ΕΠΙΣΤΗΜΟΝΙΚΟΣ ΥΠΟΛΟΓΙΣΜΟΣ Ι

3ο ΣΕΤ ΔΙΑΦΑΝΕΙΩΝ 09-16/10

Η. ΤΣΙΓΑΡΙΔΑΣ & Ε. ΓΑΛΛΟΠΟΥΛΟΣ
Τμήμα Μηχ. Η/Υ και Πληροφορικής
Παν/μιο Πατρών

Βασικές πράξεις γραμμικής άλγεβρας

Έστω $x, y \in \mathbb{R}^n$, $\alpha \in \mathbb{R}$.

sDOT $\sigma \leftarrow \sigma + x^\top y$ «σμίκρυνση» (= reduction),

sAXPY $y \leftarrow y + \alpha x$ «τριάδα».

sDOT	sAXPY
for $i = 1 : n$	for $i = 1 : n$
$\sigma = \sigma + \xi_i * \eta_i$	$\eta_i = \eta_i + \alpha * \xi_i$
end	end

Ανάλυση DOT

- $\Omega = 2n$, $\Phi_{\min} = 2n + 2$.
- Αν το μερικό άθροισμα παραμένει σε καταχωρητή μέχρι το τέλος του βρόχου

  τότε $\Phi = \Phi_{\min}$, $\mu = \mu_{\min}$

  ειδάλλως $\Phi = 3n +$ O(1).
- $\mu = \mu_{\min} = 1 + \frac{1}{n}$.

Ανάλυση: sAXPY

- $\Omega = 2n$
- $\Phi_{\min} = 3n + 1$.
- Το Φ_{min} επιτυγχάνεται, εφόσον το α παραμείνει σε καταχωρητή κατά τη διάρκεια του υπολογισμού.
- $\mu = \mu_{\min} = \frac{3}{2} + \frac{1}{2n}$.
- Το όνομα προέρχεται από τις λέξεις a x plus y.

Υλοποιήσεις σε LOAD-STORE

DOT	**sAXPY**
LOAD x, y	LOAD x, y, α
for $i = 1 : n$	for $i = 1 : n$
$\sigma = \sigma + \xi_i * \eta_i$	$\eta_i = \eta_i + \alpha * \xi_i$
end	end
STORE σ	STORE y

Επιτυγχάνεται το $\Phi_{\min}$ με κρυφή μνήμη O(n).

Υλοποίηση με κρυφή μνήμη $O(1)$

DOT	**sAXPY**
LOAD σ	LOAD α
for $i = 1 : n$	for $i = 1 : n$
LOAD η_i, ξ_i	LOAD η_i, ξ_i
$\sigma = \sigma + \xi_i * \eta_i$	$\eta_i = \eta_i + \alpha * \xi_i$
end	STORE η_i
STORE σ	end

Κόστος DOT: $\Phi = 2n + 2$ και $\Omega = 2n$ με O(1) θέσεις καταχωρητών και άμεσης μνήμης.
Κόστος sAXPY: $\Phi = 3n + 1$ και $\Omega = 2n$ με O(1) θέσεις καταχωρητών και άμεσης μνήμης.
ΠΡΟΣΟΧΗ Επιτυγχάνεται $\Phi = \Phi_{\min}$ με κρυφή μνήμη O(1) ...!

Κοινά χαρακτηριστικά DOT, sAXPY

- Πράξεις διανυσμάτων και βαθμωτών.
- Ειδική περίπτωση του ΑΓΑ.0
- όπου $n_i = n_j = 1$, $n_k > 1$ για $i, j, k \in \{1, 2, 3\}$.
- $\Omega = O(n) = \Phi_{\min}$: Γραμμική πολυπλοκ. γραμμικές στην κυρίαρχη διάσταση
- $\mu_{\min} = O(1)$: Μη αποδοτική υλοποίηση σε ιεραρχική μνήμη

Χαρακτηρίζονται ως

Βασικές πράξεις γραμμικής άλγεβρας 1ου επιπέδου: BLAS-1

Αν λάβουμε σαν βασική διάσταση το n, το κόστος είναι γραμμικό - λέμε ότι οι πράξεις αυτές είναι *1ου επιπέδου*.

Παράδειγμα υλοποίησης sAXPY

```
      subroutine daxpy(n,da,dx,incx,dy,incy)
c     constant times a vector plus a vector. c     uses unrolled
loops for increments equal to one. c     jack dongarra, linpack,
3/11/78. c     modified 12/3/93, array(1) declarations changed to
array(*)
      double precision dx(*),dy(*),da
      integer i,incx,incy,ix,iy,m,mp1,n
c
      if(n.le.0)return
      if (da .eq. 0.0d0) return


      if(incx.eq.1.and.incy.eq.1)go to 20
c        code for unequal increments or equal increments not equal
to 1
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        dy(iy) = dy(iy) + da*dx(ix)
        ix = ix + incx
        iy = iy + incy
10    continue
      return
c        code for both increments equal to 1 c        clean-up
loop 20    m = mod(n,4)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dy(i) = dy(i) + da*dx(i)
30    continue
      if( n .lt. 4 ) return
40    mp1 = m + 1
      do 50 i = mp1,n,4
        dy(i) = dy(i) + da*dx(i)
        dy(i + 1) = dy(i + 1) + da*dx(i + 1)
        dy(i + 2) = dy(i + 2) + da*dx(i + 2)
        dy(i + 3) = dy(i + 3) + da*dx(i + 3)
50    continue
      return
      end
```

Ανανέωση 1ης τάξης: $n_1, n_2 > 1, n_3 = 1$ (συχνά ... ονομάζεται GER αν και η ρίζα του ονόματος είναι απλά το R!!)

$$C \leftarrow C + ab^\top, \;\; C \in \mathbf{R}^{n_1 \times n_2}, a \in \mathbf{R}^{n_1}, b \in \mathbf{R}^{n_2},$$

- $\Omega = 2n_1n_2$
- $\Phi_{\min} = 2n_1n_2 + n_1 + n_2$
- $\mu_{\min}^{\text{rank-1}} = 1 + \frac{1}{2n_1} + \frac{1}{2n_2}$

```
for j = 1 : n2
    for i = 1 : n1
        γij = γij + αi * βj
    end
end
```

Υλοποίηση βέλτιστου $\Phi_{\min}$

```
LOAD a, b, C
for j = 1 : n2
    for i = 1 : n1
        γij = γij + αi * βj
    end
end
STORE C
```

απαιτεί $\mathcal{K} = O(n_1 \times n_2)$

Υλοποίηση με $\mathcal{K} = O(1)$)

```
for j = 1 : n2
    LOAD βj
    for i = 1 : n1
        LOAD γij, αi
        γij = γij + αi * βj
        STORE γij
    end
end
```

- $\Phi = \sum_{j=1}^{n_2}(1 + \sum_{i=1}^{n_1} 3) = n_2 + 3n_1n_2$
- κατά $n_1n_2 - n_1$ μεγαλύτερο του $\Phi_{\min}$ με $\mathcal{K} = O(1)$.
- Μειώσαμε το χώρο, αυξήσαμε το χρόνο.

Παρατήρηση: Αλγόριθμος 1 για $\mathcal{K} = O(n_1)$.

LOAD a
for $j = 1 : n_2$
  LOAD β_j
  for $i = 1 : n_1$
    LOAD γ_{ij}
    $\gamma_{ij} = \gamma_{ij} + \alpha_i * \beta_j$
    STORE γ_{ij}
  end
end

- $\Phi = n_1 + \sum_{j=1}^{n_2}(1 + \sum_{i=1}^{n_1} 2) = n_1 + n_2 + 2n_1n_2$
- Επιτυγχάνεται $\Phi_{\min}$ με $\mathcal{K} = O(n_1)$

Εναλλακτικά: Αλγόριθμος 2 για $\mathcal{K} = O(n_2)$

LOAD b
for $i = 1 : n_1$
  LOAD α_i
  for $j = 1 : n_2$
    LOAD γ_{ij}
    $\gamma_{ij} = \gamma_{ij} + \alpha_i * \beta_j$
    STORE γ_{ij}
  end
end

- $\Phi = n_2 + \sum_{i=1}^{n_1}(1 + \sum_{i=1}^{n_2} 2) = n_2 + n_1 + 2n_1n_2$
- Επιτυγχάνεται $\Phi_{\min}$ με $\mathcal{K} = O(n_2)$.
- Ιδέα για «υβριδικό» αλγόριθμο που λειτουργεί ανάλογα με το μέγεθος των n_1, n_2:

Υλοποίηση με $\mathcal{K} = O(\min(n_1, n_2))$

Αν $\mathcal{K} = O(n_1)$ τότε
  επιλέγουμε τον Αλγόριθμο 1
αλλοιώς
  επιλέγουμε τον Αλγόριθμο 2

Τι κάνουμε αν $\mathcal{K} < O(\min(n_1, n_2))$;

Γενικός τεμαχισμός → «Ορμαθοποίηση» Τεμαχίζουμε το C σε τμήματα μεγέθους $m_1 \times m_2$ όπου $n_j = m_j k_j$ και αντίστοιχα τα a, b:

$$\begin{aligned} C_{IJ} &:= \gamma_{(I-1)m_1+1:Im_1,\ (J-1)m_2+1:Jm_2}, I = 1:k_1, J = 1:k_2, \\ a_I &:= \alpha_{(I-1)m_1+1:Im_1}, \\ b_J &:= \beta_{(J-1)m_2+1:Jm_2}, \end{aligned}$$

$$\begin{pmatrix} C_{11} & \dots & C_{1,k_2} \\ & \ddots & \vdots \\ \vdots & C_{I,J} & \vdots \\ \vdots & \ddots & \vdots \\ C_{k_1,1} & \dots & C_{k_1,k_2} \end{pmatrix} = \begin{pmatrix} C_{11} & \dots & C_{1,k_2} \\ & \ddots & \vdots \\ \vdots & C_{I,J} & \vdots \\ \vdots & \ddots & \vdots \\ C_{k_1,1} & \dots & C_{k_1,k_2} \end{pmatrix} + \begin{pmatrix} a_1 \\ \vdots \\ a_I \\ \vdots \\ a_{k_1} \end{pmatrix} \begin{pmatrix} \cdots & b_J^\top & \cdots \end{pmatrix}$$

Το τελικό αποτέλεσμα αποτελείται από όλα τα

$$C_{IJ} = C_{IJ} + a_I b_J^\top, \qquad I = 1:k_1, J = 1:k_2$$

for $J = 1:k_2$
  for $I = 1:k_1$
    (* Υλοποίηση της $C_{IJ} = C_{IJ} + a_I b_J^\top$ *)
end
end

Συνθέσαμε την ανανέωση μητρώου μεγέθους $n_1 \times n_2$ από ανανεώσεις μητρώων μεγέθους $m_1 \times m_2$, όπου $n_j = m_j k_j$.

Μονοδιάστατος τεμαχισμός ως προς n_2 Αν, για παράδειγμα, επιλέξουμε $m_2 = O(\mathcal{K})$ και τεμαχίσουμε το b σε k_2 τμήματα:

for $J = 1:k_2$
  LOAD b_J
    $C_{:,J} = C_{:,J} + a_{:} b_J$
end

Κάθε $C_{:,J} = C_{:,J} + ab_J^\top$ υλοποιείται με το βέλτιστο αριθμό μεταφορών,

$$\Phi(J) = n_1 + m_2 + 2n_1 m_2$$

Συνολικά

$$\Phi = \sum_{J=1}^{k_2} \Phi(J) = n_1 k_2 + n_2 + 2n_1 n_2$$

και

$$\mu = 1 + \frac{1}{2n_1} + \frac{1}{2m_2}$$

Επομένως με τον παραπάνω τεμαχισμό οι παραπάνω όροι ελαχιστοποιούνται λαμβάνοντας $m_2 = \mathcal{K}$:

$$\mu = 1 + \frac{1}{2\mathcal{K}} + \frac{1}{2n_1}$$

Εναλλακτικά: Μπορούμε να τεμαχίσουμε το a και να συνθέσουμε μέσω βέλτιστων υλοποιήσεων της $C_{I,:} = C_{I,:} + a_I b^\top$.
Συμπεράσματα

- ο τεμαχισμός που οδηγεί στην καλύτερη υλοποίηση καθορίζεται από το μέγεθος της κρυφής μνήμης και των καταχωρητών·
- η ελαχιστοποίηση έγινε εξετάζοντας την υλοποίηση που επιτυγχάνει το $\mu_{\min}$ για το μικρότερο πρόβλημα,
- η αύξηση της απόδοσης οφείλεται σε υλοποιήσεις που παρουσιάζουν αυξημένη χρονική τοπικότητα.
- εξετάζουμε και υλοποιήσεις που βασίζονται σε αναστροφή των βρόχων.

Μητρώο × διάνυσμα: MV

$$c \leftarrow c + Ab, c \in \mathbb{R}^{n_1 \times 1}, A \in \mathbb{R}^{n_1 \times n_3}, b \in \mathbb{R}^{n_3 \times 1}$$

ή

$$c \leftarrow c + a^\top B, c \in \mathbb{R}^{1 \times n_2}, a \in \mathbb{R}^{n_3 \times 1}, B \in \mathbb{R}^{n_3 \times n_2}$$

Στη συνέχεια εξετάζουμε την πρώτη περίπτωση.

- MV = A x plus y,
- $\Omega = 2n_1 n_3$, $\Phi_{\min} = n_1 n_3 + 2n_1 + n_3$,
- $\mu_{\min} = \frac{1}{2} + \frac{1}{2n_1} + \frac{1}{n_3}$.

Υλοποίηση

Μορφή DOT/ ij/ κ. γραμμές: τα στοιχεία του y υπολογίζονται από το εσωτερικό γινόμενο των γραμμών του A με το x,

$$\eta_i \leftarrow \eta_i + a_{i,:}^\top x = \eta_i + \sum_{k=1}^{n_3} \alpha_{ik}\xi_k, \quad i = 1, ..., n_1.$$

Μορφή sAXPY/ ji / κ. στήλες τα στοιχεία του y υπολογίζονται από τον γραμμικό συνδυασμό των στηλών του A:

$$y \leftarrow y + \sum_{k=1}^{n_3} a_{:,k}\xi_k.$$

Μορφή DOT / ij/ κ. *γραμμές*	*Μορφή* sAXPY / ji/ κ. *στήλες*

for $i = 1 : n_1$
  for $j = 1 : n_3$
  $\eta_i = \eta_i + \alpha_{ij} * \xi_j$
  end
end
	for $j = 1 : n_3$
	  for $i = 1 : n_1$
	  $\eta_i = \eta_i + \alpha_{ij} * \xi_j$
	  end
	end

Υλοποιήσεις

```
LOAD A, x, y
for i = 1 : n1
    for j = 1 : n3
        ηi = ηi + αij * ξj
    end
end
STORE y
```

- $\Phi = \Phi_{\min} = 2n_1 + n_1n_3 + n_3$.
- Απαιτείται $\mathcal{K} = O(n_1n_3)$

Υλοποιήσεις

```
for i = 1 : n1
    LOAD ηi
    for j = 1 : n3
        LOAD αij, ξj
        ηi = ηi + αij * ξj
    end
    STORE ηi
end
```

- $\Phi = \sum_{i=1}^{n_1}(2 + \sum_{j=1}^{n_3} 2) = 2n_1 + 2n_1n_3$
- Απαιτείται $\mathcal{K} = O(1)$

Υλοποίηση αν $\mathcal{K} = O(n_3)$

```
LOAD x
 for i = 1 : n_1
     LOAD η_i
      for j = 1 : n_3
          LOAD α_ij
          η_i = η_i + α_ij * ξ_j
      end
     STORE η_i
 end
```

- $\Phi = n_3 + \sum_{i=1}^{n_1}(2 + \sum_{j=1}^{n_3} 1) = n_3 + 2n_1 + n_1 n_3 = \Phi_{\min}$
- Απαιτείται $\mathcal{K} = O(n_3)$
- Υλοποίηση sAXPY αν $\mathcal{K} = O(n_1)$
- Υβριδική υλοποίηση για $\mathcal{K} = O(\min(n_1, n_3))$

Γενική υλοποίηση
Έστω $n_3 = m_3 k_3$ και

$$A = [A_1, \cdots, A_{k_3}], x = [x_1; \cdots; x_{k_3}]$$

Διάσπαση σε υπογινόμενα:

$$y = y + A_1 x_1 + A_2 x_2 + \cdots A_{k_3} x_{k_3}$$

Αν $\mathcal{K} = O(m_3)$ τότε κάθε μερικό MV γίνεται με βέλτιστο $\Phi(J) = 2n_1 + n_1 m_3 + m_3$ άρα

$$\begin{aligned} \Phi &= \sum_{J=1}^{k_3} \Phi(J) \\ &= 2n_1 k_3 + n_1 n_3 + n_3. \end{aligned}$$

Εναλλακτική υλοποίηση
Έστω $n_1 = m_1 k_1$ και

$$y = [y_1; \cdots, ; y_{k_1}], A = [A_1; \cdots, ; A_{k_1}]$$

Διάσπαση:

$$y_I = y_I + A_I x, \quad I = 1 : k_1$$

Αν $\mathcal{K} = O(m_1)$ τότε κάθε μερικό MV γίνεται με βέλτιστο $\Phi(J) = 2m_1 + m_1 n_3 + n_3$ άρα

$$\begin{aligned} \Phi &= \sum_{J=1}^{k_1} \Phi(J) \\ &= 2n_1 + n_1 n_3 + k_1 n_3. \end{aligned}$$

Συμπερασμα

- Η επιλογή εξαρτάται από τη σχέση n_1, n_3 με $\mathcal{K}$.
- Αν $n_1 = n_3 = n$ οι τιμές είναι $n^2 + 2n + k_1 n$ και $n^2 + 2k_3 n + n$

  Για δεδομένο $\mathcal{K}$ και εφόσον $n_1 = n_3 = n$ ισχύει ότι $k_1 = k_3$ επομένως ο τεμαχισμός προς την n_3 απαιτεί λιγότερες μεταφορές.

Έχουμε

$$\begin{aligned} \Phi &= \min(\frac{2n_1n_3}{m_3} + n_1n_3 + n_3, 2n_1 + n_1n_3 + \frac{n_1}{m_1}n_3) \\ &= \min(\frac{2n_1n_3}{\mathcal{K}} + n_1n_3 + n_3, 2n_1 + n_1n_3 + \frac{n_1}{\mathcal{K}}n_3) \end{aligned}$$

Παρατήρηση Η καλύτερη υλοποίηση εξαρτάται από τη σχέση των διαστάσεων με το $\mathcal{K}$ και η επιλογή απαιτεί να εξετάσουμε το Φ για τις τιμές των δεικτών.

Κοινά χαρακτηριστικά MV, rank-1

- Ειδική περίπτωση του ΑΓΑ.0
- $n_i, n_j > 1, n_k = 1$ για $i, j, k \in \{1, 2, 3\}$
- $\Omega = O(n^2)$, $\Phi_{\min} = O(n^2)$, $\mu_{\min} = O(1)$
- Πολυπλοκότητες τετραγωνικές στην πρωταρχική διάσταση
- Μη αποδοτική υλοποίηση σε ιεραρχική μνήμη
- Καλύτερο $\mu_{\min}$ από τις BLAS-1

Βασικές πράξεις γραμμικής άλγεβρας 2ου επιπέδου: BLAS-2

Πολλαπλασιασμός μητρώων

$$C = C + AB, \quad C \in \mathbf{R}^{n_1 \times n_2}, A \in \mathbf{R}^{n_1 \times n_3}, B \in \mathbf{R}^{n_3 \times n_2}.$$

- $\Omega := 2n_1n_2n_3$, $\Phi_{\min} := 2n_1n_2 + n_1n_3 + n_2n_3$,
- $\mu_{\min} = \frac{1}{n_3} + \frac{1}{2n_1} + \frac{1}{2n_2}$.
- Αν τα n_j είναι ίσα με n τότε $\mu_{\min} = O(\frac{1}{n})$, που μπορεί να είναι πολύ μικρότερο των τιμών που συναντήσαμε στις προηγούμενες πράξεις.
- Δυνατότητα για αποδοτικότερες υλοποιήσεις από BLAS-1, 2
- Πολυπλοκότητα: Αριθμητικής κυβική - Μετακινήσεων τετραγωνική.

Βασική πράξη γραμμικής άλγεβρας 3ου επιπέδου: BLAS-3

Υλοποίηση

LOAD C, B, A
for $i = 1 : n_1$
  for $k = 1 : n_3$
    for $j = 1 : n_2$
      $\gamma_{ij} = \gamma_{ij} + \alpha_{ik}\beta_{kj}$
    end
  end
end
STOREC

$\mathcal{K} = O(n_1n_2 + n_2n_3 + n_1n_3)$ και $\Phi = \Phi_{\min}$.

- Για n_1, n_2, n_3 ώστε $\mathcal{K} = O(n_1n_2 + n_1n_3 + n_2n_3)$ το $\Phi_{\min}$ είναι εφικτό.
- Αλλοιώς τεμαχίζουμε σύμμορφα σε «ορμαθούς».

Υλοποίηση

for $i = 1 : n_1$
  for $k = 1 : n_3$
    LOAD α_{ik}
    for $j = 1 : n_2$
      LOAD $\gamma_{ij}, \beta_{kj},$
      $\gamma_{ij} = \gamma_{ij} + \alpha_{ik}\beta_{kj}$
      STORE γ_{ij}
    end
  end
end

$\mathcal{K} = O(1)$ και $\Phi = 3n_1n_2n_3 + n_1n_3$.

Σχήμα βρόχων για MM

for $? = 1 : n_?$
  for $? = 1 : n_?$
    for $? = 1 : n_?$
      $\gamma_{ij} = \gamma_{ij} + \alpha_{ik}\beta_{kj}$
    end
  end
end

Για την υλοποίηση πρέπει να αντικαταστήσουμε τα σύμβολα ? με τους δείκτες και αριθμούς $(i, 1), (j, 2), (k, 3)$. Ο πυρήνας του βρόχου είναι ανεξάρτητος από τον τρόπο διάταξης των δεικτών.

Αν χρησιμοποιήσουμε τον τριπλά εμφωλευμένο βρόχο έχουμε $6 = 3!$ εκδοχές του πολλαπλασιασμού.

σειρά βρόχου	**εσωτερικός βρόχος**	**μεσαίος βρόχος**	**τρόπος προσπέλασης**
ijk	DOT	διάνυσμα × μητρώο	A κ. στήλ., B κ. γραμμ.
jik	DOT	μητρώο × διάνυσμα	B κ. στήλ., A κ. γραμμ.
ikj	sAXPY	MV κ. γραμμές	B κ. γραμμές
jki	sAXPY	MV κ. στήλες	A κ. στήλες
kij	sAXPY	εξωτερικό γινόμενο	B κ. γραμμές
kji	sAXPY	εξωτερικό γινόμενο	A κ. στήλες

Για κάθε διάταξη των βρόχων, προκύπτει ο πολλαπλασιασμός μητρώων ως σύνθεση από διαφορετικές πράξεις των BLAS-1 και BLAS-2.

Παράδειγμα Ας δούμε τι εννοούμε στον προηγούμενο πίνακα, όταν λέμε, για παράδειγμα ότι
kji sAXPY εξωτερικό γινόμενο B κ. στήλες
Κατ' αρχήν, χρησιμοποιούμε το γεγονός ότι $C = C + AB$ μπορεί να γραφτεί ως εξής:

$$\begin{aligned} C &= C + \sum_{k=1}^{n_3} a_{:,k} b_{k,:} \\ &= (\cdots (C + a_{:,1} b_{1,:}) + a_{:,2} b_{2,:}) \cdots) + \cdots a_{:,n_3} b_{n_3,:} \end{aligned}$$

Για παράδειγμα,

$$\begin{aligned} C &= \begin{pmatrix} \gamma_{11} & \gamma_{12} \\ \gamma_{21} & \gamma_{22} \end{pmatrix} + \begin{pmatrix} \alpha_{11}\beta_{11} + \alpha_{12}\beta_{21} & \alpha_{11}\beta_{12} + \alpha_{12}\beta_{22} \\ \alpha_{21}\beta_{11} + \alpha_{22}\beta_{21} & \alpha_{21}\beta_1 + \alpha_{22}\beta_{22} \end{pmatrix} \\ &= \begin{pmatrix} \gamma_{11} & \gamma_{12} \\ \gamma_{21} & \gamma_{22} \end{pmatrix} + \begin{pmatrix} \alpha_{11} \\ \alpha_{21} \end{pmatrix} \begin{pmatrix} \beta_{11} & \beta_{12} \end{pmatrix} + \begin{pmatrix} \alpha_{12} \\ \alpha_{22} \end{pmatrix} \begin{pmatrix} \beta_{21} & \beta_{22} \end{pmatrix} \end{aligned}$$

δηλαδή, μπορούμε να γράψουμε το ΜΜ ως n_3 *διαδοχικές ανανεώσεις 1ης τάξης* του C. Αυτές οι διαδοχικές ανανεώσεις περιέχονται σε κάθε επανάληψη του εξωτερικού βρόχου.

Ο εξωτερικός δείκτης επανάληψης είναι το k που διατρέχει τη διάσταση n_3.
Ο μεσαίος δείκτης είναι το j που διατρέχει το n_2.
Ποια πράξη εκτελείται εντός του μεσαίου βρόχου; Η υλοποίηση κάθε

$$\underbrace{C = C + a_{:,k} b_{k,:}}_{\text{rank-1}}$$

γίνεται ως εξής: Αφού

$$[c_{:,1}, ..., c_{:,n_2}] = [c_{:,1}, ..., c_{:,n_2}] + a_{:,k} b_{k,:}$$

τότε η j στήλη του C ανανεώνεται ως εξής:

$$\underbrace{c_{:,j} = c_{:,j} + a_{:,k}\beta_{k,j}}_{\text{sAXPY}}$$

Προσέξτε ότι χρησιμοποιούμε το σύμβολο $\beta_{k,j}$ ως το j στοιχείο του διανύσματος $b_{k,:}$. Άρα μέσα στο μεσαίο δείκτη εκτελείται sAXPY.
Τέλος, μέσα στον εσωτερικο βρόχο, με δείκτη i που διατρέχει από 1 ως n_1, εκτελείται η (FMA !!)

$$\gamma_{ij} = \gamma_{ij} + \alpha_{ik}\beta_{kj}$$

Παρατηρήσεις

- Οι πολλαπλασιασμοί έχουν όλοι σα βάση πράξεις BLAS-1, BLAS2 που έχουν, όπως είδαμε, μικρή τοπικότητα.
- Σε συστήματα με ιεραρχική μνήμη, δοκιμάζουμε γενικεύσεις των παραπάνω από πολλαπλασιασμούς υπομητρώων.

Πολλαπλασιασμός μητρώων μέσω δισδιάστατης ορμαθοποίησης
Έστω ότι οι A, B, C έχουν τεμαχιστεί σε «ορμαθούς»:

$$C_{IJ} \in \mathbb{R}^{m_1 \times m_3}, A_{IK} \in \mathbb{R}^{m_1 \times m_3}, B_{KJ} \in \mathbb{R}^{m_1 \times m_2}$$

```
for I = 1 : k1
    for K = 1 : k3
        for J = 1 : k2
            C_IJ = C_IJ + A_IK B_KJ
        end
    end
end
```

Αναφορές:

- Βλ. στο βιβλίο παράδειγμα (από Golub, Van Loan) που συγκρίνει μονοδιάστατη και διδιάστατη ορμαθοποίηση.
- Βλ. τις αναφορές Ueberhuber και Daydé και Duff.

Πολλαπλασιασμός μητρώων μέσω δισδιάστατης ορμαθοποίησης
Έστω ότι οι A, B, C έχουν τεμαχιστεί σε «ορμαθούς»:

$$C_{IJ} \in \mathbb{R}^{m_1 \times m_3}, A_{IK} \in \mathbb{R}^{m_1 \times m_3}, B_{KJ} \in \mathbb{R}^{m_1 \times m_2}$$

for $I = 1 : k_1$
  for $K = 1 : k_3$
    for $J = 1 : k_2$
      $C_{IJ} = C_{IJ} + A_{IK}B_{KJ}$
    end
  end
end

Αναφορές:

- Βλ. στο βιβλίο παράδειγμα (από Golub, Van Loan) που συγκρίνει μονοδιάστατη και διδιάστατη ορμαθοποίηση.
- Βλ. τις αναφορές Ueberhuber και Daydé και Duff.

Πολλαπλασιασμός μητρώων μέσω δισδιάστατης ορμαθοποίησης

Πότε/πού κάνουμε LOAD;

... με φειδώ!

for $I = 1 : k_1$
  for $K = 1 : k_3$
    LOAD A_{IK}
    for $J = 1 : k_2$
      LOAD C_{IJ}, B_{KJ}
      $C_{IJ} = C_{IJ} + A_{IK}B_{KJ}$
      STORE C_{IJ}
    end
  end
end

Ορμαθοποίηση Αν επιλέξουμε τις διαστάσεις ώστε

$$\mathcal{K} \approx O(m_1m_2 + m_1m_3 + m_2m_3)$$

η εσωτερική πράξη υλοποιείται με βέλτιστο αριθμό μεταφορών

$$\Phi(I, J, K) = 2m_1m_2 + m_1m_3 + m_2m_3.$$

Συνολικά

$$\begin{aligned} \Phi &= \sum_{I=1}^{k_1}\sum_{K=1}^{k_3}(m_1m_3 + \sum_{J=1}^{k_2}(2m_1m_2 + m_2m_3)) \\ &= n_1n_3 + n_1n_2n_3(\frac{1}{m_1} + \frac{2}{m_3}). \end{aligned}$$

- Η επόμενη υλοποίηση επιλέγει το «χαρακτηριστικό» μέγεθος ορμαθού (blocksize) NB ώστε
$$3\mathrm{NB}^2 \approx \mathcal{K}$$
- Αυτό επιτυγχάνεται με ειδικό αρχείο «προμετωπίδα» ...
- ... που περιέχει πίνακα με τιμές που θεωρούνται «καλές» για τα σημαντικότερα υπολογιστικά συστήματα.

Αναφορά Διαθέσιμη και ηλεκτρονικά μέσω της ιστοσελίδας της κετρικής βιβλιοθήκης:

DaydeDuff99 M.J. Daydé and Iain S. Duff, "The RISC BLAS: A Blocked Implementation of Level 3 BLAS for RISC Processors", *ACM Transactions on Mathematical Software (TOMS)*, 25(3):316:349, 1999.

Παράδειγμα υλοποίησης (Dayde, Duff)

```
      SUBROUTINE DGEMM (TRANSA,TRANSB,M, N, K, ALPHA, A, LDA, B, LDB,
     $                   BETA, C, LDC )
*     .. Array Arguments ..
      DOUBLE PRECISION A( LDA, * ), B( LDB, * ), C( LDC, * )
*
*  DGEMM  performs one of the matrix-matrix operations
*
*     C := alpha*op( A )*op( B ) + beta*C,
*  where  op( X ) is one of
*     op( X ) = X   or   op( X ) = Xt,
*
*  alpha and beta are scalars, and A, B and C are matrices, with op( A )
*  an m by k matrix,  op( B )  a  k by n matrix and  C an m by n matrix.
.....
*           Form  C := alpha*A*B + beta*C.
.....
            DO 70, L = 1, K, NB
               LB = MIN( K - L + 1, NB )

               DO 60, I = 1, M, NB
                  IB = MIN( M - I + 1, NB )

                  DO 62 II = I, I + IB - 1
                     DO 61 LL = L, L + LB - 1
                        AA(LL-L+1,II-I+1)=ALPHA*A(II,LL)
61                   CONTINUE 62               CONTINUE

                  DO 50, J = 1, N, NB
```

```
                    JB = MIN( N - J + 1, NB )
*
*  Performs matrix-matrix multiplications
*
         IF ( ( MOD(IB,2).EQ.0 ) .AND. ( MOD(JB,2).EQ.0 ) ) THEN
            CALL DGEMML2X2_NN( IB, JB, LB, AA, NB,
     $                         B( L, J ), LDB, C( I, J ), LDC )
         ELSE
            CALL DGEMML_NN( IB, JB, LB, AA, NB,
     $                      B( L, J ), LDB, C( I, J ), LDC )
         END IF
*
 50              CONTINUE
 60           CONTINUE
 70         CONTINUE
 .....
```

Παρατηρήσεις

- Στην πραγματικότητα δεν χρειάζεται τόσο μεγάλο $\mathcal{K}$
- Η εσωτερική πράξη είναι μια σειρά απο MV για την βέλτιστη υλοποίηση των οποίων θέλουμε μόνο χώρο για μια στήλη του B_{JK}, δηλ. m_3.
- Επίσης χρειαζόμαστε χώρο για το A_{IK}.
- Δηλ. επιλέγουμε $m_3 m_1 + m_3 \leq \mathcal{K}$
- Η πράξη $C \leftarrow C + AB$ με $r = n_3 \ll n_1, n_2$ είναι *ανανέωση τάξης-r*.

Τα BLAS

> Basic Linear Algebra Subprograms (BLAS) is a de facto Application Programming Interface standard for publishing libraries to perform basic linear algebra operations such as vector and matrix multiplication. [1]

- Πρώτη δημοσίευση το 1979, χρησιμοποιήθηκαν για τη σύνθεση των βιβλιοθηκών LINPACK, EISPACK. Σήμερα χρησιμοποιούνται στην LAPACK.
- Χρησιμοποιούνται εκτενέστατα στον ΕΥ και σε υπολογισμούς υψηλών επιδόσεων.
- Ειδικές βιβλιοθήκες που υλοποιούν τα BLAS έχουν αναπτυχθεί σε ερευνητικά και βιομηχανικά εργαστήρια.

Μερικές υλοποιήσεις

[1] http://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms

refblas Υλοποίηση αναφοράς της Netlib σε εκδοχές C, Fortran.

ATLAS Αυτορυθμιζόμενο Λογισμικό Γραμμικής Άλγεβρας, ανοικτός κώδικας από BLAS API σε C, Fortran.

CUDA SDK Παρέχει λειτουργικότητα BLAS σε C σε κάρτες γραφικών GeForce 8 Series.

Intel MKL Λειτουργικότητα BLAS για συστήματα Pentium, Itanium.

GSL BLAS στην GNU Scientific Library.

Goto BLAS Υλοποιήσεις BLAS του Kazushige Goto.

ESSL Υλοποιήσεις BLAS στην IBM's Engineering and Scientific Subroutine Library για το PowerPC.

Κατηγοριοποίηση

BLAS-1**:** Όταν μόνο μια από τις διαστάσεις n_j είναι μεγαλύτερη της μονάδας, δηλ. πράξεις του τύπου διάνυσμα/διάνυσμα. Π.χ. το εσωτερικό γινόμενο DOT ($n_3 > 1$), και οι πράξεις sAXPY ($n_2 > 1$ ή $n_3 > 1$.) Ο αριθμός πράξεων και μεταφορών τότε είναι $O(n)$ και επομένως $\mu_{\min} = \mathrm{O}(1)$. άρα έχουμε μικρή τοπικότητα. Χρησιμοποιήθηκε για την κατασκευή των βιβλιοθηκών LINPACK, EISPACK (≈ 1975).

BLAS-2**:** Όταν ακριβώς δυο από τις διαστάσεις είναι μεγαλύτερες της μονάδας, δηλ. πράξεις του τύπου μητρείο/διάνυσμα. Λαβαίνοντας τις δυο αυτές διαστάσεις ίσες με n έχουμε πως ο αριθμός των πράξεων και των μεταφορών είναι $\mathrm{O}(n^2)$, άρα έχουμε καλύτερη τοπικότητα χώρου και χρόνου από την BLAS-1. Προτάθηκε για αποτελεσματική επεξεργασία σε διανυσματικούς ΗΥ (μέσα του 1980).

BLAS-3**:** Όταν $n_j > 1$ για $j = 1, 2, 3$, δηλ. πράξεις του τύπου μητρείο/μητρείο, οπότε έχουμε και αυξημένη τοπικότητα π.χ. αν $n_1 = n_2 = n_3 = n$ τότε έχουμε $\mathrm{O}(n^2)$ μεταφορές για $\mathrm{O}(n^3)$ πράξεις. Προτάθηκε στα τέλη του 1980 για την καλύτερη χρήση παράλληλων ΗΥ και ΗΥ με ιεραρχική μνήμη. Χρησιμοποιήθηκε για την κατασκευή της LAPACK ($\approx 1987-$).

Έστω $C \leftarrow C + AB$ για ένα μόνο μέγεθος n:

Πράξη	(n_1, n_2, n_3)	$\Phi_{\min}$	$\mu_{\min}$
DOT	$(1, 1, n)$	$2n + 2$	$1 + \frac{1}{n}$
sAXPY	$(n, 1, 1)$	$3n + 1$	$\frac{3}{2} + \frac{1}{2n}$
	$(1, n, 1)$		
Rank-1	$(n, n, 1)$	$2n^2 + 2n$	$1 + \frac{1}{n}$
MV	$(n, 1, n)$	$n^2 + 3n$	$\frac{1}{2} + \frac{3}{2n}$
	$(1, n, n)$		
MM	(n, n, n)	$4n^2$	$\frac{2}{n}$

Παρατηρήσεις για τα BLAS

- Τα BLAS παρέχουν (Application Programming Interfaces (API)) με συγκεκριμενες *προδιαγραφές* και όχι υλοποιήσεις ⇒ μεταφερσιμότητα
- *Υλοποίηση* επαφίεται στους σχεδιαστές λογισμικού/υλικού ⇒ υψηλή επίδοση
- Οι ονομασίες είναι τυποποιημένες και αναδεικνύουν το είδος της πράξης.

Τι γίνεται σήμερα (2000-); Υψηλότερη επίδοση και επεκτάσεις:

- ATLAS για αυτόματη σύνθεση BLAS.
- BLAS πολύ υψηλής επίδοσης μέσω ιδεών του K. Goto που στοχεύουν στη μείωση του πλήθους των TLB (Translation Look-aside Buffer) misses.
- Επεκτάσεις:
  - { διεπαφή με ποικιλία γλωσσών (π.χ. Java).
  - { πράξεις για αραιές δομές (αραιά μητρώα)
  - { πράξεις προσανατολισμένες σε παράλληλα συστήματα με κατανεμημένη μνήμη
  - { επέκταση σε περισσότερες πράξεις που χαρακτηρίζονται υπολογιστικοί πυρήνες
  - { αριθμητική εκτεταμένης ακρίβειας, αριθμητική διαστημάτων

«Επιχείρηση» ATLAS. (Dongarra et. al. 1998-)

Automatically Tuned Linear Algebra Software

SANS: Self-adaptive numerical software.

- Λογισμικό για την αυτόματη παραγωγή και βελτιστοποίηση αριθμητικού λογισμικού για επεξεργαστές με ιεραρχική μνήμη και αριθμητικές μονάδες με pipelining.
- Ειδικότερη έμφαση στις BLAS πιο σύνθετη πράξη, `DGEMM`
- Kasparov εναντίον του Deep Blue

Υπάρχει τρόπος να πετύχουμε ταχύτερο MM;

Όσοι τρόποι συζητήσαμε: $2n^3 - n^2$ α.κ.υ. $\rightarrow \Theta(n^3)$

Γίνεται τίποτα καλύτερο;

$\Downarrow$

Θεωρία αλγορίθμων $\rightarrow$ «Υπερταχείς» μέθοδοι

$\Downarrow$

$O(n^\beta), \quad \beta = 2.***?$

–

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$
$$= \begin{pmatrix} B_{11}C_{11} + B_{12}C_{21} & B_{11}C_{12} + B_{12}C_{22} \\ B_{21}C_{11} + B_{22}C_{21} & B_{21}C_{12} + B_{22}C_{22} \end{pmatrix}$$

$$T(n) = 8T(\frac{n}{2}) + 4(\frac{n}{2})^2$$
$$T(n) = \Theta(n^3)$$

- 8 «×» και 4 «+», κανένα *γινόμενο* δεν επαναλαμβάνεται

Ιδέα Πολλαπλασιασμός μιγαδικών:

$$(a + ib)(c + id) = (ac - bd) + i(bc + ad)$$

φαίνεται να στοιχίζει 2(+) και 4(*) όμως αν γράψουμε

$$\begin{aligned} p &= (a + b)c = ac + bc \\ q &= (c - d)a = ac - da \\ r &= (a - b)d = ad - bd \end{aligned}$$

επομένως

$$(a + ib)(c + id) = (q + r) + i(p - q)$$

στοιχίζει 5(+) και 3(*)
Χρήσεις Αν τα a, b, c, d είναι μητρώα:

- το κόστος πολλαπλασιασμού πραγματικών μητρώων μεγέθους n είναι $2n^3 - n^2$, το κόστος πρόσθεσης είναι n^2 επομένως: το κόστος πολλαπλασιασμού μιγαδικών μητρώων σε πράξεις πραγματικών είναι:

  $$8n^3 - 2n^2$$

  ενώ με το τρυκ

  $$6n^3 + 2n^2$$

  οπότε, π.χ. ακόμα και για ένα «μικροσκοπικό» μητρώο $n = 10$, 7800 ενώ με το τρυκ 6200.

- Ιδιαίτερα χρήσιμο σε περιπτώσεις που πρέπει να υπολογίσουμε επανειλημμένα $XY(1), XY(2), ...$ όπου $X, Y(k) \in \mathbb{C}^{n\times n}$ και $X = A + iB$, $Y(k) = C(k) + iD(k)$, το $Y(k)$ είναι συνάρτηση του k γιατί τότε μπορούμε να επαναχρησιμοποιήσουμε τα $A + B, A - B$.

Τι κάνουμε; Καθοριστική η συμβολή του Volker Strassen @Berkeley (1969) - Με την εργασία *Gaussian Elimination is not Optimal*, Numerische Mathematik
Αναδρομικά υπολογίζονται τα υπομητρώα

$$\begin{aligned}
P_1 &= (B_{12} - B_{22})(C_{21} + C_{22}) \\
P_2 &= (B_{11} + B_{22})(C_{11} + C_{22}) \\
P_3 &= (B_{11} - B_{21})(C_{11} + C_{12}) \\
P_4 &= (B_{11} + B_{12})C_{22} \\
P_5 &= B_{11}(C_{12} - C_{22}) \\
P_6 &= B_{22}(C_{21} - C_{11}) \\
P_7 &= (B_{21} + B_{22})C_{11}
\end{aligned}$$

τότε

$$\begin{aligned}
A_{11} &= P_1 + P_2 - P_4 + P_6 \\
A_{12} &= P_4 + P_5 \\
A_{21} &= P_6 + P_7 \\
A_{22} &= P_2 - P_3 + P_5 - P_7
\end{aligned}$$

δηλ. 7 '$\times$' μητρώων και 18 '$+$' μητρώων μεγέθους $n/2$.
Υπολογισμός πολυπλοκότητας Επομένως το κόστος σε πράξεις α.κ.υ. ικανοποιεί την παρακάτω αναδρομή:

$$T(n) = 7T(\frac{n}{2}) + 18(\frac{n}{2})^2$$

Η λύση της είναι:

$$T(n) = \Theta(n^{\log_2 7}) = \Theta(n^{2.81}).$$

Θεώρημα 1 (Strassen**).** *Ο πολλαπλασιασμός δύο τετραγωνικών πινάκων μεγέθους $n = 2^k$ μπορεί να γίνει με $\Theta(n^{2.81})$ αριθμητικές πράξεις.* □

Σχετικά με την επίλυση της αναδρομής Από τη θεωρία: Μάθημα Αλγορίθμων ή Διακριτών Μαθηματικών
Με Matlab/Maple: Χρησιμοποιώντας το Symbolic Toolbox και ειδικότερα:

- Τη συνάρτηση `rsolve` για την επίλυση αναδρομών
- Θέτουμε ως `Strs (`n`)` το κόστος πολλαπλασιασμού με τη μέθοδο για μητρώα μεγέθους n
- ... και ως αρχική συνθήκη `Strs (`2`) = 12` βάση του ότι τότε ο πολλαπλασιασμός γίνεται με τον κλασικό τρόπο (κόστος 12 πράξεις).

```
> maple('rsolve','{Stras(n)=7*Stras(n/2)+18*(n/2)^2,Stras(2)=12}','Stras(
> 12/7*n^(log(7)/log(2))+n^(log(7)/log(2))*(-21/2*(4/7)^(log(n)/log(2)+1)
```

δηλαδή:

$$T(n) = \frac{36}{7}n^{\log_2 7} - n^{\log_2 7}\left(\frac{21}{2}(\frac{4}{7})^{\log_2 n+1}\right)$$

Παρατηρήσεις Παρατηρήστε ότι τα mflop/s της `strassen` είναι πολύ χαμηλότερα. Δεν πρέπει να μας επλήσσει αφού αντικαταστήσαμε πολλαπλασιασμούς μητρώων (που εκτελούνται αποτελεσματικά με BLAS-3) με προσθέσεις μητρώων. Ο Higham χρησιμοποιεί το 8 ως μέγεθος μητρώων κάτω από το οποίο η πράξη πολλαπλασιασμού γίνεται με κλασικό αλγόριθμο αντί με Strassen αναφέροντας ότι αυτό είναι το βέλτιστο (θεωρητικό) μέγεθος. Μπορείτε να ερευνήσετε αυτόν τον ισχυρισμό·
Υλοποίηση Χρησιμοποιούμε τον πολλαπλασιασμό Strassen

- όταν τα μητρώα είναι πολύ μεγάλο ...
- ... για μερικά βήματα ...
- ... μέχρι η διάσταση των υπομητρώων να γίνει αρκετά μικρή
- ώστε να συμφέρουν αλγόριθμοι BLAS-3.

Μπορούμε να υπολογίσουμε το θεωρητικό μέγεθος n_0 για το οποίο ελαχιστοποιείται το πλήθος των πράξεων α.κ.υ. μέσω Strassen και κλασικού αλγορίθμου.
Πως εφαρμόζεται ο αλγόριθμος αν $2^{k-1} < n < 2^k$;

- Απευθείας εμφύτευση σε μητρώα διάστασης 2^k, όπου $k := \lceil n \rceil$:

$$A \to \tilde{A} := [A, 0_{2^k-n,n}; 0_{n,2^k-n}, 0_{2^k-n,2^k-n}]$$

- (είτε) διαδοχική εμφύτευση σε μητρώα με ζυγή διάσταση.

Είναι ο αλγόριθμος πρακτικός;

- Δύσκολο να μεταφρασθεί η χαμηλότερη πολυπλοκότητα σε πραγματικά ταχύτερο αλγόριθμο
- Το έμπροσθεν σφάλμα μπορεί να είναι μεγαλύτερο από τον κοινό πολλαπλασιασμό (βλ. σημειώσεις)
- Ανοικτό πρόβλημα πόσο κοντά στο n^2 μπορούμε να φθάσουμε. Οι Coppersmith και Winograd έχουν φθάσει το $O(n^{2.376...})$.
- Οι περαιτέρω μειώσεις των πράξεων (μετά από τους Strassen, Winograd) οδηγούν σε πολύ μεγάλη σταθερά και μη πρακτικούς αλγορίθμους.

Παράδειγμα υλοποίησης sAXPY

```
      subroutine daxpy(n,da,dx,incx,dy,incy)
c     constant times a vector plus a vector.
c     uses unrolled
loops for increments equal to one.
c     jack dongarra, linpack,
3/11/78.
c     modified 12/3/93, array(1) declarations changed to array(*)
      double precision dx(*),dy(*),da
      integer i,incx,incy,ix,iy,m,mp1,n
c
      if(n.le.0)return
      if (da .eq. 0.0d0) return



      if(incx.eq.1.and.incy.eq.1)go to 20
c        code for unequal increments or equal increments not equal
to 1
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        dy(iy) = dy(iy) + da*dx(ix)
        ix = ix + incx
        iy = iy + incy
10    continue
      return
c        code for both increments equal to 1 c        clean-up
loop 20    m = mod(n,4)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dy(i) = dy(i) + da*dx(i)
30    continue
      if( n .lt. 4 ) return
40    mp1 = m + 1
      do 50 i = mp1,n,4
        dy(i) = dy(i) + da*dx(i)
        dy(i + 1) = dy(i + 1) + da*dx(i + 1)
        dy(i + 2) = dy(i + 2) + da*dx(i + 2)
        dy(i + 3) = dy(i + 3) + da*dx(i + 3)
```

```
50    continue
      return
      end
```

Τεχνικές

- Έλεγχος δεδομένων για τετριμμένες περιπτώσεις ($\alpha = 0$)
- Ξεδίπλωμα βρόχου

Ξεδίπλωμα Dongarra et al, 1976

Αλγόριθμος. [Αλγ 1] (έστω $0 \equiv n \bmod 2$)

```
    DO i = 1, n
       x(i) = a*x(i) + y(i)
    ENDDO
```

Αλγόριθμος. [Αλγ 2]

```
    DO i = 1, n, 2
       x(i) = a*x(i) + y(i)
       x(i+1) = a*x(i+1) + y(i+1)
    ENDDO
```

Αλγόριθμος. [Αλγ 1σ] (έστω $0 \equiv n \bmod 4$)

```
 sum = 0.
 DO i = 1, n
    sum = sum + x(i)
 ENDDO
```

Αλγόριθμος. [Αλγ 2σ]

```
 sum = 0.
 DO i = 1, n, 4
    sum = sum + x(i) + x(i+1) + x(i+2) + x(i+3)
 ENDDO
```

Χαρακτηριστικά Πλεονεκτήματα

- μείωση επαναλήψεων (ταξιδιών) και σχετικού overhead
- αύξηση τοπικότητας: π.χ. αν ο μεταφραστής σε κάθε επανάληψη LOAD/STORE στοιχεία όπως το `a` και το `sum` στους προηγούμενους αλγ.
- αύξηση παραλληλισμού

Προβλήματα Δεν είναι χρήσιμος όταν

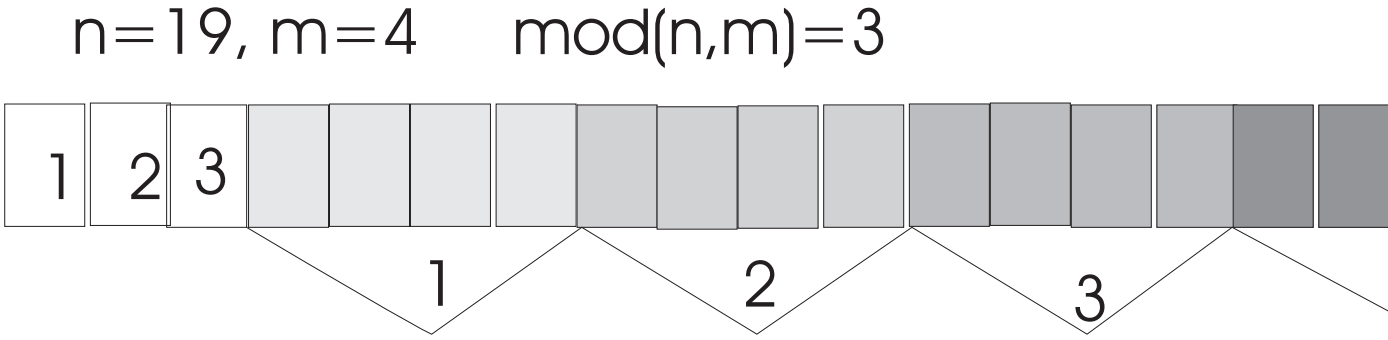
- μείωση τοπικότητας αν ο κορμός γίνει πολύ μεγάλος

- αν ο αριθμός επαναλήψεων είναι μικρός
- Κίνδυνοι σφάλματος στον μετασχηματισμό

Αρχικοποίηση π.χ. για ξεδίπλωμα του αλγ. 1ς με βάθος `nb = 4`:

Αλγόριθμος. [Αλγ 3σ]

```
nrem = MOD(n,nb)
DO i = 1, nrem
   sum = sum + x(i)
ENDDO
n1 = nrem + 1
DO i = n1, n, nb
   sum = sum + x(i) + x(i+1) + x(i+2) + x(i+3)
ENDDO
```



Από το SAXPY

```
20    m = mod(n,4)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dy(i) = dy(i) + da*dx(i)
30    continue
      if( n .lt. 4 ) return
40    mp1 = m + 1
      do 50 i = mp1,n,4
        dy(i) = dy(i) + da*dx(i)
        dy(i + 1) = dy(i + 1) + da*dx(i + 1)
        dy(i + 2) = dy(i + 2) + da*dx(i + 2)
        dy(i + 3) = dy(i + 3) + da*dx(i + 3)
50    continue
```

Σύντηξη βρόχων (**Loop fusion**)

Αλγόριθμος. [Αλγ 1]

```
DO i = 1, n
   x(i) = a*x(i) + y(i)
ENDDO
DO i = 1, n
   z(i) = y(i)/2
ENDDO
```

Αλγόριθμος. [Αλγ 2]

```
DO i = 1, n
   x(i) = a*x(i) + y(i)
   z(i) = y(i)/2
ENDDO
```

Χαρακτηριστικά Αν η σύντηξη είναι έγκυρη:

- Αν οι βρόχοι χρησιμοποιούν τα ίδια στοιχεία μειώνεται ο αριθμός μεταφορών (αύξηση τοπικότητας)
- Η τοπικότητα μειώνεται αν ο κορμός μεγαλώσει πολύ είτε χρησιμοποιούνται πολλά διαφορετικά στοιχεία

  Απαιτείται προσοχή ώστε αποφευχθούν σφάλματα οφειλόμενα στις εξαρτήσεις μεταξύ δεδομένων.

Οι παρακάτω αλγόριθμοι δίδουν διαφορετικές απαντήσεις:

Αλγόριθμος. [Αλγ 1]

```
DO i = 1, n
   z(i) = a*x(i) + y(i)
ENDDO
DO i = 1, n
   x(i+1) = y(i)/2
ENDDO
```

Αλγόριθμος. [Αλγ 2]

```
DO i = 1, n
   z(i) = a*x(i) + y(i)
   x(i+1) = y(i)/2
ENDDO
```

Αλλά τι γίνεται με τα

Αλγόριθμος. [Αλγ 1]

```
DO i = 1, n
   z(i) = a*x(i) + y(i)
ENDDO
DO i = 1, n
   w(i) = z(F(i))
ENDDO
```

Αλγόριθμος. [Αλγ 2]

```
DO i = 1, n
   z(i) = a*x(i) + y(i)
   w(i) = z(F(i))
ENDDO
```

Όταν οι δείκτες είναι και αυτοί (ακέραιες) συναρτήσεις (π.χ. έμμεση διευθυνσιοδότηση) οι μετασχηματισμοί γίνονται δύσκολοι

Ενδιαφέρον Πως μετασχηματίζονται οι εντολές τύπου APL, MATLAB, Fortran 90, C++;

```
z(1:n) = a*x(1:n) + y(1:n)
x(1:n) = y(1:n)/2
```

Παράδειγμα

```
for k=1:n
    for j=1:n
        for i = 1:n
            c(i,k) = c(i,k) + a(i,j)*b(j,k)
        end
    end
end
```

Κλάσμα πράξεων/ μεταφορές στο βρόχο i = 2/3

Παράδειγμα

```
for k=1:2:n
    for j=1:2:n
        for i = 1:n
            c(i,k) = c(i,k) + a(i,j)*b(j,k)
        end
        for i = 1:n
            c(i,k) = c(i,k) + a(i,j+1)*b(j+1,k)
        end
    end
    for j=1:2:n
        for i = 1:n
            c(i,k+1) = c(i,k+1) + a(i,j)*b(j,k)
        end
        for i = 1:n
            c(i,k+1) = c(i,k+1) + a(i,j+1)*b(j+1,k)
        end
    end
end
```

```
for k=1:2:n
    for j=1:2:n
        {LOAD b(j,k), b(j+1,k), b(j+1,k+1)}
        for i = 1:n
            {LOAD c(i,k), c(i,k+1), a(i,j), a(i,j+1)}
            c(i,k) = c(i,k) + a(i,j)*b(j,k) + a(i,j+1)*b(j+1,k)
            c(i,k+1) = c(i,k) + a(i,j)*b(j,k) + a(i,j+1)*b(j+1,k+1)
            {STORE c(i,k), c(i,k+1) }
        end
    end
end
```

Κλάσμα πράξεων/ μεταφορές στο βρόχο i = 8/6

Unswitching

Αλγόριθμος. [Αλγ 1]

```
DO i = 1, n
   IF (iadd.EQ.1) THEN
      x(i) = a*x(i) + y(i)
   ELSE
      x(i) = a*x(i) + y(i)
   ENDIF
ENDDO
```

Αλγόριθμος. [Αλγ 2]

```
IF (iadd.EQ.1) THEN
   DO i = 1, n
      x(i) = a*x(i) + y(i)
   ENDDO
ELSE
   DO i = 1, n
      x(i) = a*x(i) - y(i)
   ENDDO
ENDIF
```

Προσεταιριστική ιδιότητα

$$(a+b)+c \stackrel{?}{=} a+(b+c)$$

Αλγόριθμος. [Αλγ 1]

```
sum = 0.
DO i = 1, n
   s = s + x(i)
ENDDO
```

Αλγόριθμος. [Αλγ 2]

```
sum1 = 0.
sum2 = 0.
   DO i = 1, n, 2
      sum1 = sum1 + x(i)
      sum2 = sum2 + x(i+1)
   ENDDO
sum = sum1 + sum2
```

Βελτιστοποιήσεις
Τι;

Κλασσικοί μετασχηματισμοί: Στο επίπεδο των εντολών

HPC μετασχηματισμοί: Στο επίπεδο των βρόχων και υψηλότερα.

Γιατί;

- Έρευνα Kuck et al (1970ς) → μεγαλύτερο μέρος του χρόνου σε πρόγραμμα ΕΥ αναλώνεται σε εκτέλεση βρόχων.
- Σπάνια βρίσκονται εντολές που περιέχουν πολύ μεγάλο αριθμό πράξεων.

Η τεχνολογία οφείλει την ανάπτυξη της στη μελέτη του λογισμικού των υπερυπολογιστών.
Πως;

- Με το χέρι (συστηματικοποίηση, κωδικοποίηση)
- Ημιαυτόματα
- Αυτόματα

Σκοπιμότητα:

- μείωση του loop overhead
- βελτίωση της τοπικότητας / είδους αναφορών
- αύξηση παραλληλίας

Μετασχηματισμοί (κυρίως στους βρόχους) και αναδόμηση κώδικα

Ξεδίπλωμα (loop unrolling)

Σύντηξη βρόχων loop fusion

unswitching

προσεταιριστικοί μετασχηματισμοί

εναλλαγή βρόχων

ορμαθοποίηση

προφόρτωση (software) prefetching

software pipelining

Προφόρτωση
Παρατήρηση: Στα σύγχρονα συστήματα επιτρέπεται η ταυτόχρονη εκτέλεση εντολών μεταφοράς και αριθμητικής.
Είναι εφικτό να ζητήσουμε την μεταφορά στοιχείων που θα είναι χρήσιμα σε *μελλοντική* αριθμητική επεξεργασία.

Μέσω του μεταφραστή: με προεπεξεργασία και εισαγωγή εντολών

Μέσω δικών μας εντολών: Καλούμε τα στοιχεία έγκαιρα

Αν τα στοιχεία χρειάζονται στον κύκλο T και χρειάζονται τ κύκλοι το πολύ για να φορτωθούν τα στοιχεία από τη μνήμη, η εντολή μεταφοράς στην κρυφή μνήμη δίδεται στον $T - \tau$.
«Προφόρτωση» συνιστάται και στη MATLAB .
Παράδειγμα: DEC Visual Fortran Επίπεδα βελτιστοποιήσεων του μεταφραστή:

$\geq$ `/optimize:1` Local (minimal) optimizations occur within the source program unit and include recognition of common subexpressions and the expansion of multiplication and division.

$\geq$ `/optimize:2` Global optimizations include such optimizations as data-flow analysis, code motion, strength reduction, split-lifetime analysis, and instruction scheduling.

$\geq$ `/optimize:3` Additional global optimizations improve speed at the cost of extra code size. These optimizations include *loop unrolling*, code replication to eliminate branches, and padding certain power-of-two array sizes for more efficient cache use.

$\geq$ `/optimize:4` Automatic inlining applies interprocedure analysis and inline expansion of small procedures, usually by using heuristics that limit extra code size.

$\geq$ `/optimize:5` Loop transformation and software pipelining include a group of loop transformation optimizations and, on Alpha systems, also include the software pipelining optimization. The loop transformation optimizations apply to array references within loops and can apply to multiple nested loops. Loop transformation optimizations can improve the performance of the memory system.

Επιδόσεις της `daxpy`

lib Υλοποίηση μέσω κλήσης στη ρουτίνα που περιέχεται στη βιβλιοθήκη BLAS για το σύστημά μας.

ref Υλοποίηση μέσω του «κώδικα αναφοράς» BLAS που παρουσιάστηκε πιο πάνω.

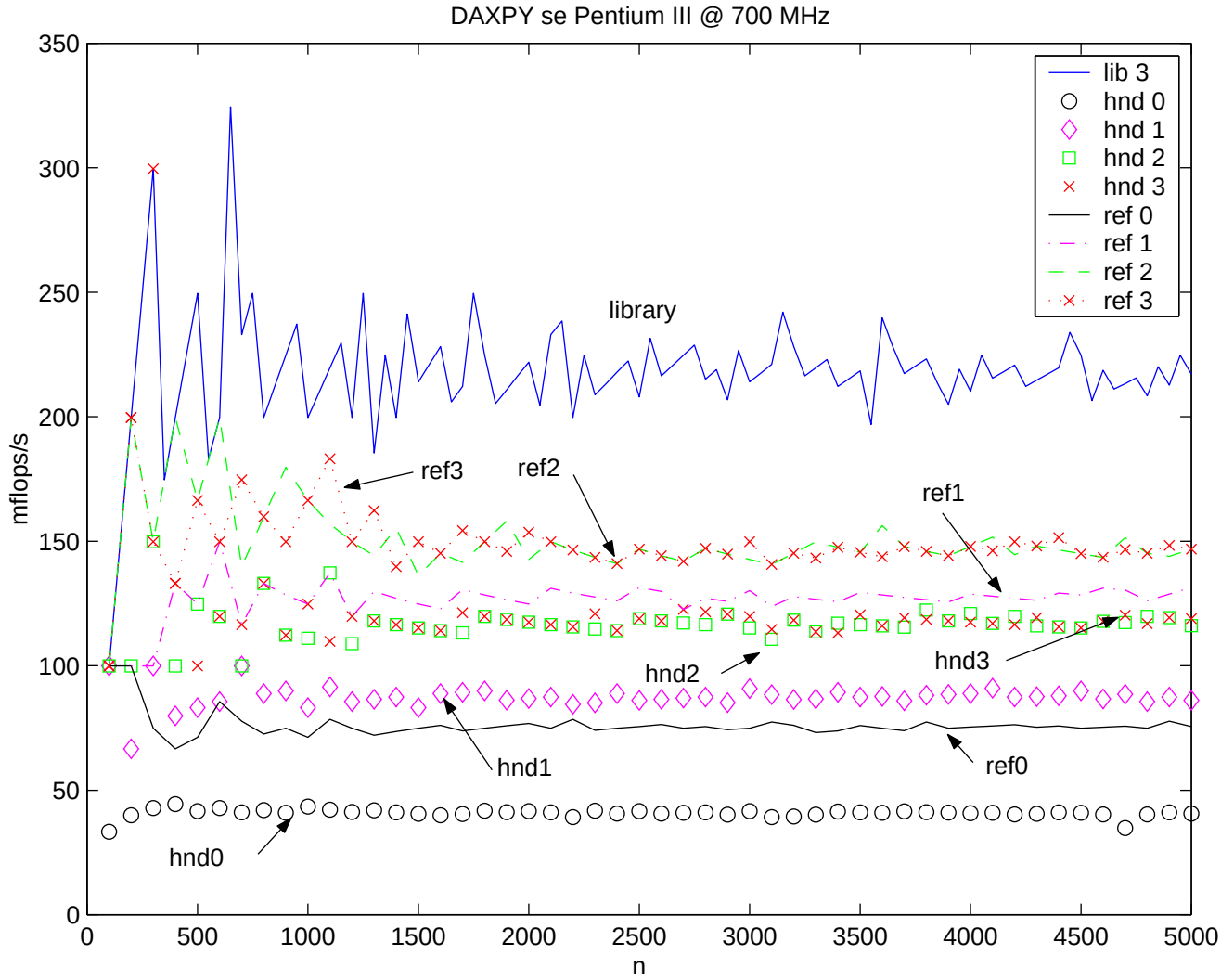
hnd Απλή υλοποίηση «με το χέρι».

Σύστημα φορητό IBM T20 με Intel Pentium III στα 700 MHz.

Λειτουργικό Windows 2000

Γλώσσα/μεταφραστής Fortran 77 με Digital Fortran της Compaq.

Το σύστημα περιλαμβάνει βιβλιοθήκη με τις υπορουτίνες BLAS. Οι εκδοχές ref και lib μεταφράστηκαν με τα 4 επίπεδα βελτιστοποίησης που περιγράφτηκαν παραπάνω.



Παρατηρήσεις

- Η καλύτερη επίδοση επιτυγχάνεται με τον κώδικα που περιέχεται στη βιβλιοθήκη του συστήματος.
- Για κάθε επίπεδο βελτιστοποίησης, οι επιδόσεις της υλοποίησης που αντιστοιχεί στον κώδικα αναφοράς είναι καλύτερες από τις επιδόσεις του απλού κώδικα.
- Η επίδοση του κώδικα που αντιστοιχεί στην απλή υλοποίηση χωρίς καμμία βελτιστοποίηση από τον μεταφραστή (hand0) είναι περί τις 5 φορές υποδεέστερη από τον κώδικα που αντιστοιχεί στον κώδικα βιβλιοθήκης (lib3).

Ενδιαφέρουσες παρατηρήσεις (από το 1975!) Από την εργασία των Beresford Parlett και Y. Wang με τίτλο *The Influence of the Compiler on the Cost of Mathematical Software - in Particular on the Cost of Triangular Factorization*:

Από τη Σύνοψη (Abstract) : ... it is almost impossible to compare to sensible implementations of a numerical algorithm in Fortran or in Algol and assert that one of them will lead to a more efficient program in machine language. Thus is so because the way the computer makes use of registers in the arithmetic unit has a strong influence on running time. Further, the writer of a subprogram to be used by other people is faced with the fact that his program will be used in a variety of compilers and computers.

Από τα Συμπεράσματα We are not implying that programs ought to be written in assembly language code in order to avoid inept use of registers. We are not castigating Fortran compilers for producing imperfect code. We are not blaming anyone nor complaining that this state of affairs is deplorable and should be rectified. The facts that we have assembled do indicate that nasty problems which beset those who try to disseminate high class numerical programs. Isn't this dissemination one of the aims of the numerical analysis community? Some proliferation of programs is inevitable because of the special forms (such as positive definite matrices, banded matrices) of which advantage can be taken. To check this proliferation our purveyor[2] feels obliged to choose *one* high level (Fortran) implementation for each case. What are his efforts worth if the user comes back to say that the new program seems to be fully accurate as his 10 year old program but 30 percent slower (or 50 percent slower, or 10 percent slower)? Of course there are two qualities more important than efficiency, namely, recognition of failure and adequate accuracy. Moreover there are important classes of computation were the scientific calculations are insignificant compared to the other parts. Nevertheless there are a great many installations which use Fortran libraries and where certain subroutines are called repeatedly by other subroutines. We conclude that the writeups of these subroutines should indicate the difference made by using an optimizing compiler or the standard one. Also helpful would be time estimates not based solely on counts of arithmetic operations. What turns out to be crucial is the way in which the Fortran program influences the compiler's exploitation of the operand registers.

[2] purveyor = Προμηθευτής.