Μεταγλωττιστές II

Βελτιστοποιήσεις για την εκμετάλλευση της παραλληλίας και ενίσχυση της τοπικότητας

Νικόλαος Καββαδίας
nkavv@uop.gr

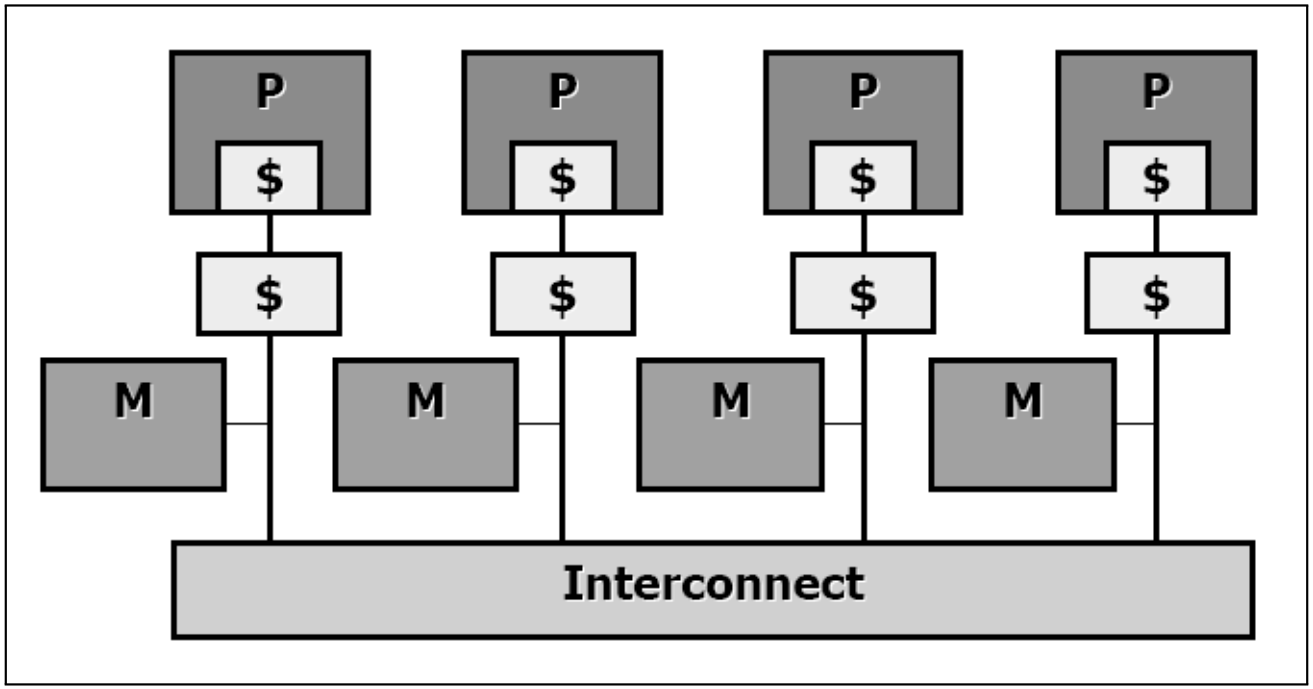
22 Δεκεμβρίου 2010

Βελτιστοποιήσεις για την εκμετάλλευση της παραλληλίας και ενίσχυση της τοπικότητας

- Οι βελτιστοποιήσεις για βαθμωτούς επεξεργαστές (scalar processors) έχουν γενικά ως στόχο τη μείωση των εκτελούμενων εντολών (και κατά επέκταση του χρόνου εκτέλεσης ή/και τη βελτίωση άλλων μετρικών)
- Οι βελτιστοποιήσεις για παράλληλες αρχιτεκτονικές έχουν ως στόχο τη βελτίωση χαρακτηριστικών όπως η αύξηση της διαθέσιμης παραλληλίας (πολλές λειτουργίες βρίσκονται ταυτόχρονα σε εξέλιξη), η τοποθέτηση των δεδομένων στη μνήμη και η οργάνωση της ιεραρχίας μνήμης
- Είδη επεξεργαστών
  - Επεξεργαστές VLIW
  - Υπερβαθμωτοί επεξεργαστές (superscalar processors)
  - Διανυσματικοί επεξεργαστές (vector processors)
  - Πολυπύρηνοι επεξεργαστές (multicore processors)
  - Πολυεπεξεργαστές (multiprocessors)
- Στόχος: Αύξηση της παραλληλίας και της τοπικότητας

Παραλληλία και τοπικότητα σε μία παράλληλη αρχιτεκτονική

Πολυεπεξεργαστής διαμοιρασμένου πεδίου διευθύνσεων (shared memory multiprocessor): μία ενιαία κεντρική μνήμη δεδομένων



Η διαδικασία της βελτιστοποίησης

- Η βελτιστοποίηση υψηλού επιπέδου έχει μεγαλύτερα οφέλη για την επίτευξη παραλληλίας
  - Ενδιάμεση αναπαράσταση υψηλού επιπέδου (πριν τη μετατροπή της σε συμβολική assembly για την εφαρμογή περασμάτων χαμηλού επιπέδου όπως επιλογή κώδικα, καταμερισμός καταχωρητών κ.λ.π.)
  - Στο επίπεδο του πηγαίου κώδικα πιθανόν με τη μεσολάβηση της μετατροπής σε AST
- Η βελτιστοποίηση από την άποψη του μεταγλωττιστή
  1. **Διερεύνηση** για τα κατάλληλα τμήματα του προγράμματος εισόδου που είναι ωφέλιμο να βελτιστοποιηθούν
  2. **Πιστοποίηση** του ότι η εφαρμογή της βελτιστοποίησης δεν αλλοιώνει τη φύση (σημασιολογία) του προγράμματος
  3. **Μετασχηματισμός** του προγράμματος με έξοδο ενδιάμεσης αναπαράστασης του ίδιου επιπέδου

ℹ Εστιάζουμε στη βελτιστοποίηση διαδικαστικών γλωσσών προγραμματισμού όπως είναι η ANSI/ISO C

Ανάλυση εξαρτήσεων δεδομένων (data dependence analysis)

- Η ανάλυση εξαρτήσεων θα πρέπει να αναγνωρίζει τους επιβαλλόμενους περιορισμούς και να καθορίζει το αν ένας συγκεκριμένος μετασχηματισμός μπορεί να εφαρμοστεί χωρίς την αλλοίωση της συμπεριφοράς του προγράμματος
- Εξαρτήσεις ανάμεσα σε δεδομένα
  - Βαθμωτές εξαρτήσεις (τύποι RAW, WAW, WAR)
  - Εξαρτήσεις μεταφερόμενες από βρόχο (loop-carried dependencies)
- Εξάρτηση δεδομένων: η εντολή i χρειάζεται τουλάχιστον μία τιμή που υπολογίζεται από την j

Εξαρτήσεις μεταφερόμενες από βρόχο (loop-carried dependencies)

- Το σώμα ενός βρόχου μπορεί να εκτελεστεί πολλές φορές
- Οι εξαρτήσεις που είναι μεταφερόμενες από βρόχο υφίστανται ανάμεσα σε διαφορετικές (συνήθως διαδοχικές) επαναλήψεις του βρόχου
- Παράδειγμα

```
for (i = 2; i <= n; i++) {
  S1: a[i] = a[i] + c;
  S2: b[i] = a[i-1] * b[i];
}
```

- Στην ίδια επανάληψη του βρόχου, δεν υπάρχει εξάρτηση ανάμεσα στις S1 και S2
- Σε δύο διαδοχικές επαναλήψεις, π.χ. όταν `i = k` η δήλωση S2 διαβάζει την τιμή του `a[k-1]` η οποία γράφεται από τη δήλωση S1 κατά την επανάληψη `k - 1`
- Για την ανάλυση εξαρτήσεων χρειάζεται η καταγραφή όλων των εκφράσεων δεικτοδότησης των πινάκων

Γενικευμένη δομή βρόχων

- Δομή πλήρως φωλιασμένων βρόχων σε γενική μορφή

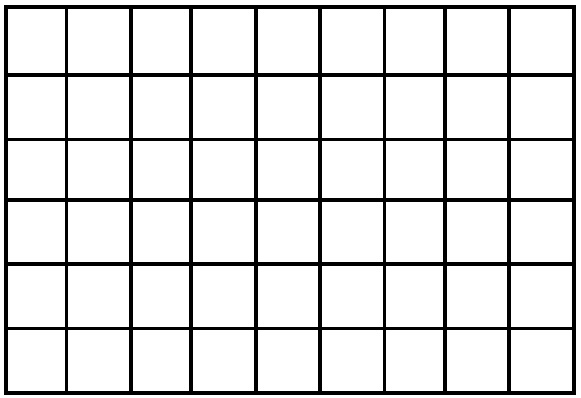
```
      do i_1 = l_1, u_1
         do i_2 = l_2, u_2
            ...
               do i_d = l_d, u_d
1                 a[f_1(i_1, ..., i_d), ..., f_m(i_1, ..., i_d)] = ...
2                 ... = a[g_1(i_1, ..., i_d), ..., g_m(i_1, ..., i_d)]
               end do
            ...
         end do
      end do
```

- Πεδίο επανάληψης (iteration space):
  το διάνυσμα των d στοιχείων $I = (i_1, \ldots, i_d)$ με κάθε στοιχείο του διανύσματος να αναπαριστά τη μεταβλητή δείκτη του αντίστοιχου βρόχου

Πεδίο επανάληψης (iteration space)

- Έστω οι διπλά φωλιασμένοι βρόχοι:

```
for (i = 0; i < 6; i++) {
  for (j = 0; j < 9; j++) {
  ...
  }
}
```



- Δομή βρόχων με n φωλιασμένους βρόχους αναπαρίσταται από ΠΟΛΥΤΟΠΟ (polytope) n διαστάσεων
- Ισοδύναμα χρησιμοποιείται η έκφραση 'πολύεδρο' (polyhedron)
- Οι επαναλήψεις εκφράζονται ως συντεταγμένες στο πεδίο επανάληψης
- Ακολουθιακή σειρά εκτέλεσης:
  [0,0], [0,1], ..., [0,7], [0,8], [1,0], [1,1], ..., [1,8], ..., [5,0], ..., [5,8]

Μετασχηματισμοί βρόχου (loop transformations) (1)

- Loop reordering transformations
  - Loop unswitching
  - Loop skewing
  - Loop reversal
  - Strip mining
  - Cycle shrinking
  - Loop tiling
  - Loop distribution
  - Loop fusion
- Loop restructuring transformations
  - Loop unrolling
  - Loop coalescing
  - Loop collapsing

Μετασχηματισμοί βρόχου (2)

- Loop restructuring transformations (cont.)
  - Software pipelining
  - Loop peeling
  - Loop normalization (canonicalization)
  - Loop spreading
- Loop replacement transformations
  - Reduction recognition
  - Loop idiom recognition
  - Array statement scalarization

Loop *unswitching*

- Εφαρμόζεται όταν ένας βρόχος περιέχει έλεγχο ο οποίος εξετάζει συνθήκη αμετάβλητη ως προς το βρόχο
- Τότε ο βρόχος μεταφέρεται (δημιουργώντας τα αντίστοιχα αντίγραφα) εντός των δύο κλάδων της συνθήκης (π.χ. μία δήλωση ελέγχου `if-else` στην C). Δηλαδή υπάρχει αντιμετάθεση μεταξύ του βρόχου και της δήλωσης ελέγχου
- Μειώνεται το μέγεθος του σώματος βρόχου (loop body)

```
for (i = 2; i <= n; i++) {
  a[i] = a[i] + c;
  if (x < 7)
    b[i] = a[i] * c[i];
  else
    b[i] = a[i-1] * b[i-1];
}
```

```
if (n > 1)
  if (x < 7)
    for (i = 2; i <= n; i++) {
      a[i] = a[i] + c;
      b[i] = a[i] * c[i];
    }
  else
    for (i = 2; i <= n; i++) {
      a[i] = a[i] + c;
      b[i] = a[i-1] * b[i-1];
    }
```

Loop reordering (αναδιάταξη βρόχου) (1)

- Μετασχηματισμός που μεταβάλλει τη σχετική σειρά εκτέλεσης των επαναλήψεων σε ένα βρόχο ή μία δομή βρόχων
- Χρησιμοποιείται για την ανάδειξη της παραλληλίας και τη βελτίωση της τοπικότητας των προσπελάσεων στη μνήμη
- Για γενικευμένες (μη τέλειες) δομές βρόχων, εφαρμόζεται σε συνδυασμό με την κατανομή βρόχων (loop distribution), η οποία και τον ενεργοποιεί
- Ο μεταγλωττιστής εξετάζει τις μεταφερόμενες από βρόχο εξαρτήσεις ώστε να καθορίσει αν ο συγκεκριμένος βρόχος/βρόχοι είναι παραλληλοποιήσιμος/οι

Loop reordering (αναδιάταξη βρόχου) (2)

Ο εξωτερικός βρόχος είναι παραλληλοποιήσιμος

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

Ο εσωτερικός βρόχος είναι παραλληλοποιήσιμος

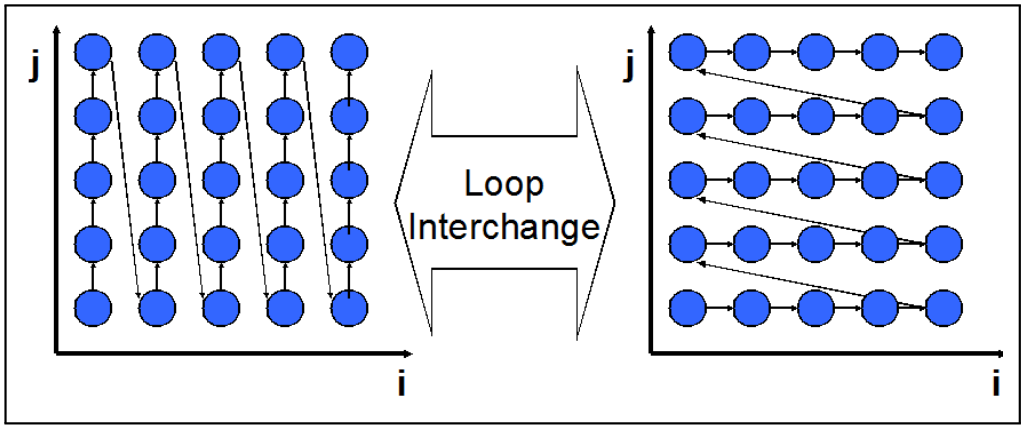
```
for (i = 1; i <= n; i++) {
  for (j = 1; j <= n; j++) {
    a[i][j] =
      a[i-1][j] + a[i-1][j+1];
  }
}
```

Loop interchange (ανταλλαγή βρόχων) (1)

- Μετακινεί έναν από τους εξωτερικότερους βρόχους στην εσώτερη θέση (inner loop)
- Μεταβάλλει τη σχετική σειρά εκτέλεσης των επαναλήψεων σε ένα βρόχο ή μία δομή βρόχων
- Χρησιμοποιείται για την ανάδειξη της παραλληλίας και τη βελτίωση της τοπικότητας των προσπελάσεων στη μνήμη
- Για γενικευμένες (μη τέλειες) δομές βρόχων, εφαρμόζεται σε συνδυασμό με κατανομή βρόχων (loop distribution) η οποία και τον ενεργοποιεί
- Χρήσεις
  - Ενεργοποίηση και βελτίωση διανυσματοποίησης
  - Μείωση του βήματος δείκτη (stride), ιδανικά στο 1
  - Αύξηση του αριθμού των εκφράσεων που είναι αμετάβλητες στον εσώτερο βρόχο

Loop interchange (ανταλλαγή βρόχων) (2)

- Μεταβολή του τρόπου σάρωσης διευθύνσεων λόγω της ανταλλαγής δύο βρόχων



- Διαδοχικά στοιχεία του `total` με βήμα n

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

- Διαδοχικά στοιχεία του `total` με βήμα 1

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

Loop skewing (ολίσθηση βρόχου) (1)

- Χρησιμοποιείται κυρίως για την ενεργοποίηση του μετασχηματισμού ανταλλαγής βρόχων
- Υλοποιείται με την πρόσθεση του δείκτη του εξώτερου βρόχου, πολλαπλασιασμένου με το συντελεστή ολίσθησης f (skewing factor) και στα δύο όρια των επαναλήψεων του δείκτη του εσώτερου βρόχου
- Στη συνέχεια μεταβάλλονται αντίστοιχα οι εκφράσεις διευθυνσιοδότησης σε όλους τους πίνακες που διαβάζονται ή γράφονται στο σώμα του εσώτερου βρόχου
- Ο μετασχηματισμός αυτός δεν μεταβάλλει τη συμπεριφορά του προγράμματος και μπορεί να χρησιμοποιηθεί σε κάθε περίπτωση

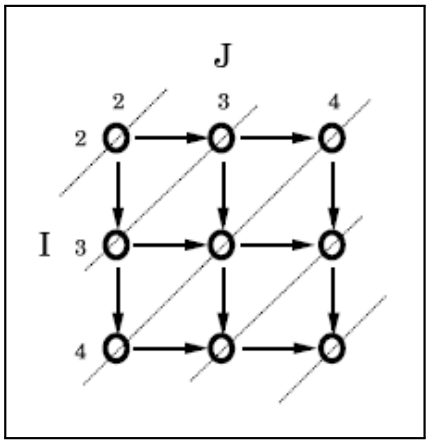
Loop skewing (ολίσθηση βρόχου) (2)

- Παράδειγμα εφαρμογής του loop skewing

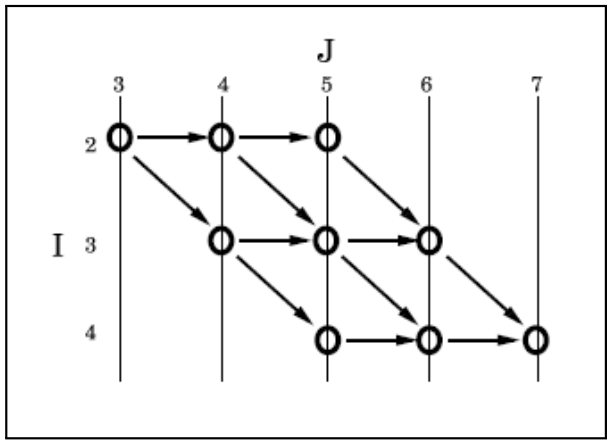
```
for (i = 2; i <= n-1; i++) {
  for (j = 2; j <= m-1; j++) {
    a[i][j] = (a[i-1][j] + a[i][j-1] +
               a[i+1][j] + a[i][j+1])/4;
  }
}
```

```
for (i = 2; i <= n-1; i++) {
  for (j = i+2; j <= i+m-1; j++) {
    a[i][j-i] = (a[i-1][j-i] + a[i][j-1-i] +
                 a[i+1][j-i] + a[i][j+1-i])/4;
  }
}
```

- Αρχικό πεδίο επαναλήψεων



- Ολισθημένο πεδίο επαναλήψεων



Loop reversal (αναστροφή βρόχου)

- Προκαλεί την αλλαγή της κατευθυντικότητας με την οποία ένας βρόχος διασχίζει το πεδίο επανάληψής του
- Μειώνει την επιβάρυνση για την εκτέλεση των λειτουργιών αύξησης δείκτη και ελέγχου του βρόχου
- Π.χ. εντολή `BNEZ (branch if not equal to zero)` στον επεξεργαστή DLX
- Ενεργοποιεί την εφαρμογή μετασχηματισμών όπως ο loop interchange: Παράδειγμα

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

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

Strip mining

- Ομαδοποιεί δεδομένα του ίδιου τύπου ώστε αν αυτά είναι ελεύθερα από εξαρτήσεις να μπορούν να χρησιμοποιηθούν από διανυσματική εντολή του επεξεργαστή
- Βασική παράμετρος είναι το strip length που αντιστοιχεί στο εύρος του διανύσματος

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

```
TN = (n/64)*64;
for (TI = 1; TI < TN; TI += 64) {
  a[range(TI:TI+63)] =
    a[range(TI:TI+63)] + c;
}
for (i = TN+1; i <= n; i++) {
  a[i] = a[i] + c;
}
```

- Διανυσματικός κώδικας (Vector DLX)

```
; R9 = address of a[TI]
LV    V1, R9        ; V1 <- a[TI:TI+63]
ADDSV V1, F8, V1    ; V1 <- V1 + c (F8 = c)
SV    V1, R9        ; a[TI:TI+63] <- V1
```

Loop tiling (πλακόστρωση βρόχων)

- Αποτελεί την πολυδιάστατη γενίκευση του μετασχηματισμού strip mining
- Ονομάζεται και loop blocking
- Χρησιμοποιείται για τη βελτίωση της επαναχρησιμοποίησης δεδομένων από την κρυφή μνήμη (cache reuse)
- Λειτουργεί με τη διαίρεση του πεδίου επανάληψης σε επιμέρους τμήματα (tiles: πλακίδια), και με την εφαρμογή του αντίστοιχου μετασχηματισμού της δομής βρόχων που αναπαρίσταται από το πεδίο

Loop tiling: Παράδειγμα

- Αρχικός βρόχος

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

- Πλακοστρωμένος βρόχος

```
for (TI = 1; TI <= n; TI += 64) {
  for (TJ = 1; TJ <= n; TJ += 64) {
    for (i = TI; i <= min(TI+63,n); i++) {
      for (j = TJ; j <= min(TJ+63,n); j++) {
        a[i][j] = b[j][i];
      }
    }
  }
}
```

Loop distribution (κατανομή βρόχων)

- Χρησιμοποιείται για την κατανομή ενός βρόχου σε πολλούς
- Εναλλακτικά ονομάζεται loop fission ή loop splitting
- Κάθε νέος βρόχος έχει το ίδιο πεδίο επανάληψης με τον αρχικό αλλά περιέχει ένα υποσύνολο των δηλώσεων του σώματος του αρχικού βρόχου
- Χρήσεις
  - Δημιουργία τέλεια φωλιασμένων βρόχων
  - Δημιουργία βρόχων με λιγότερες εξαρτήσεις
  - Βελτίωση της τοπικότητας της κρυφής μνήμης εντολών
  - Ελάττωση των απαιτήσεων μνήμης και της πίεσης στο αρχείο καταχωρητών

Loop distribution: Παράδειγμα

- Αρχικός βρόχος

```
for (i = 1; i <= n; i++) {
  a[i]   = a[i] + c;
  x[i+1] = x[i]*7 + x[i+1] + a[i];
}
```

- Κατανομή του αρχικού βρόχου

```
for (i = 1; i <= n; i++) {
  a[i] = a[i] + c;
}
for (i = 1; i <= n; i++) {
  x[i+1] = x[i]*7 + x[i+1] + a[i];
}
```

Loop fusion (μείξη βρόχων)

- Πρόκειται για τον αντίστροφο μετασχηματισμό από αυτόν της κατανομής βρόχου
- Χρησιμοποιείται για τη βελτίωση επιδόσεων με την εκμετάλλευση
  - της μείωσης της επιβάρυνσης βρόχου
  - της αύξησης της παραλληλίας στο σώμα του βρόχου
  - της βελτίωσης της τοπικότητας

- Αρχικός κώδικας

```
for (i = 1; i <= n; i++) {
  a[i] = a[i] + c;
}
for (i = 1; i <= n; i++) {
  x[i+1] =
    x[i]*7 + x[i+1] + a[i];
}
```

- Μείξη (συνένωση) των αρχικών βρόχων

```
for (i = 1; i <= n; i++) {
  a[i]   = a[i] + c;
  x[i+1] =
    x[i]*7 + x[i+1] + a[i];
}
```

Loop unrolling (ξετύλιγμα βρόχων)

- Αναδομητικός (restructuring) μετασχηματισμός ο οποίος μεταβάλλει τη δομή του βρόχου, χωρίς να επηρεάζει τη σχετική σειρά των λειτουργιών στο σώμα του βρόχου
- Αντιγράφει το σώμα ενός βρόχου για αριθμό φορών ίσο με τον παράγοντα ξετυλίγματος (unroll factor: u). Ο νέος βρόχος έχει βήμα u αντί για 1
- Χρήσεις
  - μείωση της επιβάρυνσης βρόχου
  - αύξηση της παραλληλίας στο σώμα του βρόχου
  - βελτίωση της τοπικότητας

- Αρχικός βρόχος

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

- Unroll factor: 2

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

Loop unrolling: Εφαρμογή σε δομές επανάληψης for και while

- Γενικά υπάρχει μία θεμελιώδης δομή επανάληψης, η `repeat...until` (ισότιμη της do...while)

- Δομή `for`

```
for (i = 0; i < N; i++) {
  S(i);
}
```

- Unrolled for loop

```
for (i = 0 ; i+4 < N; i += 4) {
  S(i);
  S(i+1);
  S(i+2);
  S(i+3);
}
for ( ; i < N; i++) {
  S(i);
}
```

- Δομή `repeat-until`

```
repeat
  S;
until (condition);
```

- Unrolled repeat-until loop

```
repeat
  S;
  if (condition) break;
  S;
  if (condition) break;
  S;
  if (condition) break;
  S;
until (condition);
```

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) + 1;
  j = MOD(T-1, m) + 1;
  a[i][j] = a[i][j] + 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 spreading (διεύρυνση βρόχου)

- Ο μετασχηματισμός αυτός μετακινεί ορισμένες λειτουργίες από το σώμα ενός βρόχου στο σώμα ενός παρακείμενου ώστε στην τελική τους μορφή, τα σώματα των δύο βρόχων να μπορούν να εκτελεστούν παράλληλα
- Χρησιμοποιείται για την αποκάλυψη ευκαιριών για loop peeling και fusion
- Γενικά διευκολύνει την ανάδειξη παραλληλίας επιπέδου εντολών

Loop spreading: Παράδειγμα

- Αρχική δομή βρόχων

```
for (i = 1; i <= n/2; i++) {
  a[i+1] = a[i+1] + a[i];
}
for (i = 1; i <= n-3; i++) {
  b[i+1] = b[i+1] + b[i] + a[i+3];
}
```

- Τελική δομή βρόχων

```
for (i = 1; i <= n/2; i++) {
  a[i+1] = a[i+1] + a[i];
  if (i > 3) {
    b[i-2] = b[i-2] + b[i-3] + a[i];
  }
}
for (i = (n/2)-3; i <= n-3; i++) {
  b[i+1] = b[i+1] + b[i] + a[i+3];
}
```

Μετασχηματισμοί προσπέλασης μνήμης (memory access transformations)

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

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 αντικατάσταση)

- 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;
}
```

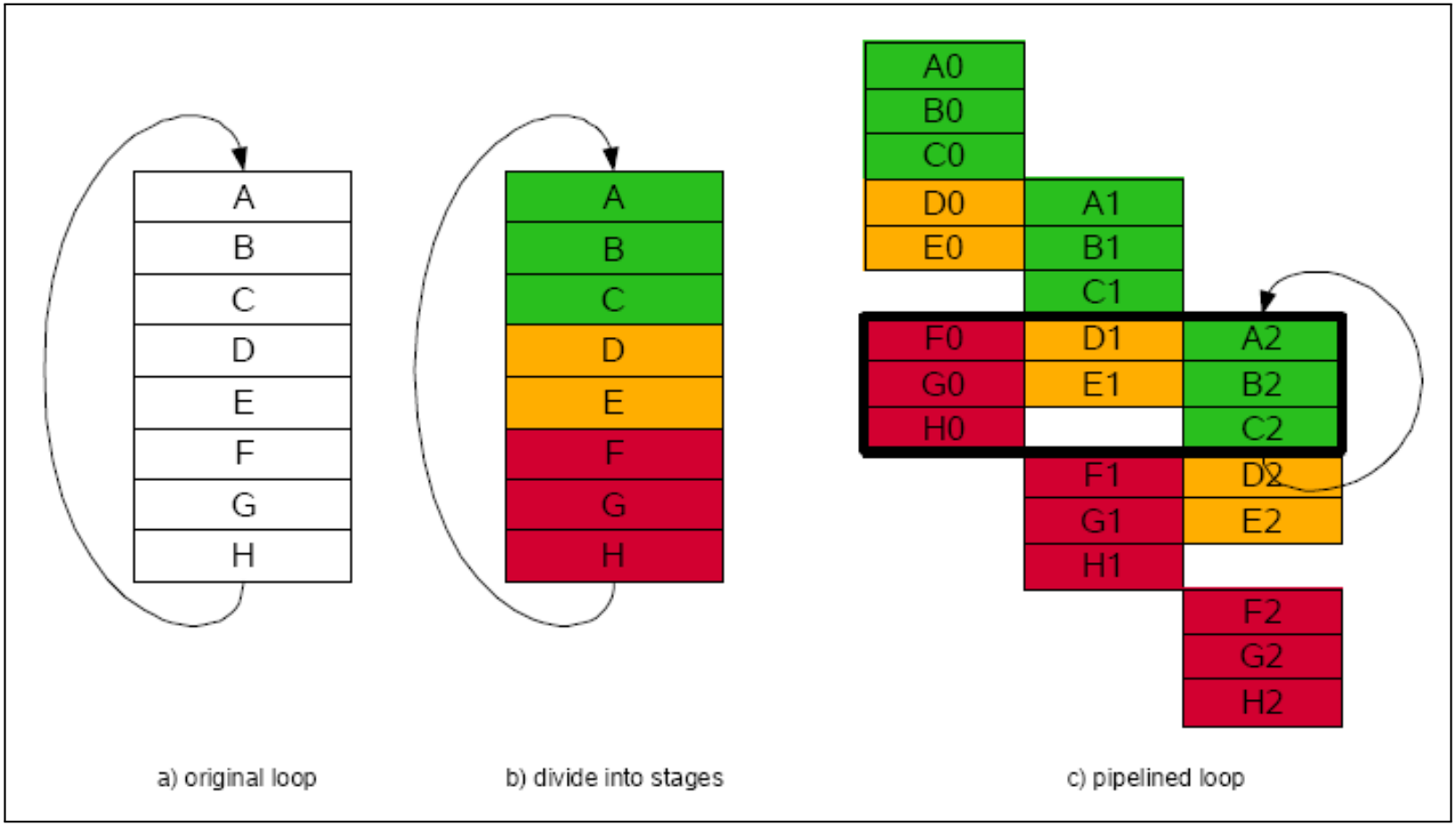
Χρονοπρογραμματισμός σε κυκλικές περιοχές

- 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

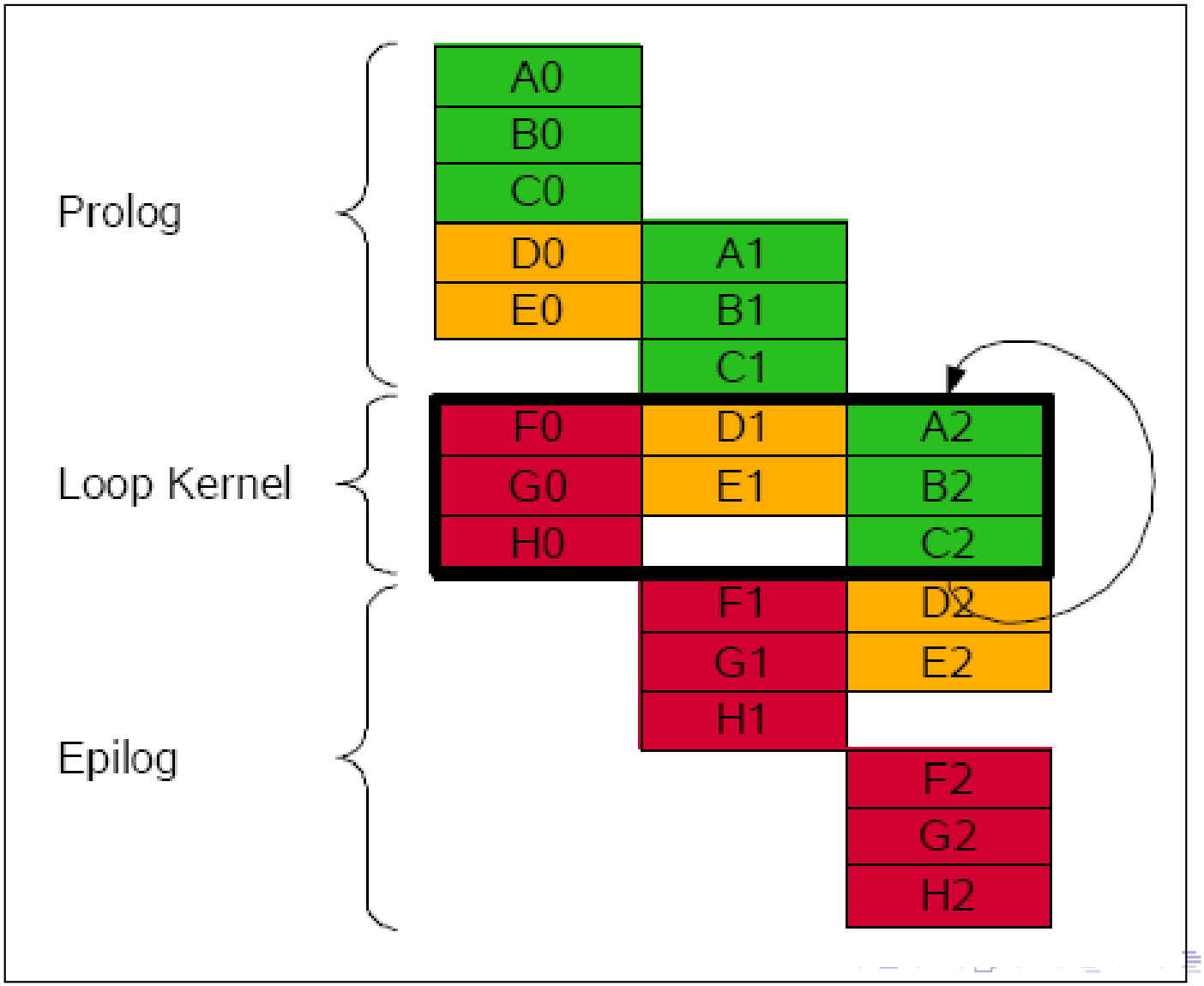
Software pipelining (Λογισμική διοχέτευση)

- 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

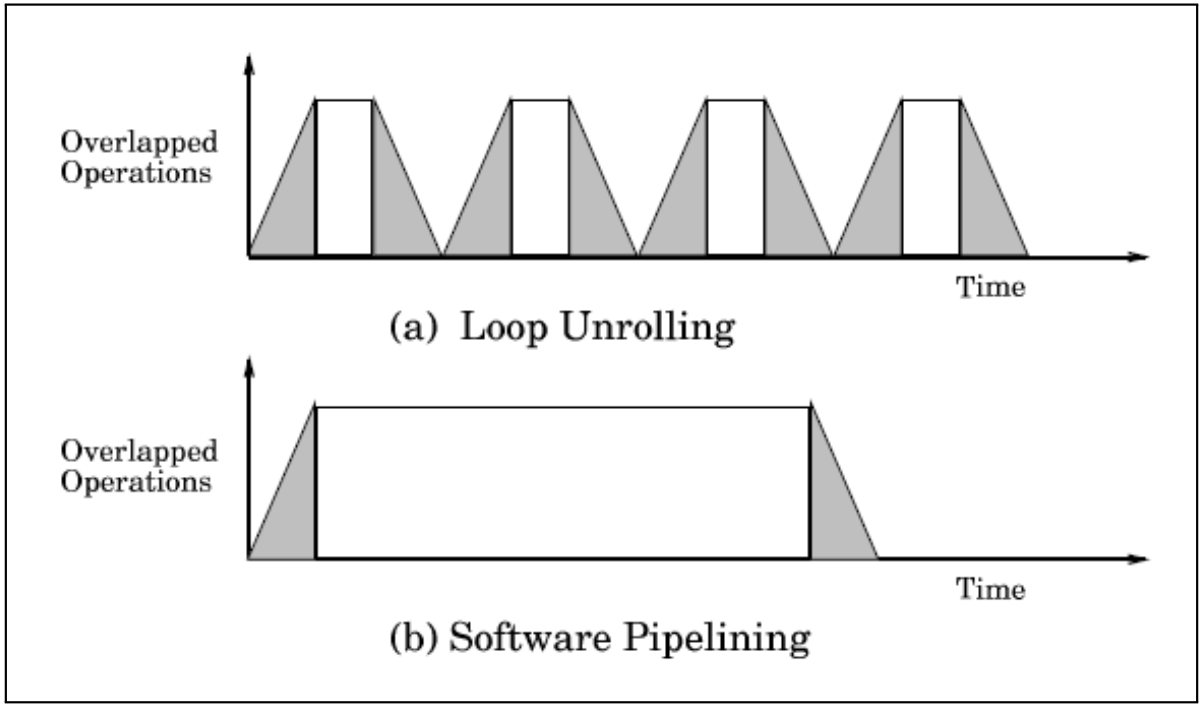
Software pipelining: Παράδειγμα (1)



Software pipelining: Παράδειγμα (2)

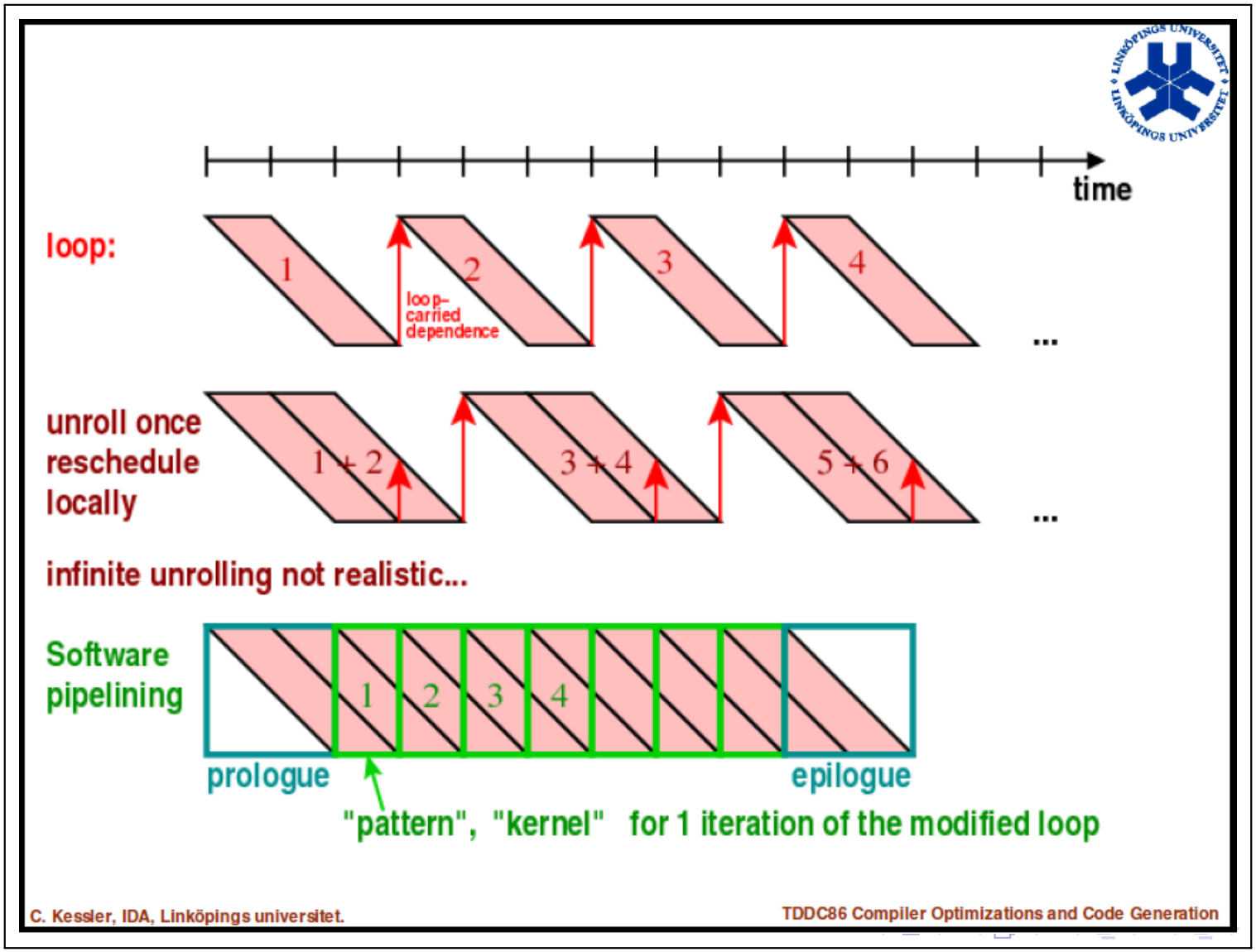


Σύγκριση loop unrolling με software pipelining (1)



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)



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 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/`