# Ε-85: Ειδικά Θέματα Λογισμικού
## Προγραμματισμός Συστημάτων Υψηλών Επιδόσεων

Χειμερινό Εξάμηνο 2009-10

«Ιεραρχία Μνήμης και
Τεχνικές Βελτιστοποίησης»

Παναγιώτης Χατζηδούκας
(Π.Δ. 407/80)



## Ζητήματα Απόδοσης

- Οι σύγχρονοι υπολογιστές χρησιμοποιούν μία ποικιλία τεχνικών για καλύτερη απόδοση
  - caches
    - μικρού μεγέθους γρήγορες μνήμες όπου τα δεδομένα αποθηκεύονται με την ελπίδα να επαναχρησιμοποιηθούν αυτά ή γειτονικά τους δεδομένα
  - παραλληλισμός
    - οι επεξεργαστές μπορούν να έχουν πολλαπλές λειτουργικές μονάδες που εργάζονται παράλληλα
  - pipelining
    - ένα είδος παραλληλισμού, όπως μία γραμμή παραγωγής σε εργοστάσιο
- Θεωρητικά οι compilers μπορούν να βελτιστοποιήσουν ένα πρόγραμμα, όχι όμως και στην πράξη



## Ιεραρχία Μνήμης

- Στα σύγχρονα υπολογιστικά συστήματα τα ίδια δεδομένα είναι αποθηκευμένα σε διάφορες συσκευές κατά την επεξεργασία
- Οι συσκευές αποθήκευσης μπορούν να περιγραφούν και να ταξινομηθούν με βάση την ταχύτητα και την απόσταση τους από τον επεξεργαστή
- Υπάρχει μία ιεραρχία αντικειμένων μνήμης
- Ο προγραμματισμός σε μία μηχανή με ιεραρχία μνήμης απαιτεί βελτιστοποίηση για αυτή τη δομή μνήμης

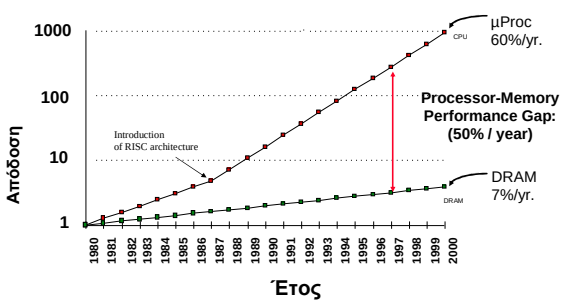


## Αναγκαιότητα για Ιεραρχία Μνήμης

- Ο επεξεργαστής (CPU) λειτουργεί σε πολύ ταχύτερο ρυθμό από ότι η μνήμη μπορεί να παραδίδει δεδομένα.
  - Η διαφορά αυτή μεγάλωσε αρκετά τα τελευταία χρόνια
- Είναι δυνατή η δημιουργία συσκευών μνήμης που να λειτουργούν στην ταχύτητα του επεξεργαστή, αλλά με μικρή αποθηκευτική χωρητικότητα και πολύ μεγαλύτερο κόστος
- Η ιεραρχία μνήμης είναι η μηχανική βελτιστοποίηση για το πρόβλημα “μέγεθος έναντι ταχύτητας“
- Βασικές ιδιότητες των περισσοτέρων αλγορίθμων, π.χ. τοπικότητα των αναφορών, προάγουν τη χρήση της ιεραρχίας μνήμης

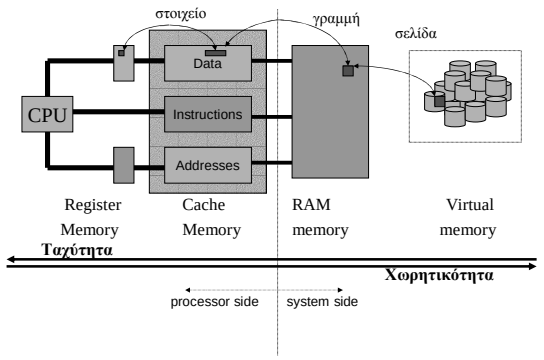


## Processor-DRAM Gap (latency)



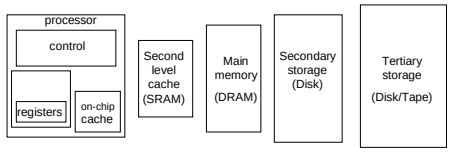



## Ιεραρχία Μνήμης



## Ιεραρχία Μνήμης

- Στοιχεία για το σύστημα Origin2000

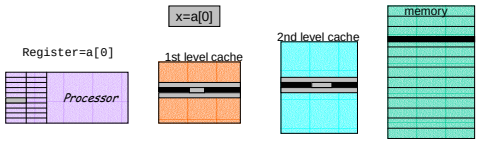



| | | | | | |
|---|---|---|---|---|---|
| Ταχύτητα (ns): | ~5 | ~75 | ~500 | ~10 ms | ~10 sec |
| Χωρητικότητα : | ~KB | ~MBs | ~GBs | ~TBs | 100 TBs |



## Cache Lines

Κανονικά περισσότερα από ένα στοιχεία μεταφέρονται κάθε φορά




- Η μονάδα μεταφοράς καλείται cache line
- Μια cache line περιλαμβάνει διαδοχικές θέσεις μνήμης
- Το μέγεθος μίας cache line εξαρτάται από την αρχιτεκτονική
- Παράδειγμα για Origin2000:
  - 32 bytes 1st level cache
  - 128 bytes 2nd level cache



## Ορθότητα της Ιεραρχίας Μνήμης

- Κάποια δεδομένα είναι αποθηκευμένα σε πολλά μέρη:
  - δεν υπάρχει πρόβλημα με read-only δεδομένα (replication)
  - όταν όμως τροποποιούνται απαιτείται ενημέρωση τους σε όλα τα μέρη (cache → μνήμη → δίσκος)
- Η χρήση αυτών των μηχανισμών σημαίνει κόστος στην απόδοση
- Για αποδοτική χρήση της ιεραρχίας μνήμης τα προγράμματα πρέπει να εκμεταλλεύονται την τοπικότητα των αναφορών



## Τοπικότητα των Αναφορών

- Χρονική τοπικότητα (temporal locality):
  - Τα αντικείμενα (εντολές και δεδομένα) που χρησιμοποιήθηκαν πρόσφατα είναι πιθανόν να επαναχρησιμοποιηθούν στο κοντινό μέλλον:
    - επαναληπτικοί βρόχοι, συναρτήσεις, τοπικές μεταβλητές
- Χωρική τοπικότητα (spatial locality):
  - τα προγράμματα προσπελαύνουν δεδομένα που είναι κοντά μεταξύ τους:
    - πράξεις σε πίνακες και διανύσματα
    - το μέγεθος της cache line σχετίζεται άμεσα με την spatial locality
- Ακολουθιακή τοπικότητα (sequential locality):
  - ο επεξεργαστής εκτελεί εντολές σύμφωνα με τη διάταξη που ορίζει το πρόγραμμα:
    - ο λόγος εντολές διακλάδωσης / υπόλοιπες εντολές είναι συνήθως 1 προς 5



## Τεχνικές Βελτιστοποίησης

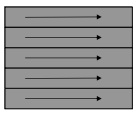
- Οι τεχνικές που θα συζητηθούν βασίζονται στους βρόχους και είναι οι ακόλουθες:
  - Εναλλαγή (Interchange)
  - Ξετύλιγμα (Unrolling)
  - Σύμπτυξη (Fusion)
  - Διάσπαση (Fission)



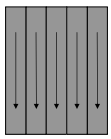
## Αποθήκευση στη Μνήμη

- Η διάταξη αποθήκευσης εξαρτάται από την γλώσσα προγραμματισμού:

C → "κατά-γραμμές"



Fortran → "κατά-στήλες"



- Η προσπέλαση των στοιχείων σύμφωνα με τη διάταξή τους βελτιώνει την απόδοση για μεγέθη προβλημάτων που δε χωράνε στην cache (spatial locality)

## Εναλλαγή Βρόγχων

- Loop Interchange
- Βασική Ιδέα: Σε ένα πολλαπλό βρόχο, εξέταση και πιθανή αλλαγή της σειράς των βρόχων
- Πλεονεκτήματα:
  - Καλύτερα διάταξη των προσπελάσεων στη μνήμη (οδηγεί σε καλύτερη χρήση της cache και της μνήμης)
  - Απαλοιφή των εξαρτήσεων των δεδομένων (αύξηση των δυνατοτήτων για βέλτιστη χρήση επεξεργαστή και παραλληλοποίηση)
- Μειονέκτημα:
  - Μπορεί ένας μικρός βρόχος να γίνει εσωτερικός



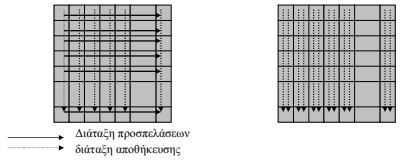
## Εναλλαγή Βρόχων – Παράδειγμα 1

*Αρχικά*

```
DO I=1,N
   DO J=1,M
     C(I,J)=A(I,J)+B(I,J)
   END DO
END DO
```

*Μετά την εναλλαγή*

```
DO J=1,M
   DO I=1,N
     C(I,J)=A(I,J)+B(I,J)
   END DO
END DO
```






## Εναλλαγή Βρόχων στην C

Στη C, η περίπτωση είναι ακριβώς η αντίθετη

```
for (j=0; j<N; j++)
    for (i=0; i<N; i++)
        C[i][j] = A[i][j] +B[i][j];
```

εναλλαγή

```
for (i=0; i<N; i++)
    for (j=0; j<N; j++)
        C[i][j] = A[i][j] +B[i][j];
```

αντιστροφή δεικτών

```
for (j=0; j<N; j++)
    for (i=0; i<N; i++)
        C[j][i] = A[j][i] +B[j][i];
```

- Το κέρδος στην απόδοση είναι το ίδιο σε αυτή την περίπτωση
- Συχνά, στην πράξη, η εναλλαγή πραγματοποιείται πολύ ευκολότερα από την αντιστροφή των δεικτών



## Εναλλαγή Βρόχων – Παράδειγμα 2

```
COMMON / SHARE /B (M,N)
. . . . . .

SUBROUTINE DO_WORK (N,M,A)
REAL A(N,M)

DO J=1,M
   DO I=1,N
      A(I,J) = A(I,J) + B(J,I)
   END DO
END DO
```

Η εναλλαγή θα είναι καλή για το B αλλά κακή για το A

*Αντιστροφή δεικτών για B;*



## Ξετύλιγμα Βρόχων

- Loop Unrolling: ταυτόχρονη εκτέλεση πολλαπλών επαναλήψεων ενός βρόγχου

```
DO I=1,N,1
   ...(I)...
END DO
```

```
DO I=1,N,unroll
   ...(I)...
   ...(I+1)...
   ...(I+2)...
   ...(I+unroll-1)...
END DO
```

- Πλεονεκτήματα της τεχνικής:
  - Περισσότερες ευκαιρίες για superscalar κώδικα (pipeline)
  - Αυξημένη επαναχρησιμοποίηση των δεδομένων
  - Αξιοποίηση - Εκμετάλλευση των cache lines
  - Μείωση του loop overhead (όχι τόσο σημαντικό πλεονέκτημα)
- Μειονεκτήματα:
  - Απαίτηση κώδικα για την εξασφάλιση του ίδιου αριθμού επαναλήψεων (do i=1,N-mod(N,unroll) & do i=N-mod (N, unroll)+1,N)



## Ξετύλιγμα Εξωτερικού Βρόχου

```
DO I=1,N
   DO J=1,N
      A(I)=A(I)+B(I,J)*C(J)
   END DO
END DO
```

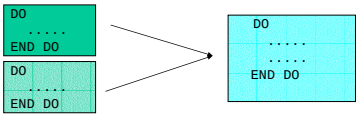
Ο βαθμός του ξετυλίγματος πρέπει να ταιριάζει με το μέγεθος της cache line

```
DO I=1,N,4
   DO J=1,N
     A(I)   =A(I)   +B(I  ,J)*C(J)
     A(I+1)=A(I+1)+B(I+1,J)*C(J)
     A(I+2)=A(I+2)+B(I+2,J)*C(J)
     A(I+3)=A(I+3)+B(I+3,J)*C(J)
   END DO
END DO
```

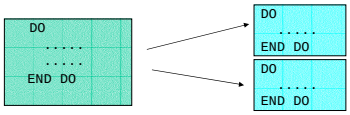
Με τον τρόπο αυτό οι τιμές του B που βρίσκονται στην cache χρησιμοποιούνται πλήρως

## Σύμπτυξη και Διάσπαση Βρόχων

- Loop Fusion and Fission
- Fusion: Ένωση πολλαπλών βρόχων σε έναν




- Fission: Διαχωρισμός ενός βρόχου σε πολλαπλούς






## Σύμπτυξη Βρόχων – Παράδειγμα 1

```
DO I=1,N
   B(I)=2*A(I)
END DO
```

Για μεγάλο N, μόνο το τελευταίο μέρος του B θα παραμείνει στη cache όταν ο βρόγχος τερματίσει

```
DO K=1,N
   C(K)=B(K)+D(K)
END DO
```

Ο πίνακας B() πρέπει να φορτωθεί ξανά

Ο B δεν χρειάζεται να φορτωθεί ξανά
Παράλληλη εκτέλεση των δύο πράξεων
Μικρότερο loop overhead
*Μειονεκτήματα:*

- Συνολικά 4 πίνακες ανταγωνίζονται για χώρο στην cache
- Απαίτηση για περισσότερους καταχωρητές

```
DO I=1,N
   B(I)=2*A(I)
   C(I)=B(I)+D(I)
END DO
```



## Διάσπαση Βρόχων – Παράδειγμα 1

```
DO I=1,N
   B(I)=2*A(I)
   D(I)=D(I-1)+C(I)
END DO
```

```
DO I=1,N
   B(I)=2*A(I)
END DO
```

```
DO I=1,N
   D(I)=D(I-1)+C(I)
END DO
```

- Ένδειξη για συγχώνευση: ανεξάρτητες πράξεις σε έναν απλό βρόχο
- Πλεονέκτημα:
  - Αποδοτικότερη δρομολόγηση του πρώτου βρόχου και πιθανή παραλληλοποίηση του
- Μειονεκτήματα:
  - Λιγότερες δυνατότητες για superscalar εκτέλεση
  - Δημιουργία πρόσθετου βρόχου



## Διάσπαση Βρόχων – Παράδειγμα 2

```
DO I=1,N
   A(I) = EXP(-L*B(I))
   DO J = 1,N
      C(I,J)=C(I,J)+A(I)*D(J)
   END DO
END DO
```

χρονοβόρα πράξη

κακή προσπέλαση μνήμης

Αποδοτικότερη δρομολόγηση

```
DO I=1,N
   A(I)= EXP(-L*B(I))
END DO
```

Βέλτιστη προσπέλαση μνήμης

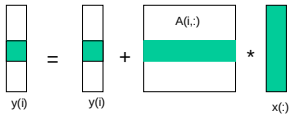
```
DO J=1,N
   DO I=1,N
      C(I,J)=C(I,J)+A(I)*D(J)
   END DO
END DO
```



## Πολλαπλασιασμός Πίνακα x Διάνυσμα

- y = y + A*x

```
for i = 1:n
    for j = 1:n
        y(i) = y(i) + A(i, j)*x(j)
```






## Πολλαπλασιασμός Πίνακας x Διάνυσμα

```
{μετάφερε το x(1:n) στη γρήγορη μνήμη}
{μετάφερε το y(1:n) στη γρήγορη μνήμη}
for i = 1:n
    {μετάφερε τη γραμμή i του A στην γρήγορη μνήμη}
    for j = 1:n
        y(i) = y(i) + A(i,j)*x(j)
{γράψε το y(1:n) πίσω στην αργή μνήμη}
```

- Αριθμός αναφορών στην αργή μνήμη = $3*n + n^2$
- Αριθμός αριθμητικών πράξεων = $2*n^2$
- Η απόδοση της συνολικής πράξης περιορίζεται από την ταχύτητα της αργής μνήμης

## Παρατηρήσεις

- Η πραγματική απόδοση ενός απλού προγράμματος μπορεί να είναι μία περίπλοκη συνάρτηση της αρχιτεκτονικής
- Μικρές αλλαγές στη δομή του προγράμματος μπορούν να αλλάξουν σημαντικά την απόδοσή του
- Για τη δημιουργία γρήγορων προγραμμάτων πρέπει να ληφθεί υπόψη η υποκείμενη αρχιτεκτονική