Προηγμένα Θέματα Θεωρητικής Πληροφορικής

Βελτιστοποιήσεις ανεξάρτητες από την αρχιτεκτονική

Νικόλαος Καββαδίας
nkavv@uop.gr

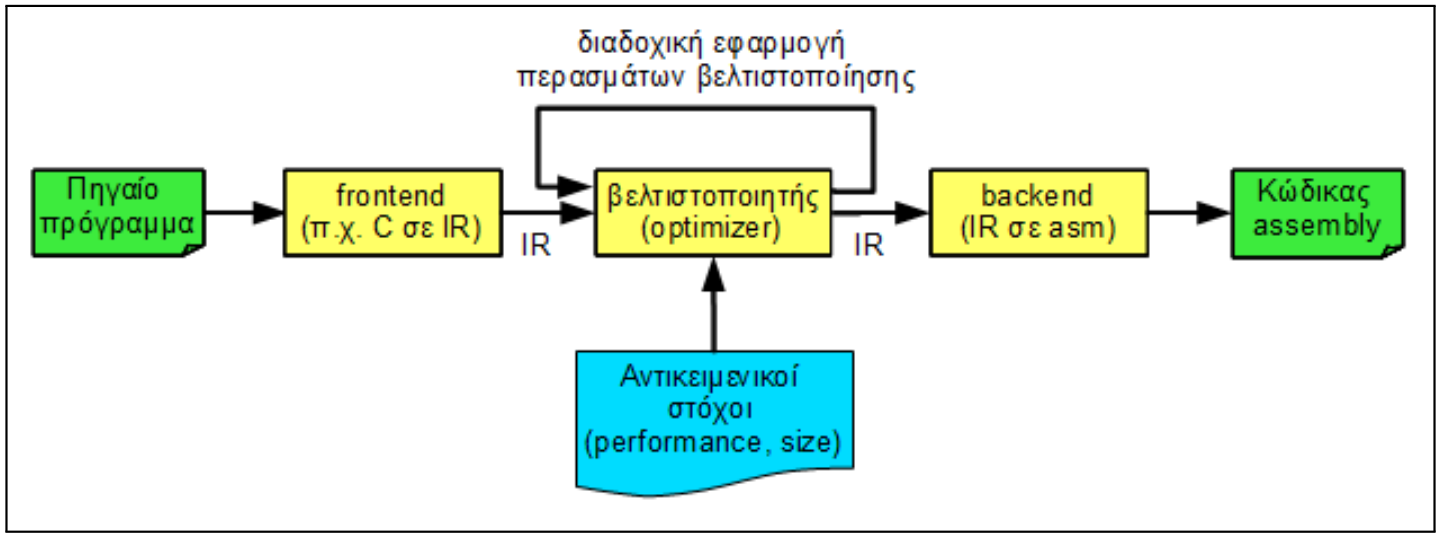
28 Απριλίου 2010

Η έννοια της βελτιστοποίησης προγράμματος

- Βελτιστοποίηση προγράμματος (program optimization): ο μετασχηματισμός ενός προγράμματος προκειμένου τη βελτίωση των επιδόσεών του
- Στόχοι βελτιστοποίησης
  1. **Performance**: ταχύτητα εκτέλεσης του κώδικα π.χ. σε αριθμό κύκλων μηχανής
  2. **Size**: μικρότερο μέγεθος εκτελέσιμου, μικρότερες απαιτήσεις σε μνήμη προγράμματος
- Φαινόμενα ανταλλαγής (trade-offs)
  1. Performance vs Size
  2. Ταχύτητα μεταγλώττισης vs απαιτήσεις μνήμης για τη διαδικασία

ℹ Δεν υπάρχει ο τέλειος βελτιστοποιητής για όλους τους επιδιωκόμενους στόχους

Η χρήση του βελτιστοποιητή στα πλαίσια ενός δομημένου μεταγλωττιστή



- Ο απεικονιζόμενος βελτιστοποιητής είναι ανεξάρτητος από την αρχιτεκτονική
  - Ένας βελτιστοποιητής εξειδικευμένος σε μία συγκεκριμένη αρχιτεκτονική επεξεργαστή θα αποτελούσε μέρος του backend

Εμβέλεια της βελτιστοποίησης (optimization scope)

- Δήλωση (statement): αριθμητικές εκφράσεις στη δεξιά πλευρά (RHS) μιας ανάθεσης
- Βασικό μπλοκ: βελτιστοποίηση ευθύγραμμου κώδικα
- Εσώτερος βρόχος (innermost loop): αύξηση παραλληλίας στα πιο συχνά εκτελούμενα τμήματα κώδικα
- Τέλεια φωλιασμένοι βρόχοι (perfect loop nest): δομή όπου το σώμα κάθε βρόχου αποτελείται μόνο από το σώμα του άμεσα εσωτερικότερου βρόχου. Αναδιοργάνωση βρόχων για την επίτευξη παραλληλίας
- Γενικευμένη δομή βρόχων (general loop nest): βελτιστοποιήσεις γενικά εφαρμόσιμες σε μία δομή βρόχων
- Διαδικασία (procedural): βελτιστοποίηση του γράφου ροής ελέγχου και της προσπέλασης μνήμης
- Υπερδιαδικαστικής εμβέλειας (inter-procedural): βελτιστοποίηση σε όλη την έκταση του προγράμματος

Είδη βελτιστοποιήσεων

- Scalar optimizations (βαθμωτές βελτιστοποιήσεις)
- Code inlining (εσωγράμμιση κώδικα)
- Procedural abstraction (αφαίρεση υποπρογράμματος)
- Loop optimizations (βελτιστοποιήσεις βρόχου)
- Register allocation (καταμερισμός κώδικα)
- Instruction scheduling (χρονοπρογραμματισμός κώδικα)
- Peephole optimization (βελτιστοποίηση κλειδαρότρυπας)
- Superoptimization (υπερβελτιστοποίηση)
- Vectorization (διανυσματοποίηση)
- Link-time optimizations (βελτιστοποιήσεις κατά τη σύνδεση αντικείμενου κώδικα)

Βελτιστοποιήσεις της IR ανεξάρτητες από την αρχιτεκτονική

- Στη διάλεξη αυτή εστιάζουμε σε βαθμωτές βελτιστοποιήσεις (scalar optimizations) οι οποίες εφαρμόζονται στην IR
- Συχνά χρησιμοποιούμενες βελτιστοποιήσεις αυτού του τύπου
  - Constant folding (δίπλωση σταθεράς)
  - Constant propagation (διάδοση σταθεράς)
  - Copy propagation (διάδοση αντιγράφου)
  - Algebraic simplifications (αλγεβρικές απλοποιήσεις)
  - Operator strength reduction (ελάττωση ισχύος τελεστή)
  - Dead code elimination (εξουδετέρωση νεκρού κώδικα)
  - Common subexpression elimination (εξουδετέρωση κοινής υποεκφράσεως)
  - Partial redundancy elimination (εξουδετέρωση μερικού πλεονασμού)
  - If conversion (μετατροπή δηλώσεων υπό συνθήκη)
  - Code motion (μετακίνηση κώδικα)

Βασικές διαφορές μεταξύ βελτιστοποιήσεων υψηλού και χαμηλού επιπέδου

- Βελτιστοποιήσεις ανεξάρτητες από την αρχιτεκτονική
  - Εφαρμόσιμες σε ευρύ φάσμα αρχιτεκτονικών επεξεργαστή
  - Ελαττώνουν το χρόνο εκτέλεσης ή την καταλαμβανόμενη μνήμη από τον τελικό κώδικα
  - Παράδειγμα: εξουδετέρωση νεκρού κώδικα
- Βελτιστοποιήσεις εξαρτημένες από την αρχιτεκτονική
  - Κεφαλοποιούν στο έπακρο τα χαρακτηριστικά του υποκείμενου επεξεργαστή
  - Βελτιστοποιούν τη χαρτογράφηση της IR σε εντολές του επεξεργαστή
  - Παράδειγμα: χρονοπρογραμματισμός εντολών
- Ο διαχωρισμός των δύο τύπων δεν είναι πάντα σαφής
- Για παράδειγμα, η βελτιστοποίηση εξουδετέρωσης πλεονασμού είναι εφαρμόσιμη και στα δύο επίπεδα, χρησιμοποιώντας διαφορετικής αναλυτικότητας γνώση για την αρχιτεκτονική

Κατηγοριοποίηση βελτιστοποιήσεων εκ του αποτελέσματος

- Ενέργειες που πραγματοποιούνται κατά τη βελτιστοποίηση για ταχύτητα
  - Ανεξάρτητα από την αρχιτεκτονική
    - Διαγραφή ενός πλεονάζοντος υπολογισμού
    - Μετακίνηση κώδικα σε λιγότερα συχνά εκτελούμενα βασικά μπλοκ
    - Εξουδετέρωση μη προσβάσιμου κώδικα
    - Ενεργοποίηση της εφαρμογής άλλων βελτιστοποιήσεων
  - Σε εξάρτηση από την αρχιτεκτονική
    - Απόκρυψη της καθυστέρησης των προσπελάσεων στη μνήμη δεδομένων
    - Εκμετάλλευση ιδιαίτερων χαρακτηριστικών όπως ειδικών αριθμητικών μονάδων (π.χ. για τη γέννηση ψευδοτυχαίων αριθμών)
    - Κατάλληλη διαχείριση πόρων με στόχο την αύξηση του ποσοστού χρησιμοποίησής τους (resource utilization)

Δίπλωση και διάδοση σταθεράς

- Δίπλωση σταθεράς: υπολογισμός σταθερών εκφράσεων κατά το χρόνο μεταγλώττισης

```
c = 1 + 3;

if (!true && 0) {}
```

```
c = 4;

if (false) {}
```

- Διάδοση σταθεράς: απορρόφηση μεταβλητών οι οποίες έχουν σταθερή τιμή από σταθερές εκφράσεις

```
b = 3;
c = 1 + b;
d = b + c;
```

```
b = 3;
c = 1 + 3;
d = 3 + c;
```

Διάδοση αντιγράφου

- Δεδομένης της ανάθεσης `x = y;` γίνεται αντικατάσταση των μεταγενέστερων εμφανίσεων της x από την y εφόσον δεν ξαναορίστηκε η x από κάποια άλλη ανάθεση

```
x = y;
c = 1 + x;
d = x + c;
```

```
x = y;
c = 1 + y;
d = y + c;
```

Αλγεβρικές απλοποιήσεις

- Γίνεται χρησιμοποίηση γνωστών αλγεβρικών ιδιοτήτων για την απλοποίηση εκφράσεων
- Για παράδειγμα εκφράσεις προσθαφαίρεσης με το 0 και πολλαπλασιασμού με το 1 απλοποιούνται καθώς αυτά είναι αντίστοιχα τα ουδέτερα στοιχεία των πράξεων
- Κανόνες απλοποίησης

Έκφραση	Αντικατάσταση
x + 0	x
x - 0	x
- (- x)	x
x * 0	0
x * 1	x
x / 1	x
x / x	1
x + c	c + x
x * c	c * x
(x + c) + y	(x + y) + c
(x * c) * y	(x * y) * c
c * x + c * y	c * (x + y)

Ελάττωση ισχύος τελεστή

- Αντικατάσταση εκφράσεων με υψηλό κόστος (π.χ. σε εκτιμώμενους κύκλους μηχανής) από απλούστερες εκφράσεις με μικρότερο κόστος
- Κανόνες απλοποίησης

Έκφραση	Αντικατάσταση
2 * x	x + x
x * 2^n	x << n
x / 2^n	x >> n

- Πολλαπλασιασμός με σταθερά (constant multiplication)
- Δυσεπίλυτο πρόβλημα: αντιμετωπίζεται με ευριστικές μεθόδους
- Η λεγόμενη δυαδική μέθοδος: παραγωγή μίας άθροισης και μιας ολίσθησης για κάθε ψηφίο του n που είναι 1
- Παράδειγμα: Υπολογισμός του $n = 113 \times x = (1110001)_2 \times x$ [Lefèvre, 2001]

```
3x = (x << 1) + x;
7x = (3x << 1) + x;
n = 113x = (7x << 4) + x;
```

Εξουδετέρωση νεκρού κώδικα

- Απομάκρυνση περιττού κώδικα
- Αυτό συμβαίνει για παράδειγμα, όταν ανατίθενται τιμές σε μεταβλητές οι οποίες δεν διαβάζονται σε κανένα σημείο του προγράμματος

```
b = 3;
c = 1 + 3;
d = 3 + c;
```

```
c = 1 + 3;
d = 3 + c;
```

- Απομάκρυνση μη προσβάσιμου κώδικα

```
if (false) {
  a = 5;
}
```

```
if (false) {}
```

Εξουδετέρωση κοινής υποεκφράσεως

- Κοινή υποέκφραση αποτελεί κάθε έκφραση η οποία χρησιμοποιείται σε περισσότερα από ένα σημεία του προγράμματος
- Ο μετασχηματισμός αυτός αποβλέπει στην αντικατάσταση αυτών των εκφράσεων από ισοδύναμες μεταβλητές
- Παράδειγμα: χρήση της *tmp* στον παρακάτω κώδικα

```
if () {
  a = b;
  c = (b + e) * 1024;
  d = b + e;
  b = 7;
} else {
  x = a + c;
  b = 7;
}
return 7 + c + d;
```

```
if () {
  a = b;
  tmp = b + e;
  c = tmp * 1024;
  d = tmp;
  b = 7;
} else {
  x = a + c;
  b = 7;
}
return 7 + c + d;
```

Εσωγράμμιση συνάρτησης (1)

- Εσωγράμμιση συνάρτησης: αντικατάσταση μιας κλήσης σε υποπρόγραμμα (συνάρτηση) από το σώμα της συνάρτησης
- Εξουδετερώνει την επιβάρυνση από την κλήση υποπρογράμματος (πέρασμα ορισμάτων, μεταφορά καταχωρητών σε νέο πλαίσιο στη στοίβα, αποκατάσταση από προηγούμενο πλαίσιο της στοίβας)
- Διερύνει την εμβέλεια εφαρμογής άλλων βελτιστοποιήσεων
- Αυξάνει το μέγεθος κώδικα (με πιθανή αρνητική επίδραση στην κρυφή μνήμη)
- Ελέγχεται από απλά μετρικά κόστους
  - Μέγεθος κώδικα
  - Βάθος κλήσεως συναρτήσεων
  - Πληροφορία του προφίλ εκτέλεσης του προγράμματος
- Στην ANSI C χρησιμοποιείται η λέξη-κλειδί **inline**

Εσωγράμμιση συνάρτησης (2)

- Οι απόψεις για την ωφέλεια από την εφαρμογή της διαδικασίας εσωγράμμισης συνάρτησης διίστανται
  1. Σχεδόν πάντα είναι ωφέλιμη
  2. Περιστασιακά είναι ωφέλιμη, αλλά προκαλεί σημαντικά προβλήματα
  3. Προκαλεί απώλειες (misses) στην κρυφή μνήμη προγράμματος και για αυτό πρέπει να αποφεύγεται
- Πιο αντικειμενικές μελέτες δείχνουν ότι η πραγματικότητα βρίσκεται κάπου ανάμεσα: οι ευριστικές τεχνικές που λαμβάνουν υπόψη τις ιδιαιτερότητες του κάθε προγράμματος είναι και οι περισσότερο ωφέλιμες

```
a = power2(b);

power2(x) {
return x*x;
}
```

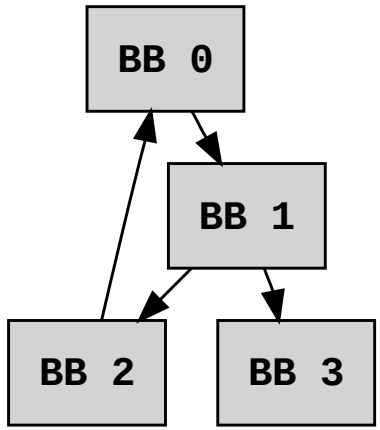
```
a = b * b;
```

Βελτιστοποίηση ροής ελέγχου

- Ζητούμενο: απλοποίηση της δομής του γράφου ροής ελέγχου (CFG)
- Κανονικοποίηση του CFG μετά την εφαρμογή μετασχηματισμών
- Ενεργοποίηση περαιτέρω μετασχηματισμών
- Βελτιστοποίηση κώδικα βρόχων
  - Ιδιαίτερης σημασίας καθώς μεγάλο ποσοστό του χρόνου εκτέλεσης ενός προγράμματος δαπανάται σε κώδικα εντός βρόχων
  - Πρώτο βήμα είναι η αναγνώριση των βρόχων στο πρόγραμμα
  - Σε δεύτερο βήμα εφαρμόζονται κατάλληλοι μετασχηματισμοί

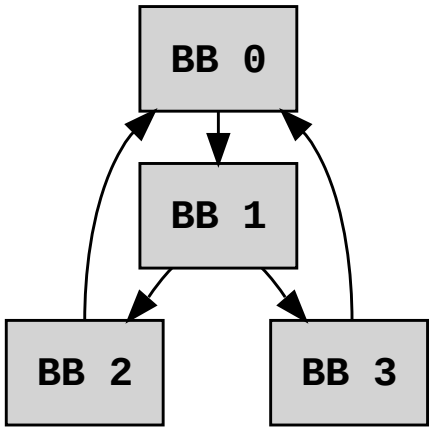
Η δομή ενός βρόχου (1)

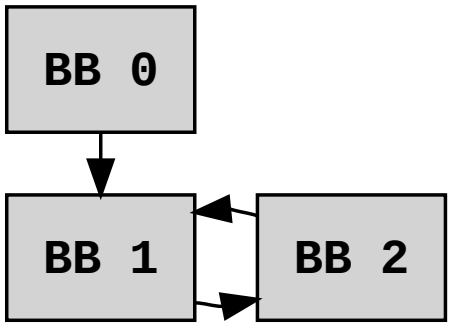
- Βρόχος: τμήμα επαναλαμβανόμενου κώδικα
- Κορυφές: σύνολο από βασικά μπλοκ
- Αποτελείται από την κεφαλίδα του βρόχου (loop header) και το σώμα του (loop body)
- Κεφαλίδα: η κορυφή από την οποία διέρχονται όλες οι επαναλήψεις του βρόχου
- Η οπισθόδρομη ακμή (back edge) επαναφέρει τη ροή ελέγχου στην κεφαλίδα του βρόχου



Η δομή ενός βρόχου (2)

- Φυσικός βρόχος (natural loop): βρόχος με ένα σημείο εισόδου (entry point) και ένα σημείο εξόδου (exit point)
- Οι περισσότεροι αλγόριθμοι βελτιστοποίησης εργάζονται μόνο σε φυσικούς βρόχους
- Μειώσιμος γράφος ροής ελέγχου (reducible CFG): CFG το οποίο περιλαμβάνει μόνο φυσικούς βρόχους
- Μη φυσικοί (γενικευμένοι) βρόχοι: οφείλονται σε αδόμητο κώδικα (χρήση goto)





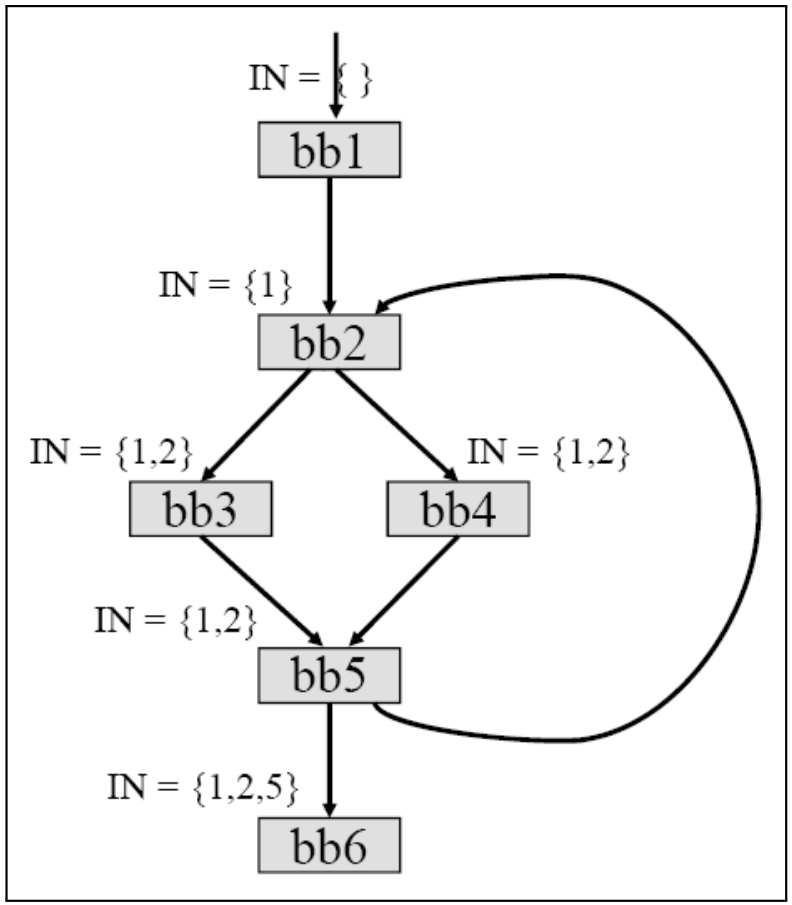
Η έννοια της κυριαρχίας (dominance) σε ένα CFG

- Δοθέντος ενός CFG με τα βασικά μπλοκ X, Y, Z και S, όπου S το BB εισόδου στο CFG
  - Κυριαρχία του X ως προς το Y: $X \geq Y$
    Κάθε μονοπάτι από το S για το Y διέρχεται από το X
  - Αυστηρή κυριαρχία (strict dominance): $X > Y$
    $X > Y \quad if \quad X \geq Y \quad \wedge \quad X \neq Y$
  - Άμεσος κυρίαρχος (immediate dominator): $idom(X)$
    $X = idom(Y) \quad if \quad X > Y \quad \wedge \quad \nexists \quad Z : X > Z > Y$
- Πόρισμα: αν οι κόμβοι $d1$ και $d2$ κυριαρχούν έναντι του m τότε:
  - ο $d1$ κυριαρχεί του $d2$, ή
  - ο $d2$ κυριαρχεί του $d1$

Υπολογισμός κυρίαρχων κόμβων

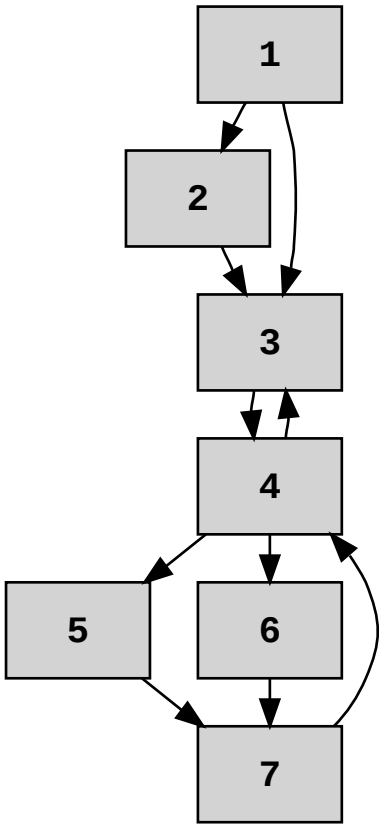
Εξισώσεις για τον υπολογισμό για κάθε βασικό μπλοκ (b_k είναι το τρέχον βασικό μπλοκ):

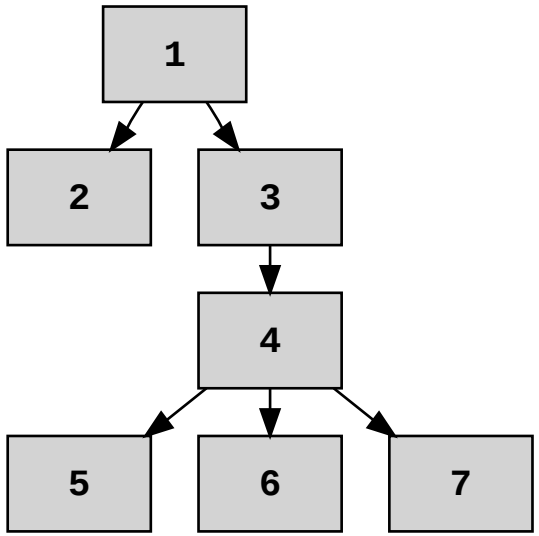
$$
\begin{aligned}
GEN &= \{b_k\} \\
KILL &= \{\emptyset\} \\
OUT &= GEN \bigcup (IN - KILL) \\
IN &= \bigcap (OUT)
\end{aligned}
\tag{1}
$$



Δένδρο κυριαρχίας (dominator tree)

- Δομή δένδρου έχοντας ως ρίζα το βασικό μπλοκ εισόδου
- Οι κορυφές του αποτελούν βασικά μπλοκ του CFG
- Υπάρχει ακμή από την κορυφή d στην n εφόσον n d είναι ο άμεσος κυρίαρχος της n





Αναγνώριση βρόχων

- Μοναδικό σημείο εισόδου (κεφαλίδα)
- Τουλάχιστον ένα μονοπάτι οδηγεί πίσω στην κορυφή κεφαλίδα
- Εύρεση ακμών των οποίων οι κεφαλές κυριαρχούν των ουρών τους
  - Αυτές οι ακμές αποτελούν οπισθόδρομες ακμές βρόχων
  - Δοθείσης μίας ακμής $n \rightarrow d$
  - Ένας βρόχος αποτελείται από την n συν όλες τις κορυφές που μπορούν να φτάσουν στην n χωρίς το αντίστοιχο μονοπάτι να διέρχεται από την d (είναι δηλαδή όλοι οι κόμβοι μεταξύ των d και n)
  - ο d αποτελεί κεφαλίδα του βρόχου

Ο αλγόριθμος αναγνώρισης βρόχων

```
INSERT(m)
    if m ∉ loop then
      loop = loop ⋃ {m};
      push m onto stack;

LOOP(d, n)
    loop = ∅;
    stack = ∅;
    INSERT(n);
    while stack not empty do
      m = pop stack;
      for all p ∈ pred(m) do
        INSERT(p);
```

Βελτιστοποίηση άλματος (1)

- Βελτιστοποίηση δηλώσεων άλματος στον κώδικα με απλοποίηση πλεονασμών και εξουδετέρωση περιττού κώδικα
- Περιπτώσεις
  - Ένα άλμα χωρίς συνθήκη που προκαλεί τη μετάβαση σε άλλη δήλωση άλματος χωρίς συνθήκη, αντικαθίσταται από άλμα στη διεύθυνση-στόχο του δεύτερου άλματος
  - Το ίδιο ισχύει και για μετάβαση από αρχική δήλωση άλματος υπό συνθήκη
  - Ένα άλμα χωρίς συνθήκη σε άλμα υπό συνθήκη αντικαθίσταται από ένα αντίγραφο του άλματος χωρίς συνθήκη
  - Ένα άλμα υπό συνθήκη σε δήλωση άλματος υπό συνθήκη αντικαθίσταται από άλμα υπό συνθήκη με κατάλληλη έκφραση υπολογισμού της συνθήκης σε TRUE ή FALSE

Βελτιστοποίηση άλματος (2)

- Παράδειγμα 1

```
if (a == 0) goto L1;
...
L1:
  if (a >= 0) goto L2;
...
L2:
```

```
if (a == 0) goto L2;
...
L1:
  if (a >= 0) goto L2;
...
L2:
```

- Παράδειγμα 2

```
if (a == 0) goto L1;
goto L2;
...
L1:
```

```
if (a != 0) goto L2;
L1:
```

Μετακίνηση αμετάβλητου κώδικα βρόχου (loop-invariant code motion)

- Μεταφορά ενός υπολογισμού εκτός του σώματος βρόχου, όταν το αποτέλεσμά του δεν μεταβάλλεται στις διαφορετικές επαναλήψεις του βρόχου
- Μπορεί να εφαρμοστεί και σε χαμηλό επίπεδο για τη βελτιστοποίηση εκφράσεων διευθυνσιοδότησης

```
int x;
...
for (i = 1; i <= n; i++)
{
  a[i] = a[i] + sqrt(x);
}
```

```
int x;
...
if (n > 0)
{
  c = sqrt(x);
}
for (i = 1; i <= n; i++)
{
  a[i] = a[i] + c;
}
```

Εξουδετέρωση επαγόμενης μεταβλητής (induction variable elimination)

- Επαγόμενη μεταβλητή: μία μεταβλητή της οποίας η τιμή λαμβάνεται από τον αριθμό επαναλήψεων του περικλείοντος βρόχου
- Παράδειγμα επαγόμενης μεταβλητής: ο δείκτης βρόχου
- Η βελτιστοποίηση αφορά την αντικατάσταση δύο ή περισσότερων επαγόμενων μεταβλητών από μόνο μία

```
int a[SIZE], b[SIZE];

void f(void)
{
  int i1, i2, i3;

  for (i1 = 0, i2 = 0, i3 = 0;
    i1 < SIZE; i1++)
  {
    a[i2++] = b[i3++];
  }
}
```

```
int a[SIZE], b[SIZE];

void f(void)
{
  int i1;

  for (i1 = 0; i1 < SIZE; i1++)
  {
    a[i1] = b[i1];
  }
}
```

Το βελτιστοποιητικό frontend LANCE v2.0

- Περιβάλλον frontend για τη γλώσσα C υλοποιημένο σε C++
- Ενδιάμεση αναπαράσταση τύπου IR-C (υποσύνολο της C)
- Περάσματα βελτιστοποίησης της IR ανεξάρτητα της αρχιτεκτονικής
- Ανάλυση ροής ελέγχου και δεδομένων
- Διεπαφή προγραμματισμού API: `lance2.h` (header file) και `liblance2.a` (στατική βιβλιοθήκη)
- Ανάπτυξη backend με τα εργαλεία IBURG και OLIVE
- Πλήρης υποστήριξη της γλώσσας C (C89)

Βελτιστοποιήσεις του LANCE

- **cfold**: constant folding
- **constprop**: constant propagation
- **copyprop**: copy propagation
- **cse**: common subexpression elimination
- **dce**: dead code elimination
- **jmpopt**: jump optimization
- **licm**: loop invariant code motion
- **ive**: induction variable elimination

LANCE: Γέννηση αρχικού ενδιάμεσου κώδικα

```
void main()
{
  int i, A[100];

  for (i = 0; i < 100; i++)
  {
    A[2] = 2;
    A[i] = i;
  }
}
```

```
/* for (i=0; i<100; i++) */
    i_3 = 0;
    t1 = i_3 < 100;
    t12 = !t1;
    if (t12) goto LL1;

/* A[2] = 2; */
LL3:
    t6 = (char *)A_4;
    t5 = 2 * 4;
    t4 = t6 + t5;
    t7 = (int *)t4;
    *t7 = 2;

/* A[i] = i; */
    t10 = (char *)A_4;
    t9 = i_3 * 4;
    t8 = t10 + t9;
    t11 = (int *)t8;
    *t11 = i_3;

/* for (i=0; i<100; i++) */
LL2:
    t2 = i_3;
    t3 = t2 + 1;
    i_3 = t3;
    t1 = i_3 < 100;
    if (t1) goto LL3;

LL1:
```

LANCE: Δίπλωση σταθεράς (cfold)

```
/* for (i=0; i<100; i++) */
    i_3 = 0;
    t1 = i_3 < 100;
    t12 = !t1;
    if (t12) goto LL1;

/* A[2] = 2; */
LL3:
    t6 = (char *)A_4;
    t5 = 2 * 4;
    t4 = t6 + t5;
    t7 = (int *)t4;
    *t7 = 2;

/* A[i] = i; */
    t10 = (char *)A_4;
    t9 = i_3 * 4;
    t8 = t10 + t9;
    t11 = (int *)t8;
    *t11 = i_3;

/* for (i=0; i<100; i++) */
LL2:
    t2 = i_3;
    t3 = t2 + 1;
    i_3 = t3;
    t1 = i_3 < 100;
    if (t1) goto LL3;

LL1:
```

```
/* for (i=0; i<100; i++) */
    i_3 = 0;
    t1 = i_3 < 100;
    t12 = !t1;
    if (t12) goto LL1;

/* A[2] = 2; */
LL3:
    t6 = (char *)A_4;
    t5 = 8;
    t4 = t6 + t5;
    t7 = (int *)t4;
    *t7 = 2;

/* A[i] = i; */
    t10 = (char *)A_4;
    t9 = i_3 * 4;
    t8 = t10 + t9;
    t11 = (int *)t8;
    *t11 = i_3;

/* for (i=0; i<100; i++) */
LL2:
    t2 = i_3;
    t3 = t2 + 1;
    i_3 = t3;
    t1 = i_3 < 100;
    if (t1) goto LL3;

LL1:
```

LANCE: Διάδοση σταθεράς (constprop)

```
/* for (i=0; i<100; i++) */
    i_3 = 0;
    t1 = i_3 < 100;
    t12 = !t1;
    if (t12) goto LL1;

/* A[2] = 2; */
LL3:
    t6 = (char *)A_4;
    t5 = 8;
    t4 = t6 + t5;
    t7 = (int *)t4;
    *t7 = 2;

/* A[i] = i; */
    t10 = (char *)A_4;
    t9 = i_3 * 4;
    t8 = t10 + t9;
    t11 = (int *)t8;
    *t11 = i_3;

/* for (i=0; i<100; i++) */
LL2:
    t2 = i_3;
    t3 = t2 + 1;
    i_3 = t3;
    t1 = i_3 < 100;
    if (t1) goto LL3;

LL1:
```

```
/* for (i=0; i<100; i++) */
    i_3 = 0;
    t1 = 0 < 100;
    t12 = !t1;
    if (t12) goto LL1;

/* A[2] = 2; */
LL3:
    t6 = (char *)A_4;
    t5 = 8;
    t4 = t6 + 8;
    t7 = (int *)t4;
    *t7 = 2;

/* A[i] = i; */
    t10 = (char *)A_4;
    t9 = i_3 * 4;
    t8 = t10 + t9;
    t11 = (int *)t8;
    *t11 = i_3;

/* for (i=0; i<100; i++) */
LL2:
    t2 = i_3;
    t3 = t2 + 1;
    i_3 = t3;
    t1 = i_3 < 100;
    if (t1) goto LL3;

LL1:
```

LANCE: Διάδοση αντιγράφου (copyprop)

```
/* for (i=0; i<100; i++) */
    i_3 = 0;
    t1 = 0 < 100;
    t12 = !t1;
    if (t12) goto LL1;

/* A[2] = 2; */
LL3:
    t6 = (char *)A_4;
    t5 = 8;
    t4 = t6 + 8;
    t7 = (int *)t4;
    *t7 = 2;

/* A[i] = i; */
    t10 = (char *)A_4;
    t9 = i_3 * 4;
    t8 = t10 + t9;
    t11 = (int *)t8;
    *t11 = i_3;

/* for (i=0; i<100; i++) */
LL2:
    t2 = i_3;
    t3 = t2 + 1;
    i_3 = t3;
    t1 = i_3 < 100;
    if (t1) goto LL3;

LL1:
```

```
/* for (i=0; i<100; i++) */
    i_3 = 0;
    t1 = 0 < 100;
    t12 = !t1;
    if (t12) goto LL1;

/* A[2] = 2; */
LL3:
    t6 = (char *)A_4;
    t5 = 8;
    t4 = t6 + 8;
    t7 = (int *)t4;
    *t7 = 2;

/* A[i] = i; */
    t10 = (char *)A_4;
    t9 = i_3 * 4;
    t8 = t10 + t9;
    t11 = (int *)t8;
    *t11 = i_3;

/* for (i=0; i<100; i++) */
LL2:
    t2 = i_3;
    t3 = i_3 + 1;
    i_3 = t3;
    t1 = t3 < 100;
    if (t1) goto LL3;

LL1:
```

LANCE: Εξουδετέρωση κοινής υποεκφράσεως (cse)

```
/* for (i=0; i<100; i++) */
     i_3 = 0;
     t1 = 0 < 100;
     t12 = !t1;
     if (t12) goto LL1;

/* A[2] = 2; */
LL3:
     t6 = (char *)A_4;
     t5 = 8;
     t4 = t6 + 8;
     t7 = (int *)t4;
     *t7 = 2;

/* A[i] = i; */
     t10 = (char *)A_4;
     t9 = i_3 * 4;
     t8 = t10 + t9;
     t11 = (int *)t8;
     *t11 = i_3;

/* for (i=0; i<100; i++) */
LL2:
     t2 = i_3;
     t3 = i_3 + 1;
     i_3 = t3;
     t1 = t3 < 100;
     if (t1) goto LL3;

LL1:
```

```
/* for (i=0; i<100; i++) */
     i_3 = 0;
     t1 = 0 < 100;
     t12 = !t1;
     if (t12) goto LL1;

/* A[2] = 2; */
LL3:
     t13 = (char *)A_4;
     t6 = t13;
     t5 = 8;
     t4 = t6 + 8;
     t7 = (int *)t4;
     *t7 = 2;

/* A[i] = i; */
     t10 = t13;
     t9 = i_3 * 4;
     t8 = t10 + t9;
     t11 = (int *)t8;
     *t11 = i_3;

/* for (i=0; i<100; i++) */
LL2:
     t2 = i_3;
     t3 = i_3 + 1;
     i_3 = t3;
     t1 = t3 < 100;
     if (t1) goto LL3;
```

LANCE: Εξουδετέρωση νεκρού κώδικα (dce)

```
/* for (i=0; i<100; i++) */
     i_3 = 0;
     t1 = 0 < 100;
     t12 = !t1;
     if (t12) goto LL1;

/* A[2] = 2; */
LL3:
     t13 = (char *)A_4;
     t6 = t13;
     t5 = 8;
     t4 = t6 + 8;
     t7 = (int *)t4;
     *t7 = 2;

/* A[i] = i; */
     t10 = t13;
     t9 = i_3 * 4;
     t8 = t10 + t9;
     t11 = (int *)t8;
     *t11 = i_3;

/* for (i=0; i<100; i++) */
LL2:
     t2 = i_3;
     t3 = i_3 + 1;
     i_3 = t3;
     t1 = t3 < 100;
     if (t1) goto LL3;
```

```
/* for (i=0; i<100; i++) */
     i_3 = 0;
     t1 = 0 < 100;
     t12 = !t1;
     if (t12) goto LL1;

/* A[2] = 2; */
LL3:
     t13 = (char *)A_4;
     t6 = t13;
     t4 = t6 + 8;
     t7 = (int *)t4;
     *t7 = 2;

/* A[i] = i; */
     t10 = t13;
     t9 = i_3 * 4;
     t8 = t10 + t9;
     t11 = (int *)t8;
     *t11 = i_3;

/* for (i=0; i<100; i++) */
LL2:
     t3 = i_3 + 1;
     i_3 = t3;
     t1 = t3 < 100;
     if (t1) goto LL3;

LL1:
```

LANCE: Βελτιστοποίηση άλματος (jmpopt)

```
/* for (i=0; i<100; i++) */
    i_3 = 0;
    t1 = 0 < 100;
    t12 = !t1;
    if (t12) goto LL1;

/* A[2] = 2; */
LL3:
    t13 = (char *)A_4;
    t6 = t13;
    t4 = t6 + 8;
    t7 = (int *)t4;
    *t7 = 2;

/* A[i] = i; */
    t10 = t13;
    t9 = i_3 * 4;
    t8 = t10 + t9;
    t11 = (int *)t8;
    *t11 = i_3;

/* for (i=0; i<100; i++) */
LL2:
    t3 = i_3 + 1;
    i_3 = t3;
    t1 = t3 < 100;
    if (t1) goto LL3;

LL1:
```

```
/* for (i=0; i<100; i++) */
    i_3 = 0;
    t1 = 0 < 100;
    t12 = !t1;
    if (t12) goto LL1;

/* A[2] = 2; */
LL3:
    t13 = (char *)A_4;
    t6 = t13;
    t4 = t6 + 8;
    t7 = (int *)t4;
    *t7 = 2;

/* A[i] = i; */
    t10 = t13;
    t9 = i_3 * 4;
    t8 = t10 + t9;
    t11 = (int *)t8;
    *t11 = i_3;

/* for (i=0; i<100; i++) */
    t3 = i_3 + 1;
    i_3 = t3;
    t1 = t3 < 100;
    if (t1) goto LL3;

LL1:
```

LANCE: Κίνηση αμετάβλητου κώδικα βρόχου (licm)

```
/* for (i=0; i<100; i++) */
     i_3 = 0;
     t1 = 0 < 100;
     t12 = !t1;
     if (t12) goto LL1;

/* A[2] = 2; */
LL3:
     t13 = (char *)A_4;
     t6 = t13;
     t4 = t6 + 8;
     t7 = (int *)t4;
     *t7 = 2;

/* A[i] = i; */
     t10 = t13;
     t9 = i_3 * 4;
     t8 = t10 + t9;
     t11 = (int *)t8;
     *t11 = i_3;

/* for (i=0; i<100; i++) */
     t3 = i_3 + 1;
     i_3 = t3;
     t1 = t3 < 100;
     if (t1) goto LL3;

LL1:
```

```
/* for (i=0; i<100; i++) */
     i_3 = 0;
     t1 = 0 < 100;
     t12 = !t1;
     if (t12) goto LL1;

/* A[2] = 2; */
LL3:
     t13 = (char *)A_4;
     t6 = t13;
     t4 = t6 + 8;
     t7 = (int *)t4;

/* A[i] = i; */
     t10 = t13;

/* for (i=0; i<100; i++) */
LL2:

/* A[2] = 2; */
     *t7 = 2;

/* A[i] = i; */
     t9 = i_3 * 4;
     t8 = t10 + t9;
     t11 = (int *)t8;
     *t11 = i_3;
```

```
/* for (i=0; i<100; i++) */
     t3 = i_3 + 1;
     i_3 = t3;
     t1 = t3 < 100;
     if (t1) goto LL2;

LL1:
```

LANCE: Εξουδετέρωση επαγόμενης μεταβλητής (ive)

```
/* for (i=0; i<100; i++) */
    i_3 = 0;
    t1 = 0 < 100;
    t12 = !t1;
    if (t12) goto LL1;

/* A[2] = 2; */
LL3:
    t13 = (char *)A_4;
    t6 = t13;
    t4 = t6 + 8;
    t7 = (int *)t4;

/* A[i] = i; */
    t10 = t13;

/* for (i=0; i<100; i++) */
LL2:

/* A[2] = 2; */
    *t7 = 2;

/* A[i] = i; */
    t9 = i_3 * 4;
    t8 = t10 + t9;
    t11 = (int *)t8;
    *t11 = i_3;
```

```
/* for (i=0; i<100; i++) */
    t3 = i_3 + 1;
    i_3 = t3;
    t1 = t3 < 100;
    if (t1) goto LL2;

LL1:
```

```
/* for (i=0; i<100; i++) */
    i_3 = 0;

/* A[2] = 2; */
    t13 = (char *)A_4;
    t4 = t13 + 8;
    t7 = (int *)t4;

/* for (i=0; i<100; i++) */
    t24 = 0;
    t29 = 1;
LL4:

/* A[2] = 2; */
    *t7 = 2;

/* A[i] = i; */
    t8 = t13 + t24;
    t11 = (int *)t8;
    *t11 = i_3;

/* for (i=0; i<100; i++) */
    t3 = t29;
    i_3 = i_3 + 1;
    t29 = t29 + 1;
    t24 = t24 + 4;
    t1 = t3 < 100;
    if (t1) goto LL4;
```

Αναφορές του μαθήματος Ι


A. V. Aho, R. Sethi, and J. D. Ullman, *Μεταγλωττιστές: Αρχές, Τεχνικές και Εργαλεία*, με την επιμέλεια των: Άγγελος Σπ. Βώρος και Νικόλαος Σπ. Βώρος και Κων/νος Γ. Μασσέλος, **κεφάλαια 9, 9.1, 9.4–9.6**, Εκδόσεις Νέων Τεχνολογιών, 2008. Website for the English version: `http://dragonbook.stanford.edu`


D. F. Bacon, S. L. Graham, and O. J. Sharp, "Compiler transformations for high-performance computing," *ACM Computing Surveys*, vol. 26, no. 4, pp. 345–420, December 1994.


V. Lefèvre, "Multiplication by an integer constant," INRIA Institute, Technical report No. 4192, May 2001.


R. Leupers, O. Whalen, M. Hahenauer, T. Kogel, and P. Marwedel, "An executable intermediate representation for retargetable compilation and high-level code optimization," in *Proceedings of the Third International Workshop on Systems, Architectures, Modeling, and Simulation (SAMOS 2003)*, Samos, Greece, July 21-23 2003, pp. 120–125.

Αναφορές του μαθήματος II


LANCE C compiler. [Online]. Available:
http://www.lancecompiler.com