Προηγμένα Θέματα Θεωρητικής Πληροφορικής

Βελτιστοποιήσεις για την εκμετάλλευση της παραλληλίας και ενίσχυση της τοπικότητας (II)

Νικόλαος Καββαδίας
nkavv@uop.gr

26 Μαΐου 2010

Μετασχηματισμοί βρόχου (loop transformations)

- Loop restructuring transformations
  - Loop unrolling (previous lecture)
  - Loop coalescing
  - Loop collapsing
  - Software pipelining
  - Loop peeling
  - Loop normalization (canonicalization)
  - Loop spreading
- Loop replacement transformations
  - Reduction recognition
  - Loop idiom recognition
  - Array statement scalarization

Loop coalescing (συνάσπιση βρόχων)

- Συνδυάζει μία δομή βρόχου ώστε τη μετατροπή της σε έναν απλό βρόχο με τις αρχικές μεταβλητές δείκτη να υπολογίζονται από μία επαγόμενη μεταβλητή
- Μπορεί να βελτιώσει το χρονοπρογραμματισμό των εντολών του βρόχου σε μία παράλληλη αρχιτεκτονική (πολυεπεξεργαστή)
- Ως μετασχηματισμός είναι πάντοτε επιτρεπτός καθώς δεν μεταβάλλει τη διαδοχή των επαναλήψεων στο βρόχο

- Αρχική δομή βρόχων

```
for (i = 1; i <= n; i++) {
  for (j = 1; j <= m; j++) {
    a[i][j] = a[i][j] + c;
  }
}
```

- Συνασπισμένη δομή βρόχων

```
for (T = 1; T <= n*m; T++) {
  i = ((T-1) / m)*m + 1;
  j = MOD(T-1, m) + 1;
  a[i][j] = a[i][j] + c;
}
```


Loop collapsing (κατάρρευση βρόχων)

- Αποτελεί απλούστερη και περισσότερο εξειδικευμένη μορφή της συνάσπισης βρόχων
- Χρησιμοποιείται και για την αύξηση του μήκους των διανυσμάτων που υπεισέρχονται στους υπολογισμούς
- Προτιμάται μόνο για βρόχους με σταθερή τιμή βήματος (stride)

- Αρχική δομή βρόχων

```
for (i = 1; i <= n; i++) {
  for (j = 1; j <= m; j++) {
    a[i][j] = a[i][j] + c;
  }
}
```

- Τελικός βρόχος

```
real TA[n*m];
equivalence (TA, a);
parfor (T = 1; T <= n*m; T++) {
  TA[T] = TA[T] + c;
}
```

Loop peeling (ξεφλούδισμα βρόχου)

- Κατά το μετασχηματισμό αυτό, απομακρύνεται μία ομάδα επαναλήψεων από την αρχή ή το τέλος του βρόχου και εκτελείται ξεχωριστά
- Χρησιμοποιείται για την απομάκρυνση εξαρτήσεων με πρώτες ή τελευταίες επαναλήψεις του βρόχου
- Επίσης και για την ταύτιση της συνθήκης ελέγχου με αυτή τυχόν γειτονικών βρόχων, προκειμένου τη συνένωση όλων σε μία ενιαία δομή βρόχου
- Μπορεί να χρησιμοποιηθεί σε κάθε περίπτωση

```
for (i = 2; i <= n; i++) {
  b[i] = b[i] + b[2];
}
parfor (i = 3; i <= n; i++) {
    a[i] = a[i] + c;
}
```

```
if (n >= 2) {
  b[2] = b[2] + b[2];
}
parfor (i = 3; i <= n; i++) {
  b[i] = b[i] + b[2];
  a[i] = a[i] + c;
}
```


Loop normalization/canonicalization (κανονικοποίηση βρόχου)

- Μετατρέπει όλους τους βρόχους έτσι ώστε η επαγόμενη μεταβλητή να έχει αρχικό όριο 1 ή 0 και να αυξάνεται κατά stride = 1 σε κάθε επανάληψη
- Χρησιμοποιείται για την αποκάλυψη ευκαιριών για loop peeling και fusion
- Επιτρέπει στο μεταγλωττιστή την ευκολότερη ανάλυση μεταβλητών δείκτη (array subscript analysis)

- Αρχική δομή βρόχων

```
for (i = 1; i <= n; i++) {
  a[i] = a[i] + c;
}
for (i = 2; i <= n+1; i++) {
    b[i] = a[i-1] * b[i];
}
```

- Οι βρόχοι πλέον μπορούν να συνενωθούν

```
for (i = 1; i <= n; i++) {
  a[i] = a[i] + c;
}
for (i = 1; i <= n; i++) {
    b[i+1] = a[i] * b[i+1];
}
```


Loop idiom recognition (αναγνώριση ιδιώματος βρόχου)

- Αναγνώριση της δομής βρόχου στο επίπεδο HLIR
- Αντικατάσταση ή εξουδετέρωση των υπολογισμών που σχετίζονται με τις λειτουργίες ελέγχου του βρόχου
  - αύξηση του δείκτη, σύγκριση με την τιμή τελικού ορίου, και εκτέλεση της εντολής διακλάδωσης
- Μηχανισμοί μηδενικής επιβάρυνσης βρόχου (zero overhead looping) στο υλικό για τη μείωση κύκλων εκτέλεσης
- Χρησιμοποιούνται εκτενώς σε DSP processors όπως οι Motorola (DSP56300) και TI C62xx
- Παράδειγμα υλοποίησης τέτοιας μονάδας σε VHDL: Hardware Looping Unit από τον υποφαινόμενο
- Zero-Overhead Loop Controller (έρευνα του ιδίου)

Array statement scalarization (βαθμωτοποίηση έκφρασης πίνακα)

- Ο μεταγλωττιστής μπορεί να μετατρέψει μία διανυσματική έκφραση, σε αντίστοιχες διανυσματικές εντολές χαμηλού επιπέδου ή σε σειριακούς βρόχους

- Αρχική έκφραση

```
a[2:n-1] = a[2:n-1] + a[1:n-2];
```

- Λανθασμένη βαθμωτοποίηση

```
for (i = 2; i <= n-1; i++) {
  a[i] = a[i] + a[i-1];
}
```

- Ορθή βαθμωτοποίηση

```
for (i = 2; i <= n-1; i++) {
  T[i] = a[i] + a[i-1];
}
for (i = 2; i <= n-1; i++) {
  a[i] = T[i];
}
```

- Αναστροφή των 2 βρόχων (ενεργοποίηση συνένωσης, απαλοιφή του T)

```
for (i = n-1; i >= 2; i--) {
  a[i] = a[i] + a[i-1];
}
```

Μετασχηματισμοί προσπέλασης μνήμης (memory access transformations)

- Ο επεξεργαστής (CPU) και η κύρια μνήμη (συχνά τεχνολογίας SDRAM) λειτουργούν σε διαφορετική συχνότητα ρολογιού
- Παράγοντες που επιδρούν στις επιδόσεις του συστήματος μνήμης
  - Η επαναχρησιμοποίηση, δηλαδή η αναλογία του αριθμού χρήσεων ενός αντικειμένου ως προς τον αριθμό φορτώσεών του, που συμβολίζεται με Q
  - Ο βαθμός παραλληλίας: οι διανυσματικοί επεξεργαστές υποδιαιρούν τη μνήμη σε συστοιχίες (banks) και επιτρέπουν τη χρήση διανυσματικών επεξεργαστών με μία εντολή
  - Το μέγεθος του συνόλου εργασίας (working set size): Αν όλα τα στοιχεία μνήμης που προσπελαύνονται κατά την εκτέλεση ενός βρόχου προγράμματος, δεν χωρούν στην κρυφή μνήμη δεδομένων, τότε κάποια από αυτά τα αντικείμενα διαγράφονται και πρέπει να ξαναφορτωθούν από την κύρια μνήμη. Αυτό προκαλεί μείωση του Q

loop unrolling, loop fusion

Οφέλη από την εφαρμογή μετασχηματισμών προσπέλασης μνήμης

- Array padding
- Scalar expansion
- Array contraction
- Scalar replacement
- Code colocation
- Displacement minimization

Array padding (γέμισμα πίνακα)

- Μετασχηματισμός ο οποίος προσθέτει ελεύθερες θέσεις μνήμης σε έναν υπάρχοντα πίνακα
- Χρήσεις
  - Διόρθωση περιπτώσεων συγκρούσεων στο σύστημα μνήμης
  - Προσαρμογή του πίνακα στις διαστάσεις ενός μπλοκ ή μιας σειράς της κρυφής μνήμης
- Παράδειγμα
- Αρχικός κώδικας

```
real a[7][512];
for (i = 1; i <= 512; i++) {
  a[1][i] = a[1][i] + c;
}
```

- Μετά το μετασχηματισμό για διανυσματική εκτέλεση

```
real a[8][512];
for (i = 1; i <= 512; i++) {
  a[1][i] = a[1][i] + c;
}
```

- Βελτιστοποίηση για μήκη βήματος που είναι δυνάμεις του δύο

Scalar expansion (βαθμωτό ανάπτυγμα)

- Βαθμωτές μεταβλητές σε ένα σώμα βρόχου δημιουργούν μία αντι-εξάρτηση με την επόμενη επανάληψη
- Καταμερισμός μιας προσωρινής μεταβλητής για κάθε επανάληψη ξεχωριστά
- Με τον τρόπο αυτό ενεργοποιούνται οι δυνατότητες παραλληλοποίησης
- Απαραίτητη προϋπόθεση είναι η διαθεσιμότητα πολύ μεγάλου αριθμού καταχωρητών
- Αρχικός βρόχος

```
for (i = 1; i <= n; i++) {
  c    = b[i];
  a[i] = a[i] + c;
}
```

- Μετά το μετασχηματισμό για διανυσματική εκτέλεση

```
real T[n];
parfor (i = 1; i <= n; i++) {
  T[i] = b[i];
  a[i] = a[i] + T[i];
}
```

Array contraction (συστολή πίνακα)

- Συρρίκνωση του αριθμού των στοιχείων και ενδεχόμενα και του αριθμού των διαστάσεων σε πίνακες
- Αρχικός βρόχος

```
real T[n][n];
for (i = 1; i <= n; i++) {
  parfor (j = 1; j <= n; j++) {
    T[i][j] = a[i][j] * 3;
    b[i][j] = T[i][j] + b[i][j]/T[i][j];
  }
}
```

- Μετά το μετασχηματισμό συστολής πίνακα

```
real T[n];
for (i = 1; i <= n; i++) {
  parfor (j = 1; j <= n; j++) {
    T[j] = a[i][j] * 3;
    b[i][j] = T[j] + b[i][j]/T[j];
  }
}
```


Scalar replacement (βαθμωτή αντικατατάσταση)

- Αντίστοιχος μετασχηματισμός με τη συστολή πίνακα για την περίπτωση που ένας συχνά αναφερόμενος πίνακας είναι αμετάβλητος ως προς τις επαναλήψεις ενός εσώτερου βρόχου
- Αντικαθιστά τις αναφορές σε στοιχείο του πίνακα από βαθμωτή μεταβλητή

- Αρχικός βρόχος

```
for (i = 1; i <= n; i++) {
  for (j = 1; j <= n; j++) {
    total[i] =
      total[i] + a[i][j];
  }
}
```

- Μετά από βαθμωτή αντικατάσταση

```
for (i = 1; i <= n; i++) {
  T = total[i];
  for (j = 1; j <= n; j++) {
    T = T + a[i][j];
  }
  total[i] = T;
}
```

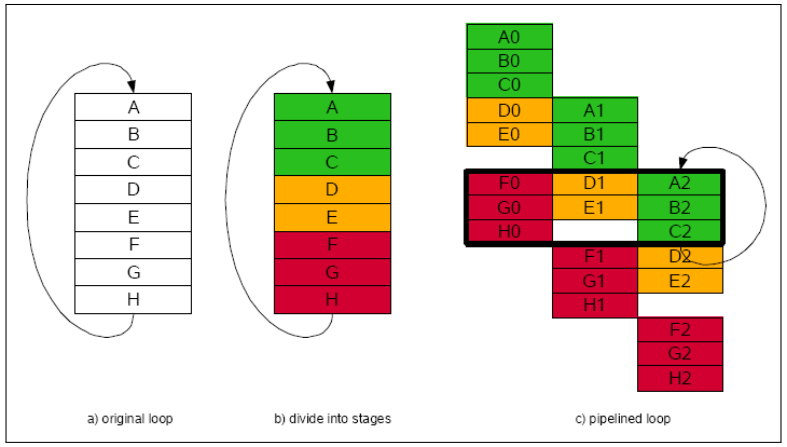

Χρονοπρογραμματισμός σε κυκλικές περιοχές

- Τεχνική χρονοπρογραμματισμού εξειδικευμένη σε κυκλικές περιοχές (βρόχοι)
- Τα περισσότερα προγράμματα εφαρμογών στην πράξη δαπανούν το μεγαλύτερο μέρος του χρόνου εκτέλεσής τους σε σώματα βρόχων
- Η βελτιστοποίηση των επιδόσεων με εκμετάλλευση της παραλληλίας του επεξεργαστή για τμήματα κώδικα βρόχου απαιτεί εξειδικευμένες τεχνικές
  - Αύξηση του μεγέθους του βρόχου με ξετύλιγμα
  - Υπέρθεση (overlapping) της εκτέλεσης διαφορετικών (διαδοχικών) επαναλήψεων παράλληλα
  - Εκτέλεση τμημάτων από διαφορετικές επαναλήψεις παράλληλα
  - Όταν αυτές οι τεχνικές εφαρμόζονται, οδηγούν σε σημαντικές βελτιώσεις
- ☣ Δεν μπορεί να χειριστεί κάθε είδος βρόχου

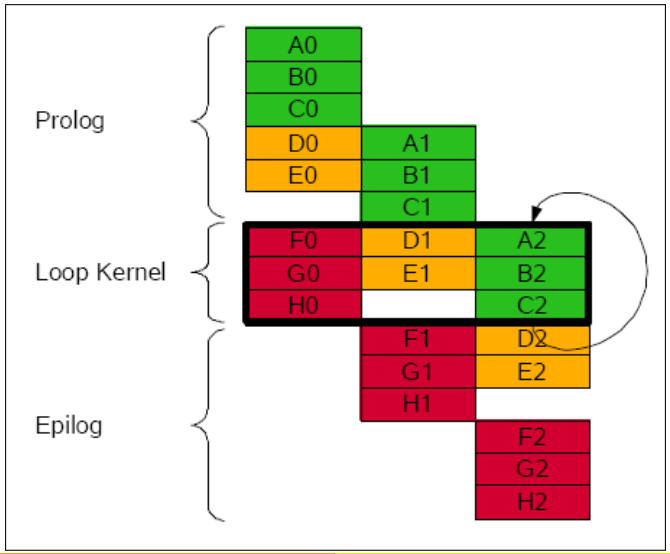
Software pipelining (Λογισμική διοχέτευση)

- Η τεχνική χρονοπρογραμματισμού με software pipelining περιγράφει μία οικογένεια από αλγορίθμους που εφαρμόζονται σε κυκλικές περιοχές κώδικα
- Υποδιαιρούν ένα βρόχο σε επιμέρους στάδια
- Τα στάδια εκτελούνται περιλαμβάνοντας διαφορετικές επαναλήψεις σε παράλληλη επεξεργασία
- Στην ουσία χωρίζει τις εντολές του βρόχου σε στάδια διοχέτευσης, όπως η αντίστοιχη τεχνική στο υλικό
- Κύρια μέθοδος: Modulo Scheduling (χρονοπρογραμματισμός ακέραιου υπόλοιπου)
- Βελτιστοποιεί για ρυθμό παραγωγής νέων αποτελεσμάτων (throughput)
- Πλεονέκτημα είναι ότι η καθυστέρηση (latency) εξυπηρέτησης της μιας επανάληψης ξεχωριστά δεν έχει σημασία

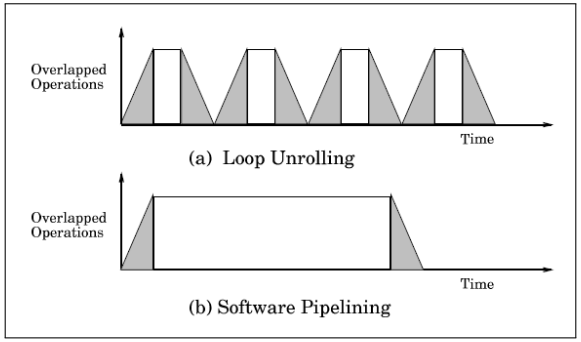
Software pipelining: Παράδειγμα (1)



Software pipelining: Παράδειγμα (2)

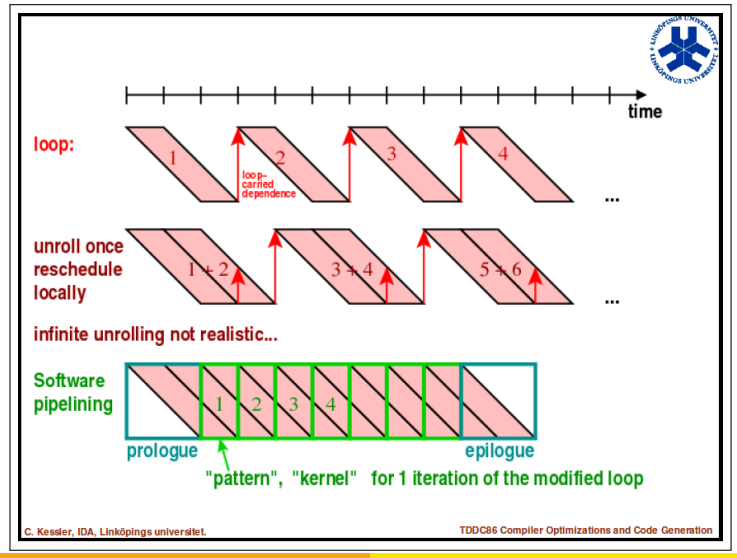


Σύγκριση loop unrolling με software pipelining (1)



Η διαφορά μεταξύ του loop unrolling και του software pipelining είναι ότι το unrolling μειώνει την επιβάρυνση του βρόχου, ελαττώνοντας τον απόλυτο αριθμό των επαναλήψεων, ενώ το sw pipelining μειώνει το κόστος έναρξης (initiation cost) κάθε νέας επανάληψης

Σύγκριση loop unrolling με software pipelining (2)



Modulo scheduling (1)

- Σημαντική τεχνική λογισμικής διοχέτευσης
  - Διερευνά το πεδίο των πιθανών πυρήνων βρόχου (loop kernels)
  - Χρησιμοποιεί την έννοια του διαστήματος αρχικοποίησης (II: Initiation Interval)
    - Σταθερό διάστημα (σε αριθμό κύκλων μηχανής) ανάμεσα στην έναρξη διαδοχικών επαναλήψεων του πυρήνα
  - Κάτω όριο του II (MinII)
    - Λαμβάνει υπόψη τους διαθέσιμους πόρους υλικού (ResII)
    - Εξαρτήσεις δεδομένων και επανανεμφανίσεις εξαρτήσεων (RecII)
  - Άνω όριο του II (MaxII)
    - Καθορίζει το μήκος ενός γραμμικού (χωρίς διακλαδώσεις/ αλλαγές ροής ελέγχου) χρονοπρογράμματος

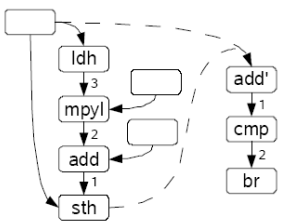
Modulo scheduling (2)

- Αναζήτηση για το κατάλληλο II ξεκινώντας από το MinII
- Η διαδικασία σταματά όταν βρεθεί έγκυρο χρονοπρόγραμμα
- Σε άλλη περίπτωση λαμβάνεται απόφαση ώστε
  - Οπισθοδρόμηση (ανακαλείται κάποια προηγούμενη απόφαση στο χρονοπρόγραμμα)
  - Αύξηση της τιμής του II
  - Η διαδικασία εγκαταλείπεται αν το II είναι πλέον μεγαλύτερο από το MaxII
  - Υψηλή υπολογιστική πολυπλοκότητα
- Οπισθοδρόμηση και αναζήτηση του II
- Τροποποιημένη μορφή του χρονοπρογραμματισμού λίστας
- Η διαθεσιμότητα πόρων καταγράφεται σε αντίστοιχο πίνακα κρατήσεων (modulo reservation table)
- Εξετάζεται η σύγκρουση μεταξύ χρήσεων πόρων για όλα τα χρονικά σημεία που αντιστοιχούν στο ίδιο ακέραιο υπόλοιπο με το II

Modulo scheduling: Παράδειγμα

```
int i,x ;
short a[];
for (i=0; i < 100; ++i)
  a[i+3] = a[i]*x + 7;
```

(a) Original C Code



(b) Data Dependence Graph

	ALU0	ALU1	MUL0	MEM	BR
0	cmp (9)	---	mpyl (3)	ldh (0)	---
1	---	---	---	sth (7)	---
2	add (5)	add' (8)	---	---	br (11)

(c) Modulo Reservation Table

Cycle mod II	Cycle				
0	0	ldh			
1	1				
2	2				
0	3	mpyl	ldh		
1	4				
2	5	add			
0	6		mpyl	ldh	
1	7	sth			
2	8	add'	add		
0	9	cmp		mpyl	ldh
1	10		sth		
2	11	br	add'	add	
0	12		cmp		mpyl
1	13			sth	
2	14		br	add'	add
0	15			cmp	
1	16				sth

(d) Modulo Schedule

Αναφορές του μαθήματος I

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 10.5, 11.1–11.3, 11.9–11.10**, 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. H. Allan, R. B. Jones, R. M. Lee, and S. J. Allan, "Software Pipelining," *ACM Computing Surveys*, vol. 27, no. 3, pp. 367–432, September 1995.

N. Kavvadias. Hardware looping unit. [Online]. Available: `http://www.opencores.org/projects.cgi/web/hwlu/overview/`