# Προηγμένα Θέματα Θεωρητικής Πληροφορικής

## Καταμερισμός καταχωρητών

Νικόλαος Καββαδίας
nkavv@uop.gr

21 Απριλίου 2010



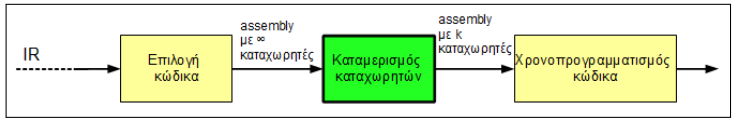
## Γενικά για τον καταμερισμό καταχωρητών

- Καταμερισμός καταχωρητών (register allocation): βελτιστοποίηση μεταγλωττιστή κατά την οποία η ενδιάμεση αναπαράσταση (IR) του πηγαίου προγράμματος μετασχηματίζεται ώστε να χρησιμοποιεί πεπερασμένο αριθμό καταχωρητών
  - Χαρτογράφηση της IR σε μια τροποποιημένη IR του προγράμματος όπου γίνεται χρήση $k$ καταχωρητών
  - Επιμέρους ζητήματα
    - Ελαχιστοποίηση της συχνότητας εμφάνισης λειτουργιών φόρτωσης από (load) και αποθήκευσης προς (store) τη μνήμη δεδομένων
    - Διαδικασίες έγχυσης (spill) και γέμισης (fill) μεταβλητών για τη μεταφορά τους από και προς τη μνήμη, όταν δεν υπάρχει επάρκεια διαθέσιμων καταχωρητών
  - Οι καταχωρητές αποτελούν την πιο φτηνή μορφή μνήμης από άποψη κόστους προσπέλασης σε ταχύτητα (κύκλοι μηχανής) και κατανάλωση ισχύος/ενέργειας



## Η θέση του καταμερισμού καταχωρητών στη ροή μεταγλώττισης

- Ο καταμερισμός καταχωρητών αποτελεί τμήμα της ροής μετατροπής της IR σε κώδικα συμβολομεταφραστή για τον στοχευόμενο επεξεργαστή
- Θέση του καταμερισμού καταχωρητών στο backend




- Ο καταμερισμός καταχωρητών για $k \geq 1$ καταχωρητές είναι δυσεπίλυτος (*NP-complete*): η βέλτιστη επίλυση ενός προβλήματος καταμερισμού καταχωρητών είναι εκθετικής πολυπλοκότητας
- Ο σχεδιασμός αλγορίθμων καταμερισμού καταχωρητών αποτελεί αντικείμενο έρευνας μέχρι και σήμερα



## Εναλλακτικές προσεγγίσεις στο πρόβλημα του καταμερισμού καταχωρητών

- Μέχρι σήμερα έχει προταθεί πλήθος διαφορετικών προσεγγίσεων για τον καταμερισμό καταχωρητών
  - Ευριστικοί αλγόριθμοι (heuristics)
  - Φορμαλισμοί ακέραιου γραμμικού προγραμματισμού
  - Άλλες τεχνικές δειγματοληψίας ενός πολυπαραμετρικού πεδίου λύσεων όπως γενετικοί αλγόριθμοι και προσομοιωμένη ανόπτηση
- Ζητούμενα στον καταμερισμό καταχωρητών
  - Χρήση του ελάχιστου αριθμού καταχωρητών
  - Ελαχιστοποίηση των spill και fill με τη μνήμη
  - Μείωση του χρόνου εκτέλεσης της εφαρμογής
  - Εμβέλεια της διαδικασίας
    - Τοπικός (local) καταμερισμός καταχωρητών: εφαρμογή σε κάθε βασικό μπλοκ ξεχωριστά
    - Καθολικός (global) καταμερισμός καταχωρητών: εφαρμογή σε ολόκληρα υποπρογράμματα

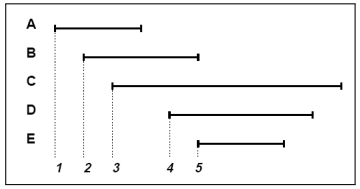
# Προετοιμασία για το κύριο μέρος του καταμερισμού καταχωρητών

- Εξαγωγή του CFG κάθε υποπρογράμματος
  - Οι ακμές εξόδου (out-edges) από τον κόμβο $n$ οδηγούν σε διάδοχους κόμβους: $succ[n]$
  - Οι ακμές εισόδου του κόμβου $n$ πηγάζουν από προηγηθέντες κόμβους: $pred[n]$
- Ανάλυση χρόνου ζωής (liveness analysis)
  - Η IR χρησιμοποιεί εντολές του επεξεργαστή (και ορισμένες συμβολικές εντολές για κώδικα κλήσης, εισόδου και εξόδου από υπορουτίνες) με απεριόριστους εικονικούς καταχωρητές
  - Οι προσωρινές μεταβλητές με μη αλληλεπικαλυπτόμενες περιοχές χρήσης στην IR μπορούν να αντιστοιχηθούν στο ίδιο φυσικό καταχωρητή
  - Η ανάλυση χρόνου ζωής πραγματοποιείται για κάθε προσωρινή μεταβλητή: είναι 'ζωντανή' (live) όταν διατηρεί μία τιμή η οποία μπορεί να ζητηθεί σε μεταγενέστερο σημείο του προγράμματος



# Παράδειγμα διαστημάτων ζωής (live intervals) προσωρινών μεταβλητών

- Έστω οι προσωρινές μεταβλητές `A, B, C, D, E` και τα αντίστοιχα διαστήματα 1 ως 5. Το παρακάτω σχήμα απεικονίζει τα live intervals για τις προσωρινές μεταβλητές για μία περιοχή ενός πηγαίου υποπρογράμματος η οποία μπορεί να εκτείνεται σε περισσότερα του ενός βασικά μπλοκ
- Δημιουργία της λίστας των ενεργών προσωρινών μεταβλητών (active list)




| | |
|---|---|
| Έναρξη I1 | active = $\langle A \rangle$ |
| Έναρξη I2 | active = $\langle A, B \rangle$ |
| Έναρξη I3 | active = $\langle A, B, C \rangle$ |
| Λήξη I1 | active = $\langle B, C \rangle$ |
| Έναρξη I4 | active = $\langle B, C, D \rangle$ |
| Λήξη I2 και έναρξη I5 | active = $\langle C, D, E \rangle$ |
| Λήξη I5 | active = $\langle C, D \rangle$ |
| Λήξη I4 | active = $\langle C \rangle$ |
| Λήξη I3 | active = $\langle \emptyset \rangle$ |



# Παράδειγμα εξαγωγής διαστημάτων χρόνου ζωής μεταβλητών

- Βασικό μπλοκ του παραδείγματος

```
1  b = 1;
2  c = 2;
3  a = b + c;
4  d = a * 2;
5  e = b / 3;
6  return (e - d);
```

- Υπολογισμός των διαστημάτων ζωής των μεταβλητών

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| a | | | X | X | | |
| b | X | X | X | X | X | |
| c | | X | X | | | |
| d | | | | X | X | X |
| e | | | | | X | X |



# Απλοϊκή προσέγγιση: Παραγωγή κώδικα assembly χωρίς καταμερισμό καταχωρητών

- Στην απλούστερη περίπτωση, μετά την επιλογή κώδικα, θεωρούμε ότι ΟΛΕΣ οι μεταβλητές διατηρούνται στη μνήμη δεδομένων
- Όλες οι μεταβλητές, καθολικής ή τοπικής εμβέλειας καλούνται από τη μνήμη
- Παράδειγμα (κώδικας assembly για την αρχιτεκτονική MIPS)

```
int x = 1;

int myfunction(int a)
{
  int b;
  b = a + x;
  return b;
}
```

```
.globl x
x:
.word 1
.text
myfunction:
addiu $sp, $sp, -8
sw    $4, 8($sp)
lw    $3, 8($sp)
lw    $2, %gp_rel(x)($28)
addu  $2, $3, $2
sw    $2, 0($sp)
lw    $2, 0($sp)
addiu $sp, $sp, 8
jr    $ra
```

- Διατήρηση των μεταβλητών $x, a, b$ στη μνήμη ενώ θα μπορούσαν να διατηρούνται σε καταχωρητές

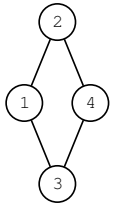
## Καθολικός καταμερισμός καταχωρητών (global register allocation)

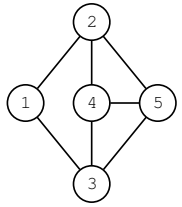
- Σε κάθε σημείο της IR (μετά την επιλογή κώδικα)
  1. Καθορισμός των μεταβλητών που πρέπει να διατηρούνται σε καταχωρητές
  2. Επιλογή ενός διαθέσιμου καταχωρητή για κάθε τέτοια μεταβλητή
- Ο στόχος είναι συνήθως η ελαχιστοποίηση του χρόνου εκτέλεσης του συνολικού προγράμματος
- Οι περισσότεροι μοντέρνοι καθολικοί καταμεριστές αντιμετωπίζουν το πρόβλημα σύμφωνα με τη θεωρία χρωματισμού γράφου (graph coloring)
  - Κατασκευή ενός γράφου παρεμβολής (interference graph) ή αλλιώς γράφου διαμάχης (conflict graph) για τις μεταβλητές του υποπρογράμματος
  - Ανεύρεση ενός $k$-χρωματισμού ($k$-coloring) για το γράφο
  - Εφόσον η ανεύρεση ενός $k$-χρωματισμού είναι αδύνατη, γίνεται κατάλληλος μετασχηματισμός της IR ώστε να αναχθεί ο γράφος παρεμβολής σε $k$-χρωματίσιμο ($k$-colorable)



## Χρωματισμός γράφου

- Ορισμός του $k$-χρωματίσιμου γράφου:
  Ένας γράφος $G$ αποκαλείται $k$-χρωματίσιμος αν και μόνο αν οι κορυφές του μπορούν να αντιστοιχηθούν με τις ετικέττες $1 \ldots k$ έτσι ώστε καμία ακμή στον $G$ να μη συνδέει δυο κορυφές με την ίδια ετικέττα
- Οι ετικέττες (labels) συνήθως αναπαρίστανται ως ακέραιοι αριθμοί ή ως χρώματα
- Παραδείγματα $k$-χρωματίσιμων γράφων






- ☞ Κάθε χρώμα χαρτογραφείται σε ένα διακριτό φυσικό καταχωρητή

## Κατασκευάζοντας το γράφο παρεμβολής

- Τι είναι παρεμβολή (διαμάχη)
  - Δύο μεταβλητές λέμε ότι 'παρεμβάλλονται' εάν υπάρχει εντολή της IR στην οποία είναι και οι δύο live ταυτόχρονα
  - Εάν οι $x$ και $y$ παρεμβάλλονται, δεν μπορούν να καταλαμβάνουν τον ίδιο φυσικό καταχωρητή
- Για τον υπολογισμό των παρεμβολών, θα πρέπει να είναι γνωστές οι αντίστοιχες περιοχές ζωής (live ranges)
- Για το γράφο παρεμβολής $G_I$
  - Οι κορυφές αναπαριστούν μεταβλητές ή ισοδύναμα περιοχές ζωής μεταβλητών
  - Οι ακμές αναπαριστούν μεμονωμένες παρεμβολές (πάντα μεταξύ δύο μεταβλητών)
    - → Για $x, y \in G_I$, υπάρχει ακμή $\langle x, y \rangle$ αν και μόνο αν οι κορυφές $x$ και $y$ παρεμβάλλονται
  - Ένας $k$-χρωματισμός του $G_I$ μπορεί να αποτυπωθεί σε έναν καταμερισμό των απεικονιζόμενων μεταβλητών προς $k$ καταχωρητές



## Περιοχή ζωής (LR: live range)

- Περιοχή ζωής: σύνολο απαρτιζόμενο από ορισμούς $\{d_1, d_2, \ldots d_n\}$ ώστε για κάθε δύο ορισμούς $d_i$, $d_j$ που είναι στο $LR$ υπάρχει κάποια χρήση $u$ η οποία είναι προσβάσιμη από τα $d_i$ και $d_j$
- Για κάθε BB υπολογίζεται το $LIVEOUT(b)$: σύνολο των ορισμών που εξέρχονται από το βασικό μπλοκ $b$
  - $d \in LIVEOUT(b)$ αν δεν υπάρχει άλλος ορισμός σε κάποιο μονοπάτι από τον $d$ μέχρι το τέλος του $b$
- Για κάθε BB υπολογίζεται το $LIVEIN(b)$: σύνολο των ορισμών που είναι live κατά την είσοδο στο βασικό μπλοκ $b$
  - $v \in LIVEIN(b)$ αν υπάρχει μονοπάτι από την είσοδο στο $b$ μέχρι σε μια χρήση της $v$ το οποίο δεν περιλαμβάνει νέο ορισμό του $v$
- Σε κάθε σημείο επανένωσης εναλλακτικών διαδρομών ροής ελέγχου του CFG, για κάθε live μεταβλητή $v$, γίνεται συνένωση των περιοχών ζωής που συνδέονται με ορισμούς στο $REACHESOUT(pred(b))$, για όλους τους προηγηθέντες του $b$ οι οποίοι αναθέτουν τιμή στην $v$

## Παρατηρήσεις για τη χρήση του χρωματισμού γράφου στον καταμερισμό καταχωρητών

- Όταν είναι διαθέσιμοι $k$ φυσικοί καταχωρητές, τότε το ζητούμενο είναι ο $k$-χρωματισμός
- Κάθε κορυφή $n$ η οποία διαθέτει λιγότερες από $k$ γειτνιάζουσες κορυφές στο γράφο παρεμβολής ($n^D < k$) μπορεί πάντα να χρωματιστεί
  - επιλογή οποιουδήποτε χρώματος δεν χρησιμοποιείται από τις γειτνιάζουσες κορυφές
- Η βασική τεχνική που χρησιμοποιείται είναι ο αλγόριθμος του Chaitin [Chaitin, 1982]
  1. Επιλογή κορυφής $n$ για την οποία $n^D < k$ και τοποθέτησή της σε στοίβα (stack)
  2. Απομάκρυνση της $n$ και όλων των προσπιπτουσών ακμών
  3. Στο τέλος αν κάποια κορυφή $n$ διαθέτει περισσότερους από $k$ γείτονες, τότε γίνεται έγχυση της περιοχής ζωής της
  4. Αν δεν συμβαίνει αυτό, γίνεται διαδοχική εξαγωγή κορυφών από τη στοίβα και ακολουθεί ο χρωματισμός αυτών με χρώμα που δεν χρησιμοποιείται από κάποιο γείτονα



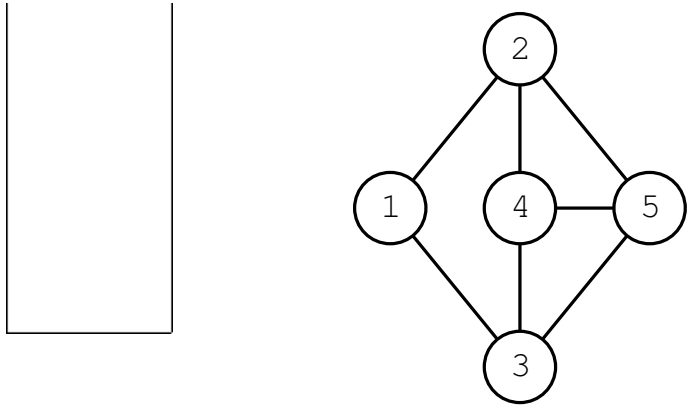
## Ο αλγόριθμος του Chaitin

1. Καθόσον ∃ κορυφές με λιγότερους από $k$ γείτονες στον $G_I$
   - Επιλογή της $n$ ώστε $n^D < k$ και τοποθέτησή της στη στοίβα
   - Διαγραφή της κορυφής και των προσπιπτουσών ακμών από τον $G_I$ (ελαττώνει το βαθμό: degree των γειτνιαζουσών κορυφών της $n$)
2. Αν ο $G_I$ δεν είναι κενός (όλες οι κορυφές έχουν $k$ ή περισσότερους γείτονες) τότε:
   - Επιλογή κορυφής $n$ και έγχυση της περιοχής ζωής που συνδέεται με την $n$
   - Διαγραφή της $n$ και των προσπιπτουσών σε αυτήν ακμών από τον $G_I$ και τοποθέτηση στη στοίβα
   - Εάν αυτές οι ενέργειες προκαλούν σε κάποια κορυφή του $G_I$ ελάττωση του αριθμού γειτόνων σε μικρότερο του $k$, μετάβαση στο βήμα 1, αλλιώς επαναλαμβάνεται το βήμα 2
3. Διαδοχική εξαγωγή κορυφών από τη στοίβα και χρωματισμός αυτών με το χρώμα χαμηλότερης απαρίθμησης που δεν χρησιμοποιείται από κάποιο γείτονα



## Ο αλγόριθμος του Chaitin στην πράξη (1)

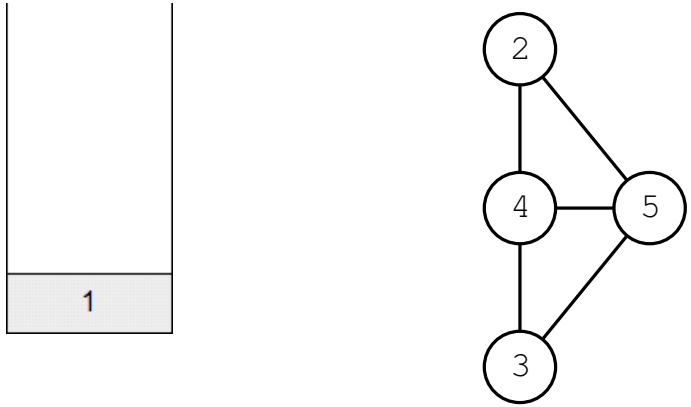
Καταμερισμός καταχωρητών με χρωματισμό γράφου για $k = 3$





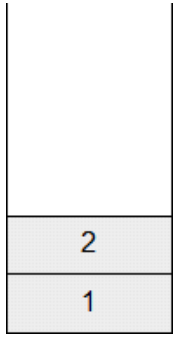

## Ο αλγόριθμος του Chaitin στην πράξη (2)

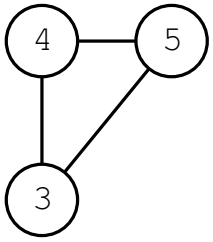
Καταμερισμός καταχωρητών με χρωματισμό γράφου για $k = 3$



## Ο αλγόριθμος του Chaitin στην πράξη (3)

Καταμερισμός καταχωρητών με χρωματισμό γράφου για $k = 3$

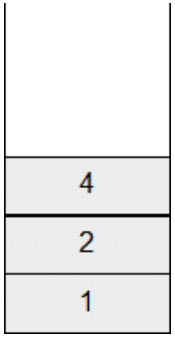


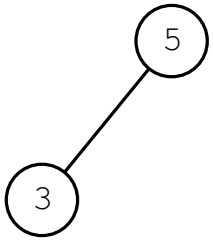






## Ο αλγόριθμος του Chaitin στην πράξη (4)

Καταμερισμός καταχωρητών με χρωματισμό γράφου για $k = 3$

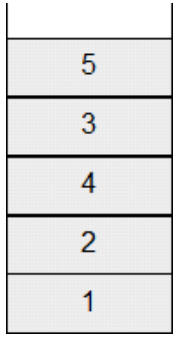





## Ο αλγόριθμος του Chaitin στην πράξη (5)

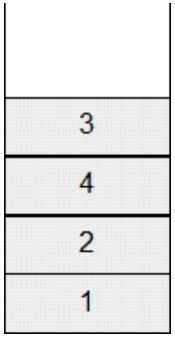
Καταμερισμός καταχωρητών με χρωματισμό γράφου για $k = 3$





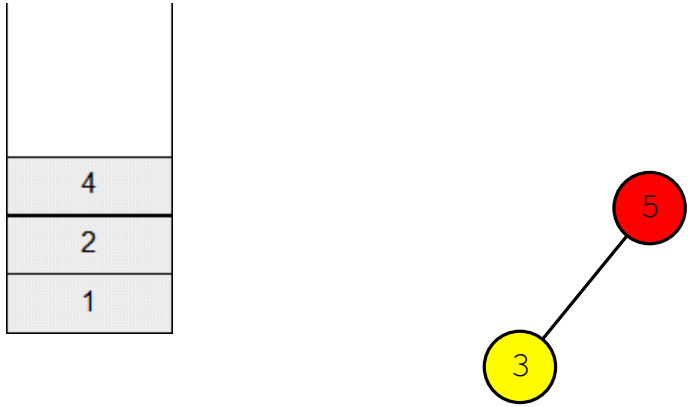

## Ο αλγόριθμος του Chaitin στην πράξη (6)

Καταμερισμός καταχωρητών με χρωματισμό γράφου για $k = 3$




## Ο αλγόριθμος του Chaitin στην πράξη (7)

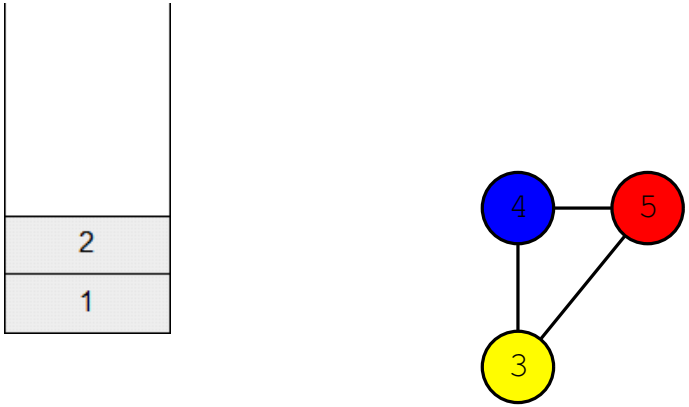
Καταμερισμός καταχωρητών με χρωματισμό γράφου για $k = 3$






## Ο αλγόριθμος του Chaitin στην πράξη (8)

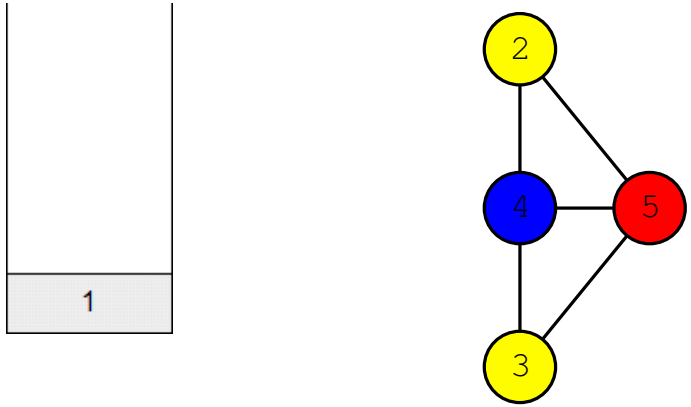
Καταμερισμός καταχωρητών με χρωματισμό γράφου για $k = 3$






## Ο αλγόριθμος του Chaitin στην πράξη (9)

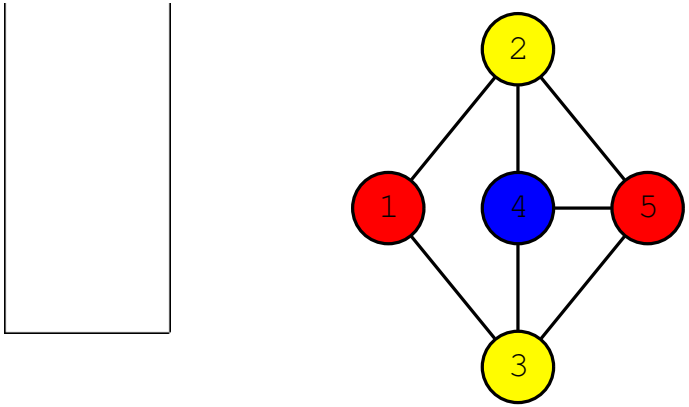
Καταμερισμός καταχωρητών με χρωματισμό γράφου για $k = 3$



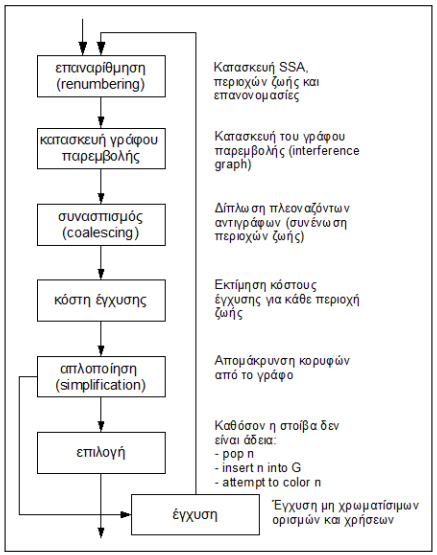



## Ο αλγόριθμος του Chaitin στην πράξη (10)

Καταμερισμός καταχωρητών με χρωματισμό γράφου για $k = 3$



## Γενική άποψη του αλγορίθμου καταμερισμού καταχωρητών με χρωματισμό γράφου






## Μια απλούστερη (και ταχύτερη) μέθοδος για τον καταμερισμό καταχωρητών

- Οι καθολικοί καταμεριστές καταχωρητών που χρησιμοποιούν τον αλγόριθμο του Chaitin είναι υπολογιστικά ακριβοί, καθώς ο γράφος παρεμβολής στη χειρότερη περίπτωση έχει διαστάσεις ανάλογες με το τετράγωνο του αριθμού των περιοχών ζωής
- Μία ιδιαίτερα δημοφιλής τεχνική στους μεταγλωττιστές δυναμικών γλωσσών προγραμματισμού αλλά και σε επαναστοχεύσιμους μεταγλωττιστές, που δεν χρησιμοποιεί την κατασκευή και το χρωματισμό γράφων παρεμβολής είναι ο **αλγόριθμος της γραμμικής σάρωσης (linear scan register allocation)** [Poletto, 1999]
- Ο αλγόριθμος, δοθέντων των περιοχών ζωής των μεταβλητών σε ένα υποπρόγραμμα, σαρώνει σε ένα πέρασμα (single pass) όλες τις περιοχές ζωής, και καταμερίζει φυσικούς καταχωρητές σε μεταβλητές με άπληστο (greedy) τρόπο



## Γενικά για τον αλγόριθμο γραμμικής σάρωσης

- Ο αλγόριθμος γραμμικής σάρωσης είναι απλός και παράγει κώδικα σχετικά υψηλών επιδόσεων
- Χρησιμοποιείται σε περιπτώσεις ανταλλαγής (trade-off) μεταξύ χρόνου μεταγλώττισης και χρόνου εκτέλεσης κώδικα όπως για μεταγλωττιστές JIT (Just-In-Time) που συνηθίζονται για δυναμικές γλώσσες προγραμματισμού
- Χρησιμοποιείται επίσης σε κάθε περίπτωση όπου είναι επιθυμητή η γρήγορη ανάπτυξη ενός μεταγλωττιστή
- Ο αλγόριθμος υποθέτει ότι οι εντολές τριών διευθύνσεων στην IR (μετά την επιλογή κώδικα) είναι αριθμήσιμες βάσει κάποιου κανόνα. Για παράδειγμα οι εντολές TAC μπορούν να αριθμηθούν κατά τη ροή του πηγαίου κώδικα ή κατά τη διάσχιση (π.χ. depth-first search) του CFG του υποπρογράμματος



## Περιοχές ζωής (live ranges) και διαστήματα ζωής (live intervals)

- Διάστημα ζωής: το διάστημα $[i, j]$ για την μεταβλητή $v$ ονομάζεται διάστημα ζωής της αν δεν υπάρχει εντολή με αριθμό $j' > j$ τέτοια ώστε η $v$ να είναι live στη θέση $j'$ και εντολή με αριθμό $i' < i$ ώστε να είναι live στη θέση $i'$
- Το διάστημα ζωής είναι μια συντηρητική προσέγγιση των περιοχών ζωής καθώς αγνοούνται τα υποδιαστήματα στα οποία η $v$ δεν είναι live
- Η αρχική τιμή του διαστήματος ζωής για κάθε μεταβλητή είναι το $[1, N]$ όπου $N$ είναι ο αριθμός των εντολών στην IR
- Η σειρά απαρίθμησης των εντολών επηρεάζει την έκταση των διαστημάτων ζωής και κατά συνέπεια τις επιδόσεις του καταμερισμού καταχωρητών

## Παρατηρήσεις για τη χρήση του αλγορίθμου γραμμικής σάρωσης

- Η παρεμβολή δύο διαστημάτων ζωής αντιπροσωπεύεται από το αν αυτά αλληλεπικαλύπτονται ή όχι
- Γραμμική σάρωση: δοθέντων $R$ διαθέσιμων φυσικών καταχωρητών και μιας λίστας διαστημάτων ζωής, ο αλγόριθμος κατανέμει καταχωρητές σε όσο το δυνατόν περισσότερα διαστήματα με τέτοιο τρόπο ώστε δύο αλληλεπικαλυπτόμενα διαστήματα να μην αντιστοιχίζονται στον ίδιο καταχωρητή
- Αν $n > R$ διαστήματα αλληλεπικαλύπτονται στην εντολή $i$, τότε $n - R$ μεταβλητές πρέπει να διατηρούνται στη μνήμη
- Ο αριθμός των αλληλεπικαλυπτόμενων διαστημάτων μεταβάλλεται μόνο στο αρχικό και το τελικό σημείο ενός διαστήματος
- ☞ Ο αλγόριθμος οφείλει να ελέγχει τη λίστα διαστημάτων για τυχόν μεταβολές μόνο στα σημεία αυτά



## Ο αλγόριθμος γραμμικής σάρωσης για τον καταμερισμό καταχωρητών

```
LinearScanRegisterAllocation
  active ← {}
  foreach live interval i in order of increasing start point
    ExpireOldIntervals(i)
    if length(active) = R then
      SpillAtInterval(i)
    else
      register[i] ← a register removed from pool of free registers
      add i to active, sorted by increasing end point
ExpireOldIntervals(i)
  foreach interval j in active, in order of increasing end point
    if endpoint[j] ≥ startpoint[i] then
      return
    remove j from active
    add register[j] to pool of free registers
SpillAtInterval(i)
  spill ← last interval in active
  if endpoint[spill] > endpoint[i] then
    register[i] ← register[spill], location[spill] ← new stack location
    remove spill from active
    add i to active, sorted by increasing end point
  else
    location[i] ← new stack location
```



## Περιγραφή του αλγορίθμου γραμμικής σάρωσης (1)

- Σε κάθε βήμα, ο αλγόριθμος διατηρεί μία λίστα (*active*) των διαστημάτων ζωής τα οποία είναι ενεργά στο τρέχον σημείο και έχουν τοποθετηθεί σε καταχωρητές. Η λίστα διατηρείται ταξινομημένη κατά αυξανόμενο σημείο λήξης (end point)
- Από την λίστα απομακρύνονται τα διαστήματα που έχουν εκπνεύσει (expired) τα οποία είναι αυτά για τα οποία το σημείο λήξης προηγείται το σημείο έναρξης του νέου διαστήματος το οποίο εξετάζει ο αλγόριθμος και οι αντίστοιχοι καταχωρητές γίνονται ξανά διαθέσιμοι
- Η σάρωση παύει προσωρινά αν φτάσει στο τέλος της λίστας *active* (και τότε η λίστα μένει κενή) ή συναντήσει ένα διάστημα το σημείο λήξης του οποίου έπεται του σημείου έναρξης του νέου διαστήματος ζωής
- Το μήκος της λίστας *active* είναι το πολύ ίσο με $R$

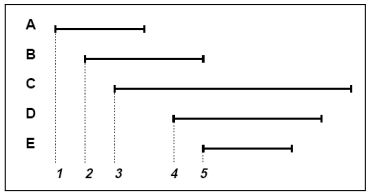


## Περιγραφή του αλγορίθμου γραμμικής σάρωσης (2)

- Στην περίπτωση που η λίστα έχει φτάσει σε μήκος $R$ στην έναρξη ενός νέου διαστήματος και ταυτόχρονα δεν εκπνέει κάποιο από τα ενεργά διαστήματα, ένα από τα ενεργά διαστήματα πρέπει να οδηγηθεί σε έγχυση στη μνήμη
- Η επιλογή του κατάλληλου (αν υπάρχουν πολλά υποψήφια) διαστήματος για έγχυση είναι κατά κανόνα ευριστική
  - Μία λύση είναι η έγχυση του διαστήματος το οποίο λήγει τελευταίο, στην περισσότερο απομακρυσμένη θέση από το τρέχον σημείο
  - Το διάστημα αυτό είναι είτε το νέο διάστημα, είτε το τελευταίο ενεργό διάστημα
  - Η λύση αυτή αποδεικνύεται ότι παράγει κώδικα με τον ελάχιστο αριθμό εγχύσεων για κώδικα χωρίς άλματα (straight-line code)

# Παράδειγμα εφαρμογής του αλγορίθμου γραμμικής σάρωσης για τον καταμερισμό καταχωρητών

- Έστω οι προσωρινές μεταβλητές `A, B, C, D, E` και τα αντίστοιχα διαστήματα 1 ως 5 του σχήματος και $R = 2$




| | | |
|---|---|---|
| Έναρξη I1 | active = $\langle A \rangle$ | $R0 \leftarrow A$ |
| Έναρξη I2 | active = $\langle A, B \rangle$ | $R1 \leftarrow B$ |
| Έναρξη I3 | active = $\langle A, B \rangle$ | $R0 \leftarrow A$, $R1 \leftarrow B$, spill $C$ |
| Έναρξη I4 | active = $\langle B, D \rangle$ | $R0 \leftarrow D$, $R1 \leftarrow B$, $C$ spilled |
| Έναρξη I5 | active = $\langle D, E \rangle$ | $R0 \leftarrow D$, $R1 \leftarrow E$, $C$ spilled |



# Υπολογιστική πολυπλοκότητα του αλγορίθμου γραμμικής σάρωσης

- Έστω $V$ ο αριθμός των μεταβλητών (διαστημάτων ζωής) σε ένα υποπρόγραμμα και $R$ ο αριθμός των φυσικών καταχωρητών που είναι διαθέσιμοι για καταμερισμό
- Δεδομένου ότι $R$ = σταθερά, ο αλγόριθμος χρειάζεται $O(V)$ χρόνο κατά την εκτέλεσή του
- Η χείριστη περίπτωση (worst case) για το χρόνο εκτέλεσης εξαρτάται από το χρόνο που απαιτείται για την εισαγωγή ενός νέου διαστήματος στη λίστα *active*
- Αν χρησιμοποιηθεί ένα ισοσταθμισμένο δυαδικό δένδρο ως αντίστοιχη δομή δεδομένων για την αποθήκευση της λίστας, η εισαγωγή ενός νέου στοιχείου στη λίστα απαιτεί $O(\log R)$ χρόνο και ο συνολικός αλγόριθμος $O(V \times \log R)$
- Η γραμμική αναζήτηση για το σημείο εισαγωγής χρειάζεται $O(R)$ χρόνο και η συνολική πολυπλοκότητα ανάγεται τότε σε $O(V \times R)$
- Συνήθως, λόγω της απλότητας της δομής δεδομένων που συνεπάγεται η γραμμική αναζήτηση, η δεύτερη προσέγγιση είναι ταχύτερη για μικρές τιμές του $R$



# Αναφορές του μαθήματος I


- A. V. Aho, R. Sethi, and J. D. Ullman, *Μεταγλωττιστές: Αρχές, Τεχνικές και Εργαλεία*, με την επιμέλεια των: Αγγελος Σπ. Βώρος και Νικόλαος Σπ. Βώρος και Κων/νος Γ. Μασσέλος, **κεφάλαια 8.8, 9.2.1-9.2.5**, Εκδόσεις Νέων Τεχνολογιών, 2008. Website for the English version: `http://dragonbook.stanford.edu`
- G. J. Chaitin, "Register allocation and spilling via graph coloring," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, June 1982, pp. 98–105.
- M. Poletto and V. Sarkar, "Linear scan register allocation," *ACM Transactions on Programming Languages and Systems*, vol. 21, no. 5, pp. 895–913, September 1999.