ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ

Μελέτη τεχνικών επιλογής κώδικα σε μεταγλωττιστές

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ
της
ΚΟΥΤΣΟΥΚΟΥ ΕΛΕΝΗΣ

Επιβλέπων καθηγητής: κ. Μασσέλος Κων/νος

Τρίπολη,Απρίλιος 2013

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ

Μελέτη τεχνικών επιλογής κώδικα σε μεταγλωττιστές

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ
της
ΚΟΥΤΣΟΥΚΟΥ ΕΛΕΝΗΣ

ΑΜ:2009014

Επιβλέπων καθηγητής: κ. Μασσέλος Κων/νος

Τρίπολη,Απρίλιος 2013

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΥΧΑΡΙΣΤΙΕΣ

Στους γονείς μου για την πολύτιμη βοήθεια και προσφορά τους στην ολοκλήρωση των μεταπτυχιακών μου σπουδών στο Πανεπιστήμιο Πελοποννήσου καθώς και στους καθηγητές μου.

ΚΕΦΑΛΑΙΟ 1: ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΕΠΙΛΟΓΗ ΚΩΔΙΚΑ ΣΤΟΥΣ ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ

1.1 ΕΙΣΑΓΩΓΗ ΣΤΟΥΣ ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ

Ο μεταγλωττιστής (compiler) ή αλλιώς μεταφραστής, είναι ένα πρόγραμμα (ή σύνολο προγραμμάτων) που μετατρέπει τον πηγαίο κώδικα (source code) μιας εφαρμογής, σε γλώσσα μηχανής (machine code). Ο κύριος λόγος για τη μετατροπή αυτή είναι η δημιουργία εκτελέσιμου προγράμματος. Στην καρδιά της διαδικασίας μετατροπής βρίσκεται η αντιστοίχιση συντακτικών δομών σε μία γλώσσα προγραμματισμού υψηλού επιπέδου σε δομές χαμηλού επιπέδου, οι οποίες βρίσκονται πλησιέστερα στο επίπεδο της συγκεκριμένης μηχανής (επεξεργαστή) που έχουμε στη διάθεσή μας κάθε φορά. Οι βασικές λειτουργίες που επιτελεί ένας μεταγλωττιστής είναι η λεκτική ανάλυση, η συντακτική ανάλυση, η παραγωγή κώδικα και η βελτιστοποίηση κώδικα. Οι δύο πρώτες από αυτές τις φάσεις θεωρείται ότι πραγματοποιούνται στο frontend (εμπροσθοφυλακή) του μεταγλωττιστή και στοχεύουν στην παραγωγή της ενδιάμεσης αναπαράστασης (Intermediate Representation: IR) για τον πηγαίο κώδικα της εφαρμογής. Οι φάσεις της παραγωγής και βελτιστοποίησης κώδικα λαμβάνουν χώρα στο backend (οπισθοφυλακή) του μεταγλωττιστή και πραγματεύονται τη μετατροπή της ενδιάμεσης αναπαράστασης σε βελτιστοποιημένο κώδικα συμβολομεταφραστή (ή κώδικα μηχανής) για τον εν λόγω επεξεργαστή.

Η παραγωγή κώδικα συνίσταται κατά κύριο λόγο στις φάσεις της επιλογής κώδικα, του καταμερισμού καταχωρητών και του χρονοπρογραμματισμού λειτουργιών. Η γεννήτρια κώδικα του μεταγλωττιστή μετατρέπει τα προγράμματα από την ενδιάμεση αναπαράσταση της μεταγλώττισης του σε συμβολική (assembly) γλώσσα ή σε δυαδικό κώδικα μηχανής. Η γέννηση κώδικα μπορεί να χωριστεί σε τρεις φάσεις: η Επιλογή Κώδικα μετατοπίζει τις εργασίες της ενδιάμεσης αναπαράστασης σε εντολές για το μηχάνημα στόχο · ο χρονοπρογραμματισμός εντολών να αναδιατάσσει τις εντολές κατά τρόπο ώστε να επιταχύνεται η εκτέλεση του προγράμματος χωρίς (το δυνατόν) αύξηση της πίεσης καταχωρητών · η κατανομή των καταχωρητών με την οποία γίνεται αντικατάσταση των εικονικών καταχωρητών που χρησιμοποιούνται στην ενδιάμεση αναπαράσταση με πραγματικούς καταχωρητές με το μικρότερο, το δυνατόν, αριθμό εγχύσεων μεταβλητών προς στη μνήμη.

Στην παρούσα εργασία θα επικεντρωθούμε στη μελέτη νέων τεχνικών για την επιλογή κώδικα. Στην επιστήμη των υπολογιστών, η επιλογή κώδικα είναι η φάση του backend μεταγλωττιστή που μετατρέπει μία αναπαράσταση σε μορφή αφηρημένου συντακτικού δένδρου (Abstract Syntax Tree: AST) όπως αυτή προκύπτει μετά τη συντακτική και σημασιολογική ανάλυση σε μια χαμηλού επιπέδου IR πολύ κοντά στην τελική γλώσσα-στόχο. Σε έναν τυπικό μεταγλωττιστή, προηγείται τόσο του καταμερισμού καταχωρητών όσο και χρονοπρογραμματισμού λειτουργιών, ώστε η έξοδος της διαδικασίας αυτής να θεωρεί την ύπαρξη ενός συνόλου από απεριόριστους, τον αριθμό, εικονικούς καταχωρητές. Στόχος της επιλογής κώδικα είναι η επικάλυψη του AST από υποδένδρα/πλακίδια τα οποία αντιστοιχούν στις εντολές που περιλαμβάνονται στην αρχιτεκτονική συνόλου εντολών του επεξεργαστή με τρόπο κατά το δυνατόν καλύτερο ως προς κάποιο μετρικό (συνήθως εκτιμώμενοι κύκλοι εκτέλεσης του κώδικα). Ορισμένες τεχνικές προσφέρουν τη βέλτιστη κάλυψη του δένδρου (optimal tree-based code selection). Οι τεχνικές όμως αυτές οι οποίες λειτουργούν στο επίπεδο του AST δεν λαμβάνουν υπόψη ισχυρές εντολές που μπορεί να διαθέτει ο επεξεργαστής, όπως είναι οι εντολές πολλαπλών εξόδων MOI (Multiple Output Instructions). Το φυσικό είδος αναπαράστασης στο οποίο μπορεί να εξεταστεί η κάλυψη με πλακίδια MOI είναι ένας γράφος εξάρτησης δεδομένων, στο επίπεδο των βασικών μπλοκ της εφαρμογής. Για αυτές οι τεχνικές που πραγματοποιούν κάλυψη γράφου, προκύπτει από την απόδειξη της ενότητας 1.2 (από τον Todd Proebstring) ότι η κάλυψη DAG με ελάχιστο κόστος είναι δυσεπίλυτο πρόβλημα.

Ο πρωτεύων στόχος για ένα μεταγλωττιστή είναι η ορθότητα, το γεγονός δηλαδή ότι όλα τα έγκυρα προγράμματα πρέπει να μεταφραστούν σωστά σε γλώσσα μηχανής. Εξίσου σημαντικός στόχος είναι η δημιουργία όσο το δυνατό πιο γρήγορα μεταφρασμένου κώδικα, δηλαδή ταχύτερου στην εκτέλεση. Σε ενσωματωμένα συστήματα σημαντικό ρόλο κατέχει και το μέγεθος του κώδικα. Ως στόχους μπορούμε επίσης να αναφέρουμε τη διαλειτουργικότητα μεταξύ των διαφόρων γλωσσών προγραμματισμού, ακόμα και την όσο το δυνατό καλύτερη μεταφορά εφαρμογών μεταξύ διαφορετικών συστημάτων. Όλα αυτά είναι σημαντικά θέματα και πρέπει να ληφθούν σοβαρά υπόψη. Ιδιαίτερο ενδιαφέρον θα δοθεί παρακάτω στους βελτιστοποιητικούς μεταγλωττιστές και συγκεκριμένα, στις τεχνικές ως προς τη παραλληλία των εντολών.

1.2 ΚΑΛΥΨΗ DAG ΕΛΑΧΙΣΤΟΥ ΚΟΣΤΟΥΣ

Το ελάχιστο Κόστος στην επιλογή εντολής σε DAG είναι NP-πλήρες

Η παραγωγή ελάχιστου κόστους στην επιλογή εντολών για DAG από την εύρεση πρότυπου «ζευγάρι» είναι NP πλήρες. Ο τετριμμένος έλεγχος ακολουθεί από την μείωση ικανοποιησιμότητας. Δημιουργεί έναν DAG για τον τύπο που θέλουμε να ικανοποιήσουμε. Έπειτα, έχουμε τους ακόλουθους κανόνες(όλοι με μοναδιαίο κόστος) :

True: VARIABLE

False: VARIABLE

False: not(True)

True: not(False)

True: or(True,True)

True: or(True,False)

True: or(False,True)

False: or(False,False)

True: and(True,True)

False: and(True,False)

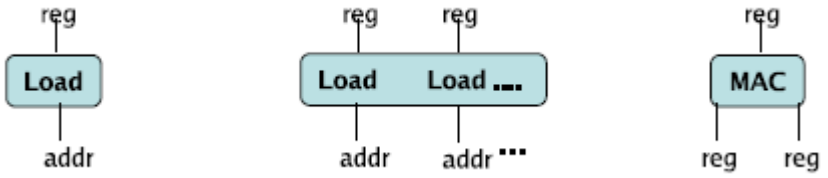
False: and(False,True)

False: and(False,False)

Εάν μπορέσουμε να αντλήσουμε μια κάλυψη του DAG που μειώνει τα True με το κόστος ακριβώς τον αριθμό των κόμβων, τότε ο τύπος είναι ικανοποιητικός. Διαφορετικά όχι. Εκτός και αν έχουμε ένα διαφορετικό τύπο βελτιστοποίησης, αυτό αποδεικνύει ότι η γέννηση του βέλτιστου κώδικα DAG είναι NP πλήρες. Σημειώνουμε ότι δεν βασίζεται στις συνηθισμένες πολυπλοκότητες :1) του αριθμού των καταχωρητών και 2)των ασύμμετρων εντολών.

Ο σχεδιασμός και η ανάπτυξη ενσωματωμένων επεξεργαστών γίνονται κάτω από πολύ αυστηρούς περιορισμούς. Οι επεξεργαστές μπορούν να χειριστούν δεδομένα σε υψηλά ποσοστά είτε ως ψηφιακό σήμα ή ως πρωτόκολλο επεξεργασίας του δικτύου καταναλώνοντας όσο γίνεται λιγότερη ενέργεια και πρέπει να προγραμματίζονται έτσι ώστε να εγγυόμαστε την επαναχρησιμοποίηση τους σε διαφορετικές εφαρμογές. Ως εκ τούτου, ιδιαίτερα στην επεξεργασία πρωτοκόλλων η εφαρμογή επεξεργαστών ειδικού σετ εντολών (Application-Specific Instruction set Processor: ASIP) γίνεται όλο και περισσότερο διαδεδομένη δεδομένου ότι προσφέρουν έναν καλό συμβιβασμό μεταξύ της αποτελεσματικότητας (επιδόσεις ταχύτητας, επιφάνειας ολοκληρωμένου και κατανάλωσης ισχύος) και της ευελιξίας (που προσφέρει η δυνατότητα επαναπρογραμματισμού). Συνήθως,

οι ASIP υλοποιούν σε επίπεδο υλικού μια σειρά από ειδικές εντολές, προσαρμοσμένες στις ανάγκες των εφαρμογών-στόχων Το κυρίαρχο χαρακτηριστικό αυτών των εντολών είναι ο τοπικός παραλληλισμός. Οι εντολές περιλαμβάνουν αρκετές εργασίες που εκτελούνται παράλληλα όπως οι ADD, MUL, MAC (πολλαπλασιασμός και συσσώρευση) κτλ και παράγουν την ίδια στιγμή περισσότερα αποτελέσματα του ενός. Συνεπώς, θα αναφέρονται ως Εντολές Πολλαπλών Εξόδων (MOI)



(a) (b) (c)

Σχήμα 1: (a)απλή εντολή , (b) πολλαπλή έξοδος εντολών, (c)αλυσιδωτές εντολές

Είναι ακριβώς αυτή η ιδιότητα που κάνει τη διαφορά μεταξύ MOI και στα άλλα είδη των εντολών. Λαμβάνοντας υπόψη τις απλές εντολές (σχήμα 1a) και τις αλυσιδωτές (chained) εντολές (σχήμα 1c) μπορούν να εκπροσωπούνται ως πρότυπα δέντρου στο IR, θα έχουν πάντα ένα θα έχουν πάντα ένα μεγαλύτερο (σχήμα 1c). Στο πεδίο της Ψηφιακής Επεξεργασίας Σήματος(Digital Signal Processing: DSP), αποτελεί ήδη ένα φυσικό τρόπο για να αυξάνεται η απόδοση του κώδικα. Διακεκριμένα παραδείγματα είναι οι εντολές οι οποίες επιτρέπουν την ταυτόχρονη πρόσβαση σε διαφορετικές θέσεις μνήμης. Για παράδειγμα, στον επεξεργαστή SONY pDSPμε μια εντολή όπως η PLDXY r1,@a,r2,@b μπορούν να φορτωθούν οι μεταβλητές a και b μέσα από τους καταχωρητές r1 και r2 ταυτόχρονα. Αυτές οι εντολές μπορούν να έχουν πρόσβαση μνήμης ταχύτερα από την εκτέλεση σε παράλληλα χωρισμένες στις θέσεις μνήμης με τη χρήση των δεδομένων και παράλληλων διαύλων διεύθυνσης. Από την ενθυλάκωση των παράλληλων ενεργειών σε αρχιτεκτονικές εντολές σε επίπεδο υλικού, εντυπωσιακή ταχύτητα μπορεί επίσης να ληφθεί για την επεξεργασία πρωτοκόλλου χωρίς να θυσιάζεται η υπερβολικά μεγάλη ευελιξία για την υλοποίηση εφαρμογών. Ωστόσο, η αυξανόμενη πολυπλοκότητα των πρωτοκόλλων δικτύου καθιστά απαγορευτικά δύσκολη την άμεση εφαρμογή όλων αυτών με χειρωνακτικό προγραμματισμό σε συμβολική γλώσσα. . Η εξελιγμένη υποστήριξη του μεταγλωττιστή ως εκ τούτου απαιτείται πιεστική, προκειμένου να εξασφαλιστεί η γρήγορη ανάπτυξη των εφαρμογών και κατά συνέπεια η αποδοχή από τους καταναλωτές για έναν επεξεργαστή. Αυτό συνεπάγεται το πρόβλημα του χειρισμού MOI που ακόμη και για δημοφιλείς μεταγλωττιστές όπως ο GCC

[GCC]. Δεν είναι δυνατή, δεδομένου ότι τυπικά οι μεταγλωττιστές εργάζονται σε δέντρο ανάλυσης [Scharwaechter01] κατά τη φάση επιλογής κώδικα. Αν η αρχιτεκτονική του επεξεργαστή στόχου περιέχει εγγενώς παράλληλες εντολές, ο κώδικας που παράγεται από το δέντρο ανάλυσης μπορεί να διαφέρει σημαντικά από το βέλτιστο. Ο λόγος για αυτό είναι ότι το πεδίο εφαρμογής της ανάλυσης δέντρου περιορίζεται σε Δέντρα Ροής Δεδομένων Data Flow Trees: DFT). Συνεπώς, οι εντολές που περιλαμβάνουν την πλήρη λειτουργικότητα υπερβαίνουν το πεδίο εφαρμογής ενός DFT με την διάβαση και κάλυψη του οποίου μπορούν να προκύψουν μόνο εντολές μονής εξόδου (Single Output Instructions: SOI.Για να ξεπεραστεί αυτό το πεδίο εφαρμογής, η έννοια του DFT πρέπει να επεκταθεί, τουλάχιστον μέχρι την εμβέλεια ενός βασικού μπλοκ το οποίο στη συνέχεια παρουσιάζεται ως γραφική παράσταση της ροής δεδομένων (Data Flow Graph: DFG). Δυστυχώς, το μοτίβο που ταιριάζει στη γραφική παράσταση δεδομένων είναι NP-πλήρες σε γενικές γραμμές (Scharwaechter02). Συνήθης προσέγγιση για την επίλυση αυτού του προβλήματος βασίζεται σε ευριστικές ή εμπειρικές, εγγενώς μη-βέλτιστες τεχνικές. Σε ορισμένες περιπτώσεις οι εφαρμογές πρέπει να τροποποιηθούν χειρωνακτικά, κάτι που μπορεί να οδηγήσει σε επιβάρυνση από άποψη ανθρωποχρόνου για μεγάλες εφαρμογές. Επιπλέον, το εξαιρετικά ετερογενές τοπίο της εφαρμογής σε ειδικές αρχιτεκτονικές όπως οι ASIP και οι DSP απαιτεί ιδιαίτερα ευέλικτα εργαλεία προγραμματισμού. Οι εν λόγω αρχιτεκτονικές αναπτύσσονται συνήθως στο πλαίσιο μιας επαναληπτικής διαδικασίας, κατά την οποία η αρχιτεκτονική υπό σχεδίαση σταδιακά τροποποιείται και εξευγενίζεται. Προφανώς, οι μεταγλωττιστές πρέπει να είναι εύκολα επαναστοχεύσιμοι προκειμένου να στηρίξουν διαφορετικές επιλογές στην κατάρτιση της αρχιτεκτονικής συνόλου εντολών και τη σχεδίαση της μικροαρχιτεκτονικής κατά τη διάρκεια της ανάπτυξης του επεξεργαστή. Υπάρχουν αρκετοί γεννήτορες κώδικα (όπως Burg [2], Iburg [3], Olive [4]) που παράγουν κώδικα επιλογής εντολών ο οποίος περιγράφεται σε C. Στον εν λόγω κώδικα επιλογής, περιγραφές μπορούν πολύ άνετα να ενταχθούν σε υφιστάμενες φάσεις του backend μεταγλωττιστή. Βασικά, ένας τέτοιος κώδικας-γεννήτρια δέχεται μια γραμματική δέντρου ως είσοδο η οποία μοντελοποιεί το σύνολο εντολών της υποκείμενης αρχιτεκτονικής στόχου όσον αφορά τα IR-πρότυπα. Με βάση αυτή τη γραμματική, παράγεται ο κώδικας C για την επιλογή χρήσης αυτών των μοτίβων/υποδένδρων. Δεδομένου ότι αυτή η γραμματική δέντρου περιλαμβάνει μόνο μοτίβα δένδρου, τα παραγόμενα από τον κώδικα επιλογής στερούνται τη δυνατότητα να εκμεταλλευτούν οποιαδήποτε MOI. Στην εργασία ένα νέο εργαλείο με το όνομα Cburg παρουσιάζεται για τη γέννηση κώδικα επιλογής από γράφο Ο Cburg επεκτείνει την έννοια

των παραδοσιακών εργαλείων που αναφέρονται στον τρόπο με τον οποίο η γραμματική μπορεί να καταγραφεί για την επιλογή κώδικα από γράφημα με βάση νέο αλγόριθμο αντιστοίχισης που παράγεται έτσι ώστε να μπορούν να διαμορφωθούν και να συνδυαστούν εντολές MOI. Έτσι, η χειροκίνητη εισαγωγή του καλεί CKF ή στην εφαρμογή παραλείπεται. Η συγκεκριμένη γεννήτρια (για την ακρίβεια πρόκειται για γεννήτορα γεννήτριας κώδικα – code generator generator) έχει βρει χρήση στα πλαίσια του μεταγλωττιστή LCC με την ενσωμάτωση του Cburg σε αυτόν [Scharwaechter 04]. Αυτή η τροποποίηση του LCC έχει ως στόχο την επιλογή κώδικα για έναν εκτεταμένο ASIP με βάση την αρχιτεκτονική MIPS [Scharwaechter 17]. Στην ίδια εργασία αναλύεται η ανάπτυξη του ευρετικού αλγόριθμου για την εκμετάλλευση MOI κατά τη φάση της επιλογής κώδικα παραδοσιακά δημοφιλής μέθοδος για την επιλογή κώδικα είναι η ανάλυση δέντρου. Επιτρέπει πολύ γρήγορους επιλογείς κώδικα, οι οποίοι προσφέρουν εγγυήσεις βέλτιστης κάλυψης (όσον αφορά την περιγραφή της μηχανής). Η ανάλυση δέντρου λειτουργεί μόνο εάν η ενδιάμεση αναπαράσταση είναι ένα δέντρο. Ωστόσο, η προτιμώμενη ενδιάμεση αναπαράσταση είναι συχνά με τη μορφή DAG (κατευθυνόμενα άκυκλα γραφήματα), ιδιαίτερα για τη γέννηση κώδικα σε αρχιτεκτονικές με ειδικά χαρακτηριστικά. Η εργασία [Ertl] επεκτείνεται στην ανάλυση δέντρου για την αντιμετώπιση των DAG. Οι κύριες συνεισφορές είναι: μια γραμμικού χρόνου μέθοδος για την ανάλυσης DAG, και ένας έλεγχος, ώστε να αποδεικνύεται το εάν αυτή η μέθοδος αναλύει όλα τα DAG με βέλτιστο τρόπο για μια δεδομένη γραμματική. Στο σχήμα 2 δίνεται η περιγραφή μίας τέτοιας, προς ανάλυση, γραμματικής που περιγράφει μία στοχευόμενη μηχανή.

	nonterminal→pattern		cost
1	start→	reg	0
2	reg→	Reg	0
3	reg→	Int	1
4	reg→	Fetch | addr	2
5	reg→	Plus / \ reg reg	2
6	addr→	reg	0
7	addr→	Int	0
8	addr→	Plus / \ reg Int	0

Σχήμα 2:Απλή γραμματική δένδρου

Το πρόβλημα της επιλογής εντολής είναι να βρούμε μια αποτελεσματική χαρτογράφηση από το μεταγλωττιστή στόχο (target compiler)-ανεξάρτητη ενδιάμεση αναπαράσταση (IR) ενός προγράμματος σε μία γλώσσα συμβολομεταφραστή που εκφράζει την αρχιτεκτονική-στόχο. Η επιλογή εντολής είναι ιδιαίτερα σημαντική, κατά τη στόχευση συνόλων εντολών που χρησιμοποιούνται από αρχιτεκτονικές με στοιχεία CISC (Complex Instruction Set Computer), όπως η αρχιτεκτονική Intel x86. Σε αυτές τις αρχιτεκτονικές υπάρχουν συνήθως αρκετές πιθανές εφαρμογές της ίδιας πράξης IR, η καθεμία με διαφορετικές ιδιότητες (π.χ., σε αρχιτεκτονική x86, η πρόσθεση της μονάδας σε μία μεταβλητή μπορεί να υλοποιηθεί από inc, add ή lea εντολή). Οι αρχιτεκτονικές CISC είναι δημοφιλείς στο πεδίο των ενσωματωμένων επεξεργαστών λόγω του εμπλουτισμένου, μεταβλητού μήκους συνόλου εντολών που μπορεί να κάνει πιο αποτελεσματική χρήση των περιορισμένων πόρων μνήμης. Το μέγεθος κώδικα, που συχνά αγνοείται σε άλλα περιβάλλοντα εκτέλεσης εφαρμογών (π.χ. desktop), αποτελεί σημαντικό στόχο στη βελτιστοποίηση κατά τη στόχευση σε ενσωματωμένο επεξεργαστή. Μια μικρή διαφορά στο μέγεθος του κώδικα θα μπορούσε να απαιτήσει ένα δαπανηρό επανασχεδιασμό ενός ενσωματωμένου συστήματος. Η επιλογή εντολών είναι ένα σημαντικό μέρος στη βελτιστοποίηση του κώδικα, καθώς το μέγεθος των επιλεγόμενων εντολών επηρεάζει την αποτελεσματική αξιοποίηση της πολυπλοκότητας του συνόλου των εντολών

στόχου. Στην ιδανική περίπτωση, ο επιλογέας εντολής θα είναι σε θέση να βρει τη βέλτιστη απεικόνιση από τον κώδικα IR για τον κώδικα assembly. Στην πιο γενική περίπτωση, η επιλογή εντολών είναι undecidable από ένα βέλτιστο επιλογέα εντολών που θα μπορούσε να λύσει το πρόβλημα της διακοπής προγράμματος. Εξαιτίας αυτού, η επιλογή εντολών είναι συνήθως ορισμένη ως η εύρεση μιας βέλτιστης πλακόστρωσης (tiling) του ενδιάμεσου κώδικα με ένα σύνολο προκαθορισμένων εντολών μηχανής/πλακιδίων. Κάθε πλακίδιο είναι η αντιστοίχιση από κώδικα IR σε κώδικα assembly και έχει ένα σχετικό κόστος. Μια βέλτιστη εντολή πλακόστρωσης ελαχιστοποιεί το συνολικό κόστος τ της πλακόστρωσης. Αν η IR είναι μια ακολουθία έκφρασης των δέντρων, τότε η βέλτιστη αποτελεσματική πλακόστρωση των αλγορίθμων υπάρχει [Koes01]. Ωστόσο, αν χρησιμοποιείται μια πιο εκφραστική αναπαράσταση κατευθυνόμενου μη περιοδικού γράφου (DAG) [Koes 02] χρησιμοποιείται το πρόβλημα γίνεται NP-πλήρες [Koes03, Koes04, Koes16]. Στην εργασία περιγράφεται ο NOLTIS, ένας σχεδόν βέλτιστος (near optimal), γραμμικού χρόνου (linear time), αλγόριθμο επιλογής εντολών (instruction selection algorithm), ο οποίος εργάζεται σε DAG. Ο NOLTIS επεκτείνει υπάρχουσες τεχνικές επιλογής εντολών. Εμπειρικά, είναι σχεδόν βέλτιστος (ένα βέλτιστο αποτέλεσμα βρίσκεται πάνω από το 99% του χρόνου και οι μη-βέλτιστες λύσεις είναι πολύ κοντά στο βέλτιστο). Οι συγγραφείς της εργασίας [NOLTIS] έχουν αποδείξει ότι ο NOLTIS μειώνει σημαντικά το μέγεθος του κώδικα σε σχέση με υφιστάμενες ευρετικές. Η κύρια συνεισφορά της εν λόγω εργασίας είναι ο σχεδόν βέλτιστος, γραμμικού χρόνου αλγόριθμος πλακόστρωσης DAG, με το όνομα NOLTIS. Επιπλέον,

• αποδεικνύεται ότι το πρόβλημα της DAG πλακόστρωσης είναι NP-πλήρες, χωρίς αυτό το πόρισμα να βασίζεται σε περιορισμούς, όπως οι δυο διευθύνσεις εντολών, οι περιορισμοί των καταχωρητών, ή η αντιστοίχηση για το πλακίδιο

• περιγράφει τη βέλτιστη διαμόρφωση 0-1 ακέραιου προγραμματισμού του προβλήματος πλακόστρωσης του DAG,

• και παρέχει μια εκτεταμένη αξιολόγηση του αλγορίθμου, καθώς και την αξιολόγηση άλλων ευρετικών DAG πλακοστρώσεων που έχουν προταθεί στο παρελθόν, συμπεριλαμβανομένων των ευρετικών (heuristics) που αποδομούν έναν DAG σε συλλογή δένδρων, και στη συνέχεια πραγματοποιούν βέλτιστη πλακόστρωση στα δένδρα της συλλογής για το αντίστοιχο βασικό μπλοκ.

ΚΕΦΑΛΑΙΟ 2 : ΑΝΑΛΥΣΗ ΠΡΟΤΕΡΗΣ ΒΙΒΛΙΟΓΡΑΦΙΑΣ

Λόγω των περιορισμών της συντακτικής ανάλυσης του δέντρου, υπάρχουν πολλά άρθρα που έχουν δημοσιευτεί στο παρελθόν ,που ασχολούνται με τη γενίκευση των δέντρων με βάση την επιλογή κώδικα. Στο [Scharwaechter6] βέλτιστο γράφημα με βάση την επιλογή κώδικα περιγράφεται η αρχιτεκτονική για την τακτική διαδρομή των δεδομένων, χωρίς παραλληλισμό. Ωστόσο, ως επί το πλείστον διαθέτουν ASIP, συνδυάζουν ακανόνιστο Σύνολο Εντολών Αρχιτεκτονικής (ISA) που αποτελείται από παράλληλες εντολές. Ειδικά για τον τομέα της επεξεργασίας σήματος, που είναι πιο φυσικό να έχουν MOI, οι περισσότερες προσεγγίσεις έχουν αναπτυχθεί για παράτυπες αρχιτεκτονικές.

Παρουσιάζει μια λύση αποτελεσματική για να σπάσει το DFG σε δέντρα έκφρασης, λαμβάνοντας υπόψη παράλληλα ακανόνιστη αρχιτεκτονική σε καταχωρητές [Scharwaechter7]. Από ότι φαίνεται στο [Scharwaechter8] το δέντρο ανάλυσης οδηγεί σε αναντίστοιχο αποτέλεσμα με την παρουσία του MOI. Για να ξεπεραστεί αυτό, μια νέα τεχνική που βασίζεται στην διαίρεση των εντολών στους καταχωρητές είναι τα αρχέτυπα και η μεταφορά αυτών σε συνδυασμό με τον βέλτιστο τρόπο, προτείνεται η χρήση ακέραιου γραμμικού προγραμματισμού. Ο Liao λύνει το ίδιο πρόβλημα με μια μαθηματική διατύπωση αυξάνοντας δυαδικά [Scharwaechter9]. Δυστυχώς, ο Liao δεν παρέχει τα αποτελέσματα για την εφαρμογή του αλγόριθμου του. Επιπλέον, οι δύο προσεγγίσεις [Scharwaechter10, Scharwaechter11] με χαρακτηριστικό την εκθετική εκτέλεση στη χειρότερη περίπτωση με βάση το μέγεθος του εξεταζόμενου DFG. Αυτό μπορεί να είναι επιζήμιο για τα μεγάλα σημεία αναφοράς. Ο Liem [Scharwaechter12] πρότεινε ένα γράφημα με βάση τη μεθοδολογία επιλογής εντολής που διαθέτει ένα ευέλικτο τρόπο στην αναπαράσταση προτύπου. Αυτό το ύφος περιλαμβάνει στοιχεία ροής, ελέγχου ροής και μικτών δεδομένων / ελέγχου ροής πληροφορίας. Μια μεθοδολογία επιλογής κώδικα περιγράφεται για την πλήρη λειτουργία, προκειμένου να λάβει τον έλεγχο της ροής υπόψη. Μια στατική Ενιαία Εκχώρηση Εκπροσώπηση (SSA) χρησιμοποιείται ως IR και την αντίστοιχη διάρθρωση λύνεται αριθμητικά. Καθώς η επιλογή κώδικα έχει δοκιμαστεί για μια πολύ μεγάλη λέξη (VLIW)-στον επεξεργαστή, οι εντολές παραλληλισμού επίπεδου στο πλαίσιο του MOI δεν είναι ένα ζήτημα για αυτή την προσέγγιση. Η πιο πρόσφατη προσέγγιση σε γράφημα με βάση την επιλογή κώδικα παρουσιάζεται . Στην εργασία απευθύνονται οι αλγόριθμοι επιλογής κώδικα για επιταχυντές υλικού με βάση unate κάλυψη. Σε αντίθεση, η Πολλαπλή Έξοδος Εντολών μπορεί επίσης να περιλαμβάνει πολλαπλά ανεξάρτητα πρότυπα στο IR-επίπεδο. Μπορούμε να συμπεράνουμε ,ότι οι υπάρχουσες προσεγγίσεις, είτε έχουν χαρακτηριστικό χρόνο

τρεξίματος εκθετικό με το μέγεθος του εξεταζόμενου DFG ή δεν θεωρούν εύκολο παραλληλισμό τις εντολές και παράγουν πολλαπλά αποτελέσματα. Η συμβολή μας σε αυτήν την εργασία είναι η παρουσίαση ενός νέου εργαλείου που ονομάζεται Cburg. Ο Cburg είναι πολύ παρόμοιο με τη γνωστή επιλογή κώδικα , όπως γεννήτορες ή Iburg ή ο Olive . Η διαφορά μεταξύ Cburg και των άλλων εργαλείων είναι, καταρχάς, το είδος των αποδεκτών στη γραμματική εισόδου και, δεύτερον, ο αλγόριθμος που παράγεται για να ταιριάζει με τα IR-πρότυπα. Σε αντίθεση με τα άλλα εργαλεία, η γραμματική Cburg δεν περιορίζεται μόνο στα μοντέλα των δέντρων και ως εκ τούτου είναι σε θέση να διαμορφώσει εγγενώς αυθαίρετα παράλληλα τα πρότυπα εντολής. Επιπλέον, ο Cburg παράγει C κώδικα για ένα γράφημα με βάση τις εντολές στην επιλογή του αλγόριθμου και την γραμματική που λαμβάνει έτσι ώστε η αντιστοίχιση με το πεδίο δεν περιορίζεται μόνο σε δηλώσεις μεμονωμένων δέντρων. Ο μέσος χρόνος εκτέλεσης του αλγορίθμου είναι γραμμικός w.r.t του μεγέθους της εισόδου DFG. Αυτή η προσέγγιση οδηγεί εύκολα σε γράφημα με βάση την επιλογή κώδικα που επιτρέπει στους σχεδιαστές του μεταγλωττιστή να παρέχουν γρήγορα νέο μεταγλωττιστή για παράτυπα συνόλων εντολών.

Η προσέγγιση των Davidson-Fraser για την επιλογή κώδικα (που χρησιμοποιείται, π.χ., στον GCC) [Ertl01, Ertl03] μπορεί να ασχοληθεί με DAG, αλλά δεν εγγυάται τη βέλτιστη (όχι ακόμη και για τα δέντρα). Ωστόσο, είναι πιο ευέλικτο στο είδος των προτύπων επιλογής κώδικα που επιτρέπονται (και όχι μόνο σχέδια δέντρο), και ως εκ τούτου μπορούν να παράγουν καλύτερα από ό, τι κώδικα ανάλυσης δέντρο. Μειονεκτήματα του είναι ότι είναι πιο δύσκολο να χρησιμοποιήσει (π.χ., προσθέτοντας ένα νέο κανόνα μπορεί να οδηγήσει σε χειρότερο κώδικα) και ότι οι απορρέοντες επιλογείς κώδικα είναι πιο αργοί. Η lcc front-end [Ertl05, Ertl06] μπορούν να παράγουν ενδιάμεση αναπαράσταση DAG ή δένδρα. Οι επιλογείς κώδικα [Ertl06] βασίζονται σε δένδρο ανάλυσης · ασχολούνται με το πρόβλημα, ζητώντας από το εμπρόσθιο άκρο του lcc να διαχωρίσει τα DAG σε δένδρα. Στο [Ertl04] μια προσέγγιση για την αντιμετώπιση DAG στο πλαίσιο της ανάλυσης δέντρου συζητείται ότι έχει τα ίδια αποτελέσματα όπως η διάσπαση DAG σε γενικές γραμμές, αλλά επιτρέπει την αναπαραγωγή του μηδενικού κόστους υποδένδρων (π.χ., σταθερές ή εκφράσεις αποτελεσματική διεύθυνση). [Ertl08] παρουσιάζει μια μέθοδο για τη διεξαγωγή μονού περάσματος ανάλυσης δέντρου επιλογής κώδικα (σε αντίθεση με την κλασική μέθοδο, η οποία απαιτεί ένα πέρασμα στον επιγράφων και ένα πέρασμα στον μειωτή). Όπως και ο τρόπος μας βέλτιστη ανάλυσης του DAG, αυτή η μέθοδος λειτουργεί μόνο με συγκεκριμένες γραμματικές, αλλά μπορεί να χρησιμοποιηθεί με ρεαλιστική επιλογή γραμματικής κώδικα.

Η κλασική προσέγγιση για την επιλογή εντολής υπήρξε για να εκτελέσει την πλακόστρωση στις εκφράσεις δένδρων. Αυτό έγινε αρχικά με την χρήση δυναμικού προγραμματισμού [Koes03,Koes05] για μια ποικιλία από μοντέλα μηχανών, συμπεριλαμβανομένων των συσκευών στοίβας, μηχανές πολλαπλών καταχωρητών, μηχανές με άπειρους καταχωρητές, και οι πολύ βαθμωτές μηχανές [Koes06]. Αυτές οι τεχνικές έχουν αναπτυχθεί περαιτέρω για να παραγάγουν τον γεννήτορα κώδικα[Koes07, Koes08], ο οποίος λαμβάνει μια δηλωτική προδιαγραφή μιας αρχιτεκτονικής και, κατά το χρόνο μεταγλώττισης, δημιουργεί μια επιλογή εντολής. Αυτός ο γεννήτορας κώδικας είτε εκτελείται κατά τον δυναμικό προγραμματισμό στο χρόνο μεταγλώττισης [Koes09, Koes10, Koes11], ή χρησιμοποιώντας BURS τη θεωρία του δένδρου ανάλυσης (κάτω προς τα πάνω σύστημα επανεγγραφής)[Koes12,Koes13] για να μετατοπιστεί ο χρόνος μεταγλώττισης του δυναμικού προγραμματισμού στον μεταγλωττιστή [Koes14,Koes15]. Σε αυτή την εργασία περιγράφουμε τον αλγόριθμο NOLTIS, που χρησιμοποιεί τη βέλτιστη αντιστοίχηση δέντρου για να βρει μια σχεδόν βέλτιστη πλακόστρωση της έκφρασης DAG. Αν χρησιμοποιούμε μια απλή αντιστοίχησης ο χρόνο μεταγλώττισης του δυναμικού προγραμματισμού, ο αλγόριθμος NOLTIS θα μπορούσε επίσης εύκολα να χρησιμοποιήσει μια προσέγγιση του BURS για να ταιριάζουν. Η έκφραση πλακόστρωσης DAG είναι σημαντικά πιο δύσκολη από ό, τι η έκφραση πλακόστρωσης δένδρων. Η πλακόστρωση DAG έχει αποδειχθεί ότι είναι NP-πλήρης για μηχανή ενός καταχωρήτη [Koes 16] και για δύο-διευθύνσεις, για μοντέλο μηχανής άπειρων καταχωρητών [Koes17]. Οι μηχανές δύο-διευθύνσεων έχουν οδηγίες του εντύπου $r_i \leftarrow r_i - op\ r_j$ και $r_i \leftarrow r_j$. Δεδομένου ότι ένας από τους τελεστές πηγής παίρνει αντικατάσταση, η δυσκολία έγκειται στην ελαχιστοποίηση του αριθμού των κινήσεων που εισάγονται για να προλαμβάνουν τις τιμές από την αντικατάσταση. Ακόμη και με άπειρους καταχωρητές και απλό, ενιαίο κόμβο με πλακίδια, το πρόβλημα της ελαχιστοποίησης της κίνηση είναι NP-πλήρες αν και υπάρχουν αλγόριθμοι για την προσέγγιση [Koes03]. Η πλακόστρωση DAG εξακολουθεί να είναι δύσκολη σε τρεις-διευθύνσεις, άπειρους καταχωρητές μηχανής εάν οι κόμβοι στην πλακόστρωση στο εξωτερικό έχουν ετικέτες που πρέπει να ταιριάζουν [Koes16]. Τα σήματα αυτά μπορούν να αντιστοιχούν στην αξία των θέσεων αποθήκευσης (π.χ. τάξεις καταχωρητών ή μνήμης) ή με τα είδη της τιμής. Τέτοια σήματα είναι περιττά, εάν η επιλογή εντολής διαχωρίζεται από τον καταμερισμό των καταχωρητών και αν η IR έχει ήδη καθοριστεί πλήρως με τους τύπους τιμών των ακμών στην έκφραση DAG. Ωστόσο, έχουμε δείξει ότι το πρόβλημα παραμένει NP-πλήρες, ακόμη και χωρίς σήματα. Αν και η πλακόστρωση DAG είναι NP-πλήρες γενικά, για κάποια σύνολα πλακιδίων μπορεί να λυθεί

σε πολυωνυμικό χρόνο [Koes17]. Εάν ένα δέντρο πλακόστρωσης αλγορίθμων προσαρμόζεται σε ένα πλακίδιο DAG και το βέλτιστο σύνολο DAG πλακιδίων χρησιμοποιείται για την εκτέλεση της πλακόστρωσης, το αποτέλεσμα είναι η βέλτιστη πλακόστρωση του DAG . Αν και τα σύνολα των πλακιδίων για αρκετές αρχιτεκτονικές βρέθηκαν να είναι βέλτιστα σε DAG [Koes 18], αυτά τα σύνολα πλακιδίων χρησιμοποιούν ένα απλό μοντέλο κόστους και το βέλτιστο DAG συνόλου των πλακιδίων δεν σώζεται αν ένα πιο σύνθετο μοντέλο του κόστους, όπως το μέγεθος του κώδικα χρησιμοποιείται. Για παράδειγμα, αν τα πλακίδια στο σχήμα 1, όλα είχαν το μοναδιαίο κόστος, θα ήταν βέλτιστος ο DAG , αλλά με το μετρικό κόστος που φαίνεται στο Σχήμα 1 δεν είναι. Παραδοσιακά, η πλακόστρωση DAG γίνεται με τη χρήση μιας εμπειρικής έκφρασης για να σπάσει τον DAG σε ένα δάσος από δέντρα έκφρασης [Koes 18]. Περισσότερες λύσεις με βάρος, οι οποίες λύνουν το πρόβλημα ιδανικά, περιλαμβάνουν τη χρήση δυαδικά καλυπτόμενο [Koes19, Koes20], χρησιμοποιώντας περιορισμούς στον λογικού προγραμματισμό [Koes21], χρησιμοποιώντας ακέραιο γραμμικό προγραμματισμό [Koes22] ή την εκτέλεση εξαντλητικής έρευνας [Koes23]. Επιπλέον, περιγράφουμε ένα 0-1 ακέραιο προγραμματισμό αναπαράστασης του προβλήματος . Οι τεχνικές αυτές παρουσιάζουν χειρότερη περίπτωση στην εκθετική συμπεριφορά. Παρά το γεγονός ότι οι τεχνικές αυτές μπορούν να είναι επιθυμητές όταν η ποιότητα του κώδικα είναι υψίστης σημασίας και θα συγκεντρώνει τον χρόνος εκτέλεσης είναι επουσιώδη, πιστεύουμε ότι στο γραμμικό χρόνο, σχεδόν βέλτιστου αλγόριθμου προσφέρει εξαιρετική ποιότητα κώδικα χωρίς να θυσιάζεται ο χρόνος μεταγλώττισης σε κλιμακωτή απόδοση. Μια εναλλακτική, μη-πλακόστρωση, στη μέθοδο της επιλογής εντολής, το οποίο είναι πιο κατάλληλο για γραμμική, σε αντίθεση με το δέντρο-όπως, IRS, είναι να ενσωματώσει την επιλογή εντολής σε βελτιστοποίηση peephole [Koes24,Koes25, Koes26, Koes27, Koes28]. Στην βελτιστοποίηση peephole [Koes 29], οι μετασχηματισμοί αντιστοιχούν σε μοτίβο που εκτελείται πάνω από ένα μικρό παράθυρο των οδηγιών, το " peephole ". Αυτό το παράθυρο μπορεί να είναι είτε ένα φυσικό παράθυρο, όπου οι εντολές θεωρούνται ότι είναι μόνο όσων έχουν προγραμματιστεί το ένα δίπλα στο άλλο στην τρέχουσα λίστα εντολών, ή ένα λογικό παράθυρο, όπου οι εντολές θεωρούνται ότι είναι μόνο αυτά που είναι δεδομένα ή ελέγχου που σχετίζονται με την εντολή σάρωσης. Κατά την εκτέλεση peephole η επιλογή με βάση τις εντολές, ο peepholer απλά μετατρέπει ένα παράθυρο της IR σε εντολές που απευθύνονται σε ειδικό. Εάν ένα λογικό παράθυρο χρησιμοποιείται, τότε αυτή η τεχνική μπορεί να θεωρηθεί ως ευρετική μέθοδος για την πλακόστρωση σε έναν DAG. Οι αλγόριθμοι επιλογής εντολών έχουν προσαρμοστεί με

επιτυχία για να λύσουν το πρόβλημα χαρτογράφησης της τεχνολογίας στον τομέα της αυτοματοποιημένης σχεδίασης κυκλωμάτων [Koes30]. Πολλές επεκτάσεις σε συγκεκριμένους τομείς στο βασικό αλγόριθμο πλακόστρωσης έχουν προταθεί (βλ. [Koes31, Koes32] για τις αναφορές), αλλά, στην καλύτερη των γνώσεών μας, όλοι οι αλγόριθμοι πλακόστρωσης DAG που προτείνονται σε αυτό τον τομέα έχουν καταφύγει σε απλούς, σε συγκεκριμένους τομείς, για την αποσύνθεση του DAG, σε δέντρα πριν από την εκτέλεση της πλακόστρωσης.

ΚΕΦΑΛΑΙΟ 3 : ΒΕΛΤΙΣΤΗ ΕΠΙΛΟΓΗ ΚΩΔΙΚΑ ΣΕ DAG

3.1. ΠΕΡΙΛΗΨΗ

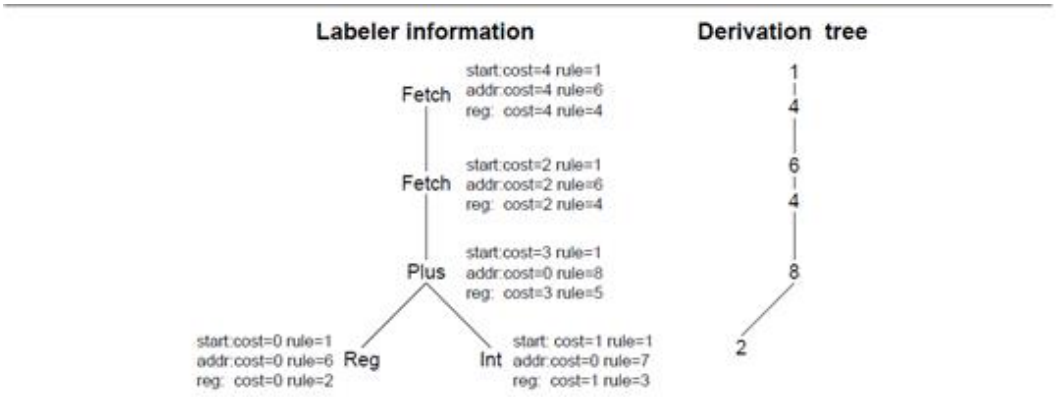
Στην εργασία [Ert99] γίνεται επέκταση της βασικής τεχνικής για την ανάλυση δένδρου προκειμένου την επιλογή κώδικα DAG. Γενικά, η προτεινόμενη επέκταση δεν παράγει τη βέλτιστη επιλογή για όλους τους κώδικες DAG (το πρόβλημα αυτό είναι NP-πλήρες), αλλά το επιτυγχάνει για ορισμένες γραμματικές επιλογής κώδικα. Στο κεφάλαιο αυτό παρουσιάζεται μία μέθοδος για τον έλεγχο του κατά πόσον μια γραμματική επιλογής κώδικα ανήκει σε αυτό το σύνολο των DAG-βέλτιστων γραμματικών, ώστε να μπορεί να χρησιμοποιηθεί για τον έλεγχο των αντίστοιχων γραμματικών που χρησιμοποιούνται στο μεταγλωττιστή LCC [LCC]. Οι γραμματικές για τις MIPS και SPARC αρχιτεκτονικές είναι DAG-βέλτιστες και η γραμματική επιλογής κώδικα είναι επίσης DAG-βέλτιστη.

	nonterminal→	pattern	cost
1	start→	reg	0
2	reg→	Reg	0
3	reg→	Int	1
4	reg→	Fetch | addr	2
5	reg→	Plus reg reg	2
6	addr→	reg	0
7	addr→	Int	0
8	addr→	Plus reg Int	0

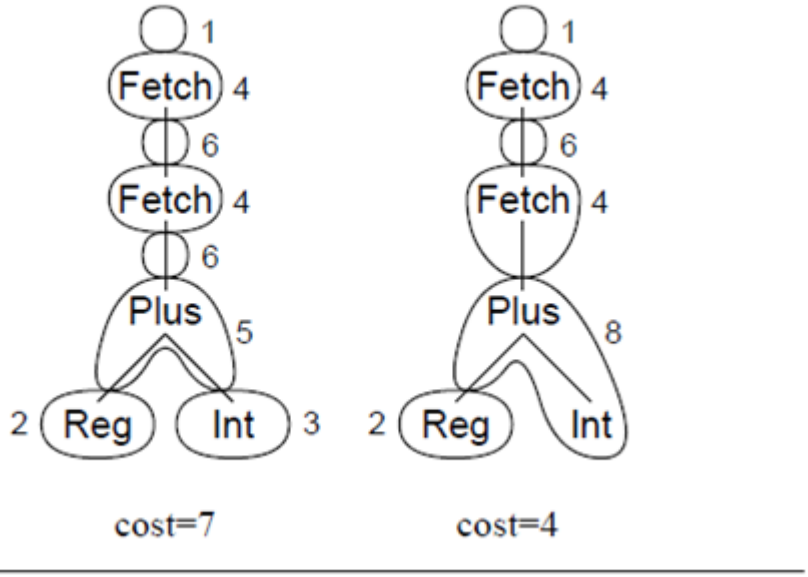
Σχήμα 3:Μια απλή γραμματική δένδρου

3.2 ΕΠΙΛΟΓΗ ΚΩΔΙΚΑ ΑΠΟ ΤΗΝ ΑΝΑΛΥΣΗ ΔΕΝΔΡΟΥ

Η περιγραφή μηχανής για την επιλογή κώδικα από την ανάλυση δένδρου αποτελεί μια γραμματική δένδρου. Στο Σχήμα 3 φαίνεται μια απλή γραμματική δένδρου [Pro95]. Κάθε κανόνας αποτελείται από την παραγωγή (με τη μορφή μη-τερματικό σύμβολο→πρότυπο), το κόστος και μερικές ενέργειες γέννησης κώδικα (Σχήμα 4). Ένα βήμα παραγωγής γίνεται με την αντικατάσταση ενός μη τερματικού σύμβολου που βρίσκεται στην αριστερή πλευρά του κανόνα με τα πρότυπα στη δεξιά πλευρά του κανόνα. Για μια πλήρη παραγωγή, θα ξεκινήσουμε με ένα μη-τερματικό σύμβολο στην αρχή, ώστε να εκτελεστούν τα βήματα παραγωγής μέχρις ότου που τα μη-τερματικά σύμβολα να είναι αριστερά. Στο Σχήμα 5 παρουσιάζονται δύο τρόποι για να παραχθεί ένα δένδρο [Pro95]. Το κόστος της παραγωγής είναι το άθροισμα από τα κόστη των κανόνων που εφαρμόζονται.



Σχήμα 4: Η πληροφορία που υπολογίζεται από τον επιγράφοντα (labeler) και το προκύπτον δένδρο παραγωγής.



Σχήμα 5: Δύο παραγωγές του ίδιου δένδρου (οι κύκλοι και οι αριθμοί απεικονίζουν τους χρησιμοποιούμενους κανόνες).

Για την επιλογή κώδικα, οι τελεστές που χρησιμοποιούνται στην γραμματική δένδρου είναι οι τελεστές της ενδιάμεσης αναπαράστασης, καθώς και τα κόστη των κανόνων αντανακλούν τα κόστη του παραγόμενου κώδικα που δημιουργείται από τους κανόνες. Το κόστος μιας ολόκληρης παραγωγής αντιπροσωπεύει το κόστος του κώδικα που προκύπτει από το παραγόμενο δένδρο. Όπως το παράδειγμα δείχνει, γραμματικές επιλογής κώδικα είναι συνήθως διφορούμενες. Το πρόβλημα στην ανάλυση δένδρου για την επιλογή κώδικα είναι να βρεθεί μία παραγωγή ελάχιστου κόστους για ένα δοσμένο δένδρο. Μια σχετικά απλή μέθοδος με γραμμική πολυπλοκότητα είναι η προσέγγιση δυναμικού προγραμματισμού που χρησιμοποιείται από τους BEG [ESL89] και Iburg [FHP93]. Λειτουργεί σε δύο φάσεις:

Labeler (Επιγράφων): Το πρώτο πέρασμα λειτουργεί από κάτω προς τα επάνω. Για κάθε συνδυασμό κόμβου/μη-τερματικού συμβόλου, καθορίζει το ελάχιστο κόστος για την παραγωγή του υποδένδρου το οποίο έχει ρίζα τον κόμβο του μη-τερματικού συμβόλου και τον κανόνα που χρησιμοποιείται στο πρώτο βήμα αυτής της παραγωγής. Επειδή το ελάχιστο κόστος για όλους τους κατώτερους συνδυασμούς κόμβων/μη-τερματικών σύμβολων είναι ήδη γνωστό, αυτό μπορεί να εκτελεστεί εύκολα με τον έλεγχο όλων των εφαρμόσιμων κανόνων, και τον υπολογισμό αυτού με το μικρότερο κόστος. Οι κανόνες της μορφής μη-τερματικό σύμβολο→ μη-τερματικό σύμβολο (κανόνες αλυσίδας) πρέπει να ελέγχονται επαναληπτικά μέχρις ότου να μην υπάρχουν αλλαγές. Εάν υπάρχουν αρκετοί βέλτιστοι κανόνες, οποιοσδήποτε από αυτούς μπορούν να χρησιμοποιηθούν.

Reducer (Μειωτής): Αυτό το πέρασμα εκτελεί ένα βηματισμό του δένδρου παραγωγής. Ξεκινά από το αρχικό μη-τερματικό σύμβολο στον κόμβο ρίζα. Εξετάζει τον εγγεγραμμένο κανόνα για αυτό το συνδυασμό κόμβου/μη-τερματικού συμβόλου. Τα μη-τερματικά σύμβολα στο πρότυπο του κανόνα αυτού προσδιορίζουν τους κόμβους και τα μη-τερματικά σύμβολα, όπου οι βηματισμοί συνεχίζονται. Σε κάποια χρονική στιγμή κατά την επεξεργασία του κανόνα (συνήθως αφού έχουν υποστεί επεξεργασία τα υποδένδρα), εκτελείται η γέννηση κώδικα.

Το Σχήμα 4 δείχνει την πληροφορία που παράγεται με αυτή τη μέθοδο. Η προκύπτουσα βέλτιστη παραγωγή είναι η ίδια με εκείνη που εμφανίζεται στη δεξιά πλευρά του Σχήματος 5.

3.3 DAGs

Τα DAGs ενδιάμεσης αναπαράστασης προκύπτουν από γλωσσικές δομές όπως ο τελεστής « + = »της γλώσσας προγραμματισμού C, από την εξουδετέρωση κοινής υποεκφράσεως, ή από την πραγματοποίηση επιλογής κώδικα σε βασικά μπλοκ. Στο Σχήμα 5 δίνεται ένα παράδειγμα για αυτές τις περιπτώσεις.

Υποθέτουμε ότι είναι αποδεκτό να γεννηθεί κώδικας για κοινούς γράφους μόνο μία φορά και ότι είναι επίσης αποδεκτό να γεννηθεί κώδικας που αντιστοιχεί σε μερική ή ολική αντιγραφή των κοινών υπογράφων. Αυτή η υπόθεση εξαρτάται από τη σημασιολογία της ενδιάμεσης αναπαράστασης, δηλαδή οι ενδιάμεσες λειτουργίες πρέπει να είναι συναρτήσεις (όταν προγραμματίζεται σωστά) ή ταυτοδύναμες. Αυτή η υπόθεση είναι συνήθως έγκυρη π.χ. ισχύει για την ενδιάμεση αναπαράσταση του LCC [FH91, FH95]. Ας σημειωθεί ότι το πρόβλημα της βέλτιστης συντακτικής ανάλυσης γίνεται ευκολότερο, αν η υπόθεση αυτή δεν είναι έγκυρη, διότι τότε δεν υπάρχει δυνατότητα επιλογής μεταξύ αντιγραφής και μη-αντιγραφής. Η εξουδετέρωση κοινής υποεκφράσεως στηρίζεται επίσης στην υπόθεση αυτή.

3.4. ΑΝΑΛΥΣΗ DAG

Εδώ αναλύεται η γενική προσέγγιση για την άμεση επέκταση του αλγορίθμου συντακτικής ανάλυσης δένδρων στα DAG.

Labeler (Επιγράφων): Η ίδια πληροφορία (όπως στον επιγράφοντα δένδρου) υπολογίζεται για κάθε κόμβο. Μόνο η συνιστώσα για την επίσκεψη των κόμβων από κάτω προς τα πάνω μπορεί να χρειάζεται προσαρμογή (π.χ., ένα αναδρομικό πέρασμα θα πρέπει να επεκταθεί με μια σημαία επίσκεψης).

Reducer (Μειωτής): Ο μειωτής έχει τώρα να αντιμετωπίσει την πιθανή ύπαρξη πολλών κόμβων-ρίζας. Επιπλέον, πρέπει να θέσει και να ελέγξει μια σημαία επίσκεψης για κάθε συνδυασμό κόμβου/μη-τερματικού σύμβολου που αυτό επισκέπτεται, εξασφαλίζοντας έτσι ότι ο μειωτής βηματίζει κάθε διαδρομή του γράφου παραγωγής ακριβώς μια φορά .

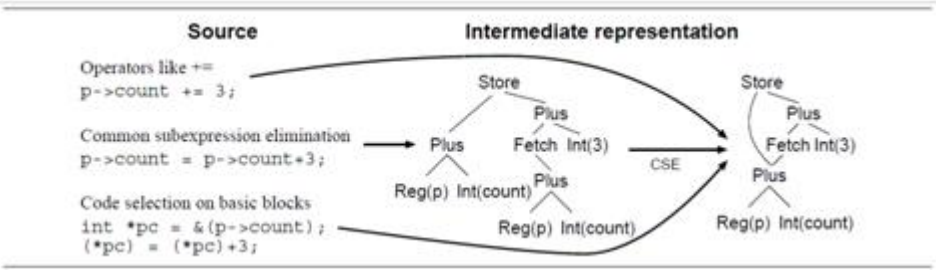
Το Σχήμα 8 δείχνει πώς αυτή η μέθοδος αναλύει ένα γράφο με τη γραμματική του Σχήματος 3.

Αυτός ο αλγόριθμος ανάλυσης DAG δεν λαμβάνει υπόψη το διαμοιρασμό κόμβου κατά το στάδιο της επισήμανσης. Στο στάδιο της μείωσης, ο διαμοιρασμός εξετάζεται μόνο για συνδυασμούς κόμβου/μη-τερματικού σύμβολου. Δηλαδή, αν ένας υπό-γράφος παράγεται πολλές φορές από το ίδιο μη-τερματικό, η μείωση του υπογράφου διαμοιράζεται. Αντίθετα, αν ένας κοινός υπό-γράφος παράγεται από διαφορετικά μη-τερματικά μέσω βασικών κανόνων, ο κόμβος-ρίζα του θα μειώνεται πολλές φορές (δηλ. η λειτουργία που αναπαρίσταται από τον κόμβο θα αντιγράφεται στον παραγόμενο κώδικα). Ένας υπό-γράφος παιδί του κόμβου αυτού επίσης θα παράγεται πολλαπλές φορές, με το διαμοιρασμό των μειώσεων να εξαρτάται και πάλι από το εάν οι παραγωγές προέρχονται από το ίδιο μη-τερματικό.

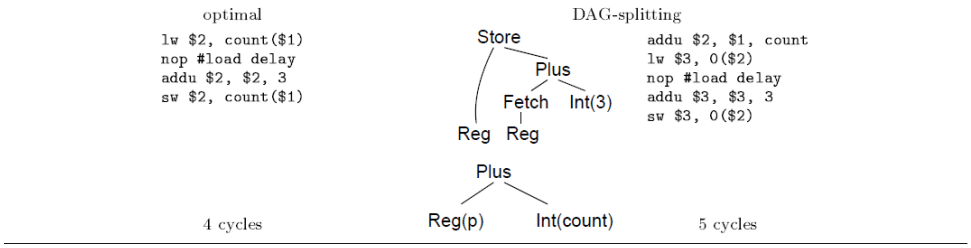
Π.χ., στο Σχήμα. 8 ο αριστερός κόμβος Plus μειώνεται δύο φορές, επειδή παράγεται από το addr μέσω κανόνα 8 (κόμβος γονέας: Fetch), και από το reg μέσω κανόνα 5 (κόμβος γονέας: δεξιός Plus). Η μείωση του αριστερού παιδιού του (αριστερό reg) διαμοιράζεται, επειδή παράγεται από το reg και στις δύο παραγωγές. Το κόστος για την παραγωγή ενός DAG είναι το άθροισμα του κόστους των εφαρμοζόμενων παραγωγών (οι κοινές παράγωγες προσμετρούνται μόνο μία φορά). Το πρόβλημα της ανάλυσης DAG για την επιλογή κώδικα είναι η εύρεση της παραγωγής ελάχιστου κόστους για το DAG. Γενικά, το πρόβλημα είναι NP-πλήρες [Pro98].

Η μέθοδος για την ανάλυση DAG είναι γραμμική στο χρόνο και το χώρο. Ο επιγράφων επισκέπτεται κάθε κόμβο μία φορά και ο χρόνος που δαπανά σε κάθε κόμβο είναι σταθερός (ανεξάρτητα από το μέγεθος του γράφου). Παρομοίως, ο μειωτής επισκέπτεται κάθε συνδυασμό κόμβου/μη-τερματικού σύμβολου το πολύ μία φορά και δαπανά σταθερό χρόνο σε κάθε συνδυασμό. Η αντιγραφή των λειτουργιών περιορίζεται από τον αριθμό των μη-τερματικών της γραμματικής, δηλ. είναι ανεξάρτητη από το μέγεθος της γραμματικής. Η προτεινόμενη μέθοδος είναι γραμμική, αλλά γενικά δεν εγγυάται βέλτιστες παραγωγές. Ευτυχώς, για μια ορισμένη κατηγορία γραμματικών, η μέθοδος αυτή αναλύει DAG με βέλτιστο τρόπο .

Για γραμματικές που θα χρησιμοποιηθούν στα δένδρα υπάρχει μια πρακτική κατανομής του κόστους μεταξύ των κανόνων που συμβάλλουν σε μια εντολή, π.χ. για την ανάθεση κάποιου κόστους σε έναν τρόπο διευθυνσιοδότησης. Κατά την ανάλυση DAG, η πρακτική αυτή θα οδηγήσει σε ένα μέρος του κόστους να υπολογιστεί μόνο μία φορά, παρόλο που στον παραγόμενο κώδικα, το κόστος εμφανίζεται δύο φορές. Ως εκ τούτου, οι γραμματικές που προορίζονται να χρησιμοποιηθούν με DAG θα πρέπει να αποδίδουν το πλήρες κόστος σε κανόνες που πραγματικά δημιουργούν κώδικα (σε γενικές γραμμές, κανόνες που πραγματικά προκαλούν κόστος), και κανένα κόστος σε άλλους κανόνες.

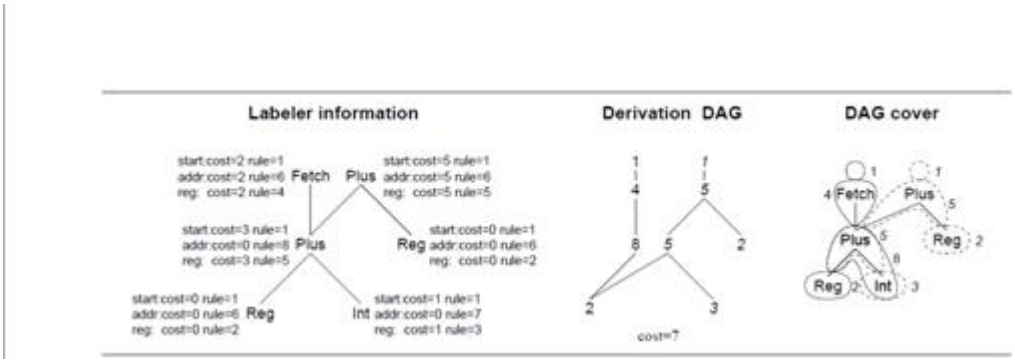


Σχήμα 6: Τρεις τρόποι για την παραγωγή ενδιάμεσης αναπαράστασης σε μορφή DAG



Σχήμα 7: Κώδικας που παράγεται από το DAG του Σχ. 6 για τον επεξεργαστή MIPS R3000 με βέλτιστη ή καταμερισμού-DAG επιλογή κώδικα. Η μεταβλητή p βρίσκεται στον καταχωρητή $1.

Μία προσέγγιση για το χειρισμό των DAG είναι να διαχωριστούν αυτά σε δένδρα (με αφαίρεση των ακμών μεταξύ διαμοιρασμένων κόμβων και των γονέων τους, Σχ. 3.5.) και να αναλυθεί κάθε δένδρο χωριστά. Τότε τα αποτελέσματα για κάθε δένδρο υποχρεώνονται να αποθηκεύονται σε έναν καταχωρητή. Κανένα έργο δεν γίνεται δύο φορές, αλλά ο προκύπτων κώδικας μπορεί να είναι υποβέλτιστος για το συνολικό DAG (Σχήμα 7).



Σχήμα 8: Πραγματοποιώντας τη συντακτική ανάλυση ενός DAG. Στην κάλυψη DAG, οι διακεκομμένες γραμμές αναπαριστούν τους κανόνες που απεικονίζονται από τους κεκλιμμένους αριθμούς.

3.5. ΚΑΘΟΡΙΣΜΟΣ ΒΕΛΤΙΣΤΟΥ

3.5.1. ΓΡΑΜΜΑΤΙΚΕΣ ΚΑΝΟΝΙΚΗΣ ΜΟΡΦΗΣ

Για να απλοποιηθεί η συζήτηση, υποθέτουμε ότι η γραμματική είναι σε κανονική μορφή [BDB90], δηλαδή, περιέχει μόνο κανόνες της μορφής $n \to n_1$ (κανόνες αλυσίδας) ή $n \to Op(n_1,...,n_i)$ (κανόνες βάσης), όπου τα ns είναι μη-τερματικά σύμβολα. Η γραμματική δένδρου μπορεί να μετατραπεί σε κανονική μορφή εύκολα με την εισαγωγή μη-τερματικών σύμβολων. Οι περισσότεροι κανόνες της γραμματικής του Σχ. 3 είναι ήδη σε κανονική μορφή, με την εξαίρεση του κανόνα 8, ο οποίος μπορεί να μετατραπεί σε κανονική μορφή με το διαχωρισμό του σε δύο κανόνες:

	nonterminal→pattern	cost
8a	addr→ Plus (reg, n1)	0
8b	n1→ Int	0

Το πλεονέκτημα της κανονικής μορφής είναι ότι δεν έχουμε να σκεφτούμε σχετικά με τους κανόνες που αναλύουν πολλούς κόμβους ταυτόχρονα και ως εκ τούτου προσπερνούν κόμβους στο μειωτή. Αντί αυτού, είναι γνωστό ότι κάθε παραγωγή του κόμβου πρέπει να περάσει μέσα από ένα μη-τερματικό σύμβολο στον κόμβο.

3.5.2. ΒΑΣΙΚΗ ΙΔΕΑ

Ο αλγόριθμος ανάλυσης DAG (Ενότητα 3.4) κάνει την επιλογή των κανόνων ως να επρόκειτο για ανάλυση δένδρου, ανεξάρτητα από το διαμοιρασμό των υπογράφων. Με άλλα λόγια, στην περίπτωση ενός κοινού κόμβου, τον βελτιστοποιεί τοπικά για κάθε γονέα. Η

δουλειά του ελεγκτή (checker) είναι να εξακριβώσει εάν αυτές οι τοπικά βέλτιστες επιλογές είναι βέλτιστες και σε καθολικό επίπεδο για κάθε DAG που μπορεί να παραχθεί από τη γραμματική.

Η βασική αρχή του ελεγκτή είναι: Δοθέντος ενός υπογράφου G, θεωρούμε ότι μπορεί να παράγεται από οποιοδήποτε αριθμό άλλων γονέων, από κάθε συνδυασμό μη-τερματικών, χρησιμοποιώντας κάθε καθολικά βέλτιστη παραγωγή · οι περιπτώσεις αυτές αντιπροσωπεύουν όλους τους τρόπους για το διαμοιρασμό του G. Στη συνέχεια, ελέγχουμε την ακόλουθη προϋπόθεση για κάθε μη-τερματικό n: Η τοπικά βέλτιστη παραγωγή του G από το n πρέπει να είναι η βέλτιστη για όλες αυτές τις περιπτώσεις. Αν αυτό συμβαίνει τότε αυτή η παραγωγή του G από το n είναι καθολικά βέλτιστη.

Υπάρχει μόνο ένα πρόβλημα: Η μέθοδος αυτή απαιτεί τη γνώση των καθολικά βέλτιστων παραγωγών, οι οποίες γενικά δεν είναι γνωστές. Χρησιμοποιούμε μια συντηρητική προσέγγιση: Υποθέτουμε ότι μια παραγωγή είναι καθολικά βέλτιστη εκτός αν γνωρίζουμε ότι δεν είναι. Αυτό εξασφαλίζει ότι ο ελεγκτής αναφέρει όλες τις γραμματικές για τις οποίες η μέθοδος ανάλυσής μας δεν είναι βέλτιστη για όλους τους DAG, αλλά μπορεί επίσης να αναφέρει ορισμένα ψευδή προβλήματα.

Κατά τον έλεγχο μιας παραγωγής, υποθέτουμε ότι οι συνδυασμοί διαμοιρασμένων κόμβων/μη-τερματικών συμβόλων καταβάλλονται πλήρως από τους άλλους γονείς. Δηλ. η αναζητούμενη παραγωγή σε αυτούς τους συνδυασμούς κόμβων/μη-τερματικών συμβόλων έχει κόστος 0. Αν ένας κανόνας είναι βέλτιστος σε αυτή την περίπτωση, τότε είναι βέλτιστος και για όλες τις κατανομές κόστους ανάμεσα στα δύο άκρα.

3.5.3. Ο ΕΛΕΓΚΤΗΣ

Ο έλεγχος μιας γραμματικής για DAG-βελτιστότητα γίνεται με την κατασκευή μιας επαγωγικής απόδειξης πάνω στο σύνολο όλων των πιθανών DAG: Οι βασικές περιπτώσεις είναι τα τερματικά, οι βηματικές περιπτώσεις κτίζουν μεγαλύτερους DAG από μικρότερους DAG. Ένας τέτοιος ελεγκτή ονομάζεται Dburg .

3.5.3.1. ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

Ένα στοιχείο αποτελεί μια εγγραφή που περιέχει ένα κόστος και ένα σύνολο κανόνων. Ένα στοιχειοσύνολο είναι ένας πίνακας από στοιχεία, με δείκτες από μη-τερματικά. Το κόστος των στοιχείων σε ένα στοιχειοσύνολο μπορεί να είναι απόλυτο ή σχετικό ως προς μία βάση δ (κοινή για όλο το στοιχειοσύνολο).

Μια κατάσταση είναι μια εγγραφή που αποτελείται από ένα στοιχειοσύνολο συνολικού κόστους και ένα σύνολο από στοιχειοσύνολα μερικού κόστους. Αυτές οι δομές δεδομένων έχουν την ακόλουθη σημασία: Μια κατάσταση αντιπροσωπεύει μια κατηγορία από γράφους που έχουν συγκεκριμένα κοινά σημεία ως προς τους κανόνες και τα κόστη για την παραγωγή τους. Κάθε κατάσταση αντιστοιχεί σε μια βασική περίπτωση ή σε μία βηματική περίπτωση της απόδειξης.

Το στοιχειοσύνολο συνολικού κόστους είναι η πληροφορία που υπολογίζεται από το πέρασμα επισήμανσης του αλγορίθμου ανάλυσης για τον κόμβο-ρίζα του γράφου που αναπαριστάται, με δύο αλλαγές: α) τα σχετικά κόστη επιτρέπουν την αναπαράσταση γράφων με διαφορετικά απόλυτα κόστη και β) τα στοιχεία αποθηκεύουν το σύνολο των βέλτιστων κανόνων αντί για έναν από αυτούς τους κανόνες.

Το στοιχειοσύνολο μερικού κόστους αντιπροσωπεύει τα πρόσθετα κόστη (και κανόνες που χρησιμοποιούνται) για την παραγωγή των γράφων από ένα συγκεκριμένο πρότυπο διαμοιρασμού. Το αυξητικό κόστος για την παραγωγή ενός μερικώς διαμοιρασμένου γράφου από ένα μη-τερματικό σύμβολο είναι το κόστος που προκύπτει από τις παραγωγές μη-διαμοιρασμένων κανόνων. Αυτό υπολογίζεται με τον ορισμό του κόστους όλων των συνδυασμών διαμοιρασμένων κόμβων/μη- τερματικών στο 0 και με την πραγματοποίηση ενός περάσματος επισήμανσης. Τα στοιχειοσύνολα μερικού κόστους περιλαμβάνουν επίσης και ένα στοιχειοσύνολο για την περίπτωση του μη-διαμοιρασμού (δηλαδή, ένα που αντιστοιχεί στο στοιχειοσύνολο συνολικού κόστους).

3.5.3.2. ΥΠΟΛΟΓΙΣΜΟΣ ΚΑΤΑΣΤΑΣΕΩΝ

Ο Dburg υπολογίζει τις καταστάσεις με έναν αλγόριθμο λίστας έργου: ο Dburg προσπαθεί να κατασκευάσει μια νέα κατάσταση με την εφαρμογή κάθε n-πλου τελεστή σε κάθε n-άδα καταστάσεων, έως ότου να μην προκύπτουν νέες καταστάσεις. Στην αρχή, δεν υπάρχουν καταστάσεις, παρά μόνον τελεστές χωρίς τελεστές (δηλαδή, τερματικά) να μπορούν να εφαρμοστούν.

Ο Dburg εφαρμόζει έναν τελεστή o σε διατεταγμένη πλειάδα από καταστάσεις παιδιού $(s_1,...s_k)$, με τον εξής τρόπο:

Πρώτον, υπολογίζει το στοιχειοσύνολο συνολικού κόστους από τα στοιχειοσύνολα συνολικού κόστους των καταστάσεων παιδιού όπως ακριβώς γίνεται με την επισήμανση: για κάθε βασικό κανόνα r της μορφής $n \to o(n_1,,..,n_k)$, υπολογίζει:

$$r.\text{cost} + \sum s_i.\text{fullcost}[n_i].\text{cost}$$

Για κάθε μη-τερματικό, ο Dburg κατασκευάζει ένα στοιχείο που περιέχει το ελάχιστο δυνατό κόστος για το μη-τερματικό, και το σύνολο των βέλτιστων κανόνων για αυτό το μη-τερματικό. Το αποτέλεσμα είναι το προκαταρκτικό στοιχειοσύνολο I. Τότε ο Dburg εφαρμόζει επαναληπτικά αλυσιδωτούς κανόνες έως ότου δεν υπάρχει καμία αλλαγή: για έναν κανόνα αλυσίδας r της μορφής $n \to n_1$ υπολογίζει:

$$r.\text{cost} + I[n_1].\text{cost}$$

Εάν το προκύπτον κόστος c είναι πολύ μικρότερο από ό, τι το $l[n].\text{cost}$, το στοιχείο $l[n]$ αντικαθίσταται από ένα με το κόστος c και σύνολο κανόνων $\{r\}$; Αν το κόστος που προκύπτει είναι ίσο με $l[n].\text{cost}$, ο κανόνας προστίθεται στους βέλτιστους κανόνες για το στοιχείο $l[n]$.

Ο υπολογισμός του στοιχειοσυνόλου μερικού κόστους λειτουργεί ουσιαστικά με τον ίδιο τρόπο, με τις εξής διαφορές: Για τον υπολογισμό των στοιχειοσυνόλων μερικού κόστους για έναν τελεστή και για μια διατεταγμένη πλειάδα καταστάσεων, ο Dburg χρησιμοποιεί κάθε συνδυασμό στοιχειοσυνόλων μερικού κόστους (που εκπροσωπούν διάφορες παραλλαγές διαμοιρασμού των υπογράφων).

Ο Dburg παράγει επίσης στοιχειοσύνολα μερικού κόστους για κάθε υποσύνολο N των μη-τερματικών, όπου τα μέλη του υποσύνολου έχουν (απόλυτο) κόστος 0. Αυτό αναπαριστά το διαμοιρασμό του συνολικού γράφου που εκπροσωπείται από την κατάσταση μέσω των μη-τερματικών στο N. Για να παράγει ένα τέτοιο στοιχείο μηδενικού κόστους, οι κανόνες μπορεί να εφαρμόζονται για κόστος 0, ανεξάρτητα από το κανονικό τους κόστος (επειδή δεν συμβάλλουν στο αυξητικό κόστος), αλλά τα στοιχεία που χρησιμοποιούνται για την παραγωγή του στοιχείου πρέπει να έχουν επίσης κόστος 0 (ούτε πρέπει να συμβάλλουν στο αυξητικό κόστος). Ωστόσο, εάν ένας κανόνας δεν είναι μεταξύ των βέλτιστων κανόνων για οποιοδήποτε στοιχειοσύνολο μερικού κόστους όπου το κόστος των κανόνων μετράει, ούτε χρησιμοποιείται για την παραγωγή στοιχείου μηδενικού κόστους (αυτό αποφεύγει ορισμένα πλασματικά λάθη και προειδοποιήσεις κατά τον έλεγχο).

Επειδή το απόλυτο κόστος είναι σημαντικό για αυτούς τους υπολογισμούς, ο Dburg δεν χρησιμοποιεί τα σχετικά κόστη των στοιχειοσυνόλων μερικού κόστους που περιέχουν ένα στοιχείο κόστους 0. Για όλα τα υπόλοιπα στοιχειοσύνολα, ο Dburg χρησιμοποιεί το σχετικό

κόστος $(\cos t + \delta)$. Αυτό επιτρέπει στον αλγόριθμο να τερματίσει (στις περισσότερες περιπτώσεις), επειδή τα σχετικά κόστη επιτρέπουν στον αλγόριθμο λίστας έργου να προσδιορίσει ότι μια κατάσταση δεν είναι καινούρια.

Στο Σχήμα 9 φαίνονται οι καταστάσεις υπολογισμένες για την γραμματική του παραδείγματος. Στο υπόλοιπο της ενότητας δίνεται παράδειγμα για τον υπολογισμό των στοιχειοσυνόλων μερικού κόστους. Θα εξεταστεί μια συγκεκριμένη περίπτωση, και συγκεκριμένα ο τελεστής Plus με το A ως η αριστερή κατάσταση-παιδί και το B ως το δεξί παιδί (δηλαδή, κατάσταση D).

Η κατάσταση A αντιπροσωπεύει το γράφο Reg, η κατάσταση B αντιπροσωπεύει το γράφο Int, και η κατάσταση D αντιπροσωπεύει το Plus(Reg,Int). Η A έχει ένα στοιχειοσύνολο μερικού κόστους (Al), που αντιπροσωπεύει όλους τους τρόπους διαμοιρασμού αυτού του γράφου. Ο B έχει δύο στοιχειοσύνολα μερικού κόστους:

• Το B2 αντιπροσωπεύει όλους τους τρόπους διαμοιρασμού όπου το reg δεν καταβάλλεται (άμεσα ή έμμεσα) από κάποιον άλλο γονέα τα μη-τερματικά start και reg έχουν κόστος 1 (δεδομένου ότι ο κανόνας 3 έχει κόστος 1).

• Το Bl αντιπροσωπεύει όλους τους τρόπους διαμοιρασμού όπου το reg μοιράζεται και καταβάλλεται από κάποιον άλλο γονέα τα μη-τερματικά σύμβολα start και reg έχουν μηδενικό κόστος.

Υπάρχουν δύο συνδυασμοί των στοιχειοσυνόλων μερικού κόστους των παιδιών: Plus (Al, Bl) και Plus (Al, B2). Υπάρχουν 16 υποσύνολα του συνόλου των μη-τερματικών, δηλ., 16 δυνατές περιπτώσεις διαμοιρασμού, αλλά το μόνο σημαντικό ζήτημα για αυτό το παράδειγμα είναι εάν το reg είναι στο υποσύνολο ή όχι:

• Αν ναι, τότε ο κανόνας 5 (ο μόνος ένας εφαρμόσιμος για την παραγωγή του Plus από το Reg) πρέπει να εφαρμόζεται με κόστος 2· αυτό παράγει την D2 από το Plus (Al, Bl) και την D3 από το Plus (Al, B2).

• Αν όχι, τότε κανόνας 5 πρέπει να εφαρμοστεί με κόστος 0· αυτό είναι έγκυρο μόνο για το Plus (Al, Bl), το οποίο δίνει την Dl. Στο B2, το reg έχει μη μηδενικό κόστος, για αυτό και δεν διαμοιράζεται, και δεν μπορεί να χρησιμοποιηθεί για ένα διαμοιρασμένο γράφο Plus (Al, B2).

Στην Dl, δύο κανόνες είναι βέλτιστη για την παραγωγή του δένδρου από το addr· επιπρόσθετα του κανόνα 8α, ο κανόνας 6 είναι βέλτιστος στην D1, επειδή το reg με μηδενικό κόστος.

state	pattern	itemset	cost	rules	cost	rules	cost	rules	cost	rules
A	Reg	full-cost	0+δ	1	0+δ	2	0+δ	6		
		A1	0	1	0	2	0	6		
B	Int	full-cost	1+δ	1	1+δ	3	0+δ	7	0+δ	8b
		B1	0	1	0	3	0	6,7	0	8b
		B2	1	1	1	3	0	7	0	8b
C	Fetch(*)	full-cost	0+δ	1	0+δ	4	0+δ	6		
		C1	0	1	0	4	0	6		
		C2	0+δ	1	0+δ	4	0+δ	6		
D	Plus(Reg,Int)	full-cost	3+δ	1	3+δ	5	0+δ	8a		
		D1	0	1	0	5	0	6,8a		
		D2	2	1	2	5	0	8a		
		D3	3	1	3	5	0	8a		
E	Plus(*,*)	full-cost	0+δ	1	0+δ	5	0+δ	6		
		E1	0	1	0	5	0	6		
		E2	0+δ	1	0+δ	5	0+δ	6		
F	Plus(*,Int)	full-cost	3+δ	1	3+δ	5	0+δ	8a		
		F1	0	1	0	5	0	6,8a		
		F2	2	1	2	5	0	8a		
		F3	3	1	3	5	0	8a		
		F4	2+δ	1	2+δ	5	0+δ	8a		
		F5	3+δ	1	3+δ	5	0+δ	8a		

Figure 7: States for the tree grammar in Fig. 1 and their itemsets

Σχήμα 9: Καταστάσεις για τη γραμματική δένδρου του Σχ. 3 και τα στοιχειοσύνολά τους.

3.5.3.3. ΈΛΕΓΧΟΣ

Μετά τον υπολογισμό μιας κατάστασης, ο Dburg εκτελεί έναν έλεγχο για κάθε μη-τερματικό που μπορεί να παράγει DAG που αναπαρίσταται από την κατάσταση: Αν η τομή των συνόλων των κανόνων στα στοιχεία για αυτό το μη-τερματικό είναι κενή, τότε δεν υπάρχει κανόνας που να είναι βέλτιστος για όλους τις περιπτώσεις διαμοιρασμού, και ο τρόπος μας DAG ανάλυσης μπορεί να οδηγήσει σε υποβέλτιστη ανάλυση για ορισμένα DAG. Αυτό το αποτέλεσμα έχει δύο αιτίες: Είτε η καθολικά βέλτιστη παραγωγή είναι τοπικά μη βέλτιστη για αυτό το μη-τερματικό, ή πρόκειται για μια ψευδή σύγκρουση που προκύπτει από την εξέταση μιας παραγωγής ως δυνητικά καθολικά βέλτιστη για κάποιο πρότυπο διαμοιρασμού ενώ στην πραγματικότητα δεν είναι καθολικά βέλτιστο για οποιοδήποτε πρότυπο διαμοιρασμού.

Ο ελεγκτής παράγει επίσης ένα άλλο αποτέλεσμα: Αν ο κανόνας ο οποίος είναι βέλτιστος για το στοιχείο συνολικού κόστους δεν είναι για όλα τα στοιχείο μερικού κόστους, τότε η χρήση αυτού του κανόνα θα οδηγούσε σε υποβέλτιστη ανάλυση για κάποια DAG. Για το λόγο αυτό ο κανόνας αυτός δεν θα έπρεπε να χρησιμοποιείται για την ανάλυση αυτής της κατάστασης. Μία τροποποίηση του αναλυτή που επιτρέπει την αποφυγή αυτού του κανόνα είναι η γέννηση ενός αυτόματου ανάλυσης δένδρου από τις καταστάσεις τις οποίες έχει υπολογίσει ο Dburg. Αυτός ο αναλυτής θα χρησιμοποιούσε τους κανόνες που είναι βέλτιστοι για όλα τα στοιχειοσύνολα της κατάστασης.

Αν δεν υπάρχουν τέτοιοι μερικώς βέλτιστοι κανόνες, μπορούμε να χρησιμοποιήσουμε τη γραμματική με κάθε γεννήτορα συντακτικού αναλυτή δένδρου (όπως είναι οι Burg και Iburg).

Όσον αφορά το παράδειγμα του Σχ. 9: εύκολα φαίνεται ότι τα στοιχεία μερικού κόστους περιέχουν υπερσύνολα των συνόλων κανόνων των αντίστοιχων στοιχείων συνολικού κόστους. Έτσι η γραμματική του παραδείγματος μπορεί να χρησιμοποιηθεί για τη βέλτιστη ανάλυση DAG με οποιοδήποτε γεννήτορα δένδρου ανάλυσης.

ΚΕΦΑΛΑΙΟ 4:ΓΕΝΝΗΤΡΙΑ ΓΕΝΝΗΤΟΡΑ ΚΩΔΙΚΑ ΓΙΑ ΕΝΤΟΛΕΣ ΠΟΛΛΑΠΛΩΝ ΕΞΟΔΩΝ

4.1 ΠΕΡΙΛΗΨΗ

Διαπραγματευόμαστε το πρόβλημα επιλογής εντολής για Εντολές Πολλαπλών Εξόδων(MOI),παράγοντας περισσότερα από ένα αποτελέσματα. Τέτοιες εγγενώς παράλληλες εντολές υλικού είναι πολύ συνηθισμένες στην περιοχή των εφαρμογών των Επεξεργαστών Ειδικού Συνόλου Εντολών(ASIP) και των Επεξεργαστών Ψηφιακού Σήματος (DSP) οι οποίοι συχνά χρησιμοποιούνται στο Σύστημα των Ολοκληρωμένων Κυκλωμάτων σαν προγραμματιζόμενοι πυρήνες. Προκειμένου να παρέχουν υψηλού επιπέδου προγραμματισμό, και κατά συνέπεια να χαίρουν ευρείας αποδοχής, η υποστήριξη από εξελιγμένους μεταγλωττιστές είναι υψίστης σημασίας για αυτούς τους προγραμματιζόμενους πυρήνες. Δεδομένου ότι δεν είναι δυνατόν να μοντελοποιηθούν οι Εντολές Πολλαπλών Εξόδων ως δέντρα στον μεταγλωττιστή Ενδιάμεσης Αναπαράστασης(IR),οι παραδοσιακές προσεγγίσεις για την επιλογή κώδικα δεν είναι επαρκείς. Επεκτείνοντας τις παραδοσιακές προσεγγίσεις γεννήτορα κώδικα για επιλογή Εντολών Πολλαπλών Εξόδων (MOI)- είναι ουσιαστικά ένας γράφος που καλύπτει το πρόβλημα, το οποίο είναι γνωστό ως NP πλήρες. Παρουσιάζουμε ένα νέο ευρετικό αλγόριθμο ,ενσωματωμένο σε ένα εκ νέου στοχευόμενο γεννήτορα κώδικα γεννήτορα ικανού να αξιοποιήσει αυθαίρετα εγγενείς παράλληλες MOI. Αποδεικνύουμε την έννοια με την ενσωμάτωση του εργαλείου σε LCC μεταγλωττιστή η οποία στοχεύει σε διαφορετικά Σύνολα Εντολών Αρχιτεκτονικής βασισμένη στην αρχιτεκτονική MIPS. Αρκετές εφαρμογές δικτύου καθώς και ορισμένα σημεία αναφοράς DSP συντάχθηκαν και αξιολογήθηκαν προκειμένου να οδηγηθούμε σε αποτελέσματα

4.2 ΑΛΓΟΡΙΘΜΟΣ ΕΠΟΛΟΓΗΣ ΚΩΔΙΚΑ

Κατά τη διάρκεια επιλογής κώδικα κάθε DFT υφίσταται επεξεργασία σε δύο φάσεις: την φάση αντιστοίχησης και τη φάση της κάλυψης. Ο αντστοιχιτής διασχίζει τον DFT με κατεύθυνση από κάτω προς τα πάνω και εφαρμόζει μια γραμματική δέντρου προκειμένου να υποσημειώσει σε κάθε κόμβο που έχει επισκεφθεί κανόνες, το αντίστοιχο κόστος και συνεπακόλουθα τα Μη Τερματικά(NT) που μπορούν να χρησιμοποιηθούν για να αντιστοιχηθεί. Ο αλγόριθμος κάλυψης διασχίζει κατόπιν τον DFT από πάνω προς τα κάτω κι έχει πλεονέκτημα της υποσημειωμένης πληροφορίας για την συλλογή του κανόνα με

ελάχιστο κόστος για κάθε κόμβο. Για κάθε εντολή hardware (υλικού), οι κανόνες υφίστανται στη γραμματική δέντρου του τύπου:

$$NT \rightarrow \underbrace{opcode(op_1, \ldots op_n)}_{\text{simple rule}}\{costs\} = \{actions\} \quad (1)$$

Στην (1), ο κώδικας πράξης $(opcode)$ προσδιορίζει τον κανόνα τελεστή (MUL, ADD, κλπ) ο οποίος χρησιμοποιείται ως το όνομα του κόμβου IR ενώ συγχρόνως $\{op_1 ... op_n\}$ αντιπροσώπευε (ή αναπαριστά?) και τους τελεστές του κανόνα. Όπως μαρτυρούν τα ονόματα τα κόστη υπολογίζονται στο κόστος-τμήμα, και το τμήμα-πράξη περιέχει κώδικα C για να εκπέμπει assembly(συμβολική) ή χαμηλού επιπέδου IR κώδικα. Εφόσον αυτή η παρουσίαση δεν είναι επαρκής για Εντολές Πολλαπλών Εξόδων ,η γραμματική Cburg εκτείνεται αυτό το σκεπτικό κανόνων κατά τρόπο που οι κανόνες έχουν τη μορφή:

$$NT_1, NT_2, \cdots \rightarrow \underbrace{opcode_1(op_1, \ldots)}_{\text{split rule}}, \underbrace{opcode_2(op_1, \ldots)}_{\text{split rule}}, \ldots \quad (2)$$

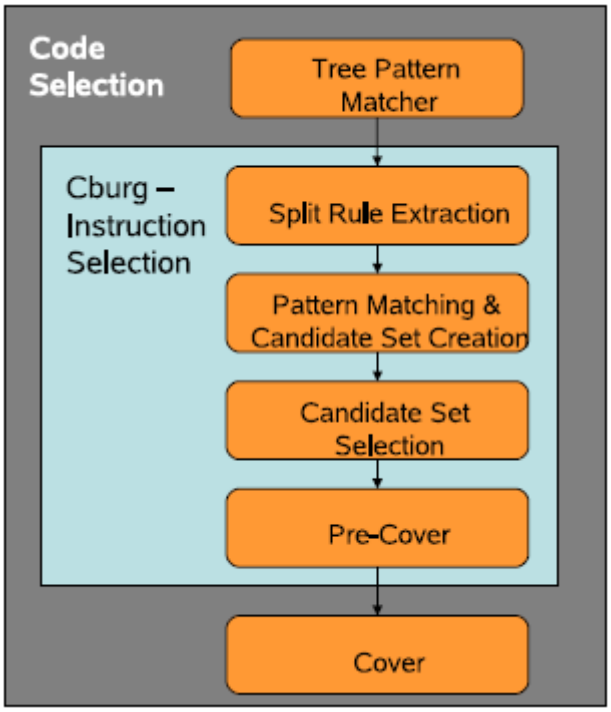
Στην (2), πολλαπλά μη τερματικά σύμβολα $NT_1, NT_2 \cdots$ παράγονται από έναν σύνθετο κανόνα. Ο σύνθετος κανόνας περιλαμβάνει διάφορους κανόνες όπως παρουσιάζεται στην (1) καθώς το κόστος-και πράξεις-τμήματα που δεν εμφανίζονται στην (2). Πριν ξεκινήσουμε να εξηγούμε τον αλγόριθμο, να παραθέτουμε τις ορολογίες που θα χρησιμοποιηθούν στο υπόλοιπο του παρόντος άρθρου :

- **κανόνας:** συμβολίζει ένα πρότυπο εντολής στην γραμματική δέντρου,
- **απλός κανόνας:** αντιπροσωπεύει ένα πρότυπο δέντρου όπως ADD, MUL,MAC
- **σύνθετος κανόνας:** αποτελείται από αρκετούς απλούς κανόνες,
- **κανόνας διαίρεσης:** είναι ένας απλός κανόνας που είναι μέρος του σύνθετου κανόνα

Όπως φαίνεται στο σχήμα 2, ο αλγόριθμος που παρουσιάζεται αποτελείται από πέντε κύριες φάσεις: την εξαγωγή κανόνων διαχωρισμού, την συνάρτηση αντιστοίχησης, την αναγνώριση κανόνων διαχωρισμού ,την επιλογή υποψήφιου συνόλου και την εκ των προτέρων κάλυψη. Πρώτον, όλοι οι σύνθετοι κανόνες στην γραμματική δέντρου αναλύονται και διαχωρίζονται στους αντίστοιχους κανόνες του διαχωρισμού (υποπαράγραφος 4.2.1).

Αυτοί οι κανόνες διαίρεσης χρησιμοποιούνται για την εύρεση υποψήφιων κόμβων στο IR, εκπροσωπώντας λειτουργίες ορισμένων MOI. Ο επιγραφών (υποπαράγραφος 4.2.1) υποσημειώνει τους απλούς κανόνες και όλους τους κανόνες διαχωρισμού σε κάθε υποψήφιο κόμβο που αντιστοιχίζεται. Μετά την υποσημείωση των κανόνων δημιουργείται ένας χάρτης απεικόνισης του κανόνα διαίρεσης ο οποίος περιλαμβάνει τους κανόνες διαχωρισμού και τους σχετικούς IR κόμβους. Χρησιμοποιώντας αυτό τον χάρτη απεικόνισης προσδιορίζεται το σύνολο των υποψήφιων κόμβων για κάθε σύνθετο κανόνα, έτσι ώστε να υπάρχει μια σαφής εικόνα σχετικά με όλες τις πιθανές λύσεις που καλύπτουν. Κατά την επόμενη φάση επιλογής συνόλου (υποενότητα 4.2.2), η ροή δεδομένων από διαφορετικά σύνολα κόμβων εξετάζονται προκειμένου να περιοριστούν τα άκυρα σύνολα.

Επιπλέον, τα υπόλοιπα σύνολα αξιολογούνται από μια νέα μέτρηση του κόστους που έχει εισαχθεί για να συγκρίνει το κόστος των απλών κανόνων και των κανόνων διαχωρισμού. Στην περίπτωση επικάλυψης των υποψηφίων συνόλων πρέπει να επιλεγεί το πιο πολύτιμο σύνολο. Η απόφαση αυτή χαρτογραφείται στο πρόβλημα της εύρεσης του Μέγιστου Σταθμισμένου Ανεξάρτητου Συνόλου (MWIS) ενός γράφου. Τελικά, τα υποψήφια σύνολα που προκύπτουν για όλους τους σύνθετους κανόνες επιλέγονται και ελέγχονται για το αν κάθε υποψήφιο σύνολο μπορεί όντως να καλυφθεί από τον αντίστοιχο σύνθετο κανόνα (υποενότητα 4.2.3). Στο τέλος, ένας κανονικός αλγόριθμος κάλυψης μπορεί να επεξεργαστεί την απόδοση ενός ευρετικού, αναδυόμενου έγκυρου κώδικα assembly.

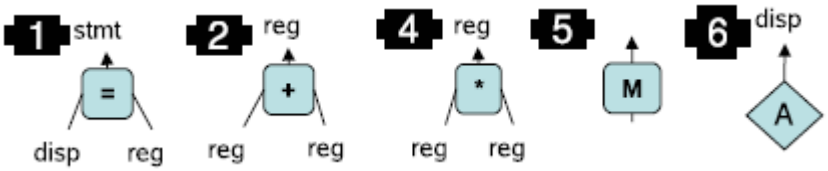


Σχήμα 10: Άποψη αλγορίθμου

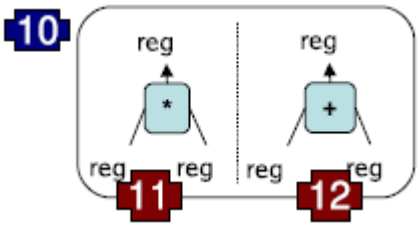
4.2.1 Σήμανση & Αναγνώριση Κανόνα Διαίρεσης

Σε αντίθεση με τον κανονικό συντακτικό αναλυτή δέντρου που υποσημειώνεται σε κάθε κόμβο για κάθε μη τερματικό σύμβολο μόνο εκείνοι οι κανόνες με την ελάχιστη τιμή κόστους, ο αναπτυγμένος επιγραφών υποσημειώνει επιπλέον όλους τους συνδυασμούς των κανόνων διαχωρισμού σε κάθε κόμβο, ανεξάρτητα από το κόστος τους και από τα μη τερματικά σύμβολα. Για το λόγο αυτό, οι κανόνες διαχωρισμού εξάγονται από κάθε σύνθετο κανόνα και οι αριθμοί των κανόνων που τους έχουν ανατίθενται σε αυτούς και λειτουργούν ως αναγνωριστικά. Στο σχήμα 3 δίνεται ένα παράδειγμα για τη διαδικασία αυτή. Στο πάνω μισό του σχήματος 2, δίνεται γραμματική δέντρου από απλούς και σύνθετους κανόνες. Οι κανόνες αυτοί υποσημειώνονται στους κόμβους του IR δένδρου στο κάτω μισό του σχήματος 3. Στους κόμβους 5 και 10 υποσημειώνονται τόσο οι κανόνες διαχωρισμού όσο και οι απλοί κανόνες, καθένας από τους οποίους παράγει το ίδιο μη τερματικό σύμβολο. Για λόγους απλούστευσης, οι τιμές κόστους και τα μη τερματικά σύμβολα δεν παρουσιάζονται σ' αυτό σχήμα. Μετά τη φάση της επισήμανσης, η απεικόνιση του κανόνα διαίρεσης δημιουργείται και αποθηκεύει τους συνδυασμούς κανόνων κόμβου-διαίρεσης. Οι διαδοχικές φάσεις μπορούν να χρησιμοποιήσουν τις πληροφορίες αυτής της απεικόνισης για να υπολογίσουν τα υποψήφια σύνολα κανόνων για κάθε κανόνα διαχωρισμού, π.χ. ο κόμβος 5 είναι υποψήφιος για τον κανόνα διαχωρισμού 11.

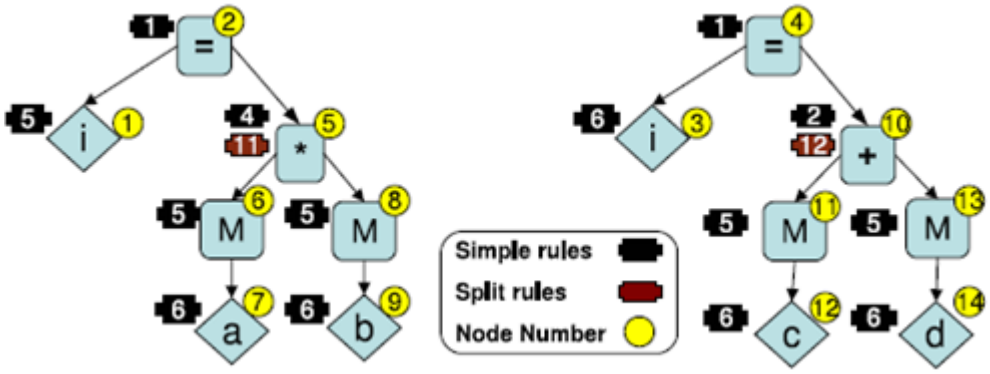
Απλοί κανόνες:



Σύνθετοι κανόνες:



Σήμανση IR δέντρου



Σχήμα 11:Σήμανση

4.2.2 ΕΠΙΛΟΓΗ ΥΠΟΨΗΦΙΟΥ ΣΥΝΟΛΟΥ

Κατά τη διάρκεια της φάσης επιλογής υποψηφίου συνόλου το καλύτερο υποψήφιο σύνολο κόμβων (CS) για όλα τα MOI αναγνωρίζεται και αξιολογείται. Για το σκοπό αυτό, εφαρμόζονται. οι πληροφορίες από την απεικόνιση του κανόνα διαχωρισμού. Ένα υποψήφιο σύνολο MOI περιέχει όλους τους υποψήφιους συνδυασμούς των κόμβων για αυτή την εντολή. Ένας έγκυρος υποψήφιος κόμβος ορίζει το σύνολο των υποψήφιων κόμβων καθένας από τους οποίους συνοδεύεται από ένα διαφορετικό κανόνα διαχωρισμού από τον ίδιο σύνθετο κανόνα. Επειδή οι υποψήφιοι κόμβοι μπορούν επίσης να συμπληρώσουν τα πρότυπα δέντρου όπως τα MAC, μόνο οι *κόμβοι της ρίζας* αποθηκεύονται μέσα στο υποψήφιο σύνολο: οι κόμβοι της ρίζας είναι εκείνοι οι οποίοι δεν έχουν διαδοχικό κόμβο μέσα στο πρότυπο τους. Για παράδειγμα που δίνεται στο σχήμα 11, το μόνο σύνολο κόμβων για τον σύνθετο κανόνα 10 είναι {5,10}.

Ο αριθμός του συνόλου των κόμβων μειώνεται περαιτέρω κατά τον έλεγχο των εξαρτημένων δεδομένων στο DFG μεταξύ των υποψήφιων κόμβων και το μειωμένο σύνολο κόμβων περιλαμβάνει εξαρτημένους κόμβους. Το υπόλοιπο σύνολο των κόμβων αξιολογήθηκε προκειμένου να μεγιστοποιηθεί το όφελος της επιλογής κώδικα. Η αξιολόγηση του συνόλου πραγματοποιείται σε σχέση με τους άλλους διαθέσιμους κανόνες για ένα συγκεκριμένο κόμβο. Η βάση για την αξιολόγηση αυτή είναι προφανώς το μετρικό κόστος των κανόνων . Δυστυχώς, το τυπικό κόστος εκτίμησης των δέντρων πρότυπο επίσης δεν είναι επαρκές για την εκτίμηση των MOI. Παραδοσιακά οι μετρήσεις του κόστους για τους κανόνες αφορούν μόνο το *σταθερό κόστος* των κανόνων, όπως ο αριθμός των εντολών που δίδονται. Ωστόσο, η εφαρμογή σύνθετων κανόνων επηρεάζει διάφορες καταστάσεις του δέντρου και ως εκ τούτου προκαλεί διαφορετικό κόστος σε διαφορετικές καταστάσεις δέντρου ενώ συγχρόνως εξαρτώνται σε μεγάλο βαθμό από τη τρέχουσα κατάσταση αντιστοίχησης. Τέτοιες τιμές κόστους μπορούν να περιγραφούν ως δυναμικές τιμές κόστους το οποίο είναι ορθογώνια του σταθερού κόστους των απλών κανόνων.

4.2.2.1 ΥΠΟΛΟΓΙΣΜΟΣ ΚΟΣΤΟΥΣ ΓΙΑ ΣΥΝΘΕΤΟΥΣ ΚΑΝΟΝΕΣ

Εάν ένας κόμβος IR μπορεί να συνδυαστεί με ένα απλό κανόνα $(rule_{smpl})$ και με ένα κανόνα διαίρεσης $(rule_{split})$, οι οποίοι μειώνονται με το ίδιο μη τερματικό σύμβολο, ο Cburg συγκρίνει τις τιμές κόστους των απλών κανόνων και τις τιμές κόστους των κανόνων διαχωρισμού για να προσδιορίσει την καλύτερη λύση για αυτόν τον κόμβο. Ο υπολογισμός του κόστους ενός σύνθετου κανόνα $(rule_{cplx})$, αποτελείται από δύο μέρη: το κόστος αποθήκευσης και το διπλασιασμό του κόστους :

Κόστος Αποθήκευσης

$Csaved(SC)$ προσδιορίζει τη διαφορά του σταθερού κόστους μεταξύ της λύσης που καλύπτει με απλούς κανόνες και μιας λύσης με ένα σύνθετο κανόνα για ένα συγκεκριμένο υποψήφιο σύνολο:

$$Csaved(CS) = (\sum_{nodes \in CS} C_{fix}(rule_{smpl})) - C_{fix}(rule_{cmpl})$$

Όπου C_{fix} περιγράφει το σταθερό κόστος των αυθαίρετων κανόνων (απλών/σύνθετων), που παράγουν το ίδιο μη τερματικό σύμβολο

Διπλασιασμός κόστους

$Cdup\ (CS)$ παράγονται από την παρουσία των Κοινών Υποέκφρασεων (CSE) εντός ενός πρότυπου κόμβου στο IR που μπορεί να καλύπτεται από έναν κανόνα διαχωρισμού. Τα Πρότυπα Κανόνων Διαίρεσης (SRP) μπορούν να εκφράσουν αυθαίρετα πρότυπα δέντρου όπως MAC ή άλλες αλυσιδωτές εντολές, αποτελούμενα από διάφορους υπό- κόμβους. Το σύνολο των υπό-κόμβων μπορούν να χωριστούν σε κόμβο-ρίζα (R) και κόμβο-παιδί (K). Σε αντίθεση με τον κόμβο-ρίζα όπου το αποτέλεσμα αντιπροσωπεύει το αποτέλεσμα του SRP, κάθε κόμβος- παιδί έχει ακριβώς ένα διάδοχο μέσα στο μοτίβο του κανόνα διαχωρισμού. Εάν ένα CSE καλύπτεται από έναν κόμβο-παιδί ενός SRP το αποτέλεσμα του δεν είναι πλέον διαθέσιμο για τους διαδόχους κόμβου έξω από το SRP επειδή οι αλυσιδωτές εντολές υπολογίζουν μόνο ένα αποτέλεσμα. Ως εκ τούτου, ο κόμβος πρέπει να είναι διπλασιασμένος προκειμένου να υπολογιστεί το αποτέλεσμα και να διατηρηθεί η εγκυρότητα του παραγόμενου κώδικα assembly. Κατά συνέπεια, ορίζουμε:

$$K(CS) = \{node \mid node\{SRP\} \wedge node \neq R\}$$

$$CSE(CS) = \{node \in K(CS) \mid nodeisaSCE\}$$

$$C_{dup}(CS) = \sum_{node \in CSE(CS)} (\sum_{fan_{out}(node)} C_{fix}(rule_{smpl}))$$

Εδώ, fan_{out} υποδηλώνει τον αριθμό των εξερχόμενων ακμών ενός κόμβου $\in K(CS)$ που προέρχεται από το SRP. Για κάθε μία από ακμές, $C_{fix}(rule_{smpl})$ αντιπροσωπεύει το κόστος, σύμφωνα με τον απλό κανόνα που παράγει εκείνα τα μη τερματικά σύμβολα για κάθε διάδοχο κόμβο.

Κόστος Ευκαιρίας

Το $Copp(CS)$ του υποψηφίου συνόλου είναι το κόστος εκείνο που μετατρέπει το συνολικό κόστος-διαφορά μεταξύ της εφαρμογής του σύνθετου κανόνα και μια εναλλακτική λύση που καλύπτεται από ένα σύνολο απλών κανόνων για τους κόμβους στο CS :

$$C_{opp}(CS) = C_{saved}(CS) - C_{dup}(CS)$$

ΚΟΣΤΟΣ ΚΑΝΟΝΩΝ ΔΙΑΙΡΕΣΗΣ

Τελικά, το κόστος των κανόνων διαίρεσης που υπολογίζεται από το μέσο κόστος ευκαιρίας του υποψήφιου συνόλου των σύνθετων κανόνων:

$$C(rule_{split}) = C_{fix}(rule_{split}) - C_{average\_opp}(CS)$$

Όπου $C_{fiix}(rule_{split})$ είναι το σταθερό κόστος των κανόνων διαχωρισμού. Επιπλέον, το μέσο κόστος ευκαιρίας $C_{average\_opp}(CS)$ ενός υποψηφίου συνόλου CS υπολογίζονται ως εξής:

$$C_{average\_opp}(CS) = C_{opp}(CS) / num_{spli}(CS) \quad ,$$

Όπου $num_{split}(CS)$ είναι ο αριθμός των κανόνων διαίρεσης του υποψήφιου συνόλου.

4.2.2.2 ΕΠΩΦΕΛΟΥΜΕΝΗ ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ

Μετά την εκτίμηση του υποψηφίου, το υποψήφιο σύνολο δεν μπορεί να μειωθεί περαιτέρω και οι υπόλοιποι κόμβοι θα πρέπει να καλύπτονται σύμφωνα με τους σύνθετους κανόνες . Ωστόσο, μπορεί να προκύψουν τομές μεταξύ των υποψήφιων συνόλων που εμφανίζονται. Σε αυτήν την περίπτωση, όταν ένας κόμβος είναι υποψήφιος για αρκετά MOI και, κατά συνέπεια, είναι εισηγμένος σε διάφορα υποψήφια σύνολα. Τέτοιους κόμβους τους καλούμε *αλυσιδωτούς κόμβους*

Δεδομένου ότι οι κόμβοι αυτοί μπορούν να καλυφθούν μόνο από ένα κανόνα, η απόφαση κάλυψης για αλυσιδωτούς κόμβους πρέπει να εξασφαλίζει ότι το όφελος από πλευράς κόστους μεγιστοποιείται. Το πρόβλημα της εύρεσης μιας βέλτιστης λύσης για την κάλυψη των αλυσιδωτών κόμβων μπορεί να μειώσει το πρόβλημα της εύρεσης ενός Μέγιστου Σταθμισμένου Ανεξάρτητου Συνόλου (MWIS) σε σταθμισμένο μη κατευθυνόμενο γράφο $G = (V, E, W)$, χωρίς βρόγχους και πολλαπλές ακμές. Στο G,V ορίζει το σύνολο των κορυφών, E το σύνολο των ακμών και W είναι η κορυφή της συνάρτησης στάθμισης.

Ένα ανεξάρτητο σύνολο σε ένα γράφο είναι η συλλογή των κορυφών εκείνων που είναι αμοιβαία μη-παρακείμενες. Το πρόβλημα της εύρεσης ενός ανεξάρτητου συνόλου μέγιστης πληθικότητας(ισχύς) είναι ένα από τα θεμελιώδη συνδυαστικά προβλήματα και είναι γνωστό ότι είναι NP-πλήρες, ακόμη κι όταν όλοι οι κόμβοι έχουν ομοιόμορφα βάρη. Λόγω αυτού, θα

εφαρμόσουμε ένα ευρετικό που ονομάζεται GWMIN2. Γενικά, GWMIN2 ανήκει στην τάξη των κατώτερων βαθμών άπληστων αλγόριθμων που κατασκευάζουν ένα ανεξάρτητο σύνολο, επιλέγοντας κάποια κορυφή κατώτερου βαθμού, αφαιρώντας αυτήν και τις γειτονικές της κορυφές από το γράφο και να επαναληφθεί έως ότου είναι άδειος . Τέτοιοι αλγόριθμοι εκτελούνται σε γραμμικό χρόνο στον αριθμό των κορυφών και των ακμών. Ο GWMIN2 επιλέγει σε κάθε επανάληψη μία κορυφή υ, έτσι ώστε να μεγιστοποιείται

$$\frac{W(\upsilon)}{\sum_{w \in N_G^+(\upsilon)} W(\omega)}, \forall u \in V$$

Είναι μέγιστη. Αποδεικνύεται ότι το αποτέλεσμα του ανεξάρτητου συνόλου έχει τουλάχιστον ένα βάρος

$$\sum_{\upsilon \in V} \frac{W(\upsilon)^2}{\sum_{u \in N_G^+(\upsilon)} W(u)}$$

Ο συμβολισμός $N_G(\upsilon)$ ορίζει τη γειτονιά των κορυφών υ στο G και $N_G^+(\upsilon)$ στο σύνολο $\{\upsilon\} \cup N_G(\upsilon)$.

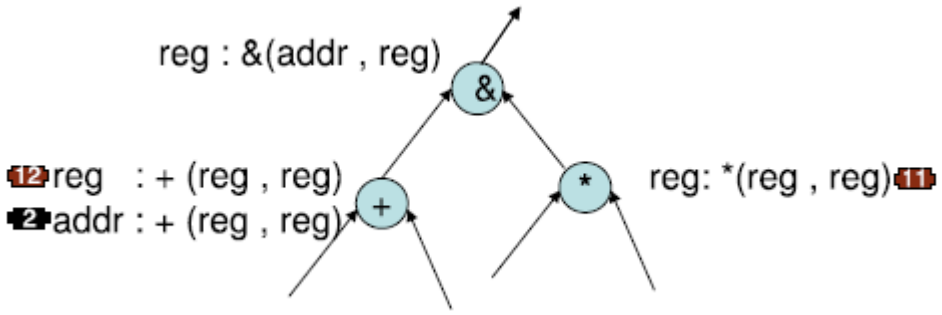
Εφαρμογή MWISP για μεγιστοποίηση οφέλους

Για κάθε μεγιστοποίηση του οφέλους στην παρουσία των επικαλυπτόμενων MOI, ο γράφος $G = (V, E, W)$ είναι κατασκευασμένος. Στο G , κάθε κορυφή που $\upsilon \in V$ αποτελεί ένα σύνθετο κανόνα και το σχετικό βάρος $W(u)$ είναι ίσο με το όφελός της. Βασικά, το όφελος ενός MOI υπολογίζεται ως το άθροισμα που αναιρείται από όλα τα κόστη των αποτελούμενων κανόνων διαίρεσης $(-1)\sum_{rule_{split}}(CS)$. Μεταξύ των δύο κορυφών του G μία ακμή υπάρχει α αν και μόνο αν τα MOI που συνδέονται έχουν ένα ή παραπάνω υποψήφιους IR-κόμβους είναι κοινοί. Ο αλγόριθμος τώρα απλά επιλέγει εκείνες τις μη-γειτονικές κορυφές

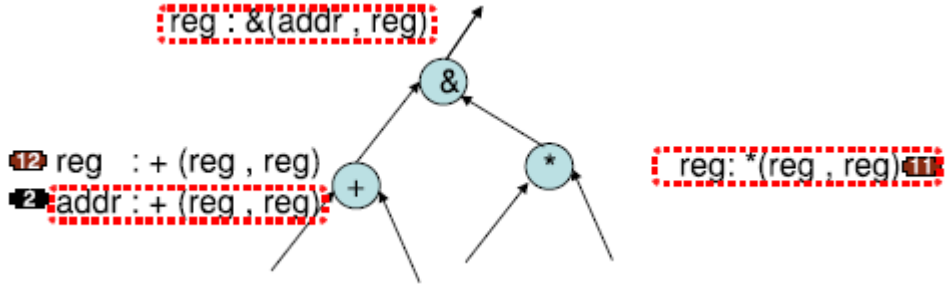
με το μεγαλύτερο βάρος (όφελος) σε ένα άπληστο τρόπο και εξαλείφει τις συμπεριλαμβανόμενες ακμές από το γράφο G.

4.2.3 ΦΑΣΗ ΕΚ ΤΩΝ ΠΡΟΤΕΡΩΝ ΚΑΛΥΨΗΣ

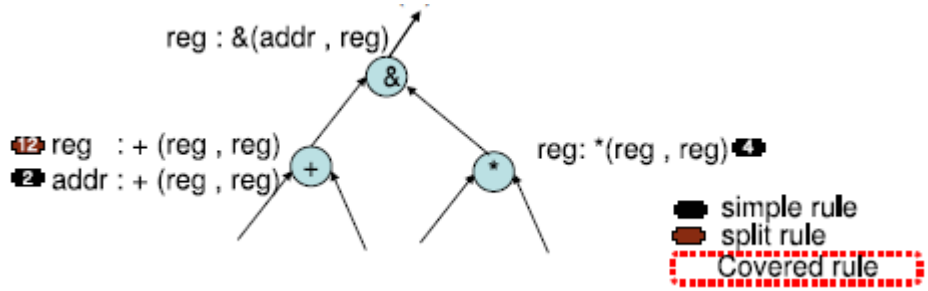
Σε αυτήν την φάση του αλγορίθμου, η επιλογή του κόμβου πρέπει να αξιολογηθεί και να καλυφθεί εκ των προτέρων, πριν ξεκινήσει η πραγματική φάση κάλυψης, δεδομένου ότι οι κανόνες διαχωρισμού δεν προσφέρουν κατ 'ανάγκη το ελάχιστο κόστος για κάθε μη τερματικό σύμβολο που δύναται να παραχθεί σε έναν υποψήφιο κόμβο. Κατά συνέπεια, λόγω των διαφορετικών απαιτήσεων των μη τερματικών σύμβολων των μεταγενέστερων IR-κόμβων, ένας υποψήφιος κόμβος μπορεί να μην καλύπτεται από τον κανόνα διαχωρισμού αν και ο κανόνας διαχωρισμού έχει ελάχιστο κόστος σχετικά με αυτό το μη τερματικό σύμβολο. Στην περίπτωση αυτή, θα πρέπει να εξασφαλιστεί ότι όλοι οι άλλοι κόμβοι του ίδιου συνόλου δεν καλύπτονται από τους κανόνες διαχωρισμού τους. Αυτό επιτυγχάνεται με την εκ προτέρων κάλυψη του IR. Όσο διαρκεί αυτό, η φάση κάλυψης του πρότυπου δέντρου προσομοιώνεται και σε περίπτωση που ένας υποψήφιος κόμβος δεν καλύπτεται από τον κανόνα διαίρεσης, όλοι οι κόμβοι του σύμφωνα με το υποψήφιο σύνολο συνδυάζονται εκ νέου με απλούς κανόνες.



(a)Μετά την επιλογή συνόλου



(b)εκ των προτέρων κάλυψη



(c)εκ νέου κάλυψη

Σχήμα 12:Εκ των προτέρων κάλυψη κόμβων

Στο σχήμα 12 (a) φαίνεται ένα παράδειγμα για μια τέτοιας κατάστασης. Αυτό παρουσιάζει ένα σύνολο από IR-δέντρα τα οποία έχουν ήδη επισημανθεί, επίσης, και ένα επιλεγμένο σύνολο των υποψήφιων κόμβων χαρακτηρίζονται από προσαρτημένους κανόνες διαίρεσης τους 11 και 12 του σχήματος 2. Στο σχήμα 12 (b), ο κανόνας διαχωρισμού 12 δεν χρησιμοποιείται για την κάλυψη, δεδομένου ότι ο διάδοχος κανόνας καταναλώνει ένα μη τερματικό σύμβολο εκείνο παράγεται φθηνότερα από τον απλό κανόνα 2. Ωστόσο, ο κανόνας διαίρεσης 11 χρησιμοποιείται για την κάλυψη ταυτόχρονα. Για την επίλυση αυτού του ανταγωνισμού, οι κανόνες διαχωρισμού απαλείφονται και όλοι οι υποψήφιοι κόμβοι εκ νέου συνδυάζονται με απλούς κανόνες, όπως παρουσιάζεται στο σχήμα 12 (c). Τώρα, τα IR-δέντρα μπορούν να διέρχονται από τη φάση της κάλυψης και ο κώδικας assembly εκπέμπεται.

4.2.4 ΑΝΑΛΥΣΗ ΠΟΛΥΠΛΟΚΟΤΗΤΑΣ

Η βάση του αλγορίθμου επιλογής κώδικα, εισήχθη στην ενότητα 1, βασίζεται στο πρώτο πέρασμα σε βάθος διαχωρισμό του $DFG = (V, E)$ σε $O(V + E)$ χρόνο στη φάση επισήμανσης. Επιπλέον, το κόστος υπολογίζεται προκειμένου να μεγιστοποιηθεί το όφελος της επιλογής κώδικα που εφαρμόζεται σε κάθε κόμβο. Οι υπολογισμοί μπορούν να χωριστούν σε δύο μέρη: ο υπολογισμός του C_{saved} και του C_{opp}.Ενώ το πρώτο μπορεί να υπολογιστεί γραμμικά εξαρτάται από τον αριθμό των κόμβων των υποψήφιων κανόνων για όλα τα MOI, ο τελευταίος υπολογισμός γίνεται:

$$O(\sum_{\substack{CS \subset V, \\ V \in DFG}} \sum_{CSE \subset CS} |Adj(CSE(CS))|)$$

χρόνο. Στην χειρότερη περίπτωση, αυτό θα μπορούσε να εξελιχθεί σε ένα εκθετικό χρόνο εκτέλεσης, εάν κάθε κόμβος $\upsilon \in V$ σε ένα CSE και στο ίδιο χρόνο καλύπτεται από τον κανόνα διαίρεσης. Επιπλέον, στην περίπτωση της επικάλυψης του πρότυπου MOI - ο MWISP λύνεται με ένα άπληστο ευρετικό που εκτελείται σε γραμμικό χρόνο, εξαρτάται από την ποσότητα των επικαλυπτόμενων MOI. Η τελική εκ των προτέρων κάλυψη φάση επισκέπτεται όλους τους κόμβους σε κάθε υποψήφιο σύνολο μια φορά. Με αυτόν τον τρόπο η πολυπλοκότητα του εκφράζεται ως εξής:

$$O(\sum_{\substack{CS \subseteq V, \\ V \in DFG}} |CS|)$$

η οποία ισούται επίσης με γραμμικό χρόνο εκτέλεσης. Συνολικά, η χειρότερη περίπτωση χρόνου εκτέλεσης είναι εκθετική για τον αριθμό των CSE που καλύπτονται από τους κανόνες διαχωρισμού, αλλά είναι γραμμική του μεγέθους των DFG που θεωρούνται.

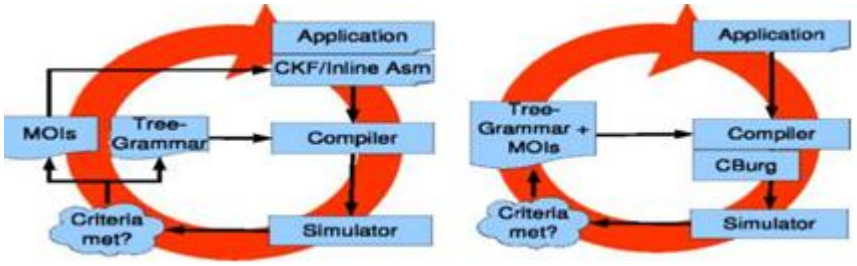
4.3 ΕΚ ΝΕΟΥ ΕΠΙΛΟΓΗ ΚΩΔΙΚΑ ΕΝΤΟΣ CBURG

Εκτός από την αποτελεσματική επιλογή κώδικα με την παρουσία του MOI, η εκ νέου στοχευμένη συλλογή είναι η δεύτερη πτυχή σε αυτό το άρθρο. Η εκ νέου στοχευμένη σύνταξη για υψηλού επιπέδου γλώσσας όπως η C / C++ είναι μία από τις μεγαλύτερες

προκλήσεις στην εξερεύνηση της αρχιτεκτονικής ASIP. Παρ 'όλα αυτά, οι επαναληπτικές προσεγγίσεις στην εξερεύνηση της αρχιτεκτονικής απαιτεί ανάπτυξη για κάθε ευέλικτο εργαλείο λογισμικού(software),προκειμένου να διερευνήσει πολλές σχεδιαστικές εναλλακτικές λύσεις αρχιτεκτονικής σε σύντομο χρονικό διάστημα. Σε αυτό το γενικό πλαίσιο, πολλές πλατφόρμες σχεδιασμού, συμπεριλαμβανομένων των εκ νέου στοχευόμενων μεταγλωττιστών που έχουν αναπτυχθεί στο παρελθόν [Scharwaecheter 12, Scharwaecheter 13, Scharwaecheter 14, Scharwaecheter 15]. Όπως παρουσιάζεται στο σχήμα 4 (α), οι επαναλήψεις ροής του σχεδιασμού περιλαμβάνουν συνήθως την σύνταξη, την προσομοίωση και τα χαρακτηριστικά των εφαρμογών της C / C++ για ένα συγκεκριμένη εικονικό πρότυπο αρχιτεκτονικής. Με βάση τα αποτελέσματα των χαρακτηριστικών, η συμφόρηση προσδιορίζεται , οι εντολές των μέσων πληροφορίας τελειοποιούνται και οι προσαρμοσμένες εντολές προστίθενται σταδιακά να βελτιώσουν την αποδοτικότητα της αρχιτεκτονικής. Οι νέες εντολές δηλώνονται στον μεταγλωττιστή, προκειμένου να αξιολογηθεί το όφελος τους για τις στοχευόμενες εφαρμογές κατά τη διάρκεια του επόμενου κύκλου σύνταξης-προσομοίωσης. Ενώ οι απλές εντολές μπορούν να δηλωθούν στην γραμματική δέντρου του γεννήτορα κώδικα, οι MOI συνήθως εφαρμόζονται ως CKF ή ενσωματωμένοι στην assembly, δεδομένου ότι δεν μπορούν να στοχεύσουν από τον μεταγλωττιστή. Αυτό συνεπάγεται την τροποποίηση του εγχειρίδιου των εφαρμογών και του μεταγλωττιστή, αντίστοιχα, γεγονός που οδηγεί σε υψηλό κόστος γενικά για μεγάλες εφαρμογές. Επειδή συχνά μόνο οι συμφορήσεις αυτές κάθε αυτές εφαρμόζονται μέσω MOI, περαιτέρω αξιοποίηση των προηγμένων προγραμματιστών του MOI παραμένει ανεκμετάλλευτη και η επαναχρησιμοποίηση προς διαφορετικές εφαρμογές δεν μπορεί να εξασφαλιστεί. Για να υποστηρίξει το μέγιστο διαφορετικών πλαισίων σύνταξης και να ξεπεράσει τους περιορισμούς που σχετίζονται με τον μεταγλωττιστή στην εξερεύνηση της αρχιτεκτονικής, η επιλογή κώδικα αλγόριθμου που περιγράφεται έχει ενσωματωθεί σε ένα εκ νέου στοχευμένο γεννήτορα κώδικα γεννήτορα που ονομάζεται Cburg. Ο Cburg βασίζεται στον Olive [Scharwaecheter 16] ο οποίος λαμβάνει ένα αρχείο ρυθμίσεων ως είσοδο και παράγει ένα σύνολο δομών δεδομένων και λειτουργίες παραγωγής κώδικα για ορισμένο στόχο ISA. Ωστόσο, σε αντίθεση με τον Olive, ο αλγόριθμος επιλογής κώδικα Cburg λειτουργεί σε DFG καλύτερα παρά σε DFT όπως περιγράφεται στην παράγραφο 1.Το αρχείο ρυθμίσεων του Cburg είναι παρόμοιο με του Olive. Το αρχείο περιέχει την περιγραφή του στόχου ISA με όρους του IR πρότυπου, καθώς και μια σειρά από λειτουργίες, απαραίτητες για πρόσβαση στον μεταγλωττιστή IR. Τα IR-πρότυπα εκπροσωπούν τη γραμματική που

χρησιμοποιείται για να προσδιορίσει πρότυπα στον IR μεταγλωττιστή και να τα απεικονίσει σε επαρκή assembly γλώσσα ή χαμηλού επιπέδου IR. Οι κανόνες μέσα σε αυτή την γραμματική έχουν τη μορφή και των δύο, κανόνα 2 και το κανόνα 1 . Οι παραγόμενες δομές δεδομένων και οι υπηρεσίες παρέχουν πλήρη μεθοδολογία που βασίζεται στον αλγόριθμο που περιγράφεται.

Οι σχεδιαστές μεταγλωττιστών μπορούν να τους χρησιμοποιήσουν για να εφαρμόσουν άνετα σε έναν αλγόριθμο επιλογής κώδικα για αυθαίρετους στόχους με μηχανήματα MOI. Έτσι είναι δυνατόν να δηλώσουν κάθε είδους εντολή hardware (υλικού) για τον μεταγλωττιστή και την ανάγκη για να τροποποιήσουν χειροκίνητα τις εφαρμογές της πηγής ή από το ίδιο το μεταγλωττιστή που παραλείπεται (σχήμα 13(b)).



Σχήμα 13(a)Παραδοσιακή ανάπτυξη ροής (b)Ανάπτυξη ροής με Cburg

4.4 ΠΕΙΡΑΜΑΤΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ

Προκειμένου να αξιολογηθεί η ποιότητα της προτεινόμενης μεθοδολογίας επιλογής κώδικα για να διευκολύνει μια πιο αποτελεσματική σχεδίαση συνόλου εντολών, Cburg έχει ενσωματωθεί στο Μικρό Μεταγλωττιστή C (LCC) [Scharwaecheter 17]. Δεδομένης της αρχιτεκτονικής του στόχου, έχει χρησιμοποιηθεί η αρχιτεκτονική MIPS [Scharwaecheter 18] . Βασισμένος στον MIPS ISA, καινούρια MOI έχουν αναπτυχθεί. Η ονοματολογία του κάθε MOI αντανακλάται μέσα από την αλληλουχία των ονομάτων των εντολών, οι παράλληλες λειτουργίες αντανακλώνται στον MOI. Για παράδειγμα, η εντολή "lwlw" περιγράφει την ταυτόχρονη εκτέλεση δύο "lw", το οποίο είναι ένα απλό φορτίο στο MIPS ISA. Η σουίτα αναφοράς περιλαμβάνει χαρακτηριστικούς συμμετρικούς αλγόριθμους κρυπτογράφησης όπως τον αλγόριθμο Data Encryption Standard (3DES), και το Advanced Encryption Standard (AES) [Scharwaecheter 19]. Και μια στοίβα πρωτοκόλλου Διαδικτύου που αποτελείτε από ένα επίπεδο IPv6, συμπεριλαμβανομένης της αυθεντικοποίησης, της κρυπτογράφησης καθώς και ένα επίπεδο Ethernet. Επιπλέον, μια διαφορική

αναπροσαρμοστική διαμόρφωση παλμού (ADPCM) DSP- η εφαρμογή λαμβάνεται από τον DSP-επίπεδο σουίτας αναφοράς [Scharwaecheter 20] έχει εξεταστεί. Για να αναπτύξουν καινούριο MOI, όλες οι αναφορές έχουν χαρακτηριστεί με ένα λεπτόκοκκο προφίλ [Scharwaecheter 21] για να προσδιορίσουν σημαντικά σημεία και υποσχόμενες υποψήφιες εντολές. Διάφορα MOI έχουν αναπτυχθεί δίνοντας ιδιαίτερη προσοχή στους συμμετρικούς αλγόριθμους κρυπτογράφησης, δηλαδή στους AES και 3DES. Δεδομένου ότι η συμμετρική κρυπτογράφηση είναι μια από τις μείζονες αναφορές επεξεργασίας IPv6, αναμένεται επίσης ότι αυτά τα MOI's θα επηρεάσει συνολικά στην εκτέλεσή της στοίβας του πρωτοκόλλου επίσης.Από τα αποτελέσματα του χαρακτηριστικού, αποδείχθηκε η μετατόπιση (sll / srl),ο μετασχηματισμός XOR και το φορτίο (LW) χρησιμοποιούνται πιο συχνά στους αλγόριθμους κρυπτογράφησης. Επομένως, τα ανεπτυγμένα MOI βασίζονται σε αυτές τις εντολές. Πρώτον, όλα τα MOI έχουν εφαρμοστεί χωριστά. Αργότερα τρία και περισσότερα MOI έχουν συνδυαστεί. Ο Πίνακας 1 περιέχει μια γενική εικόνα των πειραματικών αποτελεσμάτων που λαμβάνονται. Τα καλύτερα αποτελέσματα επιτεύχθηκαν με το MOI "lwsll" (16,96%% επιτάχυνση/-13.99 μέγεθος κώδικα) για 3DES και "lwxor" (12,83% επιτάχυνση / -9,96% μέγεθος κώδικα) για AES, αντίστοιχα. Η συνολική επίδοση των βελτιώσεων του 24,07% (3DES), 21,76% (AES) και 17,21% (IP στοίβα) ήταν πιθανή. Προφανώς, τα MOI δεν οδηγούν σε σημαντικές βελτιώσεις για τις αναφορές ADPCM , δεδομένου ότι η χρήση του διαχειριστή της διαφέρει σημαντικά από εκείνους της κρυπτογράφησης και της επεξεργασίας πρωτόκολλου

4.5ΣΥΜΠΕΡΑΣΜΑ

Παρουσιάσαμε τον γεννήτορα κώδικα γεννήτορα Cburg, που προσφέρει μια νέα μεθοδολογία για να εκμεταλλεύεται MOI κατά τη διάρκεια της εξερεύνησης της αρχιτεκτονικής. Στη θέση της κωδικοποίησης που είναι επιρρεπής σε σφάλματα ενσωματωμένα σε assembly ή CKF, οι σχεδιαστές του συστήματος μπορούν να μοντελοποιήσουν όλες τις εντολές που αναπτύχθηκαν σε ένα αρχείο γραμματικής που τροφοδοτείται μέσω του μεταγλωττιστή κώδικα-εκλέκτορα. Κατά συνέπεια, μεταγλωττίζει τις εφαρμογές που περιλαμβάνουν αυτόματα όλα τις προσαρμοσμένες εντολές χωρίς καμία τροποποίηση στο εγχειρίδιο. Έτσι το όφελος των πρόσφατων προστιθέμενων MOI να μπορεί να αξιολογηθούν ταχύτερα. Επιπλέον, η αξιολόγηση δεν περιορίζεται σε μεμονωμένα τμήματα κώδικα. Αντ 'αυτού, ολόκληρη η εφαρμογή εξετάζεται μια φορά η οποία οδηγεί σε πολύ πιο ακριβή αποτελέσματα

όσον αφορά την χρηστικότητα και την επιτευχθείσα ποιότητα του κώδικα. Συντομότεροι κύκλοι σχεδιασμού και χρόνος διάθεσης στην αγορά είναι οι συνέπειες αυτών. Το γενικό πρόβλημα στην επιλογή κώδικα για MOI στηρίζεται στο γεγονός, ότι σε κάθε IR κόμβο, σε τοπικό επίπεδο, η απόφαση επηρεάζει το συνολικό αποτέλεσμα του προτύπου που ταιριάζει στη διαδικασία. Οι τύποι που περιγράφονται στο εδάφιο 4.2.1 παρέχουν Cburg με ένα ισχυρό μετρικό κόστος που είναι σε θέση να αξιολογήσει την ποιότητα των τοπικών αποφάσεων για το συνολικό αποτέλεσμα με τη λήψη του αποκαλούμενου κόστους ευκαιρίας των σύνθετων κανόνων . Είναι η βάση για τη διαδικασία επιλογής του κώδικα μέσα στον Cburg, διότι μόνο από τον υπολογισμό του κόστους ευκαιρίας, είναι δυνατόν να γίνουν δίκαιες αποφάσεις που ταιριάζουν μεταξύ απλών και σύνθετων κανόνων, ο Cburg θα ενσωματωθεί περισσότερο σε μεταγλωττιστές για την επαλήθευση της επαναχρησιμοποίησης προς διαφορετικά IR. Επιπλέον, η προσπάθεια του υλικού που σχετίζεται με κάθε MOI πρέπει να εξεταστεί προκειμένου να αξιολογηθεί η επιτάχυνση που λαμβάνεται για να εκτελεστεί σωστά. Και τέλος, ένα ενδιαφέρον ανοιχτό ζήτημα είναι η αξιοποίηση του πολυτιμότερου συνόλου του MOI που μπορεί να υποστηρίξει ένα ορισμένο σύνολο εφαρμογών. Σε αυτή την κατάσταση, παρουσιαστήκαν τα σύνολα των MOI που μπορούν να μορφοποιηθούν χειροκίνητα , με βάση τα δυναμικά αποτελέσματα που επιτεύχθηκαν κατά το χρόνο εκτέλεσης. Σε αντίθεση με αυτό, η στατική ανάλυση του μεταγλωττιστή θα μπορούσε να χρησιμοποιηθεί για να βρεθεί μια λύση για τα "καλύτερα" σύνολα MOI.

Results of Benchmarks								
complex instructions	3DES (603 lines of C code)		AES (881 lines of C code)		IP Stack (3906 lines of C code)		ADPCM (743 lines of C code)	
	Speedup	Code Size	Speedup	Code Size	Speedup	Code Size	Speedup	Code Size
srlsll	+13.19%	-10.98%	+1.03%	-1.17%	+9.14%	-12.34%	+0.10%	-0.15%
srlsrl	+3.77%	-3.35%	+0.00%	-0.08%	+3.74%	-4.88%	+0.04%	-0.05%
sllsll	+7.64%	-6.58%	+0.69%	-0.64%	+5.09%	-6.70%	+0.04%	-0.05%
lwlw	+7.64%	-6.66%	+8.60%	-7.58%	+2.65%	-4.97%	+3.40%	-3.62%
lwsll	+16.96%	-13.99%	+1.03%	-1.00%	+6.22%	-8.10%	+0.29%	-0.50%
lwxor	+7.54%	-6.22%	+12.83%	-9.96%	+4.10%	-5.71%	+0.00%	-0.00%
xorsll	+7.54%	-6.22%	+0.69%	-0.86%	+3.75%	-4.90%	+0.00%	-0.00%
xorsrl	+7.54%	-6.22%	+8.25%	-6.63%	+3.72%	-4.85%	+0.00%	-0.00%
xorsll, xorsrl	+7.54%	-6.22%	+8.25%	-6.83%	+3.75%	-4.90%	+0.00%	-0.00%
lwsll, lwxor	+16.96%	-13.99%	+13.07%	-10.71%	+6.59%	-9.01%	+0.29%	-0.50%
srlsrl, sllsll	+7.54%	-9.89%	+1.03%	-0.89%	+8.83%	-12.44%	+0.08%	-0.10%
srlsrl, sllsll, srlsll	+13.29%	-11.42%	+1.38%	-1.47%	+9.34%	-12.69%	+0.14%	-0.20%
srlsrl, sllsll, srlsll, xorsll, xorsrl	+17.06%	-14.53%	+8.60%	-7.35%	+11.73%	-15.79%	+0.14%	-0.20%
srlsrl, sllsll, srlsll, lwsll, lwxor	+24.06%	-20.74%	+14.10%	-11.68%	+14.68%	-20.03%	+0.33%	-0.55%
srlsrl, sllsll, srlsll, lwlw	+20.83%	-17.88%	+13.41%	-11.60%	+11.98%	-17.66%	+2.57%	-2.23%
srlsrl, sllsll, srlsll, xorsll, xorsrl, lwsll, lwxor, lwlw, mvmv	+24.07%	-21.18%	+21.67%	-18.35%	+17.12%	-21.89%	+3.62%	-3.97%

Πίνακας 1:Επισκόπηση πειραματικών αποτελεσμάτων

ΚΕΦΑΛΑΙΟ 5:ΣΧΕΔΟΝ ΒΕΛΤΙΣΤΗ ΕΠΙΛΟΓΗ ΕΝΤΟΛΩΝ ΣΕ DAG

5.1ΠΕΡΙΛΗΨΗ

Η επιλογή εντολών αποτελεί τη βασική συνιστώσα στην δημιουργία κώδικα. Η υψηλή ποιότητα επιλογής εντολών έχει ιδιαίτερη σημασία στο χώρο όπου ένα ενσωματωμένο σύνθετο σύνολο εντολών είναι κοινό και το μέγεθος του κώδικα αποτελεί πρωταρχικό μέλημα. Αν και η επιλογή εντολών στις εκφράσεις δένδρου είναι ένα καλά κατανοητό πρόβλημα και εύκολα να λυθεί , η επιλογή εντολών σε κατευθυνόμενους μη περιοδικούς γράφους είναι NP-πλήρες. Στην εργασία αυτή παρουσιάζουμε τον αλγόριθμο NOLTIS, έναν σχεδόν βέλτιστο αλγόριθμο, σε γραμμικό χρόνο επιλογής εντολής για τις εκφράσεις DAG. Ο NOLTIS είναι εύκολος στην εφαρμογή, γρήγορος και αποτελεσματικός και με μια μέση βελτίωση κατά 5,1% του μεγέθους του κώδικα σε σύγκριση με την παραδοσιακή αποσύνθεση του δένδρου και στην προσέγγιση της πλακόστρωσης.

5.2 ΠΕΡΙΓΡΑΦΗ ΠΡΟΒΛΗΜΑΤΟΣ

Λαμβάνοντας υπόψη μια έκφραση DAG η οποία παρουσιάζει τον υπολογισμό του βασικού μπλοκ και ενός συνόλου συγκεκριμένης αρχιτεκτονικής εντολών, επιθυμούμε να βρούμε τη βέλτιστη πλακόστρωση DAG όπου αντιστοιχεί στο ελάχιστο κόστος με την χρήση ακολουθίας. Η έκφραση DAG αποτελείται από κόμβους που εκπροσωπούν τις λειτουργίες (όπως add ή load) και τελεστές (όπως σταθερά ή θέση μνήμης). Αναφερόμαστε σε έναν κόμβο με πολλαπλούς γονείς όπως ένας κοινόχρηστος κόμβος. Το σύνολο των πλακιδίων αποτελούν μια συλλογή από εκφράσεις δένδρου καθένα με ένα κόστος που του έχει ανατεθεί. Εάν ένα φύλλο της έκφρασης δένδρου δεν είναι ένας τελεστής, θεωρείται ότι οι είσοδοι για εκείνο το φύλλο-κόμβο, θα είναι διαθέσιμες από έναν καταχωρητή. Ομοίως, η έξοδος του δέντρου υποτίθεται ότι πρέπει να γραφτεί σε ένα καταχωρητή. Ένα πλακίδιο αντιστοιχεί σε ένα κόμβο στον DAG εάν η ρίζα του πλακιδίου είναι το ίδιο είδος του κόμβου καθώς ο κόμβος DAG και τα υποδένδρα του πλακιδίου ταιριάζουν αναδρομικά με τα παιδιά του κόμβου DAG. Προκειμένου η πλακόστρωση να ισχύει και να είναι πλήρης, οι είσοδοι για κάθε πλακίδιο πρέπει να είναι διαθέσιμες καθώς οι έξοδοι του κάθε πλακιδίου στην πλακόστρωση, και όλοι οι κόμβοι-ρίζα του DAG (εκείνοι οι κόμβοι με μηδενικό βαθμό) πρέπει να ταιριάξουν στα πλακίδια. Η βέλτιστη πλακόστρωση ισχύει και είναι πλήρης, όταν το άθροισμα του κόστους

των πλακιδίων ελαχιστοποιείται. Πιο τυπικά, ορίζουμε μια βέλτιστη πλακόστρωση εντολών ως εξής:

Ορισμός Έστω *K* ένα σύνολο από μια κατηγορία κόμβων ΄ *G = (V, E)* είναι ένας κατευθυνόμενος γράφος μη περιοδικός όπου κάθε κόμβος $\upsilon \in$ V έχει ένα είδος $k(\upsilon) \in K$, ένα σύνολο από παιδιά $ch(\upsilon) \in 2^{V}$ έτσι ώστε $\forall_{c \in ch(\upsilon)} (\upsilon \to c) \in E$, και μια μοναδική διάταξη για κάθε κόμβο παιδί o_{υ}: $ch(\upsilon) \to \{1,2,...|ch(\upsilon)|\}$ Τ είναι ένα σύνολο από πλακίδια δένδρου $t_i = (V_i, E_i)$, όπου ομοίως κάθε κόμβος $\upsilon_i \in V_i$ έχει ένα είδος $k(\upsilon_i) \in K \cup \{o\}$ τέτοιο ώστε $k(\upsilon_i) = o$ συνεπάγεται $out\deg ree(\upsilon_i) = 0$ (οι κόμβοι με είδος o συμβολίζουν την ακμή του πλακιδίου και αντί της αντιστοίχισης της λειτουργίας ή του τελεστή, χρησιμεύει για να συνδέσει τα πλακίδια μαζί), τα παιδιά κόμβοι $ch(\upsilon_i) \in 2^{V_i}$, η διάταξη o_{υ_i} και το κόστος: $T \to Z^+$ είναι μια συνάρτηση κόστους η οποία αποδίδει ένα κόστος για κάθε πλακίδιο δένδρου. Λέμε ένας κόμβος $\upsilon \in V$ αντιστοιχεί σε t_i δέντρο μαζί με την ρίζα $r \in V_i$ αν $k(\upsilon) = k(r), |ch(\upsilon)| = |ch(r)|$, και, για κάθε $c \in ch(\upsilon)$ και $c_i \in ch(r), o_{\upsilon}(c) = o_r(c_i)$ συνεπάγεται ότι είτε $k(c_i) = o$ ή c αντιστοιχίζει ένα δένδρο με τη ρίζα στο c_i. Για δοθείσα αντιστοίχιση του u και t_i και ένα πλακίδιο δέντρου στον κόμβο $\upsilon_i \in V_i$, ορίζουμε $m_{\upsilon,t_i} : V_i \to V$ και επιστρέφει τον κόμβο V που αντιστοιχίζεται με το υποδένδρο με ρίζα υ_i .Η αναπαράσταση $f : V \to 2^{T}$ από κάθε κόμβο DAG σε ένα σύνολο από πλακίδια δέντρου ισχύει αν και μόνο αν:

$f \forall \upsilon \in V$

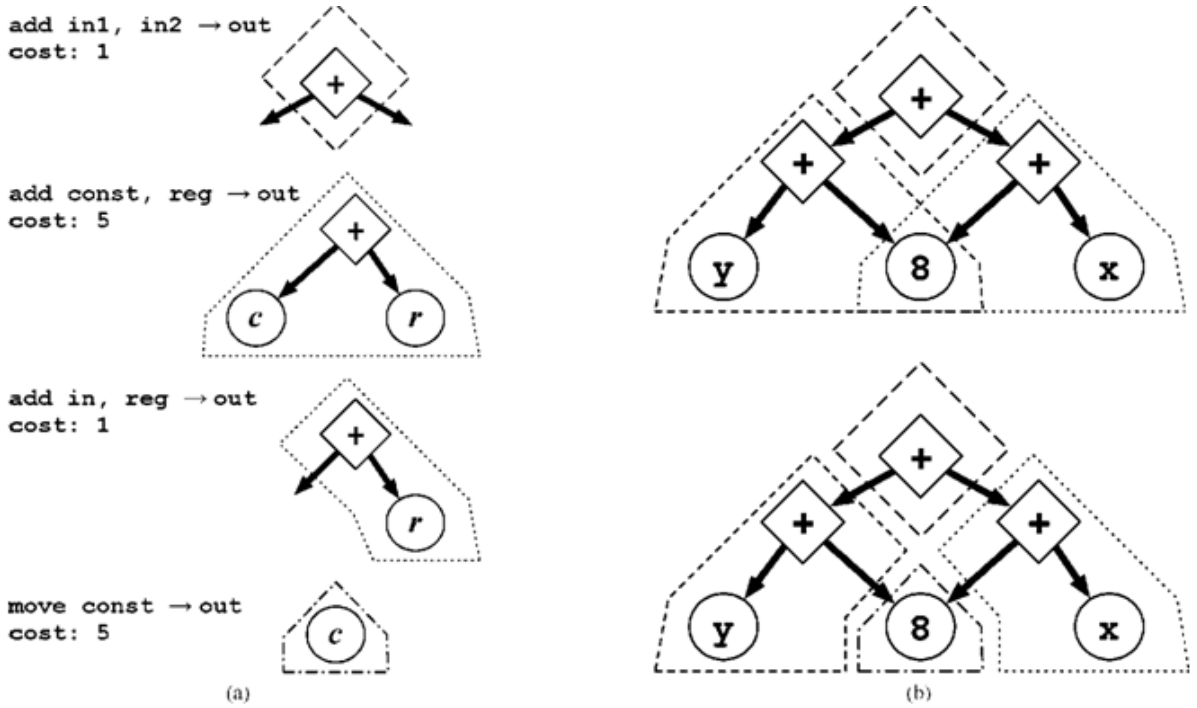
$t_i \in f(\upsilon) \Rightarrow \upsilon \ \ matches \ t_i$

$in\deg ree(\upsilon) = 0 \Rightarrow |f(\upsilon)| > 0$

$\forall t_i \, f(\upsilon), \forall \upsilon_i \in t_i, k(\upsilon_i) = o \Rightarrow |f(m_{\upsilon,t_i}(\upsilon_i))| > 0$

Μια βέλτιστη πλακόστρωση εντολών είναι μια αναπαράσταση f που ισχύει και ελαχιστοποιεί την:

$$\sum_{v \in V} \sum_{t_i \in f(v)} \cos t(t_i)$$



Σχήμα 14:Παράδειγμα επιλογής εντολής σε DAG(a)το σύνολο των πλακιδίων χρησιμοποιούνται(αντιμεταθετικά πλακίδια παραλείπονται)(b)Δυο πιθανές πλακοστρώσεις. Στο κόστος απλού μοντέλου όπου κάθε πλακίδιο έχει ένα ενιαίο κόστος στην κορυφή της πλακόστρωσης μπορεί να είναι βέλτιστη, αλλά με το κόστος του μοντέλου που παρουσιάζεται στην κατώτερη πλακόστρωση είναι βέλτιστο.

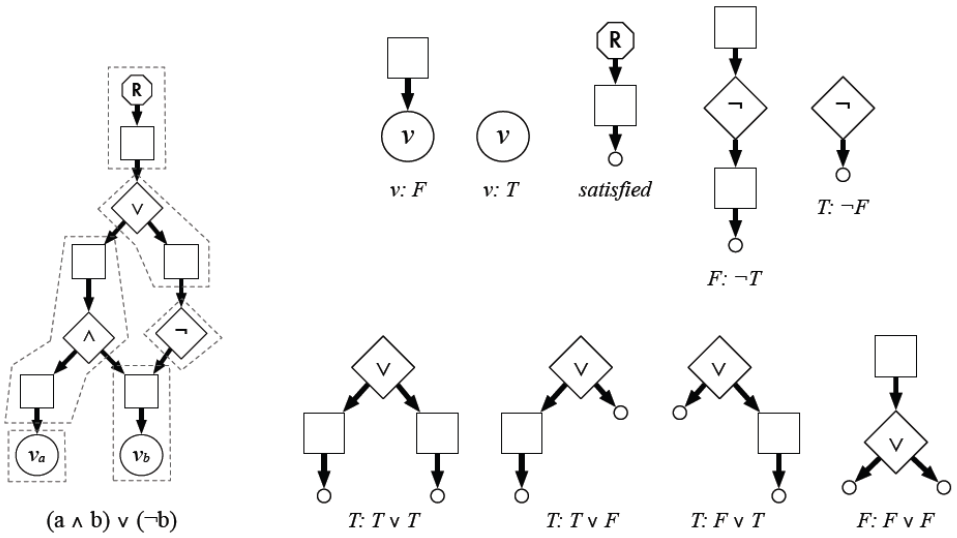
Σε ορισμένες περιπτώσεις του προβλήματος πλακόστρωσης εντολών, το όνομα της θέσης αποθήκευσης ενός πλακιδίου που γράφει ή διαβάζει είναι σημαντική. Για παράδειγμα, μερικά πλακίδια θα μπορούσαν να γράψουν στην μνήμη ή να διαβάσουν από μια συγκεκριμένη κατηγορία καταχωρητή. Στην περίπτωση αυτή, υπάρχει ένας πρόσθετος περιορισμός ότι οι είσοδοι ενός πλακιδίου πρέπει όχι μόνο να ταιριάζουν με τις εξόδους των άλλων πλακιδίων, αλλά τα ονόματα των αντίστοιχων εισόδων και εξόδων θα πρέπει επίσης να ταιριάζουν. Στην πράξη, εάν η επιλογή εντολών εκτελείται ανεξάρτητα από την κατανομή

των καταχωρητών, τα ονόματα των θέσεων αποθήκευσης είναι άνευ σημασίας. Αν και από τους προηγούμενους ελέγχους της επιλογής εντολών έχουν στηριχθεί σε επιλογή όπως η ονομασία της θέσης αποθήκευσης [Koes12] ή σε δύο διευθύνσεις εντολών [Koes13], τώρα δείχνουμε ότι ακόμη και χωρίς τους περιορισμούς αυτούς, το πρόβλημα παραμένει NP-πλήρες.

Θεώρημα 1.1. Η βέλτιστη εντολή του προβλήματος πλακόστρωσης (είναι μια βέλτιστη εντολή πλακόστρωσης του ελάχιστου κόστους έναντι k_{const}) είναι NP-πλήρες. **ΑΠΟΔΕΙΞΗ:** Σύμφωνα με [Koes12], εκτελούμε μια αναγωγή από την ικανοποίηση των εκφράσεων Boolean [Koes29]. Λαμβάνοντας υπόψη μια Boolean έκφραση που αποτελείται από τις μεταβλητές $\upsilon \in U$ και συνδέσμους Boolean $\{\vee,\wedge,\neg\}$, έχουμε κατασκευάσει ένα παράδειγμα της βέλτιστης χρήσης του προβλήματος της πλακόστρωσης ως εξής: Έστω το σύνολο των ειδών κόμβων K είναι $\{\vee,\wedge,\neg,$, □, $\Re,\upsilon\}$. Αναφερόμαστε στους κόμβους με το είδος □ ως κόμβοι box .Για κάθε μεταβλητή $u \in U$, να δημιουργούμε δύο κόμβους u_1 και u_2 και μια κατευθυνόμενη ακμή $(u_1 \to u_2)$ στο G έτσι ώστε $k(u_1) =$ □ και $k(u_2) = \upsilon$ κατά τον ίδιο τρόπο, για κάθε τελεστή Boolean op δημιουργούμε δύο κόμβους op_1 και op_2 και μια κατευθυνόμενη ακμή $(op_1 \to op_2)$ έτσι ώστε $k(op_1) =$ □ και $k(op_2)$ είναι αντίστοιχα οι λειτουργίες. Στην συνέχεια, για κάθε λειτουργία $a \ \ op \ b$ δημιουργούνται ακμές $(op_2 \to a_1)$ και , όπου $k(a_1) = k(b) =$ □ (στην περίπτωση του μοναδιαίου συμβόλου $\neg$ δημιουργείται μια χωριστή ακμή).Προσοχή, η διάταξη των κόμβων παιδιού είναι άσχετοι, όταν οι τελεστές Boolean αντιμετατίθενται. Τελικά, δημιουργείται ένας κόμβος r και μια ακμή $(r \to op)$ έτσι ώστε $k(r) = R$ και op είναι η λειτουργία της ρίζας της έκφρασης. Ένα παράδειγμα τέτοιου DAG, φαίνεται στο Σχήμα 15 (a). Σημειώνουμε ότι μόνο οι κόμβοι με ενδεχομένως περισσότερα του ενός γονέα σε αυτό το DAG είναι οι κόμβοι που αντιστοιχούν σε box μεταβλητές.

Τώρα, το σύνολο των πλακιδίων του δέντρου T ας είναι όπως στο Σχήμα 2 (β) όπου κάθε πλακίδιο περιλαμβάνει ένα μοναδικό μη-box κόμβο και έχει μοναδιαίο κόστος. Αυτά τα πλακίδια σχεδιάζονται έτσι ώστε να μπορεί να αποδειχθεί ότι η ανάθεση της έκφρασης Boolean που αντιστοιχίζεται με μια σε ισχύ πλακόστρωση όπως περιγράφεται παραπάνω. Αν η μεταβλητή είναι αληθής, τότε ο αντίστοιχος κόμβος μεταφέρεται μαζί με το πλακίδιο

κατά $u:T$, διαφορετικά μεταφέρεται με $u:F$. Το υπόλοιπο της πλακόστρωσης δημιουργείται με φυσικό τρόπο που προτείνεται από το όνομα του πλακιδίου στο Σχήμα 15 (b). Αυτή η πλακόστρωση είναι η βέλτιστη από κάθε κόμβο-φύλλο του DAG θα έχουμε ακριβώς ένα πλακίδιο που μεταφέρεται(αντίστοιχης των τιμών αληθείας αυτών των μεταβλητών) και από τους γονείς των κόμβων-φύλλων μόνο οι αλυσιδωτοί κόμβοι στοDAG(που μπορούν να έχουν πολλαπλούς γονείς), κανένας άλλος μη-box κόμβος στο DAG μπορεί να μεταφερθεί από περισσότερα από ένα πλακίδια στην πλακόστρωση. Ως εκ τούτου, το κόστος της πλακόστρωσης είναι ίσο με τον αριθμό των μη-box κόμβων και είναι βέλτιστο. Λαμβάνοντας υπόψη τη βέλτιστη πλακόστρωση του DAG, που προέρχεται από μια λογική (Boolean)έκφραση, αν το κόστος της πλακόστρωσης είναι ίσο με τον αριθμό των μη-box κόμβων, τότε μπορούμε να κατασκευάσουμε εύκολα την απόδοση τιμών αλήθειας που ικανοποιούν την έκφραση παρατηρώντας τα πλακίδια που χρησιμοποιούνται για να καλύψουν τα φύλλα του DAG. Αν το κόστος της πλακόστρωσης είναι μεγαλύτερο από τον αριθμό των μη-box κόμβων τότε οι εκφράσεις δεν ικανοποιούνται. Αν συνέβαινε αυτό, μια φθηνότερη πλακόστρωση θα πρέπει να υπάρχει .Έχουμε δείξει ότι η ικανοποιησιμότητα Boolean μειώνει τη βέλτιστη εντολή πλακόστρωσης του προβλήματος, και, ως εκ τούτου, η βέλτιστη εντολή του προβλήματος της πλακόστρωσης είναι NP-πλήρες.



(a) (b)

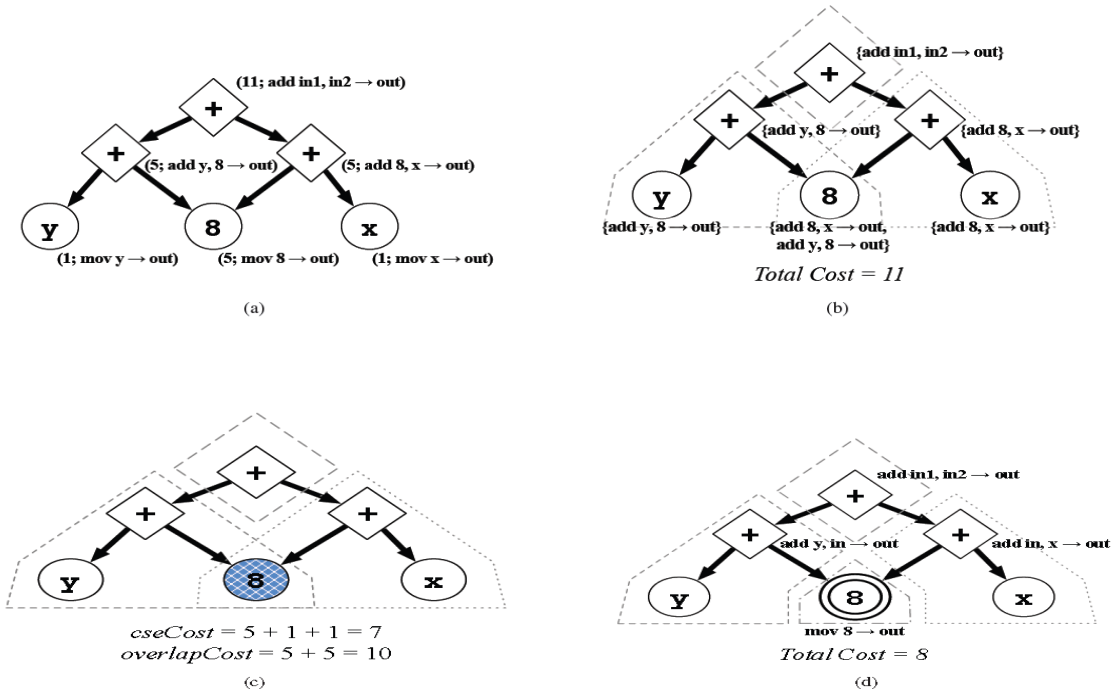
Σχήμα 15: (a)Ένα παράδειγμα της έκφρασης DAG που συμβολίζει μια έκφραση Boolean(b)τα πλακίδια χρησιμοποιούνται για την κάλυψη μιας τέτοιας έκφρασης DAG.Κάθε πλακίδιο έχει μοναδιαίο κόστος. Η εκπροσώπηση των πλακιδίων $\wedge$ παραλείπεται, αλλά είναι παρόμοια με τα πλακίδια $\vee$ με δύο μέσα πλακίδια να έχουν ένα πρόσθετο κουτί κόμβου στη ρίζα

5.3 NOLTIS

Ο NP πλήρες χαρακτήρας του προβλήματος της βέλτιστης πλακόστρωσης εντολών απαιτεί τη χρήση ευρετικών κατά την εκτέλεση της επιλογής εντολής. Μια κοινή προσέγγιση είναι πρώτα να αποσυντίθεται ο DAG σε ένα πλήθος από δένδρα και στη συνέχεια να χρησιμοποιεί έναν βέλτιστο αλγόριθμο πλακόστρωσης για να πλακοστρώσει κάθε δένδρο. Κάθε κοινή υποέκφραση του DAG, ως εκ τούτου είναι στη ρίζα ενός δένδρου στο πλήθος. Ωστόσο, όπως θα δείξουμε στην υποενότητα 7, η προσέγγιση αυτή δεν είναι τόσο επιτυχής όσο οι αλγόριθμοι οι οποίοι δουλεύουν κατευθείαν για τον DAG. Για παράδειγμα, αν όλα τα πλακίδια στο σχήμα 14 ανατέθηκαν με μοναδιαίο κόστος, η λύση αποσύνθεσης του δέντρου είναι υποβέλτιστη. Στην ενότητα αυτή παρουσιάζουμε τον NOLTIS, έναν αλγόριθμο γραμμικού χρόνου που εξασφαλίζει σχεδόν βέλτιστη πλακόστρωση των DAG εκφράσεων. Ο αλγόριθμος εφαρμόζεται στην πλακόστρωση δένδρου απευθείας στον DAG, χωρίς προηγουμένως την εκτέλεση αποσύνθεσης του δέντρου, χρησιμοποιώντας την πλακόστρωση να αποφασίσει σε ποια τμήματα του DAG, μπορούν να αναλυθούν παραγωγικά σε δένδρα, και στη συνέχεια να πλακοστρώσουν ξανά μερικώς την αποσύνθεση DAG.Πρώτα εφαρμόζουμε δυναμικό προγραμματισμό για τον DAG, αγνοώντας την παρουσία των αλυσιδωτών κόμβων χρησιμοποιώντας τη διαδικασία από BOTTOMUPDP από την Καταχώριση 1.Συνεπώς,

πλακοστρώνουμε το δένδρο που θα σχηματιζόταν, αν κάθε αλυσιδωτός κόμβος (και τους απογόνους του) διπλασιάστηκε για να μετατρέψει το DAG σε δυνητικά εκθετικά μεγαλύτερο δένδρο. Ωστόσο, ο αλγόριθμος παραμένει γραμμικός αφού κάθε κόμβος τον επισκέπτεται μόνο μια φορά. Ο δυναμικός προγραμματισμός έχει επισημανθεί μια φορά σε κάθε κόμβο με το καλύτερο πλακίδιο για εκείνον τον κόμβο, ένα πέρασμα πάνω προς τα κάτω, TOPDOWNSELECT , δημιουργεί μια πλακόστρωση DAG. Η ύπαρξη των αλυσιδωτών κόμβων μπορεί να οδηγήσει σε μια πλακόστρωση όπου οι κόμβοι καλύπτονται από πολλαπλά πλακίδια (δηλαδή, τα πλακίδια επικάλυψης). Ωστόσο, δεδομένου ότι δεν υπάρχει κανένας κόμβος θα είναι στη ρίζα των δύο πλακιδίων (αυτό θα σήμαινε ότι η ίδια ακριβώς τιμή θα υπολογίζεται δύο φορές), ο αριθμός των πλακιδίων στην πλακόστρωση είναι ανάλογη με τον αριθμό των κόμβων. Κατά συνέπεια, το πάνω προς τα κάτω πέρασμα, που διασχίζει τα πλακίδια, έχει γραμμική πολυπλοκότητα. Η πλακόστρωση που διαπιστώθηκε από το πρώτο πέρασμα των πλακιδίων αγνοεί τις επιπτώσεις των αλυσιδωτών κόμβων του DAG και ως εκ τούτου μπορεί να έχει ακριβή ποσότητα αλληλεπικαλυπτόμενων κόμβων. Στο επόμενο βήμα του αλγορίθμου, προσδιορίζουμε τους κοινούς κόμβους όπου αφαιρώντας τους τοπικά αλληλεπικαλυπτόμενους βελτιώνει την συνολική πλακόστρωση. Αυτοί οι κόμβοι προστίθενται στο σύνολο των fixedNodes. Τότε εκτελούμε ένα άλλο πέρασμα στην πλακόστρωση. Σε αυτό το πέρασμα, τα πλακίδια είναι απαγορευμένα από τους συνδεδεμένους κόμβους που στο σύνολο fixedNodes · αυτοί οι κόμβοι πρέπει να αντιστοιχηθούν στη ρίζα του πλακιδίου. Η διαδικασία IMPROVECSEDECISIONS (Καταχώριση 2) χρησιμοποιείται για να προσδιορίσει αν ο κοινός κόμβος πρέπει να είναι σταθερός. Για κάθε κοινό κόμβο n υπολογίζουμε το κόστος της αλληλοεπικάλυψης χρησιμοποιώντας την συνάρτηση GETOVERLAPCOST στην Καταχώριση 3. Η συνάρτηση αυτή υπολογίζει το κόστος της περιοχής τοπικά αλληλοεπικάλυψης στο n. Παρατηρούμε ότι, σε σπάνια περίπτωση σε εκείνη την περιοχή της επικάλυψης συναθροίζει άλλο αλυσιδωτό κόμβο · είναι πιθανόν ότι η IMPROVECSEDECISIONS θα έχει υπέρ-γραμμική πολυπλοκότητα ωστόσο, αυτό μπορεί να αντιμετωπιστεί με τη χρήση των Memoization, μια λεπτομέρεια η οποία δεν φαίνεται στον ψευτοκώδικα. Το επόμενο βήμα είναι να υπολογίσουμε το κόστος που θα προέκυπτε αν τα πλακίδια n που καλύπτουν κόπηκαν, έτσι ώστε μόνο ένα χωριστό πλακίδιο, ριζωμένο στο n,μεταφέρθηκε στο n. Το κόστος του ριζωμένου πλακιδίου δένδρου στο n μπορεί να προσδιοριστεί από τα αποτελέσματα του δυναμικού προγραμματισμού. Σε αυτό το κόστος να προσθέτουμε τα κόστη των τομών των πλακιδίων την στιγμή μεταφοράς του n, όπου υπολογίζεται χρησιμοποιώντας την συνάρτηση GETTILECUTCOST όπως φαίνεται στην

Καταχώριση 4. Κατά τον προσδιορισμό του κόστους της τομής το πλακίδιο t με τη ρίζα r ρίζα στο κόμβου n, θεωρούμε ότι κάθε πλακίδιο το οποίο επίσης αντιστοιχίζει στο r και έχει N ως έναν κόμβο των ακμής. Στη συνέχεια υπολογίζουμε τη διαφορά μεταξύ του κόστους σε αυτό το πλακίδιο για να αντιστοιχηθεί με r και χρησιμοποιώντας t. Επιλέγουμε το ελάχιστο κόστος της διαφοράς ως το κόστος της τομής του πλακιδίου. Αν το κόστος της τρέχουσας επικάλυψης της πλακόστρωσης είναι παραπάνω του κόστους της αφαίρεσης των επικαλύψεων και της τομής των πλακιδίων, τότε έχουμε βρει ένα τοπικό μετασχηματισμό που βελτιώνει την υφιστάμενη πλακόστρωση. Αντί της άμεσης εφαρμογής του μετασχηματισμού αυτού, επιλέγουμε να καθορίσουμε τον κόμβο n, απενεργοποιώντας την επικάλυψη ενώ υπολογίζουμε μια νέα πλακόστρωση. Αυτό οδηγεί σε μια δυνητικά καλύτερη λύση με την νέα πλακόστρωση να μην χρειάζεται να περιορίζεται σε ριζωμένα πλακίδια στο r. Στο σχήμα 3 παρουσιάζεται η εκτέλεση του αλγορίθμου NOLTIS για το παράδειγμα από το Σχήμα 1.Ο αλγόριθμος NOLTIS δεν είναι ο βέλτιστος, δεδομένου ότι εξαρτάται από διάφορες υποθέσεις τις οποίες δεν τις κατέχουν απαραιτήτως. Υποθέτουμε ότι πάντα είναι πιθανόν να τέμνουν τα πλακίδια σε ένα κοινό κόμβο χωρίς να επηρεάζουν την ικανότητα των πλακιδίων του DAG. Υποθέτουμε ότι είναι η καλύτερη θέση για να τέμνει τα πλακίδια για να εξαλειφθούν οι επικαλύψεις που είναι σε κοινό κόμβο. Αναλαμβάνουμε την απόφαση να καθορίσουμε ένα κοινό κόμβο που μπορεί να γίνει ανεξάρτητος από τους άλλους κοινούς κόμβους. Όταν αποφασίσουμε να καθορίσουμε ένα κοινό κόμβο υποθέτουμε ότι μπορεί να αντιπροσωπεύσει τις επιπτώσεις για τον καθορισμό του κόμβου από την εξέταση των τομών των απλών κόμβων. Παρά αυτές τις υποθέσεις, στην πράξη ο αλγόριθμος NOLTIS επιτυγχάνει σχεδόν βέλτιστα αποτελέσματα.



Σχήμα 16:Η εφαρμογή του αλγόριθμου NOLTIS του παραδείγματος από το Σχήμα 1.(a)BOTTOMPUPDP υπολογίζει τη λύση του δυναμικού προγραμματισμού για DAG με αρχικοποίηση *bestChoiceForNode* **(b)TOPDOWNSELECT προσδιορίζει την πλακόστρωση από την λύση του δυναμικού προγραμματισμού, με αρχικοποίηση** COV *eringTiles* **(c)IMPROVECSEDECISIONS αξιολογείται ο αλυσιδωτός κόμβος και προσδιορίζει αυτό που θα έπρεπε να είναι σταθερό(d)Ο δυναμικός προγραμματισμός μετά υπολογίζεται ξανά με την αξίωση ο σταθερός κόμβος να μην επικαλύπτεται και η βέλτιστη λύση να βρίσκεται**

5.4 ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΗ ΕΠΙΛΥΣΗ

Προκειμένου να καθοριστεί ο σχεδόν βέλτιστος αλγόριθμος μας, διατυπώνουμε το πρόβλημα της πλακόστρωσης ως ένα 0-1ακέραιο πρόγραμμα που μπορεί να λυθεί με τη βέλτιστη χρήση ενός εμπορικού λύτη. Η διατύπωση του προβλήματος είναι απλή. Για κάθε κόμβο i και πλακίδιο j έχουμε δυαδική μεταβλητή $M_{i,j}$ το οποίο είναι ένα πλακίδιο j αντιστοιχίζει τον κόμβο i (η ρίζα του πλακιδίου j στον κόμβο i) στην πλακόστρωση, διαφορετικά μηδέν. Το $\cos t_j$ είναι το κόστος των πλακιδίων i, οι $roots$ είναι οι ρίζες των κόμβων του DAG, και $edgeNodes(i, j)$ είναι οι κόμβοι στις ακμές των πλακιδίων j όταν είναι ριζωμένες στον κόμβο i , τότε το βέλτιστο πρόβλημα πλακόστρωσης των εντολών είναι:

$$\min \sum_{i,j} \cos t_j M_{i,j}$$

Υπόκεινται σε

$$\forall i \in roots \sum_j Mi, j >= 1 \quad (1)$$

$$\forall i, j \forall i' \in edgeNodes(i, j) Mi, j - \sum_{j'} Mi', j' \le 0 \quad (2)$$

όπου η (1) προβλέπει ότι η ρίζα των κόμβων του DAG ταιριάζουν με τα πλακίδια και η (2) απαιτεί ότι αν ένα πλακίδιο αντιστοιχεί σε ένα κόμβο, τότε όλες οι είσοδοι προς το πλακίδιο πρέπει να ταιριάζουν με τα πλακίδια.

Καταχώριση 1:Δυναμικός προγραμματισμός επιλογής εντολών
με μετασχηματισμούς για σχεδόν βέλτιστη επιλογή DAG

1:DAG: έκφραση DAG

2: $bestChoiceForNode : Node \to (Tile \times \text{int})$

3: $fixedNodes$: σύνολο Κόμβων

4: $matchedTiles$: σύνολο Πλακιδίων

5: $\text{cov}eringTiles$: Κόμβος → σύνολο Πλακιδίων

6:**διαδικασία** SELECT

7: $fixedNodes \leftarrow \{\ \}$

8: BOTTOMUPDP () αρχικοποιεί

9: TOPDOWNSELECT αρχικοποιεί $\text{cov}eringTiles$

10:IMPROVECSEDECISIONS() αρχικοποιεί $fixedNodes$

11:BOTTOMUPDP() χρησιμοποιεί $fixedNodes$

12:TOPDOWNSELECT() βάζει την τελική πλακόστρωση σε $matchedTiles$

13: **διαδικασία** BOTTOMUPDP

14: $for\ \ n \in reverseTopological(DAG)\ \ do$

15: $bestChoiceForNode[n].cost \leftarrow \infty$

16: $for\ \ t_n \in matchingTiles(n)\ \ do$

17: $if \neg hasInteriorFixedNode(t_n, fixedNodes)\ \ then$

18: $val \leftarrow cost(t) + \sum_{n' \in edgeNodes(t_n)} bestChoiseForNode[n'].cost$

19: $if\ \ val < bestChoiceForNode[n].cost\ \ then$

20: $bestChoiceForNode[n].cost \leftarrow val$

21: $bestChoiceForNode[n].tile \leftarrow t_n$

22: **διαδικασία** TOPDOWNSELECT

23 **:** $matchedTiles.clear()$

24: $coveringTiles.clear$

25: $q.push(root(DAG))$

26: $while \neg q.empty()\ \ do$

27: $n \leftarrow q.pop()$

28: $bestTile \leftarrow bestChoiceForNode[n].tile$

29: $matchedTiles.add(bestTile)$

30: $for\ \ every\ \ node\ \ n_t\ \ covered\ \ by\ \ bestTile\ \ do$

31: $coveringTiles[n_t].add(bestTile)$

32: $for\ \ n' \in edgeNodes(bestTile)\ \ do$

33: $q.push(n')$

Καταχώριση 2:Δοθείσας μιας αντιστοίχισης DAG αγνοείται η επίδραση του κοινού κόμβου, αποφασίζουμε αν η λύση μπορεί να βελτιωθεί από την έλξη των κοινών κόμβων στις κοινές υποεκφράσεις (εξαλείφοντας την επικάλυψη των πλακιδίων)

1:**διαδικασία** IMPROVECSEDECISIONS

2: $for\ \ n \in sharedNodes(DAG)\ \ do$

3: $if\ \ coveringTiles[n].size > 1\ \ then$ ▷ έχει επικάλυψη

4: $overlapCost \leftarrow getOverlapCost(n, \text{cov}eringTiles)$

5: $cseCost \leftarrow bestChoiceForNode[n].\text{cos}t$

6: *for* $t_n \in \text{cov}eringTiles[n]$ *do*

7: $cseCost \leftarrow cseCost + getTileCutCost(t_n, n)$

8: *if* $cseCost < overlapCost$ *then*

9: $fixedNodes.add(n)$

Καταχώριση 3:Δοθεντος κοινού κόμβου με επικαλυπτόμενα πλακίδια, το κόστος των πλακιδίων του δέντρου που είναι στη ρίζα στα επικαλυπτόμενα πλακίδια n χωρίς διπλό υπολογισμό των περιοχών που δεν επικαλύπτουν

1:**function** GETOVERLAPCOST(n)

2: $\text{cos}t \leftarrow 0$

3: $seen \leftarrow \{\ \}$

4: *for* $t \in \text{cov}eringTiles[n]$ *do*

5: $q.push(t)$

6: $seen.add(t)$

7: *while*$\neg q.empty()$*do*

8: $t \leftarrow q.pop()$

9: $\text{cos}t \leftarrow \text{cos}t + \text{cos}t(tile)$

10: *for* $n' \in edgeNodes(t)$*do*

11: *if* n' *is reachable from* n *then*

12: $t' \leftarrow bestChoiceForNode[t'].tile$

13 : *if* $\text{cov}eringTiles[n'].size() = 1$ *then*

14: $\text{cos}t \leftarrow \text{cos}t + bestChoiceForNode[n'].\text{cos}t$

15: *else if* $t' \notin seen$ *then*

16: $seen.add(t')$

17: $q.push(t')$

18: *return* $\text{cos}t$

5.5 ΕΚΤΕΛΕΣΗ

Έχουμε να εκτελέσουμε τον αλγόριθμο μας σε μεταγλωττιστή LLVM 2,1 [Koes30] υποδομή που στοχεύει την αρχιτεκτονική Intel x86 για το Ubuntu 7.10 με λειτουργικό σύστημα Linux. Η προεπιλογή του επιλογέα εντολών LLVM κατασκευάζει μια έκφραση DAG που έχει ως στόχο τους ανεξάρτητους κόμβους και στη συνέχεια, εκτελεί έναν μέγιστο αλγόριθμο [Koes31]. Τα πλακίδια επιλέγονται από την κορυφή προς τα κάτω. Το μεγαλύτερο πλακίδιο (το πλακίδιο εκείνο που μεταφέρει τους πιο πολλούς κόμβους) επιλέγεται άπληστα. Το κόστος των πλακιδίων χρησιμοποιείται μόνο για να σπάσει τους δεσμούς. Έχουμε τροποποιήσει τον αλγόριθμο LLVM για να χρησιμοποιήσουμε το μέγεθος του κώδικα, όταν σπάσουμε τους δεσμούς. Επιπλέον για την προεπιλογή των αλγόριθμων LLVM και DAG , έχουμε εφαρμόσει άλλους τρεις αλγόριθμους:

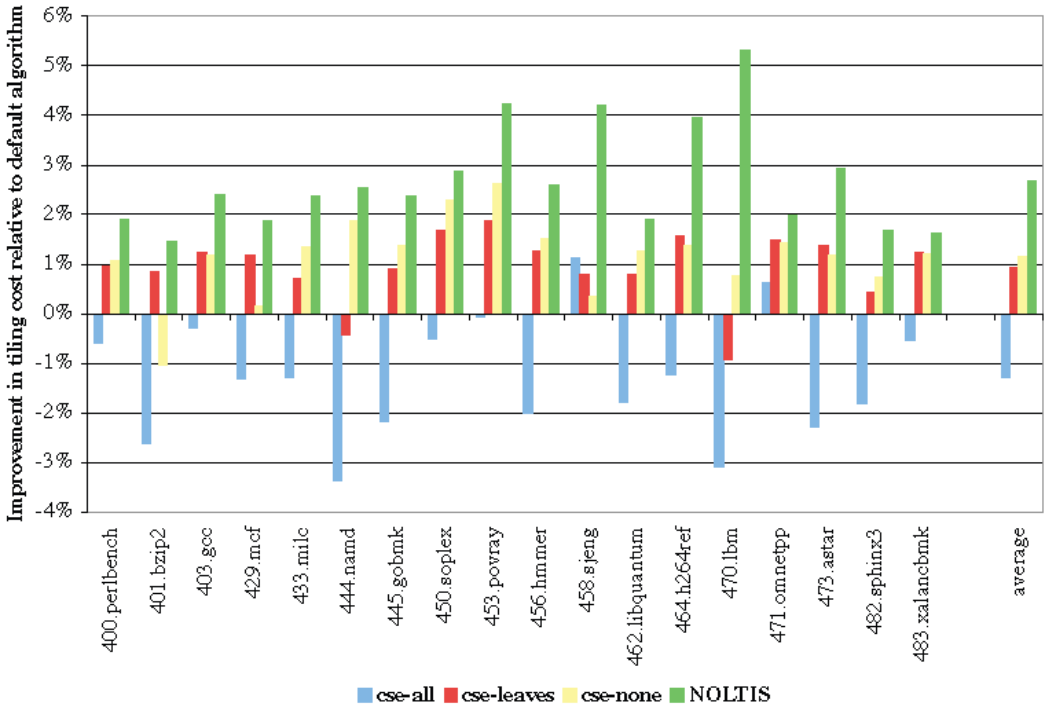
$cse-all$: Η έκφραση DAG αποσυντίθεται πλήρως σε δένδρα και ο δυναμικός προγραμματισμός εκτελείται σε κάθε δένδρο. Δηλαδή, ο κάθε κοινός κόμβος είναι σταθερός. Αυτή είναι η συμβατική μέθοδος για την εφαρμογή της πλακόστρωσης δέντρου ένα σε DAG [Koes15].

$cse-leaves$: Η έκφραση DAG αποσυντίθεται μερικώς σε δένδρα και εκτελείται σε δυναμικό προγραμματισμό γίνεται. Αν ο υπο-γράφος που είναι στη ρίζα σε κοινό κόμβος μπορεί να καλυφθεί πλήρως από ένα απλό πλακίδιο, ο κοινός κόμβος παραμένει αμετάβλητος, αλλιώς οι κοινοί κόμβοι καθίστανται ρίζες των δένδρων για να πλακοστρωθούν. Δηλαδή, οι κοινοί κόμβοι είναι σταθεροί, εκτός αν αποτελούν μια έκφραση που μπορεί να μεταφέρεται πλήρως από ένα απλό πλακίδιο.

$cse-none$: Η έκφραση DAG δεν αποσυντίθεται σε δένδρα και ο δυναμικός προγραμματισμός εκτελείται. Δηλαδή, οι μη κοινοί κόμβοι είναι σταθεροί (αυτό είναι ισοδύναμο με τη λύση που βρέθηκε πριν από τη διαδικασία IMPROVECSEDECISIONS εκτελείται στον αλγόριθμο NOLTIS).

Όλοι οι αλγόριθμοι χρησιμοποιούν το ίδιο σύνολο πλακιδίων. Το κόστος του κάθε πλακιδίου είναι το μέγεθος σε bytes των αντίστοιχων εντολών x86. Δεν επιτρέπουν τα πλακίδια να επικαλύπτουν τους τελεστές μνήμης (δηλαδή, ένα φορτίο ή η αποθήκευση κόμβων στην έκφραση DAG θα εκτελεστεί μόνο μία φορά). Ομοίως, καθώς η εκτέλεση ορίζεται, η επικάλυψη των συναρτήσεων δεν επιτρέπει να καλούνται οι διευθύνσεις. Άνευ αξίας έλαβαν οι ακμές, οι διατάξεις επιβάλλουν εξαρτήσεις στην έκφραση DAG. Ανεξάρτητα από τη φύση των δύο διευθύνσεων της αρχιτεκτονικής x86, όλα τα πλακίδια παρουσιάζουν εντολές τριών διευθύνσεων. Το πέρασμα μετά την επιλογή εντολή

μετατρέπει τον κώδικα από DAG σε μορφή καταχώρισης assembly, επιχειρεί να ελαχιστοποιήσει την πίεση του καταχωρητή του προγράμματος χρησιμοποιώντας αρίθμηση Sethi-Ullman[Koes32].



Σχήμα 17:Η βελτίωση του κόστους της πλακόστρωσης σε σχέση με τον προεπιλεγμένο αλγόριθμο αντιστοίχησης για ατομικά SPEC CPU2006 σημεία αναφοράς. Η μέση βελτίωση των $cse-all, cse-leaves, cse-none$ και του αλγόριθμου NOLTIS είναι -1,28%,0,94%,1,15% και 2,68% αντίστοιχα.

5.6 ΑΠΟΤΕΛΕΣΜΑΤΑ

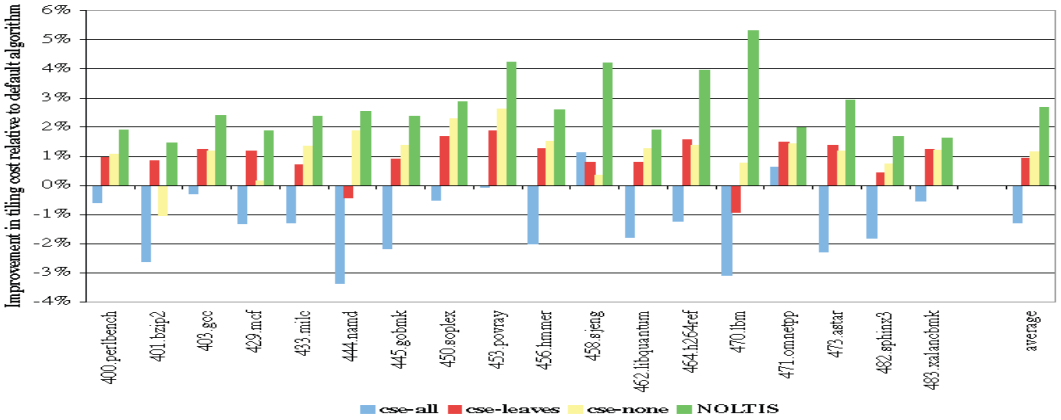
Αξιολογούμε τους διάφορους αλγόριθμους επιλογής εντολών από τη σύνταξη της C και C + + σημεία αναφοράς του SPEC CPU2006 [Koes33], MediaBench [Koes34], MiBench [Koes35], και VersaBench [Koes36] σουίτες αναφοράς και παρατηρώντας μαζί με τους πλησιέστερους το κόστος της θέσης-επιλογής της πλακόστρωση και το τελικό μέγεθος του κώδικα των σημείων αναφοράς. Αξιολογούμε το βέλτιστο του αλγορίθμου NOLTIS, αποδεικνύει την υπεροχή του σε σχέση με το υφιστάμενο ευρετικό, να εξετάσει τις επιπτώσεις στο μέγεθος του κώδικα του πλήρους μεταγλωττισμένου κώδικα, και περιγράφει την συμπεριφορά του στο χρόνο μεταγλώττισης.

5.6.1 ΒΕΛΤΙΣΤΗ

Προκειμένου να προσδιοριστεί μια βέλτιστη λύση για μια έκφραση του DAG ,δημιουργούμε ένα ακέραιο 0-1 πρόβλημα προγραμματισμού, όπως περιγράφεται στην ενότητα 5 και στη συνέχεια καθώς το επιλύουμε χρησιμοποιώντας ILOG CPLEX 10.0 [Koes37]. Αξιολογούμε όλα τα βασικά τμήματα των SPEC CPU2006 σημείων αναφοράς, με αποτέλεσμα σε σχεδόν μισό εκατομμύριο προβλήματα πλακόστρωσης. Χρησιμοποιήσαμε μια συστάδα από Pentium 4 μηχανήματα που κυμαίνονται σε ταχύτητα από 2.8GHz έως 3.0GHz για την επίλυση των προβλημάτων. Ο CPLEX μπορεί να βρει μια αποδεκτή βέλτιστη λύση σε 15 λεπτά για το 99,8% των προβλημάτων πλακόστρωσης. Από τα προβλήματα με αποδεδειγμένη βέλτιστη λύση, ο αλγόριθμος NOLTIS επιτυχώς βρήκε μια βέλτιστη λύση 99,7% του χρόνου. Επιπλέον, υποβέλτιστες λύσεις ήταν συνήθως πολύ κοντά στο βέλτιστο (μόνο μερικά bytes μεγαλύτερο). Το 0,2% των προβλημάτων όπου ο CPLEX δεν βρίσκει μια επαρκή βέλτιστη λύση , ο αλγόριθμος NOLTIS βρίσκει μια λύση τόσο καλή όσο, και σε ορισμένες περιπτώσεις καλύτερα από, την καλύτερη λύση που βρέθηκε από τον CPLEX 75% του χρόνου υπονοώντας ότι ο αλγόριθμος μας είναι αποτελεσματικός ακόμα και για πολύ δύσκολα προβλήματα πλακόστρωσης. Η συνολική βελτίωση που λαμβάνεται με τη χρήση της καλύτερης CPLEX λύσης έναντι της χρήσης του αλγόριθμο NOLTIS ήταν αμελητέα κατά 0,05%. Θεωρούμε ότι αυτά τα αποτελέσματα καταδεικνύουν σαφώς ότι ο NOLTIS αλγόριθμος είναι, σε αυτην την περίπτωση, σχεδόν βέλτιστος.

5.6.2 ΣΥΓΚΡΙΣΗ ΑΛΓΟΡΙΘΜΩΝ

Εκτός του ότι είναι σχεδόν βέλτιστος, ο αλγόριθμος NOLTIS προσφέρει σημαντικά καλύτερες λύσεις στο πρόβλημα πλακόστρωσης από ότι οι κλασικοί ευρετικοί. Τα αναλυτικά αποτελέσματα για τα σημεία αναφοράς του SPEC CPU2006 φαίνονται στο Σχήμα 17 και ο μέσος όρος στις βελτιώσεις φαίνεται στο Σχήμα 18. Ο αλγόριθμος $sce-all$, παρά την εύρεση μια βέλτιστης πλακόστρωσης για κάθε δέντρο στην πλήρη αποσύνδεση του δέντρου του DAG, εκτελεί ανεπαρκώς σε σχέση με τους άλλους αλγόριθμους γεγονός που υποδηλώνει ότι ο αλγόριθμος πλακόστρωσης του DAG είναι απαραίτητος για τη μέγιστη ποιότητα του κώδικα. Το όφελος και των δυο αλγορίθμων $sce-leaves$ και $sce-none$ από τη χρήση του δυναμικού προγραμματισμού και υπερτερούν του άπληστου αλγόριθμου, αν και δεν είναι σαφώς ανώτερος από τους άλλους. Ο αλγόριθμος NOLTIS, όπως αναμένετε, ξεπερνά σημαντικά τους άλλους αλγορίθμους και έχει την καλύτερη πρόοδο σε κάθε σημείο αναφοράς.



Σχήμα 18:Η βελτίωση στο τελικό μέγεθος κώδικα σε σχέση με το μέγιστο αλγόριθμο αντιστοίχησης για τοπικά SPEC CPU2006 σημεία αναφοράς. Η μέση βελτίωση των $cse-all$, $cse-leaves$, $cse-none$ και του αλγόριθμου NOLTIS είναι -4,03%,0,09%,0,81% και 1,20% αντίστοιχα. Η μέση βελτίωση του αλγόριθμου NOLTIS σε σχέση με το πιο παραδοσιακό $cse-all$ αλγόριθμο είναι 4,42%

5.6.3 ΕΠΙΔΡΑΣΗ ΣΤΟ ΜΕΓΕΘΟΣ ΚΩΔΙΚΑ

Η πλακόστρωση εντολών είναι μια συνιστώσα του γεννήτορα κώδικα. Το πέρασμα της μετατροπής των δύο διευθύνσεων, το πέρασμα στην χρονοδρομολόγηση , και το πέρασμα στον καταχωρητή αποθήκευσης με όλες τις περαιτέρω επιπτώσεις στην τελική ποιότητα του κώδικα που μεταγλωττίζεται. Τα αποτελέσματα στο μέγεθος του τελικού κώδικα για σημεία αναφοράς SPECPU2006 παρουσιάζονται στο Σχήμα 19 και ο μέσος όρος των βελτιώσεων φαίνονται στο Σχήμα 20. Παρόλο που οι μέσες βελτιώσεις στο μέγεθος του κώδικα που επιδεικνύει ο αλγόριθμος NOLTIS μπορεί να φαίνονται οριακά, έστω και φαινομενικά μικρές μειώσεις στο μέγεθος του κώδικα μπορεί να είναι σημαντικές, όταν έχουν ως στόχο τις ενσωματωμένες αρχιτεκτονικές. Επιπλέον, είναι σημαντικό να σημειωθεί ότι τα αποτελέσματα αυτά είναι σε σχέση με έναν αλγόριθμο που έχει ήδη προσαρμοστεί να δουλεύει απευθείας στις εκφράσεις DAG. Σε σύγκριση με την κλασική προσέγγιση του εγχειριδίου για την αποσύνδεση του δέντρου (αλγόριθμος $sce-all$), ο αλγόριθμος NOLTIS παρουσιάζει μια συνολική μέση βελτίωση του μεγέθους κώδικα του 5,1%. Η μικτή φύση των αποτελεσμάτων του τελικού μεγέθους κώδικα φαίνεται να προκαλείται κυρίως από την αλληλεπίδραση του καταχωρητή εκχώρησης, ιδίως στον αριθμό των φορτίων και στην στοίβα του καταχωρητή εκχώρησης. Η αποσύνθεση του γράφου σε δέντρα οδηγεί στη δημιουργία προσωρινών με πολλαπλές χρήσεις. Αυτά τα δυνητικά μακράς διάρκειας προσωρινά δημιουργούν περισσότερη πίεση στον καταχωρητή και περισσότερη αξία πρέπει να χαθεί στη μνήμη. Ως εκ τούτου, ο αλγόριθμος $sce-leaves$

εκτελεί ιδιαίτερα ανεπαρκώς. Επιτρέποντας απεριόριστη επικάλυψη μπορεί επίσης να έχει αρνητικές επιπτώσεις στον καταχωρητή αποθήκευσης καθώς οι είσοδοι των επικαλυπτόμενων πλακιδίων είναι επίσης δυνητικά μακράς διάρκειας προσωρινά. Ένας άλλος παράγοντας που επηρεάζει τον καταχωρητή αποθήκευσης είναι ο αριθμός των πλακιδίων. Εάν περισσότερο, τα πλακίδια, χρησιμοποιούνται μικρότερες, αντίστοιχα περισσότερα προσωρινά διατίθενται .Τελικά, η αλληλεπίδραση μεταξύ της επιλογής εντολών και του καταχωρητή αποθήκευσης δεν μπορούν εύκολα να χαρακτηριστούν και είναι πέρα από το πεδίο της εργασίας. Είναι πιθανόν εκείνες οι αρχιτεκτονικές με σύνθετα σύνολα εντολών, αλλά άφθονοι καταχωρητές θα έχουν περισσότερο όφελος από τον αλγόριθμο NOLTIS. Επιπλέον, δοθέντος ενός πλαισίου για τον χαρακτηρισμό η επέμβαση μεταξύ της επιλογής εντολών και του καταχωρητή αποθήκευσης , η σχεδόν βέλτιστοτητα του αλγορίθμου NOLTIS θα καθιστούσε τη φυσική επιλογή για την εκτέλεση της πλακόστρωσης.

5.6.4 ΑΠΟΔΟΣΗ ΧΡΟΝΟΥ ΜΕΤΑΓΛΩΤΤΙΣΗΣ

Όπως φαίνεται στο σχήμα 21,και οι δύο παίρνουν την φύση του αλγορίθμου NOLTIS που σημαίνει ότι ο χρόνος εκτέλεσης του είναι περισσότερο από το διπλάσιο των άλλων δυναμικών προγραμματισμών που είναι βασισμένοι σε αλγορίθμους. Οι αλγόριθμοι δυναμικού προγραμματισμού είναι περίπου 30% πιο αργοί από τους άπληστους αλγόριθμους, δεδομένου ότι πρέπει να εκτελέσουν μια λειτουργία επιλογής πλακιδίου σε κάθε κόμβο της έκφρασης DAG. Ο άπληστος αλγόριθμος μπορεί να αγνοήσει τυχόν κόμβους οι οποίοι μεταφέρονται πλήρως από άπληστα επιλεγμένα πλακίδια.

5.7 ΠΕΡΙΟΡΙΣΜΟΙ ΚΑΙ ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ

Οι αλγόριθμοι επιλογής εντολών έχουν χρησιμοποιηθεί επιτυχώς για να επιλύσουν το πρόβλημα της τεχνολογίας στην αναπαράσταση στον τομέα του σχεδιασμού αυτόματων κυκλωμάτων. Παραμένει ανοικτό το ερώτημα αν ο αλγόριθμος NOLTIS μπορεί να είναι επιτυχημένα προσαρμοσμένος σε αυτόν τον τομέα, όπου πολλαπλοί τελικοί στόχοι βελτιστοποιήσεων (περιοχή, καθυστέρηση, δρομολόγηση, πόροι) πρέπει να είναι συγχρόνως διευθυνδιοδοτημένα. Παρά το γεγονός ότι ο αλγόριθμος NOLTIS είναι γραμμικός στο μέγεθος του προγράμματος, ο χρόνος εκτέλεσης του καθορίζεται σε μεγάλο βαθμό από το πόσο αποτελεσματικά η αντιστοίχιση του απλού κόμβου σε ένα σύνολο από πλακίδια μπορεί να εκτελεστεί. Ο αλγόριθμος, όπως τον έχουμε παρουσιάσει,

χρησιμοποιεί ένα απλό, αλλά ανεπαρκή, αλγόριθμο αντιστοίχισης. Πιο αποδοτικοί αλγόριθμοι, όπως η συντακτική ανάλυση δέντρου[Koes38,Koes10,Koes39,Koes6], και θα πρέπει να χρησιμοποιούνται σε μια εφαρμογή παραγωγής. Επιπλέον, το δεύτερο πέρασμα του δυναμικού προγραμματισμού θα μπορούσε να καταστεί πιο αποτελεσματικό με έξυπνο εκ νέου υπολογισμό μόνο από τμήματα του DAG. Η κλασική αναπαράσταση επιλογής εντολών όπως το πρόβλημα της πλακόστρωσης βασίζεται σε εντολές που αντιπροσωπεύονται από πλακίδια δέντρου. Σε μερικές περιπτώσεις, όπως με απλή εντολή πολλαπλών δεδομένων (SIMD) οδηγίες και εντολές με πλάγια αποτελέσματα, μια εντολή δεν μπορεί να αναπαρασταθεί όπως εξαρτάται ένα δέντρο δεδομένων . Επιπρόσθετα, η μη-πλακόστρωση, οι τεχνικές που απαιτούνται για τον χειρισμό τέτοιων εντολών. Το αφηρημένο μοντέλο μηχανής που χρησιμοποιείται από τους αλγόριθμους μας πλακόστρωσης είναι τριών διεύθυνση, άπειρου καταχωρητή μηχανής. Βρίσκοντας έναν γραμμικού χρόνου, σχεδόν βέλτιστο αλγόριθμο που δεν εξαρτάται σε αυτές τις υποθέσεις παραμένει ένα ανοικτό πρόβλημα. Λόγω της σκληρότητας του προβλήματος αποθήκευσης του καταχωρητή, φαίνεται απίθανο ότι υπάρχει ένας τέτοιος αλγόριθμος. Ωστόσο , μπορεί να είναι δυνατή η δημιουργία ενός πλαισίου το οποίο ενσωματώνει τον καταχωρητή καταμερισμού και την επιλογή εντολής. Τα δύο περάσματα στη συνέχεια θα συνεργαστούν με τον επιλογέα εντολών σε κατευθυντήριες αποφάσεις και το αντίστροφο. Για παράδειγμα, ο επιλογέας εντολών μπορεί να δημιουργήσει κατά προσέγγιση μια πλακόστρωση όπου ο καταχωρητής εκχώρησης είναι υπεύθυνος για την οριστικοποίηση βάσει της διαθεσιμότητας του καταχωρητή. Ή ο καταχωρητής εκχώρησης θα μπορούσε να παρέχει ανατροφοδότηση με τον επιλογέα εντολών ο οποίος αλλάζει το κόστος των πλακιδίων. Πιστεύουμε ότι ο NOLTIS θα ήταν πολύτιμο μέρος ενός τέτοιου πλαισίου

Δοθέντος πλακιδίου t και κόμβου n ,προσδιορίζουμε το κόστος της τομής t στο κόμβο n ,έτσι ώστε η ρίζα του t παραμένει η ίδια αλλα ο n γίνεται η ακμή του κόμβου

1: λειτουργία GETTILECUTCOST (t, n)

2: $bestCost \leftarrow \infty$

3: $r \leftarrow root(tile)$

4: $for \;\; t' \in \; matching \; Tiles(r) \; do$

5: $if \quad n \in edgeNodes(t') \;\; then$

6: $\cos t \leftarrow \cos t(t')$

7: *for* $n' \in edgeNodes(t') \wedge n' \neq n$ *do*

8: $\cos t \leftarrow \cos t + bestChoiseForNode[n'].\cos t$

9: *if* $\cos t < bestCost$ *then*

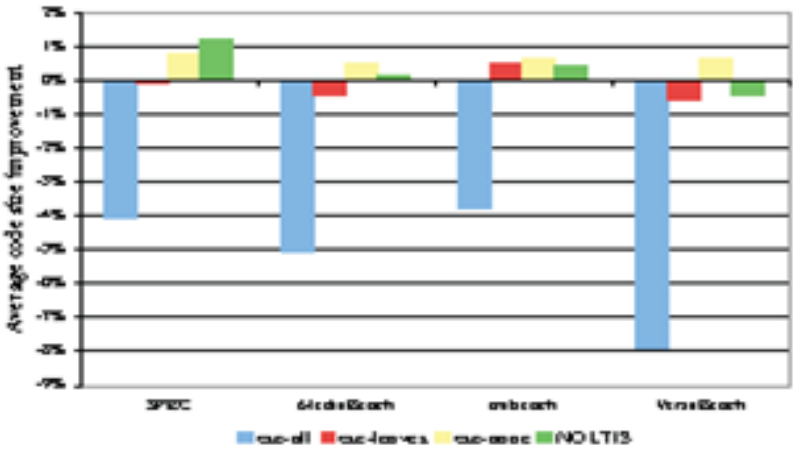
10: $bestCost \leftarrow \cos t$

11: *for* $n' \in edgeNode(t)$ *do* ▷ αφαιρεί το κόστος της ακμής του γνήσιου πλακιδίου

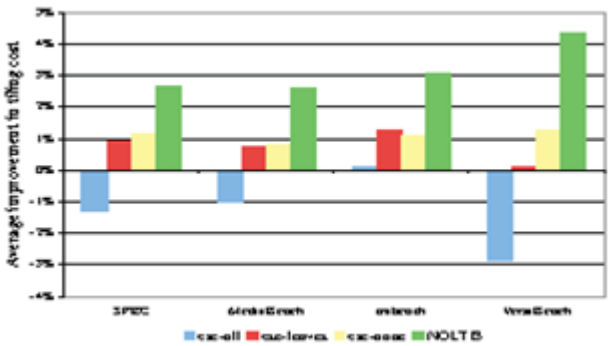
12: *if path* $r \rightarrow n' \in t$ δεν περιέχει το n *then*

13: $bestCost \leftarrow bestCost - bestChoiseForNode[n'].\cos t$

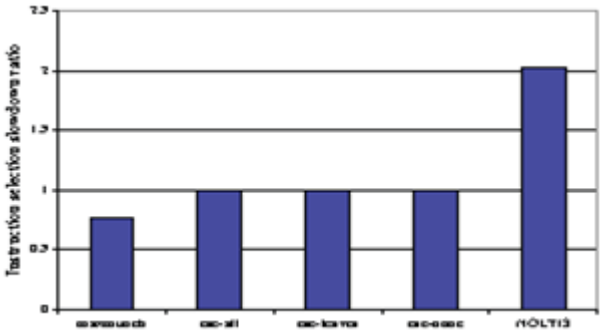
14: *return bestCost*



Σχήμα 19:Η σχετική βελτίωση του μέσου μέγεθους του τελικού κώδικα για την πρεπιλογή του μέγιστου αλγόροθμου αντιστοίχησης για τέσσερα σημεία αναφοράς



Σχήμα 20:Η μέση βελτίωση στο κόστος της πλακόστρωσης για την πρεπιλογή του μέγιστου αλγόροθμου αντιστοίχησης για τέσσερα σημεία αναφοράς



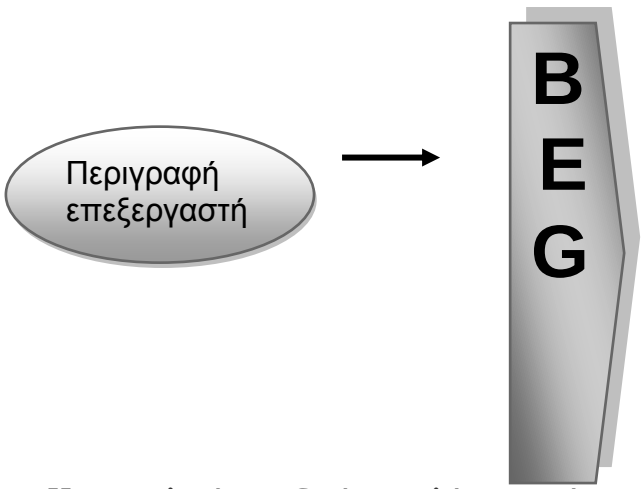
Σχήμα 21:Η συνολική επιβάρυνση του κάθε αλγόριθμου επιλογής κώδικα σχετική με τον αλγόριθμο $cse-all$.Η ιδιότητα των δυο περασμάτων του αλγόριθμου NOLTIS προκύπτει στη λήψη ελαφρώς περισσότερο από το διπλάσιο όσο του άλλου προγραμματισμού βασισμένο σε αλγόριθμος.

Σε αυτή την εργασία έχουμε περιγράψει τον NOLTIS, μια εύκολη εφαρμογή, γρήγορο και αποτελεσματικό αλγόριθμο για την εύρεση μιας εντολής πλακόστρωσης της έκφρασης DAG. Έχουμε δείξει εμπειρικά ότι ο αλγόριθμος NOLTIS επιτυγχάνει σχεδόν βέλτιστα αποτελέσματα και υπερτερεί σημαντικά από τις υφιστάμενες ευρετικές πλακοστρώσεις. Αν και η αλληλεπίδραση μεταξύ της επιλογής εντολών και του καταχωρητή καταμερισμού χρήζει περαιτέρω μελέτης, έχουμε δείξει ότι ο NOLTIS είναι ικανός να βελτιώσει το μέγεθος του κώδικα σε σχέση με υπάρχουσες τεχνικές.

ΚΕΦΑΛΑΙΟ 6: Η ΤΕΧΝΟΛΟΓΙΑ ΤΟΥ ΓΕΝΝΗΤΟΡΑ ΚΩΔΙΚΑ BEG

Η τεχνολογία του γεννήτορα κώδικα BEG είναι μέρος της κύριας ακμής, βιομηχανία αποδεδειγμένης τεχνολογίας γέννησης κώδικα. Για εσάς, καθώς ο προγραμματιστής του μεταγλωττιστή, επιτρέπει την κατασκευή καλύτερων μεταγλωττιστών για περισσότερους επεξεργαστές με στόχο το τμήμα της προσπάθειας που απαιτείται πριν.

Κάθε νέος στόχος επεξεργαστή απαιτεί την ανάπτυξη νέων γεννητόρων κωδίκων · ένα δύσκολο και επιρρεπές σε σφάλμα έργο-ειδικά για την αναδυόμενη υψηλή βαθμωτή εκτέλεση, απαιτώντας να γίνουν όλο και περισσότερο δύσκολες βελτιστοποιήσεις. Η τεχνολογία BEG επιλύει αυτό το πρόβλημα για σας με μια επαναστατική προσέγγιση: η γεννήτρια μηχανή του γεννήτορα κώδικα. Αναπτύσσουν έναν γεννήτορα κώδικα από απλά διατυπωμένη περιγραφή του επεξεργαστή στόχου. Το πρόγραμμα BEG διαβάζει αυτή την περιγραφή αυτόματα οι συμβολικές(assembly)γλώσσες του γεννήτορα κώδικα (βλέπε σχήμα).



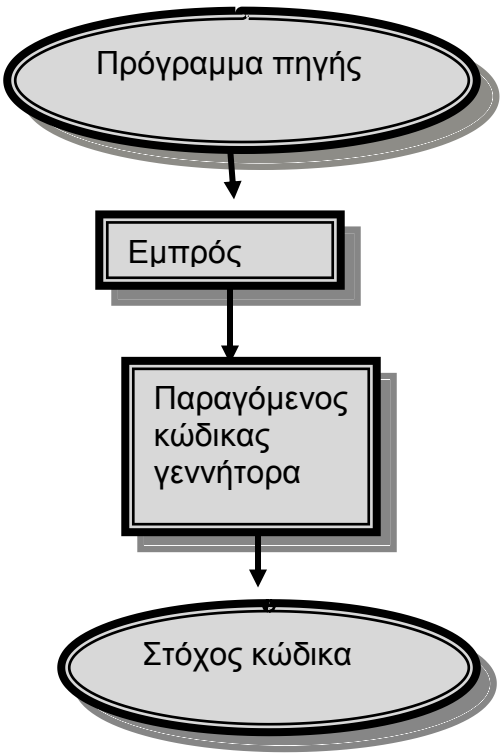
Η τεχνολογία BEG είναι πολύ περισσότερο από ότι ακριβώς ένας φορητός γεννήτορας κώδικας με ένα σταθερό ενδιάμεσο κώδικα ή μια εικόνα αντιστοίχισης εργαλείων για την επιλογή κώδικα. Συνδυάζει ένα δυναμικό πρότυπο βέλτιστου πρότυπου δέντρου με μια βιβλιοθήκη των αναφορών γέννησης των αλγόριθμων και μια νέα και δυναμική μέθοδο στην χρονοδρομολόγηση εντολών. Ο BEG χρησιμοποιεί ένα είδος να παράγουν τα μέρη του γεννήτορα κώδικα, αυτόματα δένδρα για την επιλογή κώδικα, η αντιστοίχηση DAG για ολοκλήρωση με SSA βελτιστοποίηση (π.χ. σταθερός βελτιστοποίητης)η αυτόματη προσομοίωση μηχανής για προγραμματισμό, ενσωματώνονται ο καταμερισμός καθολικού / τοπικού καταχωρήτη συν η αυτόματη επιλογή και παραμετροποίηση των στοιχείων από

την εσωτερική βιβλιοθήκη του BEG. Όλη η γέννηση καθοδηγείται από έναν απλού τύπου ενσωματωμένου επεξεργαστή, η οποία απλοποιεί τη διατύπωση του επεξεργαστή και εγγυάται την ακεραιότητα ολόκληρου του συστήματος. Η τεχνολογία BEG επιτρέπει να οικοδομήσουμε υψηλής ποιότητας γεννήτορα κώδικα σε λίγο χρόνο και η προσπάθεια που απαιτείται από τη γραφή στο χέρι. Το αποτέλεσμα του γεννήτορα κώδικα που προκύπτει είναι πολύ αξιόπιστο, διότι παράγεται από μια αυτόματα ελεγχόμενη προδιαγραφή επιπλέον προς την όποια δοκιμασμένη συνιστώσα από την βιβλιοθήκη επαναχρησιμοποιείται. Η τεχνολογία BEG έχει αποδειχθεί ώριμη σε αρκετά ερευνητικά και βιομηχανικά έργα μεταγλωττιστών, για παράδειγμα, ένας βιομηχανικός μεταγλωττιστής της οικογένειας επεξεργαστών SPARC™ .Οι γεννήτορες κώδικα για INTEL™ 80x86,MC68020, και MIPS™R3000 έχουν παραχθεί επίσης. Με τον BEG μπορείτε να έχετε μαζί πολύ γρήγορους γεννήτορες κώδικα με ικανοποιητική ποιότητα κώδικα καθώς και την παραγωγή γεννητόρων κώδικα με πολύ υψηλής ποιότητα κώδικα, σχεδόν από την ίδια περιγραφή επεξεργαστή. Ο τελευταίος γεννήτορας κώδικα περιέχει έναν καθολικό καταχωρητή εκχώρησης και μια ενιαία λίστα σε συνδυασμό και τους πόρους του προγράμματος .Η τεχνολογία BEG είναι διαθέσιμη σήμερα στην C για UNIX™ και WINDOWS™.Οι μεταγλωττιστές που παράγονται είναι προγράμματα C,που τρέχουν σχεδόν σε κάθε περιβάλλον C.

ΤΕΧΝΙΚΕΣ ΛΕΠΤΟΜΕΡΕΙΕΣ

ΓΕΝΝΗΣΗ ΜΗΧΑΝΗΣ

Η γέννηση της μηχανής σημαίνει ότι το επιθυμητό πρόγραμμα του γεννήτορα κώδικα (μια συλλογή από αρχεία C) παράγεται από μια περιγραφή του επεξεργαστή από το γεννήτορα πρόγραμμα BEG (βλέπε σχήμα). Γράφετε μια περιγραφή του επεξεργαστή αντί για το πρόγραμμα του. Γράφοντας την περιγραφή του επεξεργαστή είναι πολύ ευκολότερα, ταχύτερα, ασφαλέστερα από τη συγγραφή των ίδιων των προγραμμάτων. Η περιγραφή του επεξεργαστή σας ελέγχεται για τη συνέπεια που οδηγεί σε βελτίωση της αξιοπιστίας του γεννήτορα κώδικα που παράγεται.



ΕΝΔΙΑΜΕΣΗ ΓΛΩΣΣΑ

Μπορείτε να διασυνδεθείτε στον ήδη υπάρχων μεταγλωττιστή μπρος τέλος χωρίς αλλαγή στο μπροστινό άκρο ή μεταφράζοντας τον ενδιάμεσο κώδικα από απλή περιγραφή του ενδιάμεσου στην περιγραφή του επεξεργαστή

Ευελιξία

Για να αντιμετωπίσει την τεράστια ποικιλία των χαρακτηριστικών που προσφέρονται από τους τρέχοντες επεξεργαστές η μεγάλη ευελιξία είναι απαραίτητη. Η τεχνολογία BEG το προσφέρει αυτό χρησιμοποιώντας ένα γενικό μηχανισμό δέντρου αντιστοίχησης, με ένα γράφο με βάση χρωστικές και ιδιαίτερες παραμέτρους στον καθολικό καταχωρητή εκχώρησης, την τεχνική και τον προγραμματισμό των πόρων. Οι προϋποθέσεις και οι δράσεις στην περιγραφή του επεξεργαστή χρησιμοποιούν την πλήρη ισχύ της γλώσσας C. Η καλή διεπαφή στον προγραμματισμό στο χέρι είναι η υποστήριξη των ειδικών τμημάτων του γεννήτορα κώδικα, θα πρέπει αυτό να απαιτείται.

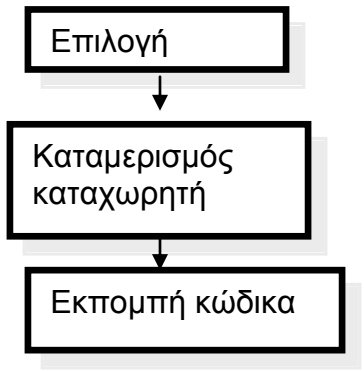
ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΓΕΝΝΗΤΟΡΑ ΚΩΔΙΚΑ

Ο χειρόγραφος γεννήτορας κώδικας έχει μια σταθερή αρχιτεκτονική, η οποία έχει συσταθεί κατά το σχεδιασμό σύμφωνα με την επιθυμητή ποιότητα του κώδικα και την ταχύτητα σύνταξης. Με τον BEG μπορεί να παράγει γεννήτορες κώδικα πολλών διαφορετικών αρχιτεκτονικών με μικρές αλλαγές στις περιγραφές του επεξεργαστή. Τα διαγράμματα δείχνουν την αρχιτεκτονική ενός απλού, υψηλής ταχύτητας σύνταξη του

γεννήτορα κώδικα σε αντίθεση με μια πιο πολύ σύνθετη βελτιστοποίηση του γεννήτορα κώδικα.

ΕΝΣΩΜΑΤΩΜΕΝΗ ΓΕΝΝΗΤΡΙΑ ΚΩΔΙΚΑ

Καθώς υπάρχουν πολλά μέρη του γεννήτορα κώδικα, υπάρχουν πολλές διαφορετικές μέθοδοι για να δημιουργηθούν αυτά τα μέρη. Σε αντίθεση με άλλα εργαλεία, ο BEG δημιουργεί όλες τις συνιστώσες από μια απλή περιγραφή ενσωματωμένου επεξεργαστή. Αυτό εγγυάται τη συνοχή μεταξύ των μερών, διευκολύνει την ενσωμάτωση, και απλοποιεί την ανάπτυξη της περιγραφής του επεξεργαστή.



Δομή υψηλής ταχύτητας γεννήτορα κώδικα

Η ΓΕΝΝΗΣΗ ΧΡΗΣΙΜΟΠΟΙΩΝΤΑΣ ΑΥΤΟΜΑΤΑ ΘΕΩΡΗΤΙΚΕΣ ΜΕΘΟΔΟΥΣ

Αυτόματα η θεωρία προσφέρει αλγόριθμους για την παραγωγή προγραμμάτων ("αυτόματα")που βασίζονται σε ορισμένα σύνολα κανόνων. Αυτές οι τεχνικές έχουν χρησιμοποιηθεί με επιτυχία σε συντακτικό αναλυτή γεννήτορα για μεγάλο χρονικό διάστημα. Η επέκταση σε δέντρα, η θεωρία αυτόματα σε δέντρα, κάνει τώρα να είναι δυνατόν να παραχθούν αυτόματα επιλογείς κώδικα. Οι καινοτόμες τεχνικές χρησιμοποιούνται αυτόματα για την προσομοίωση του επεξεργαστή στόχου, καθιστώντας έτσι τον προγραμματισμό των πόρων να είναι εφικτός.

ΠΑΡΑΜΕΤΡΟΠΟΙΗΣΗ ΓΕΝΝΗΤΟΡΑ

Τα άλλα τμήματα του γεννήτορα κώδικα (π.χ. καταχωρητή εκχώρησης) λαμβάνονται από μια εσωτερική βιβλιοθήκη BEG και παραμετροποιείται σύμφωνα με την περιγραφή του επεξεργαστή. Αυτό επιτρέπει την επαναχρησιμοποίηση αυτών των εξαιρετικά πολύπλοκων αλγορίθμων σε οποιοδήποτε παραγόμενο γεννήτορα κώδικα. Η συνοχή αυτόματα διατηρείται από το πρόγραμμα του γεννήτορα BEG.

ΕΠΙΛΟΓΗ ΚΩΔΙΚΑ ΑΠΟ ΤΟ ΠΡΟΤΥΠΟ ΑΝΤΙΣΤΟΙΧΙΣΗΣ ΔΕΝΤΡΟΥ

Η επιλογή Κώδικα επιλέγει τις εντολές μηχανής για έναν ενδιάμεσο κώδικα δέντρου από την περιγραφή του επεξεργαστή, η οποία περιγράφει κάθε εντολή και κάθε τρόπο δυευθυνσιόδοτησης από ένα κανόνα αντιστοίχισης του πρότυπου δέντρου και μια τιμή κόστους. Η παραγόμενη μηχανή επιλογέα κώδικα επιλέγει τις εντολές για έναν ενδιάμεσο κώδικα δέντρου κατά το βέλτιστο τρόπο, σύμφωνα με το κόστος που δίνεται στην περιγραφή. Έτσι, η περιγραφή του επεξεργαστή περιγράφει ακριβώς ποιος κώδικας μπορεί να επιλεγεί, μια επιλογή ενός αλγόριθμου όπου ο βέλτιστος κώδικας μπορεί να συνταχθεί αυτόματα και δεν είναι απαραίτητο να προγραμματιστεί.

ΕΠΙΛΟΓΗ ΚΩΔΙΚΑ ΑΠΟ DAG ΑΝΤΙΣΤΟΙΧΙΣΗ

είναι μια επιλογή. Είναι πολύ ισχυρή, αν η ενδιάμεση γλώσσα δεν είναι σε μορφή δέντρου, αλλά αναπαρίστανται ως DAG. Αυτό είναι η τυπική περίπτωση που ένας SSA που βασίζεται σε βέλτιστο χρησιμοποιείται. Η αντιστοίχιση DAG προσφέρει μια πολύ καλύτερη ποιότητα κώδικα, επειδή το πρότυπο αντιστοίχισης λειτουργεί πλέον σε ένα βασικό μπλοκ αντί μιας απλής έκφρασης DAG.

ΚΑΤΑΧΩΡΗΤΗΣ ΕΚΧΩΡΗΣΗΣ

Μπορείτε να επιλέξετε μεταξύ τοπικού και καθολικού καταχωρήτη εκχώρησης. Ο τοπικός καταχωρητής εκχώρησης λειτουργεί με το πεδίο της ενδιάμεσης κατάστασης κώδικα, γεγονός που τον καθιστά πολύ γρήγορο. Ο καθολικός καταχωρητής εκχώρησης λειτουργεί σε μια πλήρη διαδικασία ταυτόχρονα. Έτσι, τα ενδιάμεσα αποτελέσματα μπορούν να μείνουν πολύ περισσότερο στους καταχωρητές με αποτέλεσμα υψηλής ποιότητας κώδικα. Ο καθολικός καταχωρητής εκχώρησης χρησιμοποιείται σε μια ενσωματωμένη μέθοδο τοπικού / καθολικού. Ο καταχωρητής μηχανής όρισε και τους καταχωρητές για κάθε εντολή μηχανής που μπορούν να προσδιοριστούν, καλύπτοντας ένα ευρύ φάσμα μηχανών στόχου.

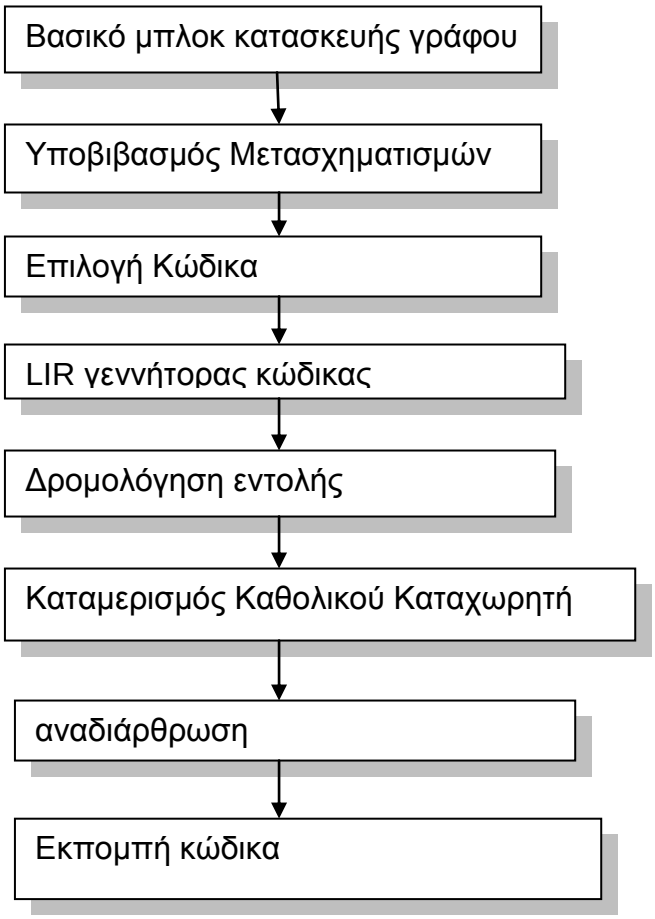
ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΕΝΤΟΛΩΝ

Η συνδυασμένη λίστα και οι πόροι του χρονοδρομολογητή παράγονται. Η λίστα χρονοδρομολόγησης αναδιατάσσει τις εντολές, έτσι ώστε τα μέσα πληροφορίας του επεξεργαστή που απαιτούνται να περιμένουν να είναι όσο το δυνατόν σπάνια για ένα

ενδιάμεσο αποτέλεσμα που δεν έχει ακόμα υπολογιστεί. Ο πόρος της χρονοδρομολόγησης παίρνει μια πολύ βαθύτερη γνώση των μέσων πληροφορίας της μηχανής υπόψη για να αποφευχθούν πιθανές συγκρούσεις των μέσων πληροφορίας βασισμένα στον περιορισμένο αριθμό των λειτουργικών μονάδων των επεξεργαστών. Ιδιαίτερα, μπορεί να χειριστεί πολλαπλά θέματα επεξεργαστών ,που μπορεί να εκτελέσει παραπάνω από μία εντολές παράλληλα. Για να κάνει δυνατή τον προγραμματισμό των πόρων, ο πόρος του επεξεργαστή προσομοιώνεται, ενώ προγραμματίζεται. Αυτό έγινε πρακτικά δυνατό μόνο μετά την εφαρμογή περιορισμένων αυτόματων τεχνικών. Η εντολή στο πρόγραμμα εξακολουθεί να απαιτεί κάποια προσαρμογή σε νέο περιβάλλον.

ΧΡΗΣΕΙΣ ΤΗΣ ΤΕΧΝΟΛΟΓΙΑΣ BEG

Η τεχνολογία BEG χρησιμοποιήθηκε για πρώτη φορά στους πολύ γρήγορους GMD Modula-2 μεταγλωττιστές Mocka ™. Οι γεννήτορες κώδικα παρήχθησαν για Motorola 68020, για SPARC ™, για MIPS ™ και Intel 386, INMOS ™ T800 και PowerPC ™. Αυτός ο μεταγλωττιστής χρησιμοποιείται ευρέως στην έρευνα και στην εκπαίδευση. Η έκδοση Linux είναι ένα σημαντικό εργαλείο που διατίθεται δωρεάν στον κώδικα πηγής. Η περιγραφή του επεξεργαστή γίνεται εντελώς από έναν φοιτητή που στο παρελθόν δεν ήταν εξοικειωμένος με την BEG μέσα σε 3 μήνες. Η Beg ήταν να χρησιμοποιηθεί για την παραγωγή στην οικογένεια των επεξεργαστών για C, σε ANSI C, Modula-2, Fortran-77, και Pascal, για τους επεξεργαστές SPARC.



Δομή Βέλτιστου Κώδικα Γεννήτορα

ΑΝΑΦΟΡΕΣ:

[Ertl01] Jack W. Davidson and Christopher W. Fraser. Code selection through object code optimization. ACM Transactions and Programming Languages and Systems, 6(4):505-526, October 1984

[Ertl02] Helmut Emmelmann, Friedrich-Wilhelm Schroer, and Rudolf Landwehr, BEG-a generator for efficient back ends. In SIGPLAN ’89 Conference on Programming Language Design and Implementation, pages 227-237,1989

[Ertl03] Alan L. Wendt. Fast code generation using automatically-generated decision trees. In SIGPLAN ’90 Conference on Programming Language Design and Implementation, pages 9-15, 1990

[Ertl04] John Boyland and Helmut Emmelmann. Discussion: Code generator specification techniques(summary). In Robert Giegerish and Susan L. Graham,editors, Code Generation-Concepts, Tools, Techniques, Workshops in Computing, pages 66-69. Sprigner,1991.

[Ertl05] Christopher W. Fraser and David R. Hanson. A code generation interface for ANSI C. Software-Practice and Experience,21(9): 963-988, September 1991.

[Ertl06] Christopher Fraser and David Hanson. A Retargetable C compiler: Design and Implementation, Benjamin / Cummings Publishing, 1995

[Ertl07] Todd A. Proebisting. BURS automata generation. ACM Translations on Programming Languages and Systems, 17(3):461-486, May 1995

[Ertl08] Todd A. Proebisting and Benjamin R. Whaley. One pass, optimal tree parsing-with or without trees. In Tidor Gyimóthy, editor, Compiler Construction(CC’96),pages 294-308, Lincoping 1996.Springer LNCS 1060

[Ertl09] Todd A. Proebisting. Least cost instruction selection in DAGs is NP-complete. http://research.microsoft.com/todpro/papers/proof.html ,1998

[Koes01] AHO, A., AND ULLMAN, J. Optimization of stright line programs. SIAM Journal on Computing 1, 1 (1972), 1–19.

[Koes02] BRUNO, J., AND SETHI, R. Code generation for a one-register machine. J. ACM 23, 3 (1976), 502–510.

[Koes03] AHO, A. V., AND JOHNSON, S. C. Optimal code generationfor expression trees. J. ACM 23, 3 (1976), 488–501.

[Koes04] AHO, A. V., JOHNSON, S. C., AND ULLMAN, J. D. Code generation for expressions with common subexpressions. J.ACM 24, 1 (1977), 146–160.

[Koes05] SETHI, R., AND ULLMAN, J. D. The generation of optimal code for arithmetic expressions. J. ACM 17, 4 (1970),715–728.

[Koes06] BOSE, P. Optimal code generation for expressions on super scalar machines. In Proc. of the ACM Fall Joint Computer Conf. (Los Alamitos, CA, USA, 1986), IEEE Computer

Society Press, pp. 372–379

[Koes07] GLANVILLE, R. S., AND GRAHAM, S. L. A new method for compiler code generation. In Proc. of ACM/SIGPLAN-SIGACT Symposium on Principles of Prog.Lang. (New York, NY, USA, 1978), ACM Press, pp. 231–254.

[Koes08] CATTELL, R. G. Automatic derivation of code generators from machine descriptions. ACM Trans. Program. Lang. Syst. 2, 2 (1980), 173–190.

[Koes09] AHO, A. V., GANAPATHI, M., AND TJIANG, S. W. K. Code generation using tree matching and dynamic programming. ACM Trans. Program. Lang. Syst. 11, 4 (1989), 491–516.

[Koes10] FRASER, C. W., HANSON, D. R., AND PROEBSTING, T. A. Engineering a simple, efficient code-generator generator. ACM Lett. Prog. Lang. Syst. 1, 3 (1992), 213–226.

[Koes11] EMMELMANN, H., SCHRÖER, F.-W., AND LANDWEHR, L.Beg: a generation for efficient back ends. In Proc. ofACM/SIGPLAN Conf. on Prog. Lang. Design and Impl.(New York, NY, USA, 1989), ACM Press, pp. 227–237.

[Koes12] PROEBSTING, T. A. Simple and efficient burs table generation. In Proc. of ACM/SIGPLAN Conf. on Prog. Lang.Design and Impl. (New York, NY, USA, 1992), ACM Press,pp. 331–340.

[Koes13] PELEGRÍ-LLOPART, E., AND GRAHAM, S. L. Optimal codegeneration for expression trees: an application burs theory. In Proc. of ACM/SIGPLAN-SIGACT Symposium on Principles of Prog. Lang. (New York, NY, USA, 1988), ACM Press,pp. 294–308.

[Koes14] FRASER, C. W., HENRY, R. R., AND PROEBSTING, T. A.Burg: fast optimal instruction selection and tree parsing. SIGPLAN Not. 27, 4 (1992), 68–76.

[Koes15] PROEBSTING, T. A. Burs automata generation. ACM Trans.Program. Lang. Syst. 17, 3 (1995), 461–486.

[Koes16] PROEBSTING, T. Least-cost instruction selection in dags isNP-complete. http://research.microsoft.com/ ~toddpro/papers/proof.htm.

[Koes17] ERTL, M. A. Optimal code selection in dags. In Proc. of ACM/SIGPLAN-SIGACT Symposium on Principles of Prog.Lang. (New York, NY, USA, 1999), ACM Press, pp. 242–249.

[Koes18] AHO, A. V., SETHI, R., AND ULLMAN, J. D. Compilers: Princiles, Techniques, and Tools. Addison-Wesley, 1986.

[Koes19] LIAO, S., DEVADAS, S., KEUTZER, K., AND TJIANG, S.
Instruction selection using binate covering for code size optimization. In Proc. of the IEEE/ACM Intl. Conf. on Computer-aided Design (Washington, DC, USA, 1995), IEEE Computer Society, pp. 393–399.

[Koes20] LIAO, S., KEUTZER, K., TJIANG, S., AND DEVADAS, S. Anew viewpoint on code generation for directed acyclic graphs. ACM Trans. Des. Autom. Electron. Syst. 3, 1 (1998),51–75.

[Koes21] NAIK, M., AND PALSBERG, J. Compiling with code-size constraints. Trans. on Embedded Computing Sys. 3, 1 (2004), 163–181.

[Koes22] KESSLER, C., AND BEDNARSKI, A. A dynamic programming approach to optimal integrated code generation. In Proc. of the ACM/SIGPLAN Workshop on Languages, Compilers and Tools for Embedded Systems(New York, NY, USA, 2001), ACM Press, pp. 165–174

[Koes23] DAVIDSON, J. W., AND FRASER, C. W. Code selection through object code optimization. ACM Trans. Program.Lang. Syst. 6, 4 (1984), 505–526.

[Koes24] FRASER, C. W., AND WENDT, A. L. Integrating code generation and optimization. In Proc. of SIGPLAN Symposium on Compiler Construction (New York, NY, USA,1986), ACM Press, pp. 242–248.

[Koes25] FRASER, C. W., AND WENDT, A. L. Automatic generation of fast optimizing code generators. In Proc. Of ACM/SIGPLAN Conf. on Prog. Lang. Design and Impl.(New York, NY, USA, 1988), ACM Press, pp. 79–84.

[Koes26] KESSLER, R. R. Peep: an architectural description driven peephole optimizer. In Proceedings of SIGPLAN Symposium on Compiler Construction (New York, NY, USA, 1984), ACM Press, pp. 106–110.

[Koes27] COOPER, K. D., AND TORCZON, L. Engineering a Compiler. Morgan Kaufmann Publishers, 2004.

[Koes28] MCKEEMAN, W. M. Peephole optimization. Commun. ACM 8, 7 (1965), 443–444.

[Koes29] KEUTZER, K. DAGON: technology binding and local optimization by dag matching. In Proc. of the ACM/IEEE Conf. on Design Automation (New York, NY, USA, 1987), ACM Press, pp. 341–347.

[Koes30] HASSOUN, S., AND SASAO, T., Eds. Logic Synthesis and Verification. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[Koes31] DE MICHELI, G. Synthesis and Optimization of Digital Circuits. McGraw-Hill Higher Education, 1994, ch. 10.

[Koes32] GAREY, M. R., AND JOHNSON, D. S. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA, 1979.

[Koes33] The LLVM compiler infrastructure project. http://llvm.org.

[Koes34] APPEL, A. W. Modern Compiler Implementation in Java: Basic Techniques. Cambridge University Press, 1997.

[Koes35] SETHI, R., AND ULLMAN, J. D. The generation of optimal code for arithmetic expressions. J. ACM 17, 4 (1970), 715–728.

[Scharwaechter01] H. Scharwaechter, D. Kammler, A. Wieferink, M.

Hohenauer, J. Zeng, K. Karuri, R. Leupers, G. Ascheid, and H. Meyr. ASIP Architecture Exploration for Efficient IPSec Encryption: A Case Study. *ACM Transactions on Embedded Computing Systems (TECS)*, 6(2), Mai 2007.

[Scharwaechter02] A. V. Aho, M. Ganapathi, and S. W. K. Tjiang. Code Generation Using TreePattern Matching and Dynamic Programing. *ACM Transactions on Programming Languages and Systems.*, 11(4):491–516, Oct. 1989.

[Scharwaechter03] A. Todd. Least-Cost Instruction Selection in DAGs is NP-Complete . In *"http://research.microsoft.com/˜toddpro/papers/proof.htm"*, Feb. 2007.

[Scharwaechter04] C. W. Fraser, D. R. Hanson, and T. A. Proebsting. Engineering Efficient Code Generators Using Tree Matching and Dynamic Programming. Technical Report TR-386-92, 1992.

[Scharwaechter05] S. W. K. Tjiang. An Olive Twig. Technical report, Synopsys Inc., 1993.

[Scharwaechter06] C. Fraser and D. Hanson. *A Retargetable C Compiler : Design and Implementation*. Benjamin/Cummings Publishing Co., 1994.

[Scharwaechter06] C. Devine. http://xyssl.org, 2007.

[Scharwaechter07] M. A. Ertl. Optimal Code Selection in DAGs . In *"Principles of Programming Languages (POPL '99)"*, 1999.

[Scharwaechter08] G. Araujo, S. Malik, and M. Lee. Using Register Transfer Paths in Code Generation for Heterogeneous Memory Register Architectures . In *Proc. of the Design Automation Conference (DAC)*, pages 591–596, June 1996.

[Scharwaechter09] R. Leupers and P. Marwedel. Instruction Selection for Embedded DSPs with Complex Instructions. In *Proc. of the European Conference on Design Automation (EDAC)*, Sept 1996.

[Scharwaechter10] S. Liao, S. Devadas, K. Keutzer, and S. Tjiang. Instruction Selection Using Binate Covering for Code Size Optimization. In *Proc. of the Int. Conf. on Computer Aided Design (ICCAD)*, pages 393–399, 1995.

[Scharwaechter11] C. Liem, T. May, and P. Paulin. Instruction-set Matching and Selection for DSP and ASIP Code Generation. In *Proc. of the European Design and Test Conference (ED & TC)*, pages 31–37, 1994.

[Scharwaechter12] N. Clark, A. Hormiri, S. Mahlke, and S. Yehia. Scalable Subgraph Mapping for Acyclic Computation Accelerators. In *Proc. of the Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*, Oct. 2006.

[Scharwaechter13] A. Halambi, P. Grun, V. Ganesh, A. Khare, N. Dutt, and A. Nicolau. EXPRESSION: A Language for Architecture Exploration through Compiler/Simulator Retargetability. In *Proc. of the Conference on Design,Automation & Test in Europe (DATE)*, Mar. 1999.

[Scharwaechter14] A. Hoffmann, H. Meyr, and R. Leupers. *Architecture Exploration for EmbeddedProcessors With Lisa*. Kluwer Academic Publishers, Jan. 2003. ISBN 1-4020-7338-0.

[Scharwaechter15] S. Kobayashi, Y. Takeuchi, A. Kitajima, M. Imai. Compiler Generation inPEAS-III: an ASIP Development System. In

Workshop on Software and Compilers for Embedded Processors (SCOPES), 2001.

[Scharwaechter16] S. W. K. Tjiang. An Olive Twig. Technical report, Synopsys Inc., 1993.

[Scharwaechter17] MIPS technologies Inc. *MIPS 4Kc Processor Core Datasheet*, Jun. 2000

[Scharwaechter18] M. Willems and V. ”ˇ Zivojnovi´c. DSP-Compiler: Product Quality for Control-Dominated Applications? In *Proc. of the Int. Conf. on Signal Processing Applications and Technology (ICSPAT)*, Oct. 1996.

[Scharwaechter19] K. Karuri, M. A. A. Faruque, S. Kraemer, R. Leupers, G. Ascheid., and H. Meyr. Fine-grained Application Source Code Profiling for ASIP Design . In *Proc. of the Design Automation Conference (DAC)*, pages 329–334, 2005