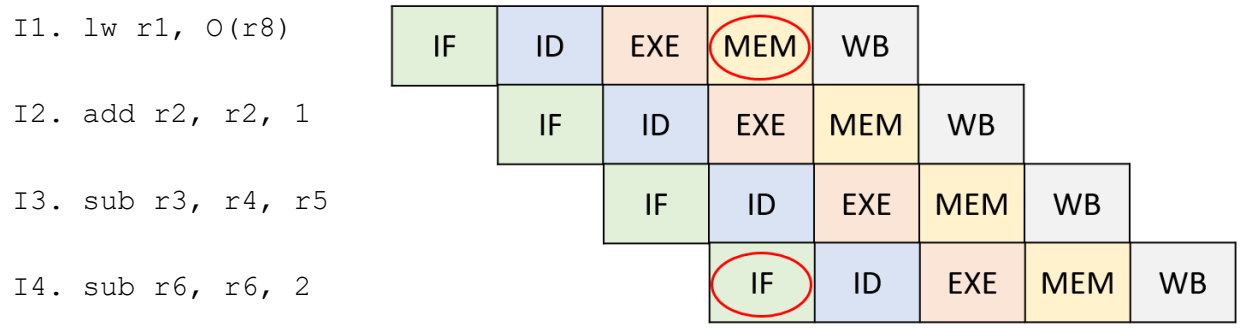
Παρά το γεγονός ότι το pipeline προσφέρει αρκετά πλεονεκτήματα και επιδιώκει στη βελτίωση της επίδοσης του επεξεργαστή, εντούτοις η αύξηση του παραλληλισμού στην εκτέλεση των εντολών μερικές φορές δημιουργεί τρεις πολύ σημαντικούς κινδύνους (**hazards**) [2].
Αυτοί οι κίνδυνοι είναι οι ακόλουθοι:

- Δομικοί Κίνδυνοι (Structural Hazards)
- Κίνδυνοι Δεδομένων (Data Hazards)
- Κίνδυνοι Ροής Ελέγχου (Control Hazards)

Υπάρχουν διάφορες μέθοδοι για αντιμετώπιση αυτών των κινδύνων έτσι ώστε να αποφευχθεί η παραγωγή λανθασμένου αποτελέσματος από τον επεξεργαστή. Η πιο απλή μέθοδος είναι η εισαγωγή stalls (ή bubbles), όπου εισάγονται κενές εντολές NOP (No-Operation) για να δημιουργήσουν κάποια καθυστέρηση μεταξύ των εντολών που δημιουργούν τα hazards, έτσι ώστε να μην υπάρχει πλέον κίνδυνος.

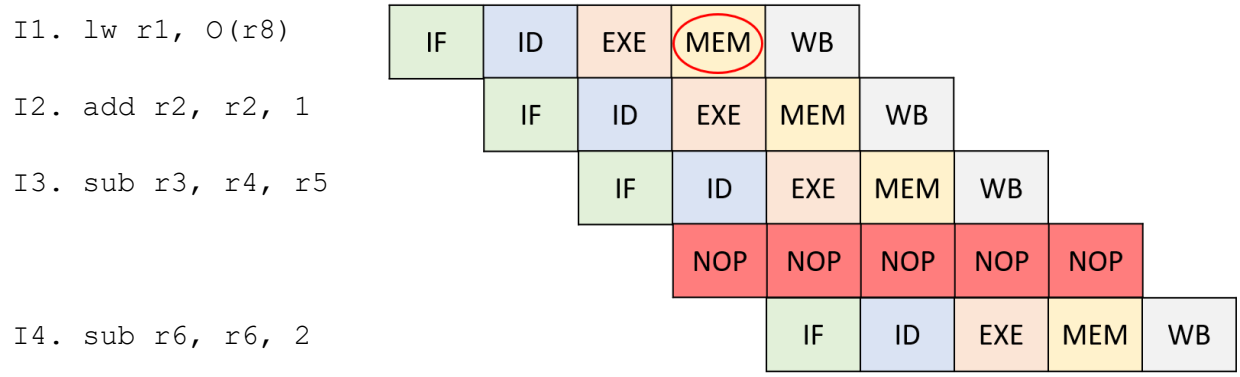
2.2.1 Structural Hazards

Οι δομικοί κίνδυνοι (structural hazards) εμφανίζονται όταν συνδυασμοί εντολών δεν μπορούν να εκτελεστούν ταυτόχρονα διότι χρειάζονται περισσότερους πόρους από αυτούς που διαθέτει η μηχανή. Για παράδειγμα αυτό μπορεί να συμβεί εάν ο επεξεργαστής διαθέτει μόνο μία κοινή πόρτα μνήμης για διάβασμα/γράψιμο σε μια μονάδα μνήμης ή στο αρχείο καταχωρητών. Κατά συνέπεια δύο εντολές οι οποίες βρίσκονται σε διαφορετικά τμήματα της διασωλήνωσης δεν μπορούν να πραγματοποιήσουν ταυτόχρονη πρόσβαση σε αυτήν. Το Σχήμα 2.2 παρουσιάζει ένα παράδειγμα structural hazard.



Σχήμα 2.2: Παράδειγμα structural hazard

Το πιο πάνω πρόβλημα μπορεί να λυθεί με την εισαγωγή stalls (ή bubbles), δηλαδή κενές εντολές NOP οι οποίες δημιουργούν καθυστερήσεις. Το Σχήμα 2.3 παρουσιάζει την εισαγωγή stalls στο πιο πάνω παράδειγμα.



Σχήμα 2.3: Παράδειγμα αντιμετώπισης structural hazard με την εισαγωγή stalls

2.2.2 Data Hazards

Τα data hazards εμφανίζονται όταν εντολές, οι οποίες βρίσκονται σε σχετικά μικρή απόσταση εκτέλεσης, έχουν μεταξύ τους εξάρτηση. Με άλλα λόγια μια προγραμματισμένη εντολή δεν μπορεί να εκτελεστεί στον κατάλληλο κύκλο ρολογιού επειδή τα δεδομένα που χρειάζεται για να εκτελεστεί η εντολή αυτή δεν είναι ακόμη διαθέσιμα.

Τα data hazards χωρίζονται σε 3 κατηγορίες:

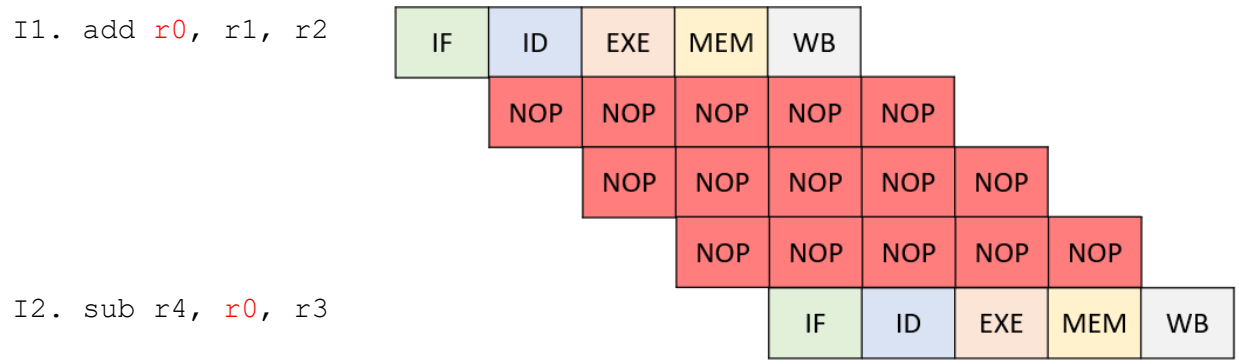
- RAW (Read After Write)
- WAW (Write After Write)
- WAR (Write After Read)

Για σκοπούς καλύτερης επεξήγησης ας υποθέτουμε ότι έχουμε δύο εντολές την I1 και την I2, με την I2 να εκτελείται πριν από την I1.

RAW (Read After Write): η εντολή I2 προσπαθεί να διαβάσει μια τιμή από ένα καταχωρητή πριν η I1 γράψει σε αυτόν. Επομένως, λανθασμένα η I2 θα πάρει την παλιά τιμή. Οι εξαρτήσεις RAW χαρακτηρίζονται ως true data dependence.

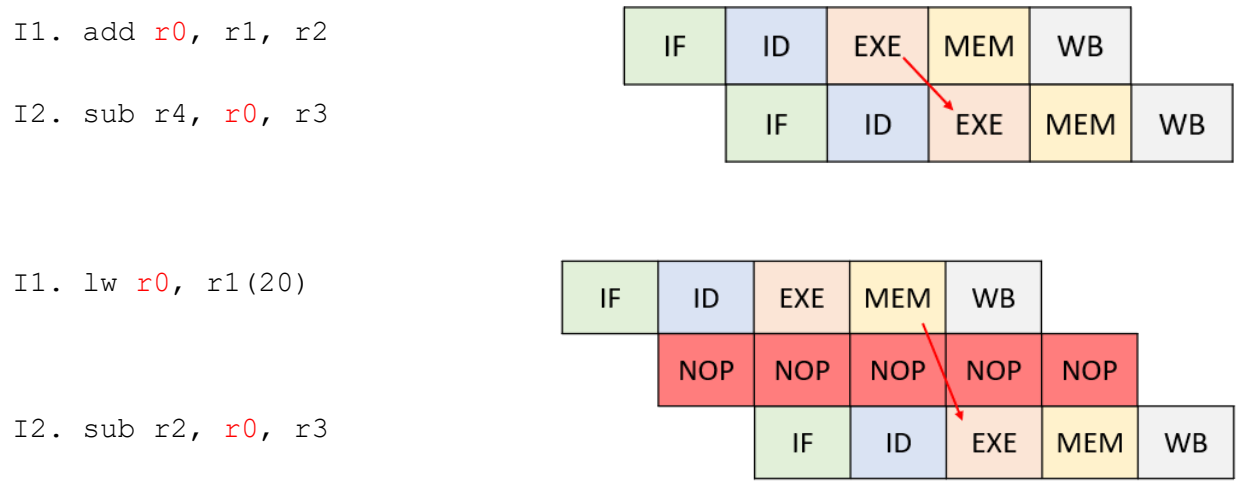
```
I1. add r0, r1, r2          ή        I1. lw r0, r1(20)
I2. sub r4, r0, r3                   I2. sub r2, r0, r3
```

Το Σχήμα 2.4 δείχνει τη λύση στην εξάρτηση RAW με την εισαγωγή stalls (ή bubbles).



Σχήμα 2.4: Αντιμετώπιση RAW με την εισαγωγή stalls

Forwarding ή Bypassing είναι μια μέθοδος για την επίλυση των data hazards με την ανάκτηση του στοιχείου δεδομένων που λείπει από εσωτερικές προσωρινές μνήμες, αντί να το περιμένουμε να αφιχθεί από ορατούς στον προγραμματιστή καταχωρητές ή από τη μνήμη. Το Σχήμα 2.5 δείχνει τη λύση στην εξάρτηση RAW με forwarding.



Σχήμα 2.5: Αντιμετώπιση RAW με forwarding

WAW (Write After Write): η εντολή I2 προσπαθεί να γράψει σε έναν καταχωρητή πριν να γραφτεί από την εντολή I1. Οι εγγραφές καταλήγουν να εκτελούνται με λάθος σειρά, αφήνοντας την τιμή που γράφτηκε από το I1 και όχι την τιμή που γράφτηκε από τον I2 στον καταχωρητή. Αυτό συμβαίνει μόνο σε διασωληνώσεις που γράφουν σε περισσότερα από ένα pipe stage. Οι εξαρτήσεις WAW χαρακτηρίζονται ως false data dependency.

```
I1. add r1, r4, 5
I2. add r1, r2, r3
```

WAR (Write After Read): η εντολή I2 προσπαθεί να γράψει έναν καταχωρητή πριν αυτός διαβαστεί από την εντολή I1 και έτσι παίρνω λανθασμένη τη νέα τιμή. Αυτό συμβαίνει μόνο σε διασωληνώσεις που γράφουν σε περισσότερα από ένα pipe stage. Οι εξαρτήσεις WAR χαρακτηρίζονται ως false data dependency.

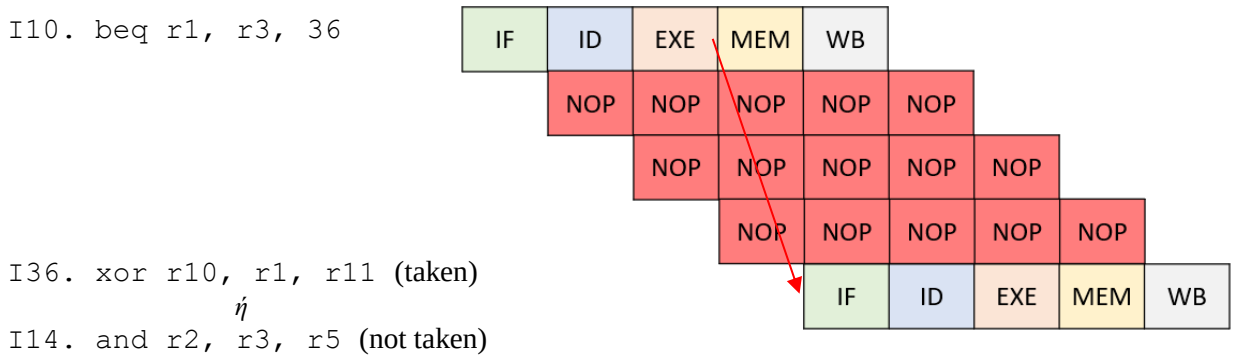
```
I1. add r4, r1, 4
I2. add r1, r2, r3
```

2.2.3 Control Hazards

Οι κίνδυνοι ροής ελέγχου (**control hazards**) εμφανίζονται κατά την είσοδο εντολών διακλάδωσης στο pipeline. Συγκεκριμένα το πρόβλημα προκύπτει όταν ο επεξεργαστής πρέπει να ορίσει το επόμενο PC και δεν ξέρει αν η διακλάδωση θα γίνει taken ή not-taken. Μια μέθοδος αντιμετώπισης αυτού του κίνδυνου, είναι η εισαγωγή stalls (ή bubbles) μέχρι η εντολή διακλάδωσης να εκτελεστεί και αποκαλυφθεί το επόμενο PC.

```
I10. beq r1, r3, 36
I14. and r2, r3, r5
I18. or r6, r1, r7
I22. add r8, r1, r9
I36. xor r10, r1, r11
```

Το Σχήμα 2.6 δείχνει την αντιμετώπιση των control hazards με εισαγωγή stalls (ή bubbles).



Σχήμα 2.6: Αντιμετώπιση control hazards με την εισαγωγή stalls

2.3 Speculative Execution

Η κερδοσκοπική εκτέλεση (**speculative execution**) [2] είναι η διαδικασία όπου ο μεταγλωττιστής ή ο επεξεργαστής μαντεύουν το αποτέλεσμα μιας εντολής για να την αφαιρέσουν ως μια εξάρτηση κατά την εκτέλεση άλλων εντολών. Η ιδέα πίσω από το speculative execution είναι ότι οι εντολές εκτελούνται πριν γνωρίσουν ότι απαιτούνται. Χωρίς speculative execution, ο επεξεργαστής θα χρειαστεί να περιμένει τις προηγούμενες εντολές να γίνουν resolve πριν εκτελέσουν τις επόμενες. Αυτή η τεχνική βοηθά την επίδοση 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. Ως αποτέλεσμα αυτού αυξάνουμε την συνολική επίδοση του επεξεργαστή. Τα αποτελέσματα μπορεί να απορριφθούν εάν διαπιστωθεί ότι οι εντολές δεν ήταν τελικά απαραίτητες.

Τα είδη του speculative execution είναι:

- *Branch Prediction*: Πρόβλεψη της επόμενης εντολής για conditional, indirect jump και return εντολές.
- *Memory Dependence Prediction*: Πρόβλεψη της διεύθυνσης για load / store
- *Value Prediction*: Πρόβλεψη τιμής που θα πάρει ο καταχωρητής

2.4 Branch Predictor

Σκοπός του μηχανισμού πρόβλεψης διακλαδώσεων (**branch predictor**) [3] είναι η πρόβλεψη του αποτελέσματος μιας διακλάδωσης που θα καθορίσει τη ροή του προγράμματος. 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 γνωρίζουμε τη σωστή κατάσταση της ροής του προγράμματος. Αυτό βοηθά την βελτίωση της επίδοσης του επεξεργαστή καθώς μπορούμε να εκμεταλλευτούμε τους πόρους του επεξεργαστή στο μέγιστο δυνατό, μειώνοντας έτσι τις καθυστερήσεις και τα stalls. Δηλαδή αντί να περιμένει όλες τις εντολές διακλάδωσης να γινουν resolve ο branch predictor, προβλέπει τη ροή ελέγχου. Με τον branch predictor μπορούμε να αντιμετωπίσουμε αποδοτικά τα control hazards που περιγράψαμε πιο πάνω.

Οι state-of-the-art branch predict έχουν πολύ ψηλή ακρίβεια. Δηλαδή συνήθως οι προβλέψεις τους είναι σωστές, πράγμα που επιτρέπει την επίτευξη υψηλών επιδόσεων στον επεξεργαστή. Ωστόσο, εάν μια πρόβλεψη είναι λάθος, τότε η εργασία που εκτελέστηκε speculative απορρίπτεται (γίνεται flush) και ο επεξεργαστής θα ανακατευθυνθεί για να εκτελέσει την σωστή διαδρομή εντολών.

Η βασική ιδέα πίσω από τους predictors είναι ότι κρατάμε διάφορους μετρητές για κάθε εντολή διακλάδωσης. Οι πληροφορίες που χρησιμοποιούν οι branch predictors προέρχονται από στάδια του pipeline πριν το renaming stage. Αυτές οι πληροφορίες είναι κυρίως το global history δηλαδή

το ιστορικό τον πρόσφατων branches και το PC που χρησιμοποιείται κυρίως για το hashing της εντολής.

2.5 Out-of-order Execution

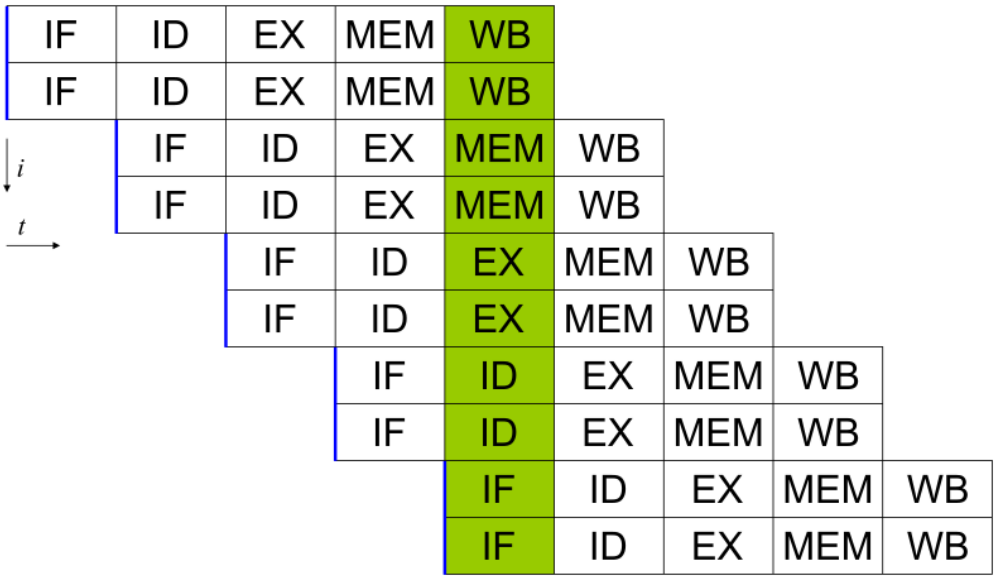
Σε μια **out-of-order execution** [2] οι εντολές μπορούν να εκτελεστούν σε διαφορετική σειρά από αυτή που εμφανίζονται στο dynamic program order. Δηλαδή, αλλάζει την σειρά των εντολών του προγράμματος που εκτελούνται με σκοπό να αποφευχθούν τα stalls.

Η βασική ιδέα του out-of-order execution είναι να επιτρέψει στον επεξεργαστή να αποφύγει τα stalls που συμβαίνουν όταν τα δεδομένα που απαιτούνται για την εκτέλεση μιας εντολής δεν είναι διαθέσιμα. Οι επεξεργαστές που υποστηρίζουν out-of-order execution εκτελούν εντολές που είναι ήδη έτοιμες για να αποφύγουν τις καθυστερήσεις. Εν τέλη παράγεται το ίδιο αποτέλεσμα με αυτό που θα είχε αν ακολουθούσε την σειριακή εκτέλεση των εντολών εφόσον οι εντολές γίνονται commit in-order.

2.6 Superscalar Processor

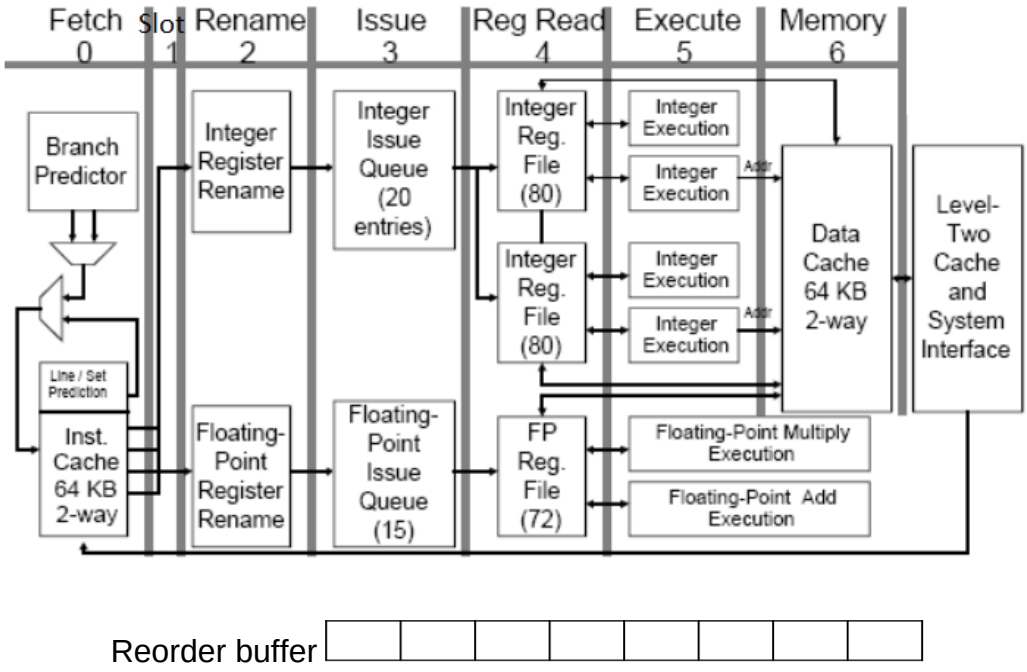
Ένας **superscalar processor** [4] είναι ένα είδος CPU που υλοποιεί μια μορφή παραλληλισμού που ονομάζεται instruction-level parallelism μέσα σε έναν ενιαίο επεξεργαστή. Σε αντίθεση με έναν scalar επεξεργαστή ο οποίος μπορεί να εκτελεί το πολύ μία μόνο εντολή ανά κύκλο ρολογιού, ένας superscalar επεξεργαστής μπορεί να εκτελέσει περισσότερες από μία εντολές κατά τη διάρκεια ενός κύκλου ρολογιού. Επομένως, η τεχνική αυτή συμβάλει σημαντικά στην αύξηση της επίδοσης του επεξεργαστή. Κάθε μονάδα εκτέλεσης δεν είναι ξεχωριστός επεξεργαστής (ή πυρήνας αν ο επεξεργαστής είναι επεξεργαστής πολλαπλών πυρήνων), αλλά ένας πόρος εκτέλεσης μέσα σε μία CPU, όπως μια αριθμητική λογική μονάδα.

Το Σχήμα 2.7 παρουσιάζει ένα διάγραμμα απλού superscalar επεξεργαστή. Κάθε ένα από τα 5 στάδια του pipeline παρουσιάζεται 2 φορές, με αποτέλεσμα να μπορούν να ολοκληρώνονται μέχρι και 2 εντολές σε κάθε κύκλο.



Σχήμα 2.7: Διάγραμμα superscalar επεξεργαστή

Το Σχήμα 2.8 παρουσιάζει τα στάδια του pipeline ενός superscalar επεξεργαστή. Αξίζει να σημειωθεί ότι τα στάδια που θα μας απασχολήσουν στην παρούσα εργασία είναι το fetch στο οποίο γίνεται η πρόβλεψη του branch predictor, το rename, το execute ή resolve στο οποίο εκτελούνται οι εντολές. Οι εντολές μπαίνουν στο reorder buffer από όπου γίνονται και commit.



Σχήμα 2.8: Στάδια pipeline superscalar επεξεργαστή

2.7 Λάθος μονοπάτι

Κατά τη διάρκεια της εκτέλεσης κάποιου προγράμματος υπάρχουν διάφορες εντολές που περνούν από το στάδιο της εκτέλεσης του pipeline, οι πλείστες από αυτές στο τέλος θα γίνουν commit αλλά αρκετές εντολές από αυτές βρίσκονται στο λάθος μονοπάτι που εκτελέστηκαν εξαιτίας κάποιας λανθασμένης πρόβλεψης που έγινε για μια εντολή διακλάδωσης.

Για σκοπούς καλύτερης επεξήγησης ας υποθέσουμε ότι ο κώδικας που φαίνεται στο Σχήμα 2.9 αποτελεί μέρος ενός προγράμματος.

```
I1. if (x > 50)
I2.     if (y < 20)
I3.         y = 0;
I4. else
I5.     if (z < 20)
I6.         z = 0;
```

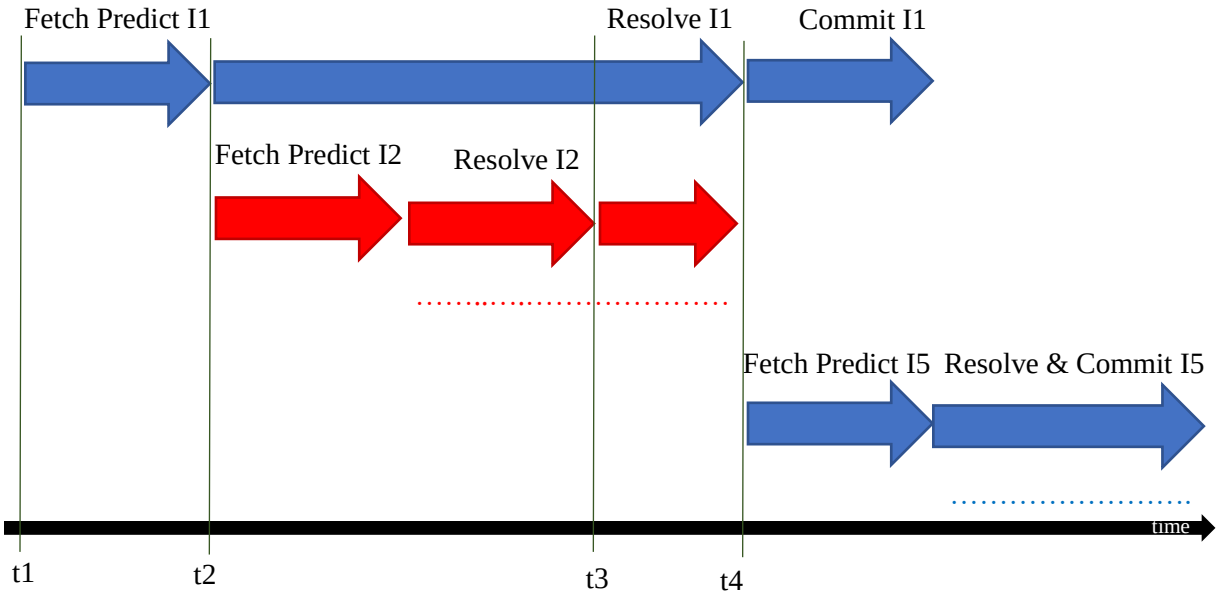
Σχήμα 2.9: Παράδειγμα κώδικα

Ας υποθέσουμε επίσης ότι ο πιο πάνω κώδικας εκτελείται σε έναν out-of-order επεξεργαστή με speculative execution. Επίσης, κατά το fetch της εντολής I1 ο branch predictor προβλέπει ότι η εντολή θα είναι not-taken. Επιπρόσθετα, η εντολή I2 εκτελείτε (γίνεται resolve) πριν την εντολή I1. Αυτό μπορεί να συμβεί διότι η τιμή του καταχωρητή x εξαρτάται από ένα μακρύ μονοπάτι εντολών που την παράγουν ενώ η τιμή του καταχωρητή y είναι έτοιμη ή μπορεί να χρειάζεται να η τιμή του x να έρθει από τη μνήμη ενώ η τιμή του y είναι ήδη στη cache.

Στο Σχήμα 2.10 απεικονίζεται η πορεία εκτέλεσης του πιο πάνω προγράμματος όπως λειτουργούν σήμερα οι επεξεργαστές σε περίπτωση που η πρόβλεψη του branch predictor ήταν λανθασμένη (δηλαδή προκάλεσε conditional branch mispredict).

Τη χρονική στιγμή t1 η εντολή I1 γίνεται fetch και ο branch predictor προβλέπει ότι η εντολή θα είναι not-taken. Επομένως, τη χρονική στιγμή t2 η εντολή I2 γίνεται fetch και μετά από κάποιους κύκλους τη χρονική στιγμή t3 η εντολή γίνεται resolved. Στη συνέχεια, τη χρονική στιγμή t4 η εντολή I1 γίνεται resolved. Στο resolution της I1 ο επεξεργαστής καταλαβαίνει ότι η πρόβλεψη του branch predictor ήταν λανθασμένη. Ως εκ τούτου οι εντολές που εκτελέστηκαν μετά την I1 (δηλαδή η I2 στο παράδειγμα μας) βρίσκονται στο λάθος μονοπάτι και θα πρέπει να γίνουν flush

(δηλαδή να αποβληθούν) από το pipeline. Μόλις ολοκληρωθεί το flush των εντολών στο λάθος μονοπάτι γίνεται fetch και εκτελείται η I5.



Σχήμα 2.10: Διάγραμμα εκτέλεσης εντολών στο λάθος μονοπάτι

Με βάση τα πιο πάνω μπορούμε να ορίσουμε ότι **conditional branch mispredict penalty** είναι ο χρόνος σε κύκλους μηχανής που χάθηκε εκτελώντας εντολές στο λάθος μονοπάτι προσδίδοντας σημαντικό κόστος στην επίδοση του επεξεργαστή. Στο παράδειγμα μας conditional branch mispredict penalty είναι ο αριθμός των κύκλων που ορίζονται από τη διαφορά της χρονικής στιγμής t4 – τη χρονική στιγμή t2.

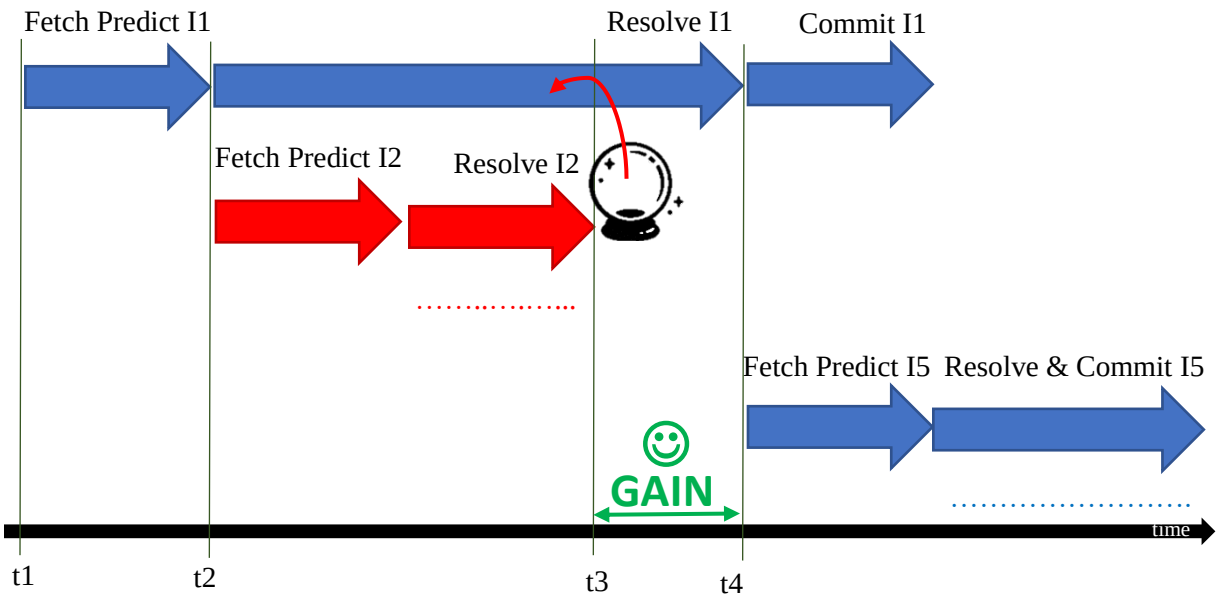
Κεφάλαιο 3

Μηχανισμός πρόβλεψης λάθος μονοπατιού

3.1 Πρόβλεψη λάθος μονοπατιού

Ας υποθέσουμε ότι έχουμε στη διάθεσή μας ένα μηχανισμό (wrong path predictor) ο οποίος κατά το resolution μιας conditional branch εντολής μπορεί να προβλέψει ότι μια οποιαδήποτε άλλη conditional branch εντολή που βρίσκεται στο reorder buffer και ακόμη είναι unresolved, ότι θα γίνει mispredict και ως εκ τούτο έχει προκαλέσει λάθος μονοπάτι. Εφόσον ο μηχανισμός πρόβλεψης εντοπίσει κάποια unresolved εντολή η οποία πιστεύει ότι θα γίνει mispredict, σε αυτή την εντολή θα εφαρμόσουμε **resteer**, δηλαδή flush των εντολών που βρίσκονται στο λάθος μονοπάτι. Η υλοποίηση του μηχανισμού πρόβλεψης θα επεξηγηθεί στο επόμενο κεφάλαιο.

Από το πιο πάνω απορρέουν δύο περιπτώσεις. Η πρώτη περίπτωση είναι ο μηχανισμός πρόβλεψης να κάνει σωστή πρόβλεψη δηλαδή να προβλέψει ότι κάποια εντολή προκάλεσε λάθος μονοπάτι εκτέλεσης και στην πραγματικότητα να προκάλεσε λάθος μονοπάτι δηλαδή ήταν mispredict. Η δεύτερη περίπτωση είναι ο μηχανισμός πρόβλεψης να κάνει λάθος πρόβλεψη δηλαδή να προβλέψει ότι κάποια εντολή προκάλεσε λάθος μονοπάτι εκτέλεσης αλλά στην πραγματικότητα να προκάλεσε σωστό μονοπάτι δηλαδή ήταν correctpredict. Έτσι όσες φορές κάποια εντολή προβλέπεται ορθά κερδίζουμε κάποιους κύκλους ενώ αν η πρόβλεψη είναι λανθασμένη πληρώνουμε κάποιο κόστος.

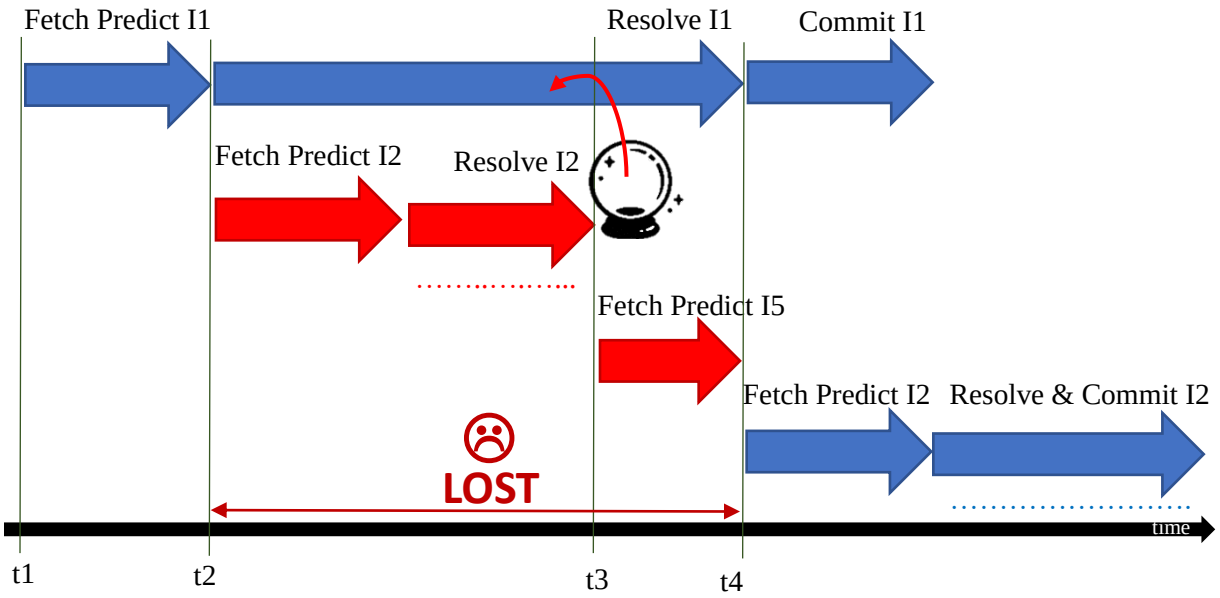


Σχήμα 3.1: Διάγραμμα εκτέλεσης εντολών με ορθή πρόβλεψη mispredict εντολής

Για σκοπούς καλύτερης επεξήγησης θα χρησιμοποιήσουμε τον κώδικα που περιγράφεται στο Σχήμα 2.9 και τις υποθέσεις που έγιναν πιο πάνω.

Το Σχήμα 3.1 παρουσιάζει την πρώτη περίπτωση όπου ο μηχανισμός πρόβλεψης έκανε σωστή πρόβλεψη. Συγκεκριμένα τη χρονική στιγμή t1 η εντολή I1 γίνεται fetch και ο branch predictor προβλέπει ότι η εντολή θα είναι not-taken. Κατά συνέπεια τη χρονική στιγμή t2 η εντολή I2 γίνεται fetch. Ακολούθως τη χρονική στιγμή t3 κατά το resolution της εντολής I2 ο μηχανισμός πρόβλεψης προβλέπει ότι η εντολή I1 είναι mispredict και ως εκ τούτο έχει προκαλέσει ένα λάθος μονοπάτι. Στη συνέχεια η εντολή I1 γίνεται resteer και γίνονται fetch οι εντολές που βρίσκονται στο μονοπάτι που πιστεύει ότι είναι σωστό. Τέλος τη χρονική στιγμή t4 η εντολή I1 γίνεται resolve και επαληθεύτε ότι η πρόβλεψη ήταν σωστή

Με βάση τα πιο πάνω ορίζουμε **gain** τους κύκλους που θα είχαν σπαταληθεί εκτελώντας εντολές στο λάθος μονοπάτι μέχρι να γίνει resolved η εντολή I1 (αυτή που προκάλεσε το λάθος μονοπάτι) – τους κύκλους που ήδη σπαταλήθηκαν σπαταληθεί εκτελώντας εντολές στο λάθος μονοπάτι.



Σχήμα3.2:Διάγραμμαεκτέλεσης εντολών με λανθασμένη πρόβλεψη mispredict εντολής

Το Σχήμα 3.2 παρουσιάζει την δεύτερη περίπτωση όπου ο μηχανισμός πρόβλεψης έκανε λανθασμένη πρόβλεψη. Συγκεκριμένα τη χρονική στιγμή t1 η εντολή I1 γίνεται fetch και ο branch predictor προβλέπει ότι η εντολή θα είναι not-taken. Κατά συνέπεια, τη χρονική στιγμή t2 η εντολή I2 γίνεται fetch. Ακολούθως, τη χρονική στιγμή t3 κατά το resolution της εντολής I2 ο μηχανισμός πρόβλεψης προβλέπει ότι η εντολή I1 είναι mispredict και ως εκ τούτο έχει προκαλέσει ένα λάθος μονοπάτι. Στη συνέχεια η εντολή I1 γίνεται resteer και γίνονται fetch οι εντολές που βρίσκονται στο μονοπάτι που πιστεύει ότι είναι σωστό δηλαδή γίνεται fetch η I5. Τη χρονική στιγμή t4 η εντολή I1 γίνεται resolve και καταλαβαίνουμε ότι η πρόβλεψη ήταν λάθος. Ως εκ τούτου πρέπει οι εντολές που πιστεύαμε ότι βρίσκονται στο σωστό μονοπάτι αλλά τελικά δεν ήταν, να γίνουν flush και να εκτελεστούν εξαρχής τις εντολές που λανθασμένα κάναμε resteer γιατί πιστεύαμε ότι βρίσκονταν στο λάθος στο μονοπάτι ενώ στην πραγματικότητα ήταν στο σωστό.

Με βάση τα πιο πάνω ορίζουμε **lost** τους κύκλους που ήδη σπαταλήθηκαν στο σωστό μονοπάτι εκτέλεσης και θα πρέπει να ξανά εκτελέσουμε τις ίδιες εντολές για δεύτερη φορά + κύκλους που σπαταλήθηκαν εκτελώντας εντολές που πιστεύαμε ότι ήταν στο σωστό μονοπάτι ενώ στην πραγματικότητα ήταν στο λάθος.

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 προκαλούν λάθος μονοπάτι θα είναι σε υψηλά επίπεδα.

Επιτυχημένο resteer (**correct resteer**) ονομάζουμε την αλλαγή της διαδρομής ενός mispredict branch στην διαδρομή όπου είναι ορθό να προχωρήσει η εκτέλεση. Αντίστοιχος, αποτυχημένο resteer (**wrong resteer**) ονομάζουμε την αλλαγή της διαδρομή ενός correctpredict branch στην λανθασμένη διαδρομή εκτέλεσης.

$$opportunity = (total\ correct\ resteers \times gain) - (total\ wrong\ resteers \times lost)$$

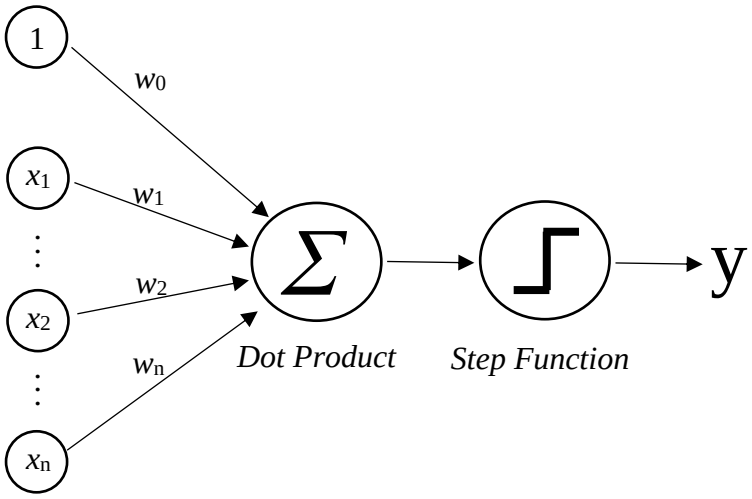
Εξίσωση 3.1: Ορισμός opportunity

Η Εξίσωση 3.1 δηλώνει τον ορισμό του **opportunity** το οποίο είναι το περιθώριο βελτίωσης της επίδοσης ενός επεξεργαστή ο οποίο χρησιμοποιεί ένα μηχανισμό πρόβλεψης των mispredicts. Αναμένουμε να έχουμε αύξηση στην επίδοση του επεξεργαστή όταν η τιμή του opportunity είναι θετική.

3.2 Λειτουργία νευρώνα perceptron

Ο μηχανισμός πρόβλεψης που επιλέξαμε να χρησιμοποιήσουμε στην παρούσα εργασία είναι ένα νευρωνικό δίκτυο perceptron. Ο perceptron είναι ένα από τα πιο απλά νευρωνικά δίκτυα το οποίο μπορεί να υλοποιηθεί αποδοτικά στο hardware.

Ένας single-layer perceptron αποτελείται από ένα τεχνητό νευρώνα ο οποίος συνδέει πολλές μονάδες εισόδου με ακμές με βάρη (**weights**) σε μία μονάδα εξόδου. Ο perceptron μαθαίνει μια Boolean συνάρτηση $t(x_1, \ldots, x_n)$ που αποτελείται από n εισόδους. Στην περίπτωση μας, x_i είναι τα bits που προκύπτουν από τη δημιουργία ενός **διανύσματος με features** που θα χρησιμοποιήσουμε. Τα features που επιλέχθηκαν καθώς και η κωδικοποίηση τους επεξηγούνται αναλυτικά στη συνέχεια. Αφαιρετικά μπορούμε να πούμε ότι ένα perceptron παρακολουθεί τις θετικές και αρνητικές συσχετίσεις μεταξύ των features και του αποτελέσματος της πρόβλεψης.



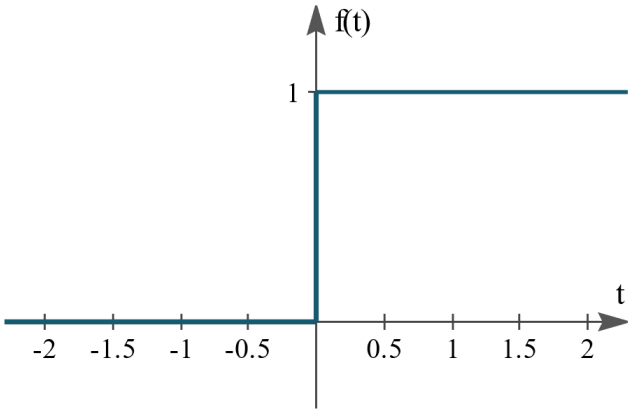
Σχήμα 3.3: Γραφική αναπαράσταση perceptron

Το Σχήμα 3.3 παρουσιάζει τη γραφική αναπαράσταση του μοντέλου ενός perceptron. Ένα perceptron αναπαρίσταται από ένα διάνυσμα του οποίου τα στοιχεία του είναι τα weights. Για τους σκοπούς μας, τα βάρη είναι προσημασμένοι ακέραιοι. Αρχικά, υπολογίζεται το εσωτερικό γινόμενο (**dot product**) του διανύσματος των weights, $w_{0...n}$, και του διανύσματος της εισόδου, $x_{1...n}$. Αξίζει να σημειωθεί ότι το x_0 πάντα ισούται με 1 εφόσον αντιπροσωπεύει το bias input. Το dot product ενός perceptron υπολογίζεται ως

$$dot\ product = w_0 + \sum_{i=1}^{n} x_i w_i.$$

Εξίσωση 3.2: Υπολογισμός dot product

Οι είσοδοι στον perceptron είναι διπολικοί. Για παράδειγμα κάθε x_i μπορεί να είναι είτε -1 είτε 1. Το **step function** το οποίο λαμβάνει σαν είσοδο την τιμή του dot product και καθορίζει την τιμή του *y*. Το step function ερμηνεύει ένα αρνητικό dot product ως πρόβλεψη ότι η εντολή δεν θα γίνει mispredict, δηλαδή δίνει την τιμή 0. Μια μη-αρνητική έξοδος ερμηνεύεται ως πρόβλεψη ότι η εντολή θα γίνει mispredict, δηλαδή δίνει την τιμή 1. Το Σχήμα 3.4 παρουσιάζει την γραφική παράσταση του step function.



Σχήμα 3.4: Γραφική παράσταση του step function

Εάν το dot product που προκύπτει είναι μεγάλο τότε σημαίνει ότι υπάρχει ισχυρή συσχέτιση ανάμεσα στο διάνυσμα εισόδου και στα weights. Όσο μεγαλύτερη είναι η τιμή του dot product τόσο πιο σίγουρος είναι ο perceptron για την πρόβλεψη του. Οπότε ορίζουμε ως **confidence** του perceptron την τιμή που δίνεται από το dot product.

Αξίζει να σημειωθεί ότι ο perceptron εκτελεί την πρόβλεψη του στο στάδιο του resolve μιας conditional branch εντολής για conditional branch εντολές που είναι unresolved. Για να μπορέσει γίνει resteer μια unresolved conditional branch εντολή πρέπει ο perceptron να προβλέψει ότι η θα γίνει mispredict και η πρόβλεψη του να ξεπερνά ένα **confidence threshold** έτσι ώστε να είναι μεγαλύτερες οι πιθανότητες η πρόβλεψη που έκανε να είναι σωστή. Σε περίπτωση που στο reorder buffer υπάρχουν περισσότερες από μια conditional branch unresolved εντολές που ικανοποιούν τα πιο πάνω δυο κριτήρια γίνεται resteer η εντολή που έχει το μεγαλύτερο confidence για να ελαχιστοποιήσουμε τις πιθανότητες λάθους πρόβλεψης.

3.3 Εκπαίδευση νευρώνα perceptron

Αφού έχει υπολογιστεί η έξοδος *y* του perceptron, στο στάδιο του commit εκτελείται ο ακόλουθος αλγόριθμος για την εκπαίδευση του perceptron. Το *t* ορίζεται ως η επιθυμητή έξοδος (**target output**) του perceptron την οποία έχουμε στη διάθεση μας στο στάδιο του commit. Στο πρόβλημα μας το target output είναι 1 εάν η πρόβλεψη του branch predictor ήταν λανθασμένη (οπότε έπρεπε να κάνω resteer) και -1 εάν η πρόβλεψη του branch predictor ήταν ορθή (οπότε δεν έπρεπε να κάνω resteer). Επίσης, ας ορίσουμε το **train threshold**, μια παράμετρος στον αλγόριθμο εκπαίδευσης, που χρησιμοποιείται για να αποφασιστεί εάν η εκπαίδευση που έγινε ήταν αρκετή.

```
1.  if (y ≠ t) or (|dot product| ≤ train threshold) then
2.      for i:=0 to n do
3.          wi:=wi + txi
4.      end for
5.  end if
```

Σχήμα 3.5: Αλγόριθμος εκπαίδευσης perceptron

Δεδομένου ότι το t και το x_i είναι είτε -1 είτε 1, αυτός ο αλγόριθμος αυξάνει το i^{th} weight όταν το target output συμφωνεί με το x_i και μειώνει το weight όταν διαφωνεί. Διαισθητικά μπορούμε να πούμε ότι όταν υπάρχει ως επί το πλείστων συμφωνία, δηλαδή θετική συσχέτιση (positive correlation), το weight γίνεται μεγάλο. Όταν υπάρχει κυρίως διαφωνία, δηλαδή αρνητική συσχέτιση (negative correlation), το weight γίνεται αρνητικό με μεγάλο μέγεθος. Και στις δύο περιπτώσεις, το weight έχει μεγάλη επιρροή στην πρόβλεψη. Όταν υπάρχει ασθενής συσχέτιση, το weight παραμένει κοντά στο 0 και έτσι δεν συμβάλλει στην έξοδο του perceptron.

3.4 Αρχιτεκτονική υλοποίησης μηχανισμού πρόβλεψης

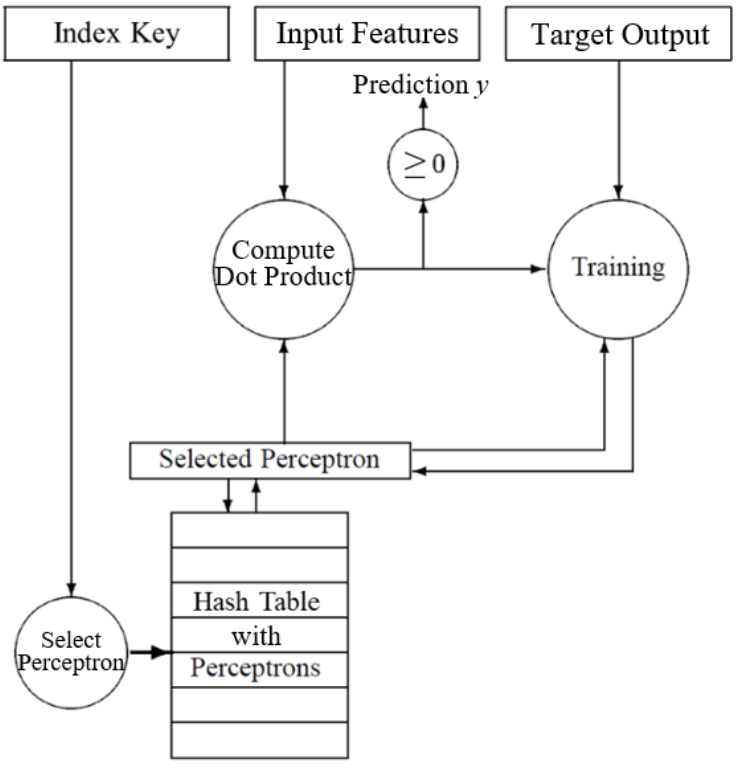
Μπορούμε να χρησιμοποιήσουμε τον perceptron για να μάθει τις συσχετίσεις μεταξύ συγκεκριμένων features. Αυτές οι συσχετίσεις αντιπροσωπεύονται από τα weights. Όσο μεγαλύτερο είναι το weight, τόσο ισχυρότερη είναι η συσχέτιση, και συμβάλει περισσότερο στην πρόβλεψη. Η είσοδος στο bias weight w_0 είναι πάντα 1, οπότε μαθαίνει τις συσχετίσεις μεταξύ των target outputs ανεξάρτητα από τα features.

Το Σχήμα 3.5 δείχνει το διάγραμμα υλοποίησης και εφαρμογής του μηχανισμού πρόβλεψης (Wrong Path Predictor). Ακολουθεί αναλυτική περιγραφή του μηχανισμού.

Κατά το resolution μιας conditional branch εντολής γίνεται προσπέλαση όλων των εντολών που βρίσκονται στο reorder buffer. Για κάθε εντολή k στο reorder buffer που είναι unresolved conditional branch:

1. Δημιουργείται ένα **index key**. Το index key μπορεί να περιέχει bits από το PC της εντολής k ή bits από το global history της εντολής k ή συνδυασμό των δύο.
2. Ο μηχανισμός διατηρεί ένα hashtable με N perceptrons. Το index key χρησιμοποιείται για να την επιλογή του perceptron από το hashtable.

3. Κάθε perceptron από τη στιγμή της δημιουργίας του (**perceptron allocation**) φέρει μαζί του ένα index key. Η δημιουργία του perceptron γίνεται στο στάδιο του commit κάτω από κάποια κριτήρια που θα περιγράψουμε στη συνέχεια. Αυτό συνεπάγει ότι κατά την εκτέλεση του βήματος 2 υπάρχει το ενδεχόμενο ο perceptron που αντιστοιχεί στο index key της εντολής *k* να μην είναι ακόμη δημιουργημένος. Σε μια τέτοια περίπτωση αγνοούμε αυτή την εντολή και προχωρούμε στην επόμενη. Σε περίπτωση που ο perceptron είναι δημιουργημένος προχωρούμε στο επόμενο βήμα και χαρακτηρίζουμε την εντολή *k* ως **candidate resteer**.
4. Δημιουργείται το διάνυσμα εισόδου με features για την εντολή *k*. Στη συνέχεια επεξηγείται αναλυτικά ο τρόπος δημιουργίας του διανύσματος.
5. Ο perceptron χρησιμοποιεί το διάνυσμα εισόδου που δημιουργήθηκε για να εκτελέσει την πρόβλεψη του όπως περιγράψαμε πιο πάνω.
6. Κάνουμε resteer την εντολή που ο perceptron πρόβλεψε ότι θα γίνει mispredict και το confidence που έχει ο perceptron ξεπερνά το confidence threshold και έχει το μεγαλύτερο confidence από όλες τις εντολές στο reorder buffer τη δεδομένη στιγμή.



Σχήμα 3.5: Αρχιτεκτονική υλοποίησης μηχανισμού πρόβλεψης (Wrong Path Predictor)

Κατά το commit μιας conditional branch εντολής:

1. Εάν ο perceptron έχει ήδη δεσμευτεί, εκτελείται ο αλγόριθμος εκπαίδευσης που ορίσαμε πιο πάνω με target output το 1 εάν η πρόβλεψη του branch predictor ήταν λανθασμένη (οπότε έπρεπε να κάνω resteer) και -1 εάν η πρόβλεψη του branch predictor ήταν ορθή (οπότε δεν έπρεπε να κάνω resteer).
2. Εάν ο perceptron δεν έχει ήδη δεσμευτεί, τότε δεσμεύεται αν ικανοποιούνται κάποια κριτήρια που θα περιγράψουμε στη συνέχεια όπως εάν η εντολή ήταν mispredicted.

Κεφάλαιο 4

Features

4.1 Κωδικοποίηση εντολών

Όπως ήδη έχουμε αναφέρει η ιδέα πίσω από την παρούσα εργασία είναι ότι κατά την διάρκεια εκτέλεσης του λανθασμένου μονοπατιού, συμβαίνουν γεγονότα μέσα στον επεξεργαστή τα οποία μπορούν να μαρτυρήσουν ότι μια εκτέλεση βρίσκεται σε λάθος μονοπάτι. Έτσι, θεωρήσαμε ότι με το να συλλέξουμε features, μετά το renaming stage στο pipeline, που σχετίζονται με τις memory και τις branch εντολές που βρίσκονται στο reorder buffer, θα καταφέρουμε να καταγράψουμε αυτά τα γεγονότα που ίσως μαρτυρήσουν την εκτέλεση στο λάθος μονοπάτι.

Κάθε memory εντολή στο reorder buffer κωδικοποιείται σε ένα 6-bit κωδικό (διάνυσμα) όπου κάθε bit αντιπροσωπεύει μια συγκεκριμένη συμπεριφορά όπως ορίζεται στο Σχήμα 4.1.

5	4	3	2	1	0
Resolved / Unresolved	Cache Hit / Cache Miss	Not Trap / Trap	Dtlb mshr miss	Load / Store	Memory

most significant bit *least significant bit*

Σχήμα 4.1: Κωδικοποίηση memory εντολής

Κάθε θέση του διανύσματος ερμηνεύεται ως εξής:

Θέση 0: Εάν η εντολή είναι τύπου memory το bit παίρνει την τιμή 1.

Θέση 1: Εάν η memory εντολή είναι load το bit παίρνει την τιμή 0 ενώ εάν είναι store παίρνει 1.

Θέση 2: Εάν η εντολή είχε dtlb mshr miss το bit παίρνει την τιμή 1 διαφορετικά παίρνει την τιμή 0. Το συγκεκριμένο στατιστικό προέκυψε από ένα correlation analysis που έγινε και φάνηκε ότι υπάρχει ισχυρή συσχέτιση των dtlb mshr miss με τα conditional branch mispredicts.

Θέση 3: Εάν η εντολή δεν είχε replay trap το bit παίρνει την τιμή 0 διαφορετικά εάν είχε replay trap το bit παίρνει την τιμή 1.
Θέση 4: Εάν η εντολή είχε cache hit το bit παίρνει την τιμή 0 διαφορετικά εάν είχε cache miss το bit παίρνει την τιμή 1.
Θέση 5: Εάν η εντολή έγινε resolved το bit παίρνει την τιμή 0 διαφορετικά εάν είναι unresolved το bit παίρνει την τιμή 1.

Κάθε branch εντολή στο reorder buffer κωδικοποιείται με επίσης ένα 6-bit κωδικό όπου κάθε bit αντιπροσωπεύει μια συγκεκριμένη συμπεριφορά όπως ορίζεται στο Σχήμα 4.2.

5	4	3	2	1	0
Resolved / Unresolved	Correctpredict / Mispredict	Not taken / Taken	Dtlb mshr miss	Conditional / Unconditional	Branch

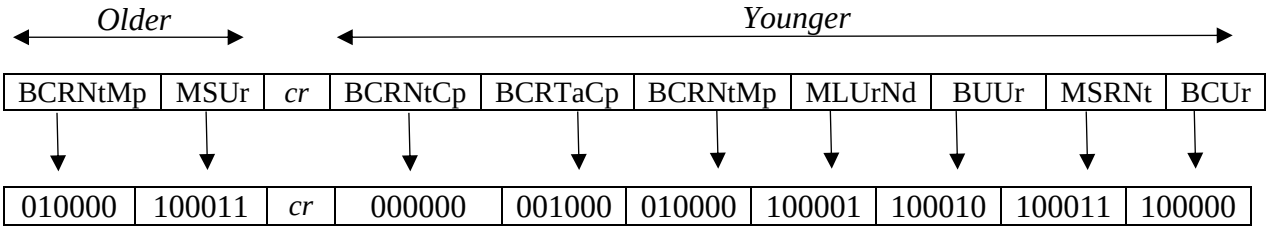
most significant bit *least significant bit*

Σχήμα 4.2: Κωδικοποίηση branch εντολής

Κάθε θέση του διανύσματος ερμηνεύεται ως εξής:
Θέση 0: Εάν η εντολή είναι τύπου branch το bit παίρνει τη τιμή 0.
Θέση 1: Εάν η branch εντολή είναι conditional το bit παίρνει την τιμή 0 ενώ εάν είναι unconditional παίρνει την τιμή 1.
Θέση 2: Στις branch εντολές δεν υπάρχουν dtlb mshr miss γι' αυτό το bit παίρνει την τιμή 0.
Θέση 3: Εάν το predicted direction της εντολή είναι not taken το bit παίρνει την τιμή 0 διαφορετικά εάν είναι taken το bit παίρνει την τιμή 1.
Θέση 4: Εάν η εντολή ήταν correct predict παίρνει την τιμή 0 διαφορετικά εάν ήταν mispredict το bit παίρνει την τιμή 1.
Θέση 5: Εάν η εντολή έγινε resolved το bit παίρνει την τιμή 0 διαφορετικά εάν είναι unresolved το bit παίρνει την τιμή 1.

Το Σχήμα 4.3 παρουσιάζει ένα απλό παράδειγμα των εντολών που περιέχει το reorder buffer μια δεδομένη στιγμή. Στη συνέχεια παρουσιάζεται η κωδικοποίηση που προκύπτει, δηλαδή ο 6-bit κωδικός, για την κάθε εντολή ξεχωριστά. Η λέξη *cr* αντιπροσωπεύει την candidate resteer εντολή που εξετάζουμε.

Αξίζει να σημειωθεί ότι οι εντολές στο reorder buffer χωρίζονται σε παλαιότερες (older) και νεότερες (younger) με σημείο αναφοράς την candidate εντολή. Με άλλα λόγια, μια εντολή είναι **older** εάν χρονικά έγινε fetch πριν να γίνει fetch candidate εντολή. Με ανάλογο τρόπο, μπορούμε να πούμε ότι μια εντολή είναι **younger** εάν χρονικά έγινε fetch αφότου έγινε fetch η candidate εντολή.



Σχήμα 4.3: Κωδικοποίηση των εντολών στο reorder buffer

Με την πιο πάνω κωδικοποίηση μπορούμε να παράγουμε 6-bit κωδικούς για κάθε memory και branch εντολή στο reorder buffer μια δεδομένη στιγμή. Πολλές φορές υπάρχουν πολλά στοιχεία στο reorder buffer έτσι παράγονται πολλοί 6-bit κωδικοί οι οποίοι θα αποτελέσουν μέρος του διανύσματος με τα features εισόδου του perceptron. Για πρακτικούς λόγους το μέγεθος του predictor είναι περιορισμένο οπότε δεν μπορούμε να χρησιμοποιήσουμε όλους τους 6-bit κωδικούς ως έχουν γι' αυτό επιβάλλεται η **συμπίεση** τους η οποία επεξηγείται στη συνέχεια.

4.2 Συμπίεση κωδικοποιήσεων

Ένα σημαντικό σημείο της παρούσας εργασίας είναι η συμπίεση όλων των 6-bit κωδικών των εντολών του reorder buffer μιας δεδομένης χρονικής στιγμής σε ένα διάνυσμα μικρότερου μεγέθους.

Ο αλγόριθμος συμπίεσης των κωδικών βασίζεται στην εφαρμογή του λογικού τελεστή OR σε υποσύνολα των κωδικών. Μια σημαντική παράμετρος στον αλγόριθμο είναι το πλήθος των ομάδων (**groups**) που θέλουμε να χωρίσουμε τα στοιχεία του reorder buffer για εφαρμόσουμε τον τελεστή OR. Μπορούσε να χωρίσουμε τα στοιχεία σε groups με older εντολές ή / και groups με younger εντολές. Στα στοιχεία που βρίσκονται στο ίδιο group εφαρμόζεται ο λογικός τελεστής OR και έτσι επιτυγχάνεται η συμπίεση τους. Ο λόγος που επιλέξαμε τον τελεστή OR είναι γιατί σε αντίθεση με άλλους λογικούς τελεστές όπως το XOR ή το AND είναι ότι μας δίνει την πληροφορία ποια features βρίσκονται στο group.

Ένα ερώτημα που προκύπτει είναι πόσα στοιχεία θα περιέχει το κάθε group. Η αρχική μας σκέψη ήταν το κάθε group να περιέχει ίσο (fixed) αριθμό από στοιχεία. Στη συνέχεια θεωρήσαμε ότι εάν ο αριθμός των στοιχείων των groups αυξανόταν γεωμετρικά (geometric) θα πετυχαίναμε τη δημιουργία καλύτερων pattern που θα βοηθούσαν το perceptron να τα μάθει καλύτερα. Η Εξίσωση 4.1 αναλύει τον μαθηματικό τύπο του αθροίσματος μιας γεωμετρικής σειράς. Στο δικό μας πρόβλημα ***s*** είναι ο αριθμός των στοιχείων του reorder buffer που θέλουμε να συμπιέσουμε, ***n*** είναι ο αριθμός των groups, ***α*** είναι το μέγεθος του πρώτου group που στην περίπτωση μας είναι παραμετρικό και ***r*** είναι το growth factor δηλαδή πόσο θα αυξάνεται κάθε φορά το μέγεθος του group. Από την εξίσωση ο μόνος άγνωστος είναι το ***r*** το οποίο υπολογίζεται δυναμικά σε κάθε περίπτωση.

$$s = a + ar + ar^2 + ar^3 + \cdots + ar^{n-1} = \sum_{k=0}^{n-1} ar^k$$

$$rs = ar + ar^2 + ar^3 + ar^4 + \cdots + ar^n$$

$$s - rs = a - ar^n$$

$$s(1 - r) = a(1 - r^n)$$

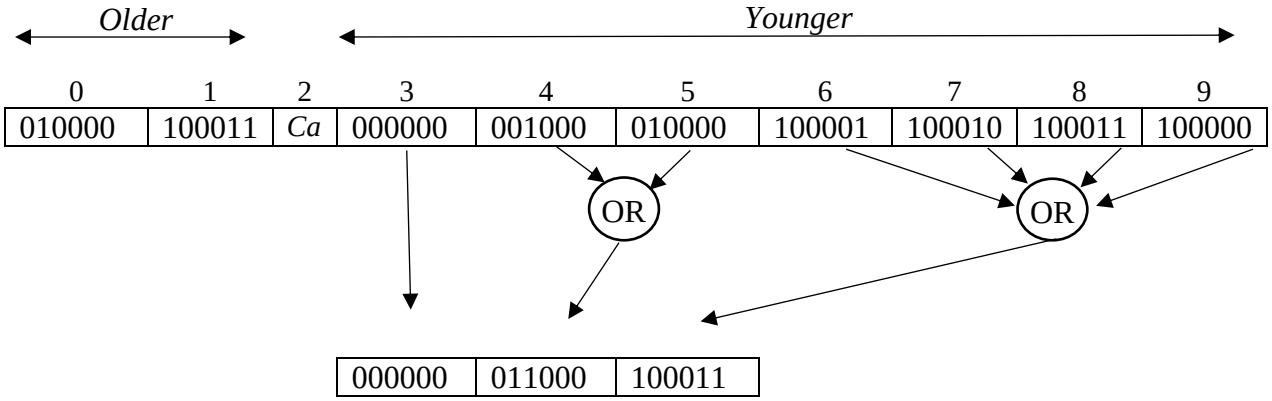
$$s = \frac{a(1 - r^n)}{1 - r}$$

Εξίσωση 4.1: Άθροισμα γεωμετρικής σειράς

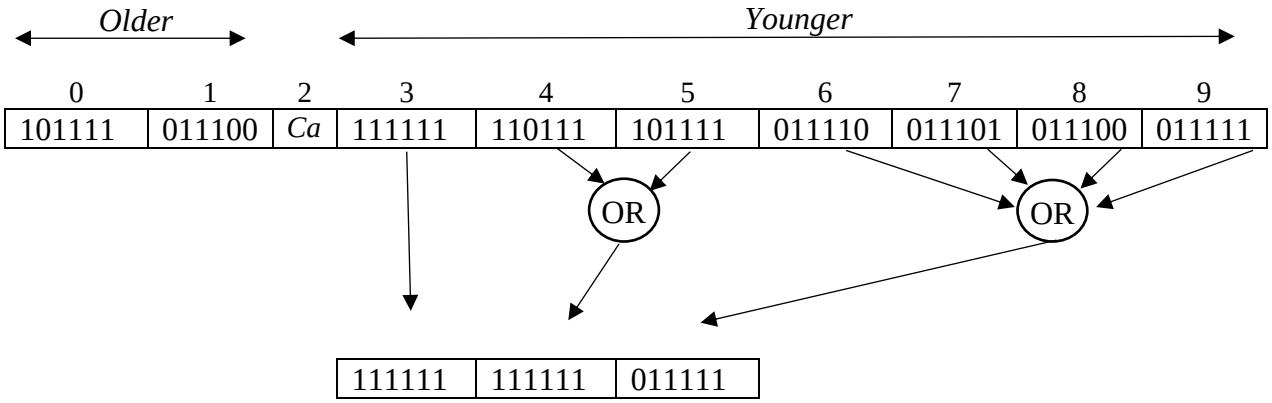
Ένα τελευταίο σημείο του αλγόριθμο συμπίεσης ο υπολογισμός του συμπληρώματος των κωδικών. Είναι απαραίτητο να υπολογίσουμε πρώτα την κωδικοποίηση της κάθε εντολής και στη συνέχεια το συμπλήρωμα της κωδικοποίησης. Έχοντας στη διάθεση μας και το συμπλήρωμα της κωδικοποίησης αποκτούμε περισσότερες πληροφορίες και έτσι βοηθούμε τον perceptron να μάθει καλύτερα. Στο τέλος της κωδικοποίησης θα δημιουργηθούν 2 σύνολα ένα με τις συμπιέσεις των κωδικών στην κανονική του μορφή και ένα με τις συμπιέσεις των συμπληρωμάτων των κωδικών.

Ο αλγόριθμος της συμπίεσης που περιγράφεται πιο πάνω επεξηγείται με το ακόλουθο παράδειγμα. Ας υποθέσουμε ότι θέλουμε να δημιουργήσουμε 4 groups με younger κωδικούς (***n***=4). Τη δεδομένη χρονική στιγμή στο reorder buffer υπάρχουν 7 younger στοιχεία (***s***=7). Επίσης,

ορίζουμε το μέγεθος το πρώτου group ίσο με 1 (***a***=1). Χρησιμοποιώντας την Εξίσωση 4.1 μπορούμε να υπολογίσουμε το growth factor το οποίο ισούται με 2 (***r***=2). Με βάση τα πιο πάνω μπορούμε να πούμε ότι τα groups θα έχουν μέγεθος 1, 2, 4, 8. Μπορούμε να δούμε ότι με την δημιουργία 3 groups τα στοιχεία που έχουμε έχουν εξαντληθεί. Στην υλοποίηση μας έχουμε προνοήσει ότι σε μια τέτοια περίπτωση θα τα bits που υπολοπονται από τα groups που δεν δημιουργήθηκαν θα συμπληρόνται με bits από το global history. Το Σχήμα 4.4 δείχνει την συμπίεση των κωδικων στην κανονική τους μορφή ενώ το Σχήμα 4.5 δείχνει την συμπίεση των συμπληρωμάτων των κωδικών. Τα παραδείγματα στο Σχήμα 4.4 και στο Σχήμα 4.5 βασίζονται στην κωδικοποίηση των ενολών του reorder buffer όπως παρουσιάστηκαν στο παράδειγμα στο Σχήμα 4.3.



Σχήμα 4.4: Συμπίεση κωδικών στην κανονική τους μορφή



Σχήμα 4.5: Συμπίεση συμπληρώματος κωδικών

4.3 Δημιουργία διανύσματος με features

Το διάνυσμα εισόδου με features που δίνεται στον perceptron αποτελείται από n bits όπου το n μια παράμετρος. Το διάνυσμα μπορεί να περιέχει τις ακόλουθες πληροφορίες:

a) Το predicted direction της candidate εντολής.
b) Ένα συγκεκριμένο αριθμό από bits του PC της candidate εντολής.
c) Ένα συγκεκριμένο αριθμό από bits του global history της candidate εντολής.
d) Τον αριθμό των στοιχείων του reorder buffer της candidate εντολής ή την λογαριθμική του τιμή με βάση το 2.
e) Η συμπίεση των κωδικών που προκύπτουν από την κωδικοποίηση των εντολών του reorder buffer, όπως περιγράφεται πιο πάνω.

Κεφάλαιο 5

Μεθοδολογία αξιολόγησης

5.1 Προσομοιωτής Sim-Alpha

Ο προσομοιωτής Sim-Alpha είναι ένας execution-driven simulator που μοντελοποιεί τους περιορισμούς υλοποίησης και τα low-level χαρακτηριστικά επίδοσης του Alpha 21264, ενός RISC μικροεπεξεργαστή με Alpha Instruction Set Architecture. Ο προσομοιωτής αυτός, παράγει συγκρίσιμα αποτελέσματα με πραγματικό hardware και αυτό επιτυγχάνετε χάρης στην ακρίβεια με την οποία μοντελοποιεί τον Alpha μικροεπεξεργαστή. Ο Sim-Alpha εκτελεί τις εντολές στο βάθος των misspredicts paths όπως θα τις εκτελούσε και ένας πραγματικός επεξεργαστής. Χρησιμοποιείτε κατά κόρων από ερευνητές για την ακρίβεια του και για το λόγω αυτό θα ήταν κατάλληλος για την έρευνα αυτής της εργασίας.

Για ακρίβεια των αποτελεσμάτων της έρευνα μας, είναι πολύ σημαντικό ο branch predictor που χρησιμοποιείτε από τον προσομοιωτή Sim-Alpha να είναι ο καλύτερος δυνατός. Για το λόγω αυτό χρησιμοποίησα μια αναβαθμισμένη έκδοση του Sim-Alpha, η οποία ενσωματώνει τον branch predictor L-TAGE, ο οποίος αποτελεί ένα state-of -the-art-predictor που προτάθηκε από τον André Seznec (2007). Επιπροσθετα, η αναβαθμισμένη έκδοση του Sim-Alpha υποστηρίζει και out-of-order brach execution.

5.2 SPEC CPU® 2006 Benchmark Suite

Τα Benchmark που περιλαμβάνονται στο πακέτο των SPEC 2006 [5] αποτελούν βιομηχανική τυποποιημένη σουίτα benchmark για CPU, που σκοπό έχουν να στρεσάρουν τον επεξεργαστή του συστήματος, το υποσύστημα μνήμης και τον μεταγλωττιστή.

Η SPEC σχεδίασε αυτή τη σουίτα για να παρέχει ένα συγκριτικό μέτρο επιδόσεων εντάσεως υπολογισμών σε όλο το ευρύτερο φάσμα υλικού που χρησιμοποιεί workloads που αναπτύχθηκαν από πραγματικές εφαρμογές χρηστών. Αυτά τα benchmarks παρέχονται ως πηγαίος κώδικας και απαιτούν από τον χρήστη να τα μεταγλωττίσει σε εκτελέσιμα binaries. Στην περίπτωση μας, τα benchmarks αυτά μεταγλωττίστηκαν για να τρέχουν σε επεξεργαστή της αρχιτεκτονικής Alpha.
Τα Benchmark από τη σουίτα SPEC 2006 που χρησιμοποιήσαμε στην μελέτη αυτή, έτρεχαν πάντα σε ντετερμινιστικό περιβάλλον και ήταν τα ακόλουθα:
perlbench, bzip2, gcc, bwaves, gamess, mcf, milc, zeusmp, gromacs, cactusADM, leslie3d, namd, gobmk, soplex, hmmer, sjeng, GemsFDTD, libquantum, h264ref, lbm, omnetpp, astar, sphinx3.

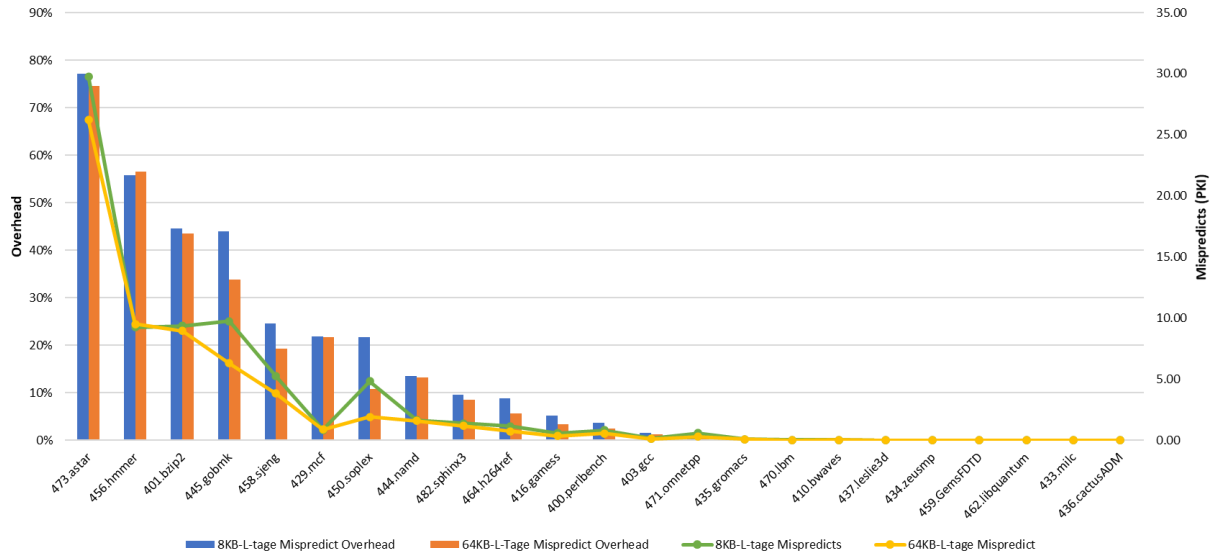
Με βάση το Σχήμα 1.1 αποφασίσαμε στην παρούσα μελέτη να ασχοληθούμε μόνο με τα 12 benchmarks τα οποία έχουν μεγαλύτερο κόστος από conditional branch mispredicts. Αυτά τα benchmarks είναι:
astar, hmmer, bzip2, gobmk, soplex, sjeng, mcf, namd, sphinx3, h264ref, gamess, perlbench.

5.3 Configurations Προσομοιωτή

Πριν ξεκινήσουμε την εκτέλεση των πειραμάτων θεωρήσαμε σκόπιμο να εντοπίσουμε τα καλύτερα configurations στον προσομοιωτή τα οποία προσδίδουν μεγαλύτερη επιτάχυνση (speedup) στις εκτελέσεις των benchmarks. Ο λόγος για τον οποίο θέλουμε να βρούμε τα καλύτερα configurations είναι για να δείξουμε ότι ακόμη και με καλά configurations το overhead που προκαλούν τα conditional branch mispredicts εξακολουθεί να υπάρχει και είναι ακόμη πολύ σημαντικό.

5.3.1 Μέγεθος Branch Predictor

Αρχικά, αυξήσαμε το μέγεθος του L-TAGE branch predictor από 8KB σε 64KB. Υπολογίσαμε τα conditional branch mispredictions committed και το κόστος (overhead) που προκαλούν. Από τη γραφική παράσταση που φαίνεται στο Σχήμα 5.1 παρατηρούμε ότι το κόστος που προκαλούν εξακολουθεί να είναι αυξημένο. Αυτό γεγονός που μαρτυρεί ότι το κόστος των mispredicts εξακολουθεί να είναι σημαντικό ακόμη και σε έναν μεγαλύτερο branch predictor.



Σχήμα 5.1: Σύγκριση overhead που προκαλούν τα conditional branch mispredicts σε branch predictor 8KB και σε 64KB

5.3.2 Skylake configurations

Η δεύτερη προσέγγιση ήταν να αλλάξουμε τα configurations του προσομοιωτή έτσι ώστε να μοιάζουν όσο το δυνατόν καλύτερα στα configurations ενός skylake επεξεργαστή. Ο Πίνακας 5.1 παρουσιάζει τα configurations που επιλέχθηκαν.

Structure	**Size**
Reorder buffer size (<number of entries>)	256
Number of integer physical registers	256
Number of fp physical registers	256
Load queue size (<number of entries>)	64
Store queue size (<number of entries>)	6

Slot width	4
Issue: intwidth / Fetch / Commit / Map width	8
Issue: fpwidth	4
Line predictor width	8
Number of regular mshrs for each cache	16
d-L1 size	64KB
L2 size	2MB
d-TLB	2KB
d-L1 associativity	8
i-L1 associativity	8
L2 associativity	16

Πίνακας 5.1: Configurations που προσεγγίζουν έναν skylake επεξεργαστή

Σε συνέχεια της αναζήτησης των καλύτερων configurations, ενεργοποιήσαμε το prefetching 1ος block στην dL1 σε περίπτωση dcache miss και αλλάξαμε κάποιες παραμέτρους για βελτιστοποίηση της επίδοσης της dL1 κατά το prefecting. Ο Πίνακας 5.2 δείχνει τις παραμέτρους που επιλέχθηκαν για αυτό τον σκοπό.

Structure	**Size**
Number of regular mshrs for each cache	16
Number of prefetch mshrs for each cache	8

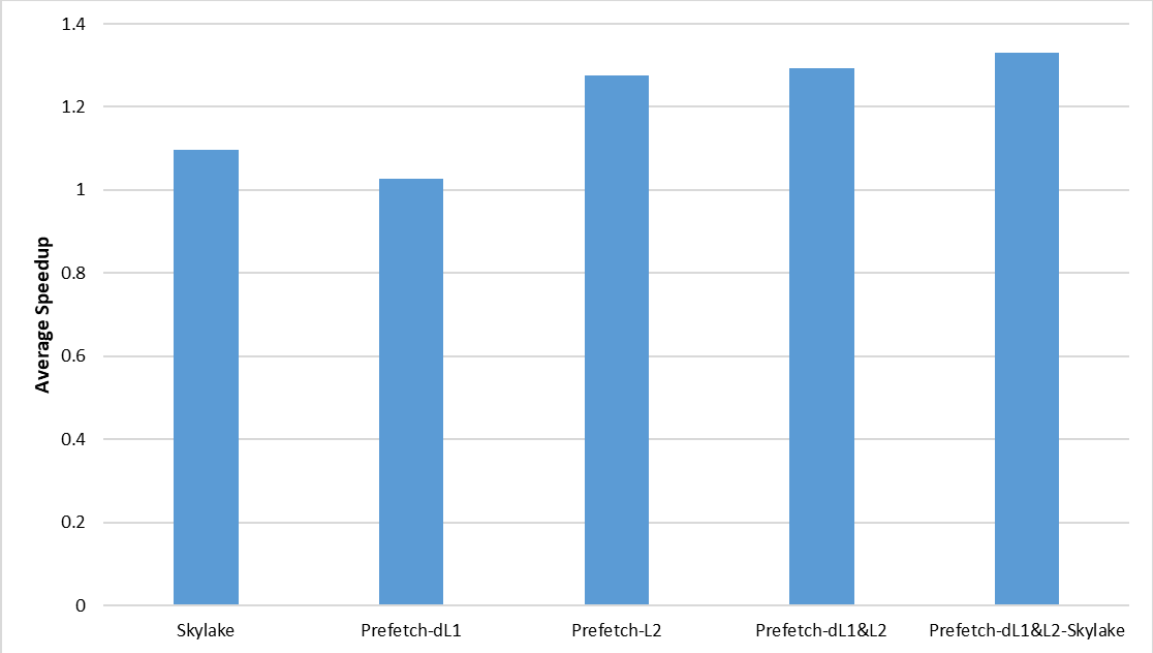
Πίνακας 5.2: Configurations για βελτιστοποίηση της dL1

Επίσης, ενεργοποιήσαμε το prefetching 1ος block στην L2 σε περίπτωση L2 miss και αλλάξαμε κάποιες παραμέτρους για βελτιστοποίηση της επίδοσης της L2 κατά το prefecting. Ο Πίνακας 5.3 δείχνει τις παραμέτρους που επιλέχθηκαν για αυτό τον σκοπό.

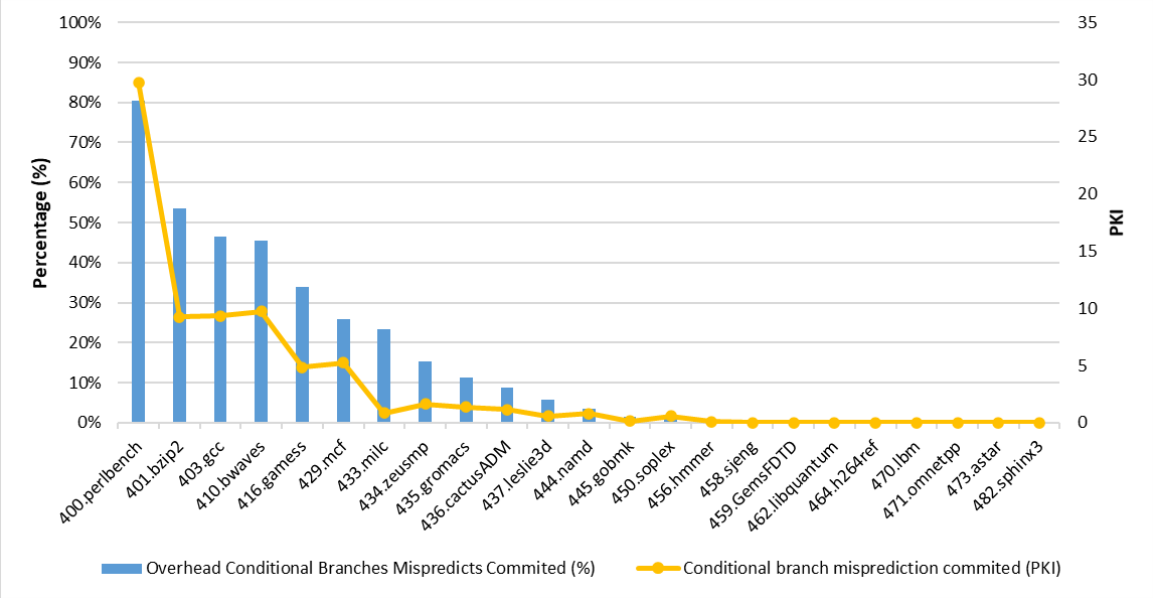
Structure	**Size**
Number of targets for each cache	16
Membus width bytes	16

Πίνακας 5.3: Configurations για βελτιστοποίηση της L2

Οι πιο πάνω περιπτώσεις εφαρμόστηκαν τόσο ανεξάρτητα όσο και συνδυάστηκα. Η γραφική παράσταση που φαίνεται στο Σχήμα 5.2 παρουσιάζει το average speedup που επιτεύχθηκε για όλα τα benchmarks. Με βάση τη γραφική παράσταση παρατηρούμε ότι η μεγαλύτερη επιτάχυνση επιτεύχθηκε κατά την εφαρμογή του συνδυασμού όλων των πιο πάνω περιπτώσεων.



Σχήμα 5.2: Average Speedup των πέντε περιπτώσεων cofnigurations που εξετάστηκαν



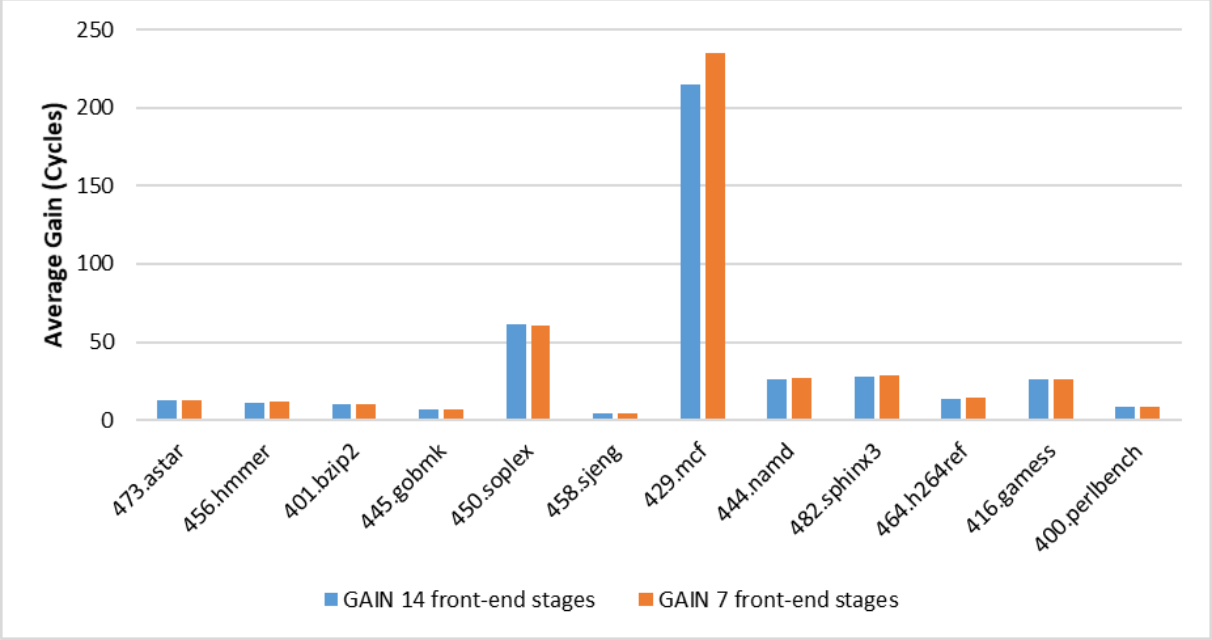
Σχήμα 5.3: Σύγκριση overhead που προκαλούν τα conditional branch mispredicts στις πέντε περιπτώσεις configurations

Για τα καλύτερα configurations υπολογίσαμε τα conditional branch mispredictions committed και το κόστος που προκαλούν. Με βάση τη γραφική παράσταση που απεικονίζει το Σχήμα 5.3 παρατηρούμε ότι το κόστος που προκαλούν εξακολουθεί να είναι αυξημένο δίνοντας μας το κίνητρο να συνεχίσουμε την ερεύνα μας ως προς τη μείωση αυτού του κόστους.

5.3.3 Front-end pipeline stages

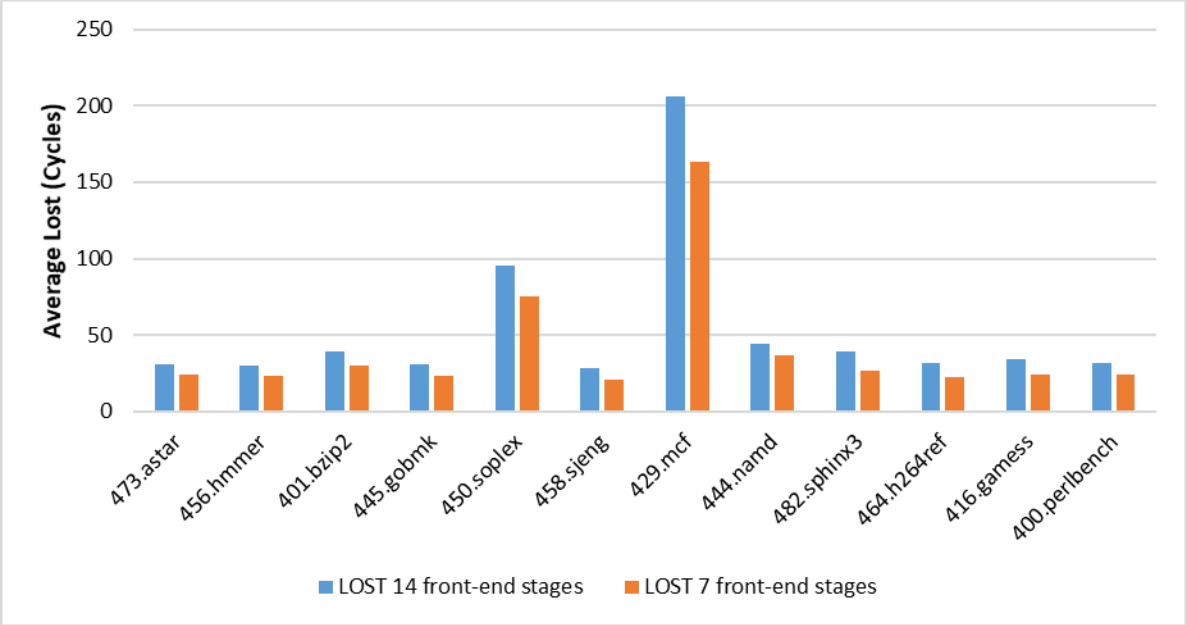
Στη συνέχεια προσπαθήσαμε να αναζητήσουμε configurations τα οποία θα βοηθήσουν στην αύξηση του gain και μείωση του lost. Γι' αυτό το λόγο αποφασίσαμε να μειώσουμε τα front-end (ή slot) pipeline stages στα configurations του προσομοιωτή. Συγκεκριμένα μειώσαμε τα front-end stages από 14 σε 7.

Παρατηρώντας τη γραφική παράσταση στο Σχήμα 5.4 μπορούμε να δούμε ότι στα περισσότερα benchmarks μειώνοντας τα front-end stages το gain είτε διατηρείται σταθερό είτε αυξάνεται.



Σχήμα 5.4: Σύγκριση gain σε επεξεργαστή με 14 και 7 front-end stages

Παρατηρώντας τη γραφική παράσταση στο Σχήμα 5.5 μπορούμε να δούμε ότι στα περισσότερα benchmarks μειώνοντας τα front-end stages το lost μειώνεται.



Σχήμα 5.5: Σύγκριση gain σε επεξεργαστή με 14 και 7 front-end stages

Με βάση τα πιο πάνω αποτελέσματα αποφασίσαμε να συνεχίσουμε την πειραματική διερεύνηση χρησιμοποιώντας 7 front-end pipeline stages.

5.4 Μετρικές αξιολόγησης

Αποφασίσαμε να αξιολογήσουμε τα πειραματικά μας αποτελέσματα χρησιμοποιώντας μετρικές όπως είναι το Speedup, το Accuracy και το Coverage.

Το **Speedup** ορίζεται ως το πηλίκο που προκύπτει από τη διαίρεση του νέου IPC μετά την εφαρμογή του Wrong Path Predictor δια το αρχικό IPC.

$$Speedup = \frac{IPC\ with\ WPP}{IPC\ without\ WPP}$$

Εξίσωση 5.1: Υπολογισμός της μετρικής του Speedup

Το **Accuracy** ορίζεται ως το πηλίκο που προκύπτει από τη διαίρεση των σωστών resteers που έγιναν διά τα συνολικά resteers.

$$Accuracy = \frac{Correct\ resteers}{Correct\ resteers + Wrong\ resteers}$$

Εξίσωση 5.2: Υπολογισμός της μετρικής του Accuracy

Το **Coverage** ορίζεται ως το πηλίκο που προκύπτει από τη διαίρεση των σωστών resteers που έγιναν διά τα συνολικά conditional branch mispredicts commited.

$$Coverage = \frac{Correct\ resteers}{Conditional\ branch\ mispredicts\ commited}$$

Εξίσωση 5.3: Υπολογισμός της μετρικής του Coverage

5.5 Design Space

Στα πλαίσια της πειραματικής διερεύνησης της εργασίας μας ορίσαμε το χώρο τον οποίο θα διερευνήσουμε έτσι ώστε να πετύχουμε την καλύτερη δυνατή λύση του προβλήματος μας.

Σχετικά με το index key υπάρχουν οι ακόλουθες δυνατότητες διερεύνησης:

a) *index_key_PC_bits*: Ένας συγκεκριμένος αριθμός από bits του PC.
b) *index_key_ghist*: Ένας συγκεκριμένος αριθμός από bits του global history.
c) *index_key_XOR*: Εάν είναι ίσο με 1 τότε θα εφαρμοστεί ο λογικός τελεστής XOR μεταξύ των bits από το PC και των bits από το global history. Εάν είναι ίσο με 0 τότε θα τοποθετηθούν πρώτα τα bits από το PC και στη συνέχεια θα τοποθετηθούν τα bits από το global history.

Σχετικά με τη δημιουργία διανύσματος με features:

a) *num_encoded_bits*: Ο συνολικός αριθμός των bits που θα περιέχει το κωδικοποιημένο διάνυσμα που θα προκύψει.

b) *predicted_direction*: Εάν είναι ίσο με 1 τότε το διάνυσμα θα περιέχει το predicted direction, διαφορετικά δεν θα το περιέχει.
c) *PC_bits*: Ένας συγκεκριμένος αριθμός από bits του PC.
d) *ghist_bits*: Ένας συγκεκριμένος αριθμός από bits του global history.
e) *num_of_elem_ROB*: Εάν είναι ίσο με [1, 0] θα τοποθετήσει στο διάνυσμα τον αριθμό των στοιχείων του reorder buffer. Εάν είναι ίσο με [1, 1] θα τοποθετήσει στο διάνυσμα την λογαριθμική τιμή με βάση το 2 του αριθμού των στοιχείων του reorder buffer. Εάν είναι ίσο με 0 δεν θα τοποθετήσει κάτι.
f) *elem_mask*: Η μάσκα με την οποία θα φιλτράρει ποια στοιχεία θα περάσουν από την 6-bit κωδικοποίηση για κάθε στοιχείο του reorder buffer.
g) *num_group_old_young*: Ο αριθμός των groups που θα δημιουργηθούν χρησιμοποιώντας older elements, ο αριθμός των groups που θα δημιουργηθούν χρησιμοποιώντας younger elements.
h) *init_group_size*: Ο αριθμός των στοιχείων που θα συμπιέσει το πρώτο group.

Σχετικά με τον μηχανισμό πρόβλεψης με perceptrons:

a) *perceptron_mode*: Εάν είναι ίσο με 0 τότε κάθε index key αντιστοιχεί σε έναν μοναδικό perceptron. Εάν είναι ίσο με 1 τότε ένα ή περισσότερα index key που έχουν το ίδιο hash αντιστοιχούν στον ίδιο perceptron.
b) *hashtable_len*: Ο αριθμός των θέσεων του hashtable που περιέχει τους perceptrons.

Σχετικά με τα resteer policies:

a) *candidate_resteer*: Εάν είναι ίσο με 1 τότε μια εντολή θεωρείται candidate resteer εάν ο perceptron που αντιστοιχεί στο index key της έχει δημιουργηθεί. Εάν είναι ίσο με 2 για να θεωρηθεί candidate resteer θα πρέπει να έχει τουλάχιστον μια νεότερη (younger) resolved conditional branch εντολή στο reorder buffer.
b) *update_resteer*: Ενημερώνει όλες τις candidate resteer εντολές.
c) *allocate_predictor_policy*: Για να δημιουργηθεί ένας perceptron θα πρέπει η εντολή να είναι mispredicted και να μην έχει δημιουργηθεί στο παρελθόν για το δεδομένο index key.
d) *conf_threshold*: Μια τιμή που αναπαριστά το threshold το οποίο ξεπερνιέται από το dot product τότε η εντολή θεωρείται confidence.

e) *train_threshold*: Μια τιμή που αναπαριστά το threshold με το οποίο ξεπερνιέται από το dot product μπορεί να σταματήσει η εκπαίδευση του perceptron.

f) *which_resteer:* Την εντολή που ο perceptron πρόβλεψε ότι θα γίνει mispredict και είναι confidence και έχει το μεγαλύτερο confidence από όλες τις εντολές στο reorder buffer τη δεδομένη στιγμή.

g) *when_resteer*:

1. Με βάση το target accuracy. Target Accuracy ορίζεται ως η θεωρητική ακρίβεια του predictor που όταν ξεπεράσει ο πραγματικός predictor εκτιμούμε ότι θα έχουμε όφελος στην επίδοση του επεξεργαστή. Η θεωρητική ακρίβεια του predictor ορίζεται με την κάτω εξίσωση.

$$\text{T}arget\ Accuracy = \frac{Θεωρητικό\ Lost}{Θεωρητικό\ Lost + Θεωρητικό\ Gain}$$

2. Με βάση το confidence threshold όπως εξηγήσαμε πιο πάνω.

Κεφάλαιο 6

Αξιολόγηση

6.1 Design Space που εξερευνήθηκε

Λόγω του ότι το design space που ορίσαμε είναι πολύ μεγάλο και προκύπτουν πάρα πολλοί συνδυασμοί αποφασίσαμε να το περιορίσουμε εξετάζοντας μόνο ένα υποσύνολο των παραμέτρων και διατηρώντας σταθερές κάποιες άλλες. Οι Πίνακες 6.1, 6.2, 6.3, 6.4 περιγράφουν το design space που επιλέξαμε να εξερευνήσουμε.

Parameter	**Value**
-index_key_PC_bits	48
-index_key_ghist	16
-index_key_XOR	0

Πίνακας 6.1: Design Space που εξερευνήθηκε για την επιλογή του Index Key

Parameter	**Value**
-num_encoded_bits	53
-predicted_direction	0, 1
-PC_bits	0
-ghist_bits	0
-num_of_elem_ROB	[1,1]
-elem_mask	0, 1, 2, 4, 8, 16, 32, 63
-num_group_old_young	[4,0], [0,4]
-init_group_size	2

Πίνακας 6.2: Design Space που εξερευνήθηκε για την επιλογή του Feature Encoding

Parameter	Value
-perceptron_mode	0
-hashtable_lenght	32768

Πίνακας 6.3: Design Space που εξερευνήθηκε για την επιλογή του Perceptron

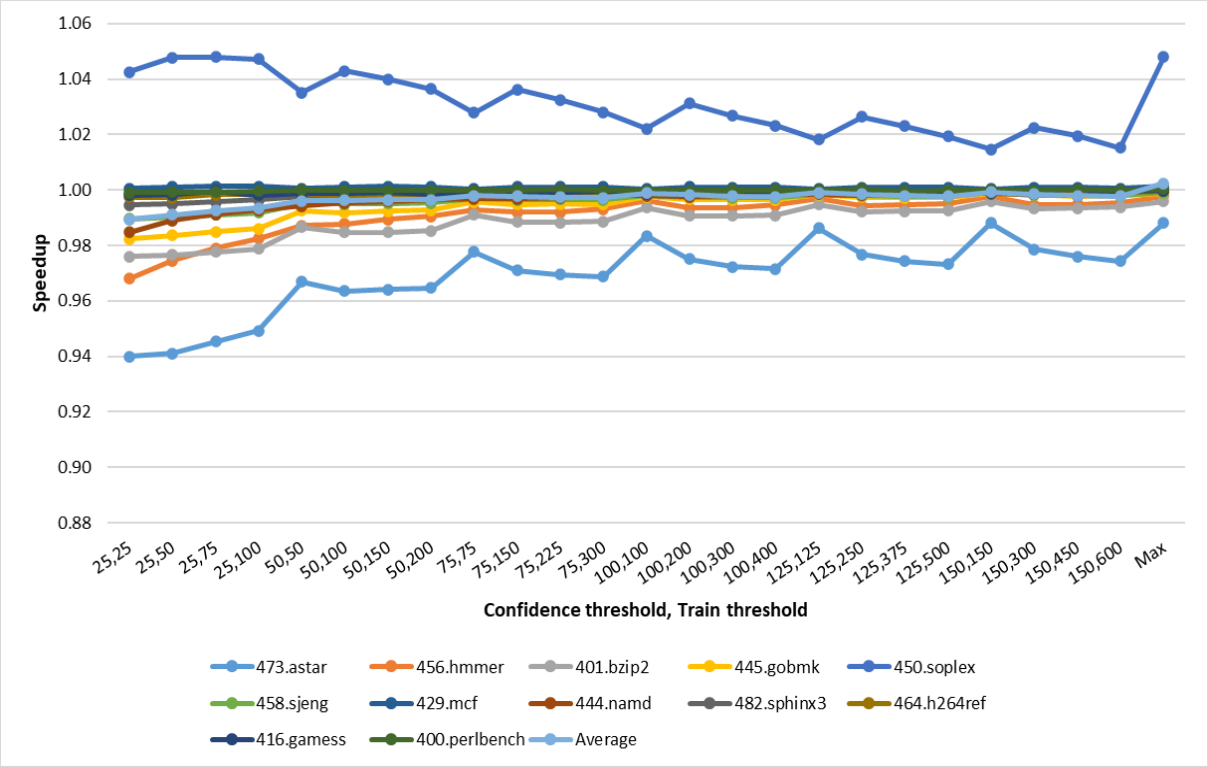
Parameter	Value
-candidate_resteer	1, 2
-update_resteer	All eligible candidates
-allocate_predictor_policy	Mispredicted & Candidate
-conf_threshold	25, 50, 75, 100, 125, 150
-train_threshold_factor	1, 2, 3, 4
-which_resteer	Most confidence
-when_resteer	2

Πίνακας 6.4: Design Space που εξερευνήθηκε για την επιλογή του Resteer Policies

6.2 Αποτελέσματα και Παρατηρήσεις

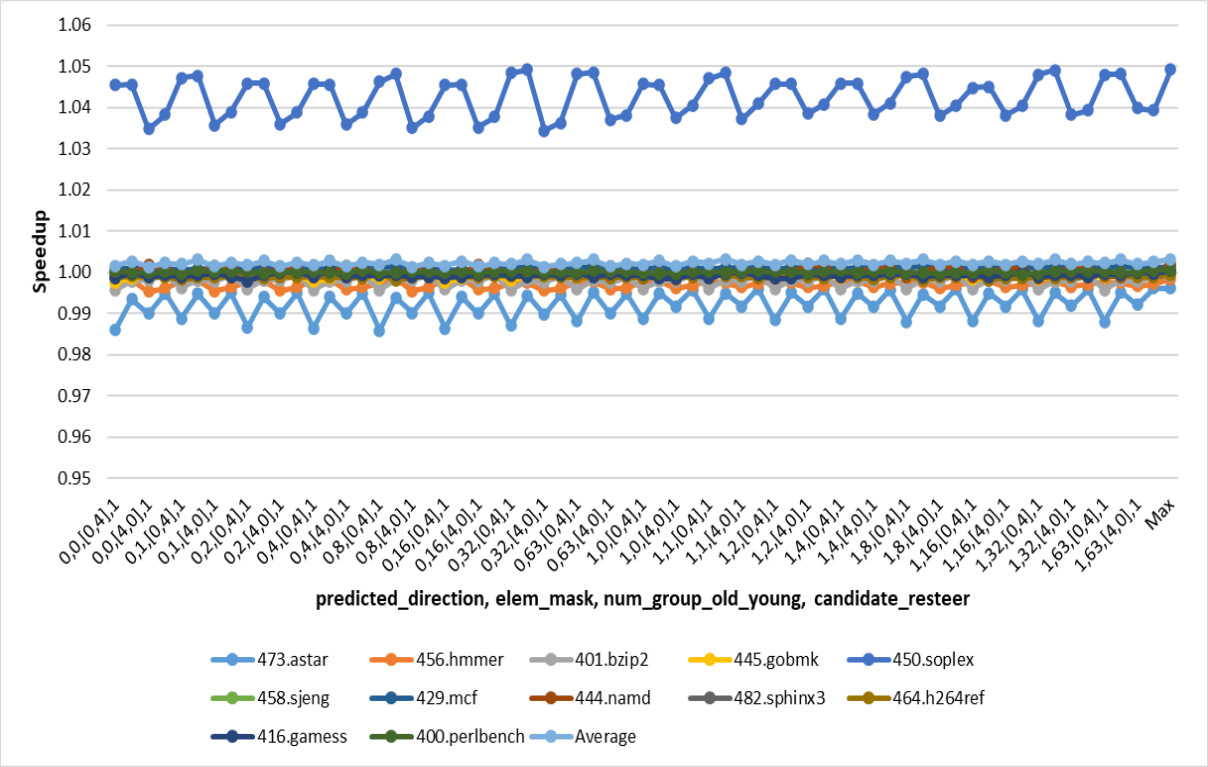
Το Σχήμα 6.1 παρουσιάζει τη γραφική παράσταση του speedup σε συνάρτηση με το confidence και train threshold. Κάθε γραμμή αναπαριστά ένα benchmark. Η αξιολόγηση αυτή έγινε με στόχο να εντοπίσουμε τα καλύτερα confidence και train threshold για κάθε benchmark διατηρώντας τις υπόλοιπες παραμέτρους σταθερές. Με βάση την πιο κάτω γραφική παράσταση μπορούμε να πούμε ότι ο συνδυασμός confidence threshold=150 και train threshold=150 πετυχαίνει κατά μέσο όρο το καλύτερο speedup. Είναι σκόπιμο να αναφέρουμε ότι το speedup δεν σημείωσε ιδιαίτερη μεταβολή γι’ αυτό συνεχίσαμε με διερεύνηση περισσότερων παραμέτρων του design space.

Αξίζει να σημειωθεί ότι για το benchmark soplex παρατηρήθηκε ψηλό speedup συγκριτικά με τα άλλα benchmarks χωρίς να έχει ιδιαίτερα ψηλό accuracy. Αυτήν την συμπεριφορά δεν μπορέσαμε να την ερμηνεύσουμε εφόσον αποκλίνει από τα άλλα benchmarks γι’ αυτό το θεωρήσαμε outlier και δεν θεωρήσαμε αντιπροσωπευτική τη συμπεριφορά του.



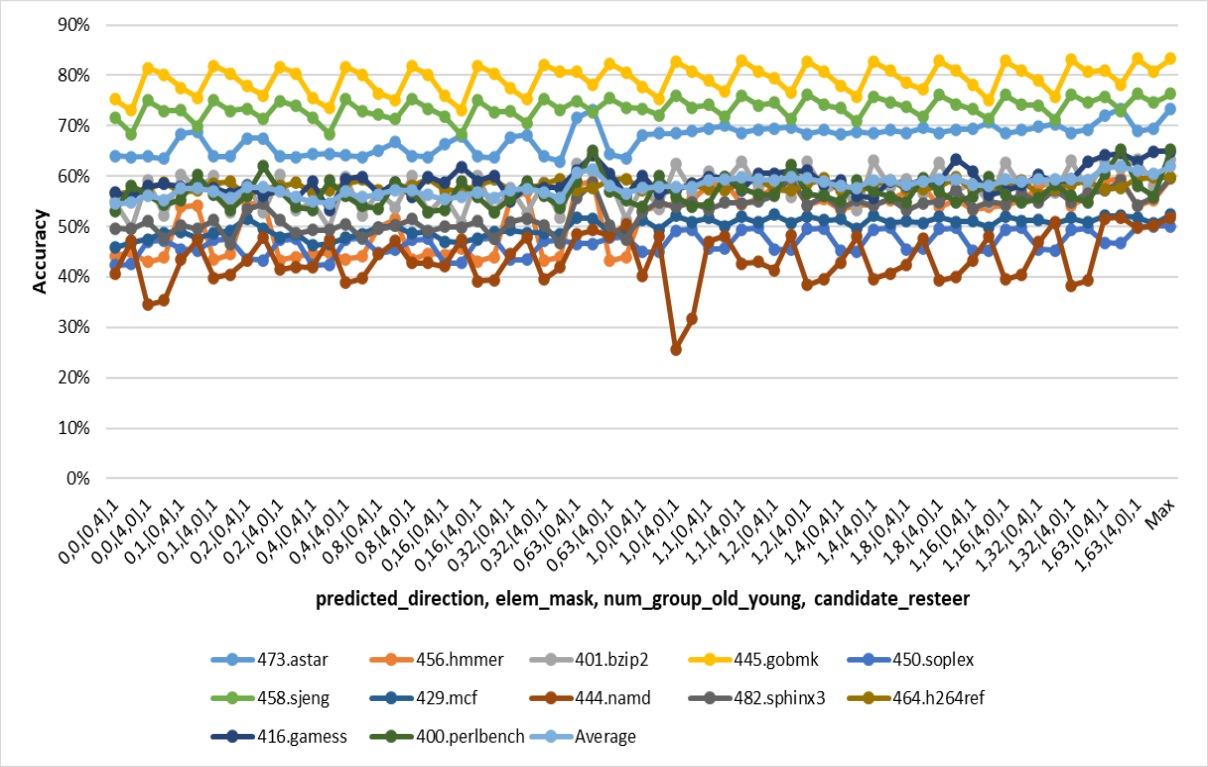
Σχήμα 6.1: Speedup σε συνάρτηση το confidence και train threshold

Το Σχήμα 6.2 παρουσιάζει τη γραφική παράσταση του speedup σε συνάρτηση με παραμέτρους που αφορούν το predicted direction, το element mask, τον αριθμό των groups που θα δημιουργηθούν και το candidate resteer. Κάθε γραμμή αναπαριστά ένα benchmark. Η αξιολόγηση αυτή έγινε με στόχο να εντοπίσουμε τις παραμέτρους που συμβάλουν στην αύξηση του speedup διατηρώντας το confidence και train threshold σταθερά, στα οποία επιλέχθηκαν οι καλύτερες τιμές με βάση την προηγούμενη ανάλυση. Κατά μέσο όρο στις καλύτερες περιπτώσεις υπήρχε η μια πολύ μικρή αύξηση του speedup σε σχέση με την προηγούμενη ανάλυση. Ένα πολύ σημαντικό εύρημα είναι ότι το μεγαλύτερο speedup σε κάθε benchmark παρουσιάζεται πάντα όταν candidate resteer εντολή είναι εκείνη που έχει τουλάχιστο ένα younger resolved conditional branch. Επίσης, φάνηκε ότι πετυχαίνουμε καλύτερο speedup όταν τα features που χρησιμοποιούνται είναι ο συνδυασμός όλων των features. Επίσης, μερικά benchmarks έχουν καλύτερο speedup όταν χρησιμοποιείται το feature Taken/Not taken ή Trap/Not trap. Επίσης φάνηκε ότι συνήθως όταν η κωδικοποίηση των features περιέχει younger εντολές του reorder buffer επιφέρει μεγαλύτερο speedup από όταν χρησιμοποιούνται older εντολές αλλά υπάρχουν και περιπτώσεις όπου όταν χρησιμοποιούνται older εντολές παρουσιάζεται καλό speedup.



Σχήμα 6.2: Speedup σε συνάρτηση με παραμέτρους του design space

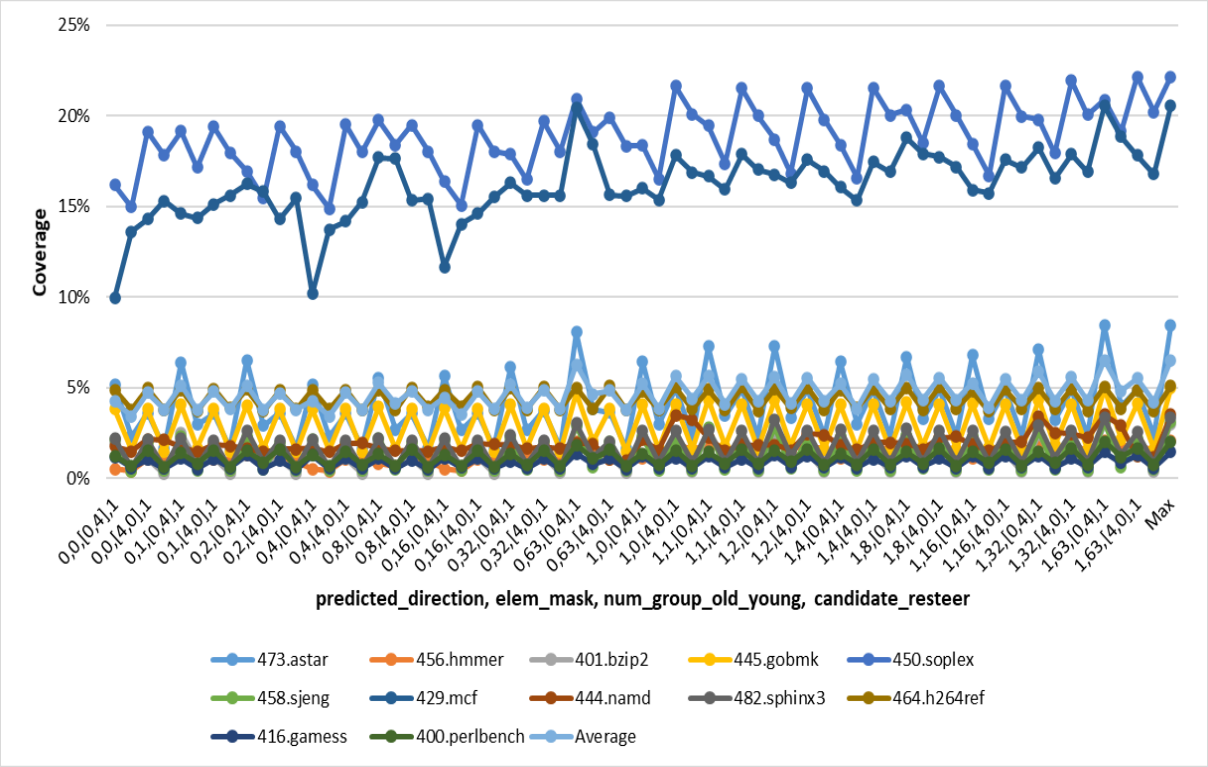
Το Σχήμα 6.3 παρουσιάζει τη γραφική παράσταση του accuracy σε συνάρτηση με παραμέτρους που αφορούν το predicted direction, το element mask, τον αριθμό των groups που θα δημιουργηθούν και το candidate resteer. Κάθε γραμμή αναπαριστά ένα benchmark. Η αξιολόγηση αυτή έγινε με στόχο να εντοπίσουμε τις παραμέτρους που συμβάλουν στην αύξηση του accuracy διατηρώντας το confidence και train threshold σταθερά, στα οποία επιλέχθηκαν οι καλύτερες τιμές με βάση την προηγούμενη ανάλυση. Οι τιμές του accuracy κυμαίνονται από 52% έως 83%. Ένα σημαντικό εύρημα είναι το μεγαλύτερο accuracy σε κάθε benchmark παρουσιάζεται πάντα όταν candidate resteer εντολή είναι εκείνη που έχει τουλάχιστο ένα younger resolved conditional branch. Επίσης, φάνηκε ότι πετυχαίνουμε καλύτερο accuracy όταν τα features που χρησιμοποιούνται είναι ο συνδυασμός όλων των features.



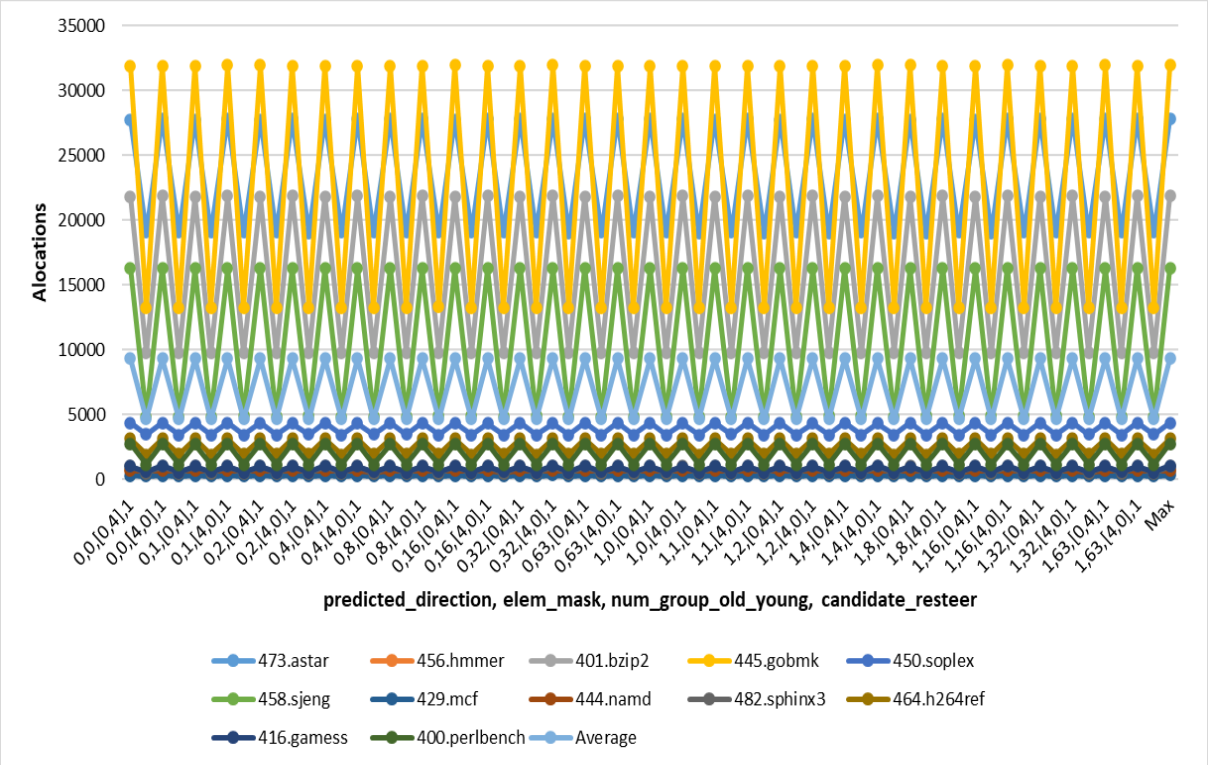
Σχήμα 6.3: Accuracy σε συνάρτηση με παραμέτρους του design space

Το Σχήμα 6.4 παρουσιάζει τη γραφική παράσταση του coverage σε συνάρτηση με παραμέτρους που αφορούν το predicted direction, το element mask, τον αριθμό των groups που θα δημιουργηθούν και το candidate resteer. Κάθε γραμμή αναπαριστά ένα benchmark. Η αξιολόγηση αυτή έγινε με στόχο να εντοπίσουμε τις παραμέτρους που συμβάλουν στην αύξηση του coverage διατηρώντας το confidence και train threshold σταθερά, στα οποία επιλέχθηκαν οι καλύτερες τιμές με βάση την προηγούμενη ανάλυση. Οι τιμές του coverage κυμαίνονται από 1% εώς 21%. Ψηλό coverage σημαίνει ότι κάνουμε resteer περισσότερα mispredicts κάτι το οποίο είναι και το επιθυμητό.

Το Σχήμα 6.5 παρουσιάζει τη γραφική παράσταση των αριθμό των perceptron που δημιουργήθηκαν σε συνάρτηση με παραμέτρους που αφορούν το predicted direction, το element mask, τον αριθμό των groups που θα δημιουργηθούν και το candidate resteer. Κάθε γραμμή αναπαριστά ένα benchmark. Είναι άξιο αναφοράς ότι όταν candidate resteer εντολή είναι εκείνη που έχει τουλάχιστο ένα younger resolved conditional branch ο αριθμός των perceptron που δημιουργούνται είναι πολύ πιο μικρός κάτι που ήταν και αναμενόμενο εφόσον με αυτό το κριτήριο περιορίζουμε τις εντολές που εξετάζουμε.



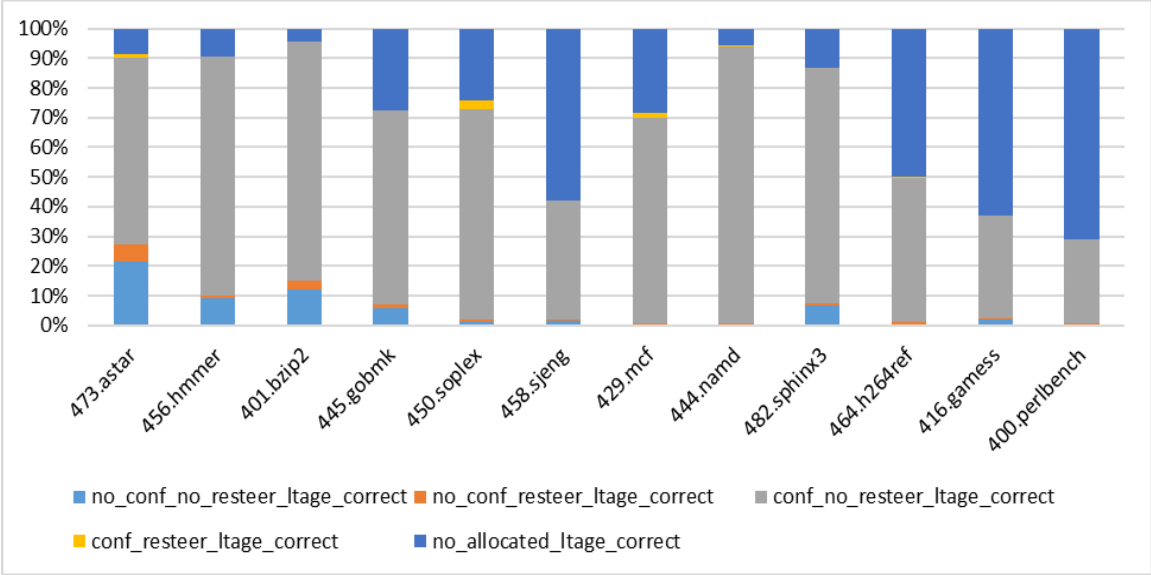
Σχήμα 6.4: Coverage σε συνάρτηση με παραμέτρους του design space



Σχήμα 6.5: Perceptron Allocations σε συνάρτηση με παραμέτρους του design space

Στη συνέχεια, για το κάθε benchmark επιλέξαμε την περίπτωση η οποία προσδίδει το μεγαλύτερο speedup και αναλύσαμε σε μεγαλύτερο βαθμό τη λειτουργία του wrong path predictor. Συγκεκριμένα εξετάσαμε: πόσες φορές ο perceptron δεν είναι confidence και προβλέπει ότι δεν πρέπει να γίνει resteer (γαλάζιο), πόσες φορές ο perceptron δεν είναι confidence και προβλέπει ότι πρέπει να γίνει resteer (πορτοκαλί), πόσες φορές ο perceptron είναι confidence και προβλέπει ότι δεν πρέπει να γίνει resteer (γκρίζο), πόσες φορές ο perceptron είναι confidence και προβλέπει ότι πρέπει να γίνει resteer (κίτρινο) και πόσες φορές δεν είχε δημιουργηθεί ο perceptron (μπλε).

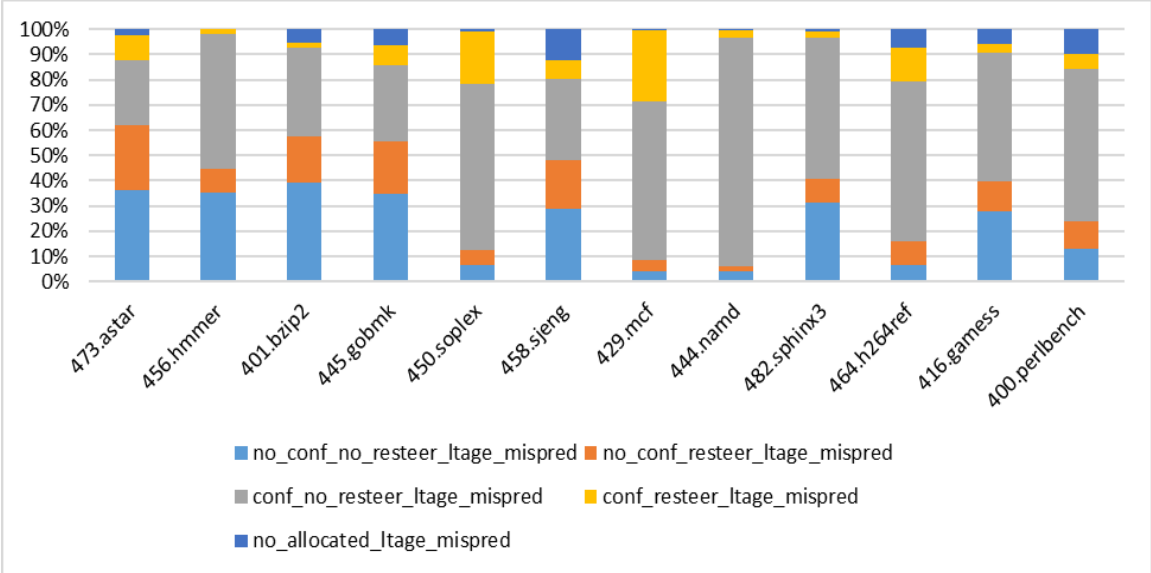
Το Σχήμα 6.6 παρουσιάζει τις περιπτώσεις όπου ο branch predictor είχε ορθή πρόβλεψη και ως τούτου δεν έπρεπε να κάνουμε resteer.Με κίτρινο φαίνεται το ποσοστό που ο wrong path predictor θα έκανε λανθασμένα resteer, το οποίο φαίνεται να είναι μικρό συγκριτικά με τα υπόλοιπα όμως είναι πολύ ακριβό όπως ήδη εξηγήσαμε. Φαίνεται ότι τις περισσότερες φορές ότι ο wrong path predictor σωστά προβλέπει ότι δεν πρέπει να γίνει resteer (γκρίζο) κάτι που είναι ιδιαίτερα ενθαρρυντικό εφόσον ο φαίνεται ότι ο perceptron καταφέρνει να μάθε καλά.



Σχήμα 6.6: Στατιστικά στα οποία η πρόβλεψη του branch predictor ήταν σωστή στις καλύτερες περιπτώσεις κάθε benchmark

Το Σχήμα 6.7 παρουσιάζει τις περιπτώσεις όπου ο branch predictor είχε λανθασμένη πρόβλεψη και ως τούτου έπρεπε να κάνουμε resteer. Με κίτρινο φαίνεται το ποσοστό που ο wrong path predictor θα έκανε σωστά resteer. Επίσης, φαίνεται ότι τις αρκετές φορές ότι ο wrong path

predictor λανθασμένα προβλέπει ότι δεν πρέπει να γίνει resteer (γκρίζο) κάτι που δηλώνει ότι ο predictor χρειάζεται καλύτερη εκπαίδευση ως προς την αναγνώριση των resteer. Με πορτοκαλί φαίνεται το ποσοστό που ο wrong path predictor πρόβλεψε να γίνει resteer αλλά δεν έγινε διότι δεν ξεπερνούσε το confidence threshold. Αυτό δηλώνει ότι με την εύρεση καλύτερων thresholds θα έχουμε αύξηση στο accuracy του predictor.

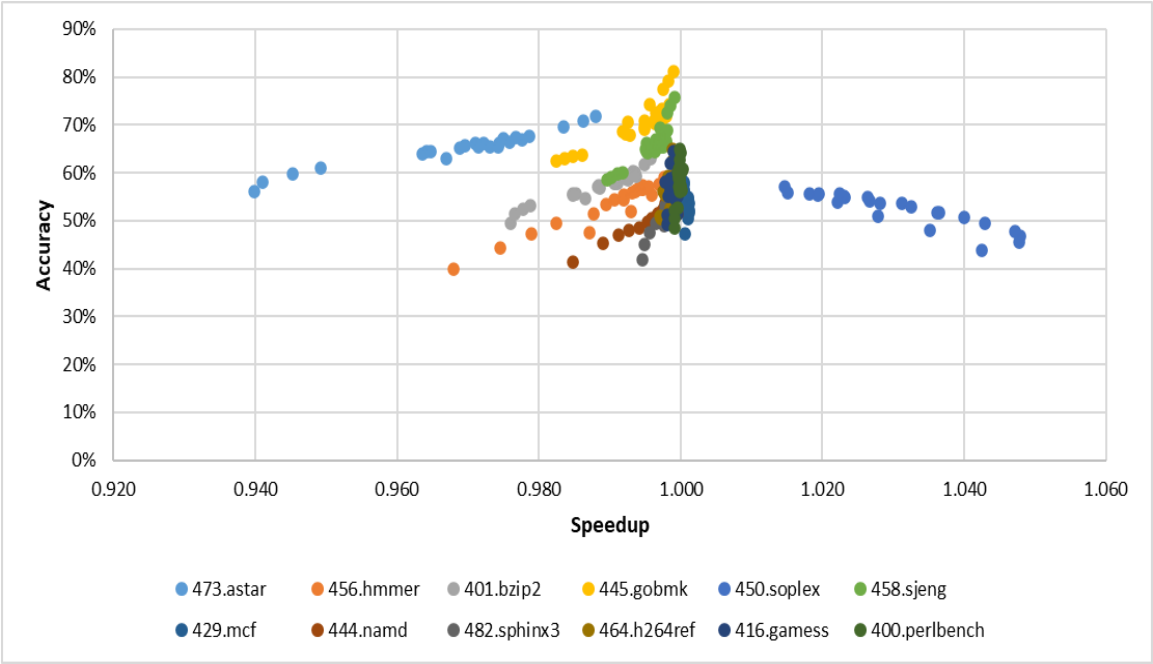


Σχήμα 6.7: Στατιστικά στα οποία η πρόβλεψη του branch predictor ήταν λάθος στις καλύτερες περιπτώσεις κάθε benchmark

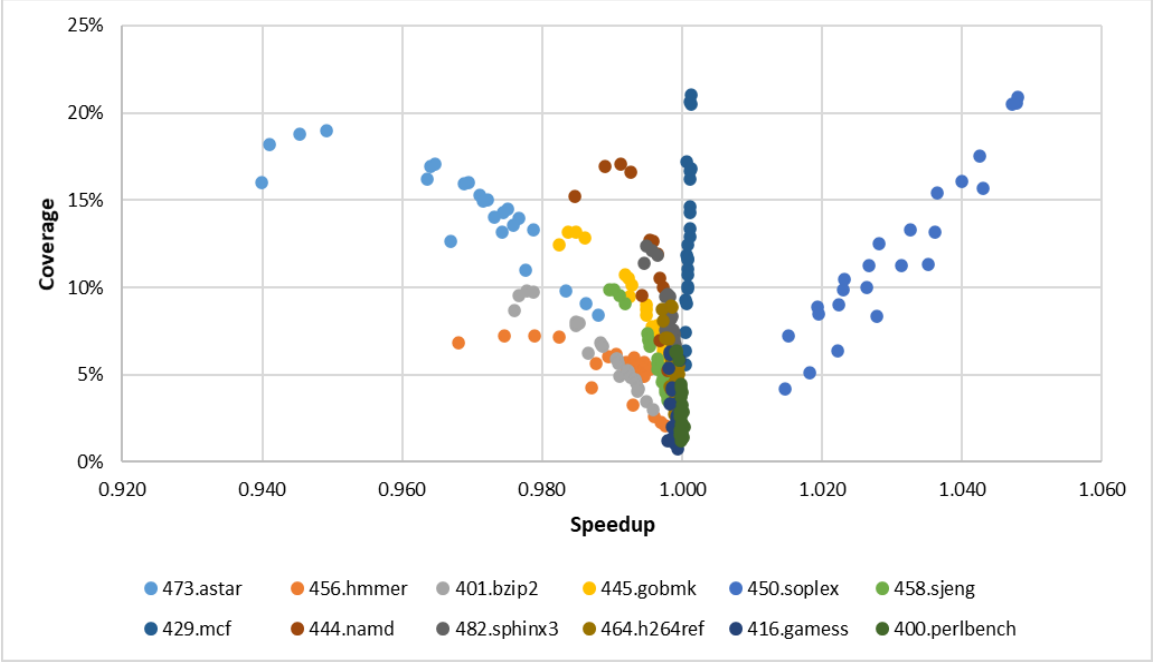
Το Σχήμα 6.8 παρουσιάζει τη γραφική παράσταση του accuracy σε σχέση με το speedup. Παρατηρούμε γραμμική συσχέτιση ανάμεσα στο accuracy και το speedup. Συγκεκριμένα, όσο αυξάνεται το accuracy αυξάνεται και το speedup, αν βέβαια αγνοήσουμε το benchmark soplex το οποίο όπως έχουμε πει το θεωρούμε outlier. Το μήνυμα που μας δίνει αυτή η γραφική είναι πολύ ενθαρρυντικό εφόσον εάν καταφέρουμε να πετύχουμε καλύτερο accuracy στον wrong path predictor τότε θα καταφέρουμε να πάρουμε και καλύτερο speedup το οποίο είναι και το επιθυμητό.

Το Σχήμα 6.9 παρουσιάζει τη γραφική παράσταση του coverage σε σχέση με το speedup. Παρατηρούμε επίσης γραμμική συσχέτιση ανάμεσα στο coverage και το speedup. Αυτή τη φορά όμως με την αύξηση του coverage μειώνεται speedup. Αυτή η παρατήρηση μπορεί να δικαιολογηθεί από το γεγονός ότι όταν το coverage είναι ψηλό γίνονται πολλά restters, άρα

γίνονται και πολλά λανθασμένα resteers κάτι το οποίο προκαλεί μείωση του speedup. Ιδανικά θα θέλαμε να έχουμε και ψηλό accuracy και ψηλό speedup.



Σχήμα 6.8: Speedup σε συνάρτηση με το Accuracy



Σχήμα 6.9: Speedup σε συνάρτηση με το Coverage

Κεφάλαιο 7

Σχετική εργασία

7.1 Wrong Path Events

Οι Armstrong et al. (2004) [6] εισηγήθηκαν την ιδέα των Wrong Path Events (WPE) και την χρήση τους για βελτίωση της επίδοσης του επεξεργαστή. Η ομάδα αυτή επέλεξε να εστιάσει την έρευνα της στον εντοπισμό WPE ανά κατηγορία εντολών.

Υπάρχουν δύο είδη WPEs. Τα Hard WPEs, όπου όταν παρατηρήσουμε μία συγκεκριμένη συμπεριφορά την οποία θεωρούμε WPE υπάρχει μια 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 μια συμπεριφορά που είναι αρκετά σπάνια να εμφανιστεί στο ορθό μονοπάτι δηλαδή υπάρχει πολύ μεγάλη πιθανότητα η εντολή να βρίσκεται λάθος μονοπάτι.

Πιο συγκεκριμένα ασχολήθηκαν και ανάλυσαν συμπεριφορές διάφορων τύπων εντολών:

Memory Instructions: Έθεσαν ως Wrong Path Event το dereference ενός NULL Pointer, γιατί αυτό δεν συμβαίνει ποτέ ότι υπό κανονικές συνθήκες στο correct path. Αν εντοπιστεί κάποιος δείχτης ο οποίος είναι NULL σηματοδοτεί ότι βρισκόμαστε σε wrong path. Επίσης παρατηρήθηκε ότι σε περιπτώσεις που μια εντολή βρίσκεται σε λάθος μονοπάτι μπορεί να χρησιμοποιήσει για διάβασμα ή γράψιμο κάποιο μέρος μνήμης το όποιο 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 μονοπάτι.

Control Flow Instructions: Για τις εντολές διακλάδωσης παρατηρήθηκε ότι αν προκύψουν τρία διαδοχικά mispredicts από τρεις εντολές διακλαδώσεις και υπάρχει στο reorder buffer εντολή διακλάδωσης η οποία ακόμα να ολοκληρώσει την εκτέλεση της, και είναι πιο παλιά 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: Παρατηρήθηκε ότι πολλές μεταβλητές που χρησιμοποιούνται για εντολές οι οποίες είναι στο λάθος μονοπάτι δεν αρχικοποιούνται, και προκύπτουν αδύνατες πράξεις όπως για παράδειγμα διαιρέσεις με το 0 ή τετραγωνικές ρίζες για αρνητικούς αριθμούς.

Κατέληξαν στο συμπέρασμα ότι εντοπίζοντας αυτά τα φαινόμενα και αποκαθιστώντας άμεσα το λάθος μονοπάτι είχαν κάποια αξιοσημείωτη αύξηση στο IPC του επεξεργαστή για μερικά benchmarks.

Η προσέγγιση τους διαφέρει αρκετά από την δική μας. Εισηγήθηκαν, τη χρήση δυο predictors. Ο πρώτος θα προβλέπει εάν μια εντολή βρίσκεται στο λάθος μονοπάτι με τη χρήση στατικών features τα οποία ονόμασαν Wrong Path Events (WPE). Ο δεύτερος θα εντοπίζει ποια ήταν η εντολή που προκάλεσε το λάθος μονοπάτι. Αντίθετα, στη δική μας εργασία εισηγηθήκαμε τη χρήση μόνο ενός predictor ο οποίος θα εκπαιδεύεται δυναμικά και θα χρησιμοποιεί features μετά το renaming stage.

7.2 Dynamic Branch Prediction with Perceptrons

Οι Jimenez και Lin (2001) [7] παρουσίασαν μια νέα προσέγγιση για έναν branch predictor. Η βασική ιδέα ήταν ότι χρησιμοποιήθηκε ένα από τα πιο απλά νευρωνικά δίκτυα, ο perceptron. Με άλλα λόγια έδειξαν ότι ένα μοντέλο μηχανικής μάθησης (machine learning) μπορεί να υλοποιηθεί στο hardware με σκοπό την αύξηση της επίδοσης του branch predictor. Ο μηχανισμός πρόβλεψης που υλοποίησαν επιτυγχάνει αυξημένη ακρίβεια κάνοντας χρήση μεγάλων branch histories, κάτι που είναι δυνατό επειδή οι πόροι του hardware για τη συγκεκριμένη μέθοδο κλιμακώνονται γραμμικά με το μήκος του history. Τα περισσότερα νευρωνικά δίκτυα είναι δύσκολο να υλοποιηθούν σαν branch predictors γι’ αυτό επιλέχθηκε ένας απλός νευρώνας perceptron.

Η προσέγγιση τους διαφέρει από την δική μας εφόσον στην παρούσα εργασία ο perceptron δεν θα εκπαιδευτεί σαν branch predictor για να προβλέπει τη κατεύθυνση της διακλάδωσης αλλά θα εκπαιδευτεί για να προβλέπει εάν κάποια unresolved conditional branch εντολή στο reorder buffer θα γίνει mispredict. Επίσης, το διάνυσμα εισόδου στον perceptron δεν θα αποτελείται μόνο από global history αλλά θα περιέχει και άλλα features τα οποία θα προέρχονται μετά το renaming stage στο pipeline.

7.3 Fast Path-Based Neural Branch Prediction

Ο Jimenez (2003) [8] εισηγήθηκε τη χρήση ενός perceptron predictor ως overriding predictor. Δηλαδή χρησιμοποιείται ως ένας δεύτερος predictor που συναγωνίζεται τον πρώτο, θα έχει μεγαλύτερο μέγεθος από τον πρώτο και θα εκπαιδεύεται με features πριν το renaming stage (πχ global history, PC)

Η προσέγγιση τους διαφέρει από τη δική μας εργασία εφόσον εισηγηθήκαμε τη χρήση μόνο ενός predictor ο οποίος θα εκπαιδεύεται δυναμικά και θα χρησιμοποιεί features μετά το renaming stage.

Κεφάλαιο 8

Συμπεράσματα και Μελλοντική εργασία

8.1 Συμπεράσματα

Στη παρούσα εργασία, μελετώντας τη συμπεριφορά των benchmarks της σουίτας SPEC CPU 2006, καταφέραμε να εξετάσουμε την επίδραση που έχουν τα mispredicts στην συνολική επίδοση αυτών των προγραμμάτων. Φάνηκε λοιπόν, πως τα conditional branch mispredicts επιβαρύνουν αρκετά τη συνολική επίδοση, προσθέτοντας μεγάλο αριθμό κύκλων στο χρόνο εκτέλεσης λόγω της εκτέλεσης εντολών στο λάθος μονοπάτι.

Επίσης, υλοποιήθηκε ένας wrong path predictor ο οποίος χρησιμοποιεί features μετά το renaming stage στο pipeline και εκπαιδεύεται δυναμικά την ώρα της εκτέλεσης. Με βάση τα αποτελέσματα μας μπορούμε να πούμε ότι το λάθος μονοπάτι είναι προβλέψιμο με ακρίβεια 52%-83% ανάλογα με το κάθε benchmark.

Από την ανάλυση μας φάνηκε ότι υπάρχει γραμμική συσχέτιση ανάμεσα στην ακρίβεια και την επιτάχυνση. Συγκεκριμένα, όσο αυξάνεται το accuracy αυξάνεται και το speedup. Το μήνυμα αυτό είναι πολύ ενθαρρυντικό εφόσον εάν καταφέρουμε να πετύχουμε καλύτερο accuracy στον wrong path predictor τότε θα καταφέρουμε να πάρουμε και καλύτερο speedup το οποίο είναι και το επιθυμητό.

Το speedup εξαρτάται τόσο από το accuracy όσο και από το coverage. Εκτιμούμε ότι αν καταφέρουμε να πέτυχουμε και ψηλό accuracy και ψηλό coverage, θα έχουμε και αύξηση του speedup.

Τέλος, το gain και το lost επηρεάζουν σημαντικά το opportunity για βελτίωση της επίδοσης του επεξεργαστή. Επομένως αξίζει να ερευνήσουμε περισσότερο την περιοχή αυτή και να αναζητήσουμε τεχνικές οι οποίες θα συμβάλουν στην αύξηση του gain και στη μείωση του lost.

8.2 Μελλοντική εργασία

Στόχος μας είναι να συνεχίσουμε και να εξελίξουμε την ιδέα μας έτσι ώστε να μπορέσουμε να πετύχουμε αύξηση της επίδοσης του επεξεργαστή με την εφαρμογή του wrong path predictor. Σίγουρα μελλοντικά μπορεί να ερευνηθεί ένα πολύ μεγαλύτερο μέρος του design space. Με την εξερεύνηση μεγαλύτερου μέρους του design space πιθανόν να εντοπιστούν καλύτερες παράμετροι οι οποίες πιθανόν θα επιφέρουν αύξηση του speedup.

Ένα σημαντικό σημείο μελλοντικής έρευνας είναι η διερεύνηση κατά πόσο μπορούμε να αυξήσουμε το gain και να μειώσουμε το lost. Αυτό θα μπορούσε να γίνει εάν ήταν δυνατόν να εκτελέσουμε την πρόβλεψη του wrong path predictor σε κάποιο στάδιο του pipeline πριν από το resolution. Με αυτό τον τρόπο θα εντοπίζεται πιο γρήγορα η εντολή που πιθανόν είναι mispredict και έτσι θα εξοικονομούμε περισσότερους κύκλους μηχανής. Σίγουρα όμως η πρόβλεψη του wrong path predictor δεν θα μπορούσε να γίνει στο στάδιο του fetch διότι αν είχαμε από το στάδιο του fetch τις πληροφορίες που θέλουμε θα μπορούσαμε απλά να βελτιστοποιήσουμε τον branch predictor.

Επίσης, θα μπορούσε να γίνει ένα sensitivity analysis όσον αφορά τον αριθμό των front-end (ή slot) σταδίων του pipeline έτσι ώστε να διερευνηθεί κατά πόσο η μεταβολή των σταδίων του pipeline θα μπορούσε να επηρεάσει το gain και το lost. Εκτιμούμε ότι με τη περαιτέρω μείωση των front-end stages θα παρατηρηθεί αύξηση του gain και μείωση του lost.

Τέλος, θα μπορούσε να γίνει μια πιο εκτενής έρευνα ως προς την επιλογή του predictor. Ο perceptron είναι ένα πολύ απλός predictor ο οποίος υλοποιείται αποδοτικά στο hardware και στην πειραματική μας αξιολόγηση είχε αρκετά ψηλό accuracy κάτι το οποίο δεν περιμέναμε. Ωστόσο, μια πιο εκτενής έρευνα ως προς την επιλογή του predictor θα μπορούμε ίσως να εντοπίσει έναν καλύτερο predictor οποίος θα μπορούσε να επιφέρει ακόμα μεγαλύτερο accuracy. Επίσης, θα μπορούσε ο predictor να έχει μεγαλύτερο μέγεθος έτσι να αποφευχθεί η διαδικασία της συμπίεσης features που πιθανόν να επιφέρει απώλειες στις πληροφορίες.

Βιβλιογραφία

[1] G. Chrysos and J. Emer, "Memory Dependence Prediction using Store Sets," *Annual International Symposium on Computer Architecture*, 1998.

[2] D. Patterson and J. Hennessy, "The Processor," in *Computer Organization and Design: The Hardware / Software Interface*, 2012.

[3] S. McFarling, "Combining Branch Predictors," *Digital Western Research Laboratory*, 1993.

[4] J. Smith and G. Sohi, "The microarchitecture of superscalar processors," *TR UW*, 1995.

[5] "SPEC CPU 2006," SPEC, [Online]. Available: https://www.spec.org/cpu2006/.

[6] D. Armstrong, H. Kim, O. Mutlu and Y. Patt, "Wrong Path Events: Exploiting Illegal and Unusual Program Behavior for Early Misprediction Recovery," *MICRO-37*, 2004.

[7] D. Jiménez and C. Lin, "Dynamic Branch Prediction with Perceptrons," *HPCA*, 2001.

[8] D. Jiménez, "Fast Path-Based Neural Branch Prediction," *MICRO-36*, 2003.