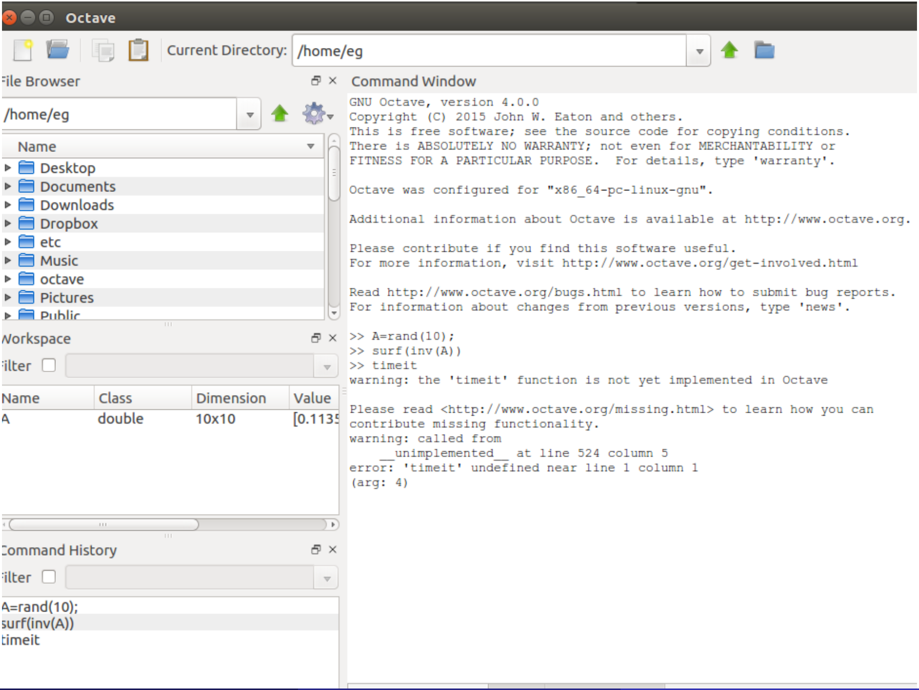
Επιστημονικός Υπολογισμός

Ε.Γαλλόπουλος

ΤΜΗΥΠ, Π. Πατρών

Διάλεξη 4: 25 Οκτωβρίου 2017

Περιεχόμενα


Octave
Current Directory: /home/eg
File Browser
/home/eg
Name
Desktop
Documents
Downloads
Dropbox
etc
Music
octave
Pictures
Public
Workspace
Filter
Name Class Dimension Value
A double 10x10 [0.1135
Command History
Filter
A=rand(10);
surf(inv(A))
timeit
Command Window
GNU Octave, version 4.0.0
Copyright (C) 2015 John W. Eaton and others.
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE. For details, type 'warranty'.
Octave was configured for "x86_64-pc-linux-gnu".
Additional information about Octave is available at http://www.octave.org.
Please contribute if you find this software useful.
For more information, visit http://www.octave.org/get-involved.html
Read http://www.octave.org/bugs.html to learn how to submit bug reports.
For information about changes from previous versions, type 'news'.
>> A=rand(10);
>> surf(inv(A))
>> timeit
warning: the 'timeit' function is not yet implemented in Octave
Please read <http://www.octave.org/missing.html> to learn how you can
contribute missing functionality.
warning: called from
__unimplemented__ at line 524 column 5
error: 'timeit' undefined near line 1 column 1
(arg: 4)

Συζήτηση

Υπόδειξη

Θα θεωρήσουμε το Ω σταθερό οπότε για την ελαχιστοποίηση επιλέγουμε την υλοποίηση που πετυχαίνει το μικρότερο Φ στους διαθέσιμους πόρους.

Δείκτης τοπικότητας

- Η τιμή μ_{min} αποτελεί ένδειξη της δυνητικής τοπικότητας στον αλγόριθμο και της δυνατότητας αποδοτικής υλοποίησης σε σύγχρονα συστήματα Η/Υ (ιεραρχικής μνήμης ή/και παράλληλα).
- Οι τιμές των μ_{min} και μ χρησιμοποιούνται επίσης για να συγκρίνουμε και να αξιολογήσουμε αλγορίθμους για διαφορετικά προβλήματα!

Τυπικά ευρήματα

$\mu_{\min}$	παράδειγμα	δυνάμει τοπικότητα
(1)	dot, sum, MV	ασήμαντη
$(1/\log n)$	FFT	μέτρια
$O(1/n)$,	MM	αξιόλογη (αν γίνει προσεκτική υλοποίηση για αξιοποίηση ιεραρχίας μνήμης)

Σχετικά με τη μείωση της καθυστέρησης των μεταφορών

επικάλυψη μεταφορών	προφόρτωση (prefetching)
βελτίωση της απόδοσης της μνήμης	διαφύλλωση (interleaving)
αξιοποίηση τοπικότητας	επαναχρησιμοποίηση από την cache
	μεταφορά ανά cache line (πλαίσιο)

Πίνακας: Μέθοδοι απόκρυψης κόστους μεταφορών

Στόχος

Προγράμματα που εκμεταλλεύονται την ιεραρχία και μεγιστοποιούν τις ευστοχίες (hits) στην cache

Πολλές τοπικότητες

Those who cannot remember the past are condemned to repeat it - George Santayana

χωρική τοπικότητα Αν τώρα ζητηθεί στοιχείο από τη θέση s, σύντομα θα ζητηθεί στοιχείο από παραπλήσια θέση.

χρονική τοπικότητα Αν τώρα ζητηθεί στοιχείο από τη θέση s, σύντομα θα ζητηθεί πάλι το ίδιο στοιχείο

αλγοριθμική τοπικότητα Όταν ένα πρόγραμμα αναφέρεται κατ΄ επανάληψη σε ορισμένα στοιχεία ή εκτελεί κατ΄ επανάληψη κάποιο συγκεκριμένο τμήμα κώδικα.

Παράδειγμα

Listing 1: pseudo-MATLAB Horner με LOAD/STORE

```
LOAD a(1:n+1), x
s = a(n+1);
for j=n:-1:1
    s = s*x + a(j);
end;
STORE s;
```

Αν η cache έχει χωρητικότητα $\mathcal{K} = O(n)$, τότε $\Omega = 2n$, $\Phi = n + 3$.

$$\text{Συνοπτικά:} [\mathcal{K}, \Omega, \Phi] = [O(n), 2n, n + 3]$$

Συγκρίσεις

Στόχοι

1. υλοποιήσεις που ελαχιστοποιούν το χρόνο εκτέλεσης, επομένως ελαχιστοποιούν το Φ
2. υλοποιήσεις που πετυχαίνουν καλή ισορροπία (trade-off) μεταξύ ταχύτητας και κόστους
3. για δεδομένους πόρους, υλοποιήσεις που ελαχιστοποιούν το χρόνο εκτέλεσης

- η υλοποίηση φαίνεται ότι χρειάζεται $= O(n)$
- μη πρακτικό όταν το K είναι συνάρτηση του n
- ... γιατί το μέγεθος του προβλήματος δεν είναι εκ των προτέρων γνωστό!

Παράδειγμα

Listing 2: pseudo-MATLAB Horner με JIT LOAD/STORE

```
LOAD a(n+1), x;
s = a(n+1);
for j=n:-1:1
    LOAD a(j);
    s = s*x + a(j);
end;
STORE s;
```

Αν η cache έχει χωρητικότητα $\mathcal{K} = O(1)$, τότε $\Omega = 2n$, $\Phi = n + 3$.

$$\text{Συνοπτικά:} [K, \Omega, \Phi] = [O(1), 2n, n + 3]$$

Βασικοί υπολογισμοί με μητρώα

Τα πρώτα προβλήματα που θα εξετάσουμε είναι όλα της μορφής

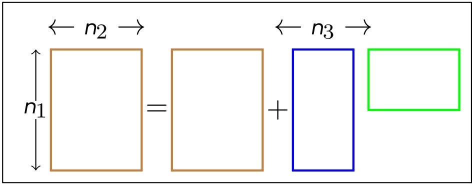
$$\boxed{C \leftarrow C + AB, \quad C \in \mathbb{R}^{n_1 \times n_2}, A \in \mathbb{R}^{n_1 \times n_3}, B \in \mathbb{R}^{n_3 \times n_2}.}$$

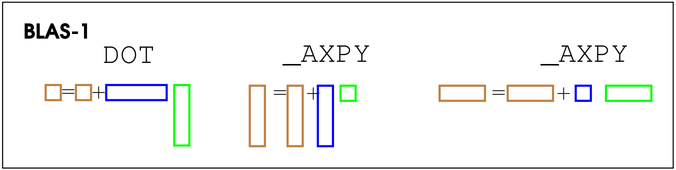
	$(n_1 > 1? \; n_2 > 1? \; n_3 > 1?)$	
DOT	$(0,0,1)$	εσωτερικό γινόμενο
DAXPY	$(1,0,0)$	διάνυσμα + βαθμωτός $\times$ διάνυσμα
	$(0,1,0)$	διάνυσμα + διάνυσμα $\times$ βαθμωτός
GER	$(1,1,0)$	ανανέωση 1ης τάξης
MV	$(1,0,1)$	διάνυσμα + μητρώο $\times$ διάνυσμα
	$(0,1,1)$	διάνυσμα + διάνυσμα $\times$ μητρώο
MM	$(1,1,1)$	μητρώο + μητρώο $\times$ μητρώο

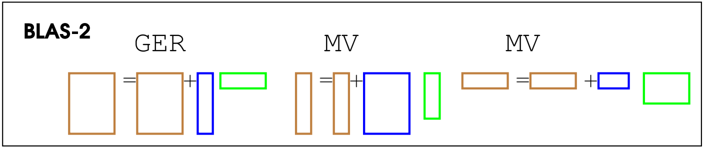
ΠΡΟΣΟΧΗ:

1. Προς το παρόν αναφερόμαστε σε **γενικά μητρώα** **χωρίς γνωστή δομή**
2. Στη MATLAB δίνεται πρόσβαση στις παραπάνω πράξεις μέσω τελεστών, π.χ. για βαθμωτό t και διανύσματα και μητρώα a, b, C, D,E εντολές όπως a+t*b, t+a'*b,C+a*b', a+C*b, E+C*D.

Template και ειδικές περιπτώσεις







BLAS: Basic Linear Algebra Subprograms

Από τη Wikipedia:

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. They were first published in 1979, and are used to build larger packages such as LAPACK. Heavily used in high-performance computing, highly optimized implementations of the BLAS interface have been developed by hardware vendors ...

Τρία επίπεδα: BLAS level 1, BLAS level 2, BLAS level 3, εν συντομία BLAS-1, BLAS-2, BLAS-3. Γενικά, αναφέρονται σε πράξεις μεταξύ αλγεβρικών αντικειμένων (μητρώων, διανυσμάτων) στα οποία μπορεί να υφίστανται 3 διαστάσεις συνολικά (οι βαθμωτοί δεν λαμβάνονται υπόψη)

- BLAS-1: μία διάστ. > 1, δηλ. πράξεις μεταξύ διανυσμάτων (π.χ. DOT, _AXPY)
- BLAS-2: δύο διαστ. > 1, δηλ. πράξεις μεταξύ μητρώων, διανυσμάτων (π.χ. MV, ανανέωση τάξης-1).
- BLAS-3: τρείς διαστ. > 1, δηλ. πράξεις μεταξύ μητρώων (π.χ. MM).

BLAS: Basic Linear Algebra Subprograms

- API για βασικές πράξεις της ΑΓΑ. Περιγράφονται η ονοματολογία, ο τρόπος κλήσης και οι πράξεις, όχι όμως η υλοποίηση.
- Συνοπτική παρουσίαση http://www.netlib.org/lapack/lug/node145.html
- Υλοποιήσεις:
  - Κώδικες αναφοράς Fortran http://netlib.org/blas/
  - ATLAS Automaticaly Tuned Linear Algebra Subroutines για C, Fortran
  - GotoBLAS
  - MKL BLAS
  - OpenBLAS
  - και πολλές άλλες ...

Δείκτης τοπικότητας

Listing 3: Pseudo-MATLAB BLAS daxpy with L/S

```
LOAD(a, x(1:n), y(1:n));
 for j=1:n
     y(j) = y(j) + a*x(j);
 end
STORE(y(1:n));
```

$$[K, \Omega, \Phi_{\text{min}}] = [2n + O(1), 2n, 3n + 1] \Rightarrow \mu_{\text{min}} = \frac{3}{2} + \frac{1}{2n}$$

Listing 4: Pseudo-MATLAB BLAS ddot with L/S

```
LOAD(s, x(1:n), y(1:n));
 for j=1:n
     s = s + x(j)*y(j);
 end
STORE(s);
```

$$[K, \Omega, \Phi_{\text{min}}] = [2n + O(1), 2n, 2n + 2] \Rightarrow \mu_{\text{min}} = 1 + \frac{1}{n}$$

Listing 5: Pseudo-MATLAB BLAS daxpy with JIT L/S

```
LOAD(a);
for j=1:n
    LOAD(x(j),y(j));
    y(j) = y(j) + a*x(j);
    STORE(y(j));
end
```

$$[\mathcal{K}, \Omega, \Phi] = [O(1), 2n, 3n+1] \Rightarrow \mu_{\min} = \frac{3}{2} + \frac{1}{2n}$$

Σημείωση: Χρησιμοποιούμε το αρκτικόλεξο JIT εννοώντας just in time, δηλ. η μεταφορά εκτελείται ακριβώς πριν χρειαστεί το δεδομένο.

ΠΡΟΣΟΧΗ: Η υλοποίηση επιτυγχάνει $\Phi = \Phi_{\min}$ μόνο με $\mathcal{K} = O(1)$ cache!

Τι μας ενδιαφέρει

Δίδονται ένα ή περισσότερα προγράμματα ή "ψευδο-κώδικες":

Listing 6: Pseudo-MATLAB

```
statement_First;
statement_Second;
...
statement_Last;
```

Τυπικά ζητούμενα από υλοποιήσεις εντός του μοντέλου

1. Οι τιμές Ω, Φ_{min}, μ_{min}
2. Δοθέντος *K*, στρατηγική ένθεση εντολών LOAD, STORE για μείωση των Φ, μ.
3. Νέα υλοποίηση που επιτυγχάνει (ακόμα) μικρότερα Φ, μ, όσο γίνεται πλησιέστερα στο Φ_{min}.
4. Συστηματική αξιολόγηση/σύγκριση υλοποιήσεων.
5. Θα θέλαμε οι βελτιώσεις να γίνονται αισθητές και στις υλοποιήσεις σε υπάρχοντα υπολογιστικά συστήματα.

Συγκρίσεις

- η δεύτερη υλοποίηση πετυχαίνει την ίδια πολυπλοκότητα πράξεων και μεταφορών με $\mathcal{K} = O(1)$ αντί $O(n)$
- μικρότερη απαίτηση σε πόρους (μικρότερο κόστος)

Θέματα

1. υπολογισμός Φ_{min} του αλγορίθμου (πώς?) ... ΕΥΚΟΛΟ
2. σχεδιασμός υλοποίησης που ελαχιστοποιεί το $\Phi \geq \Phi_{\text{min}}$ ... ΔΥΣΚΟΛΟΤΕΡΟ

Listing 7: Υλοποίηση αλγορίθμου με $\Phi = \Phi_{\text{min}}$

```
LOAD all_data_in_cache; %Phase_1
COMPUTE all;                    % Phase 2
STORE results_in_memory; % Phase 3
```

Η υλοποίηση προυποθέτει απεριόριστη cache αλλά δείχνει ότι το Φ_{min} είναι το πλήθος των (χρήσιμων) δεδομένων εισόδου που φορτώνονται στο LOAD και εξόδου που αποθηκεύονται στο STORE και είναι εύκολο να την υπολογίσετε!

Κοινά χαρακτηριστικά DOT, _AXPY

- $n_i, n_j = 1, n_k > 1$ για $i, j, k \in \{1, 2, 3\}$
- $\Omega = O(n), \Phi_{\text{min}} = O(n), \mu_{\text{min}} = O(1)$
- Πολυπλοκότητες γραμμικές στην κυρίαρχη διάσταση
- Μη αποδοτική υλοποίηση σε ιεραρχική μνήμη
- Level-1 Basic Linear Algebra Subprograms (BLAS-1)

Βασικές πράξεις γραμμικής άλγεβρας 1ου επιπέδου

Listing 8: Pseudo-MATLAB BLAS-2 with L/S

```
%        rank-1  update
% C <- C + a*b'
LOAD(C,a,b)
 for j = 1:n2
     for i=1:n1
          C(i,j) = C(i,j) + a(i)*b(j);
     end
 end
STORE(C)
```

$$[\mathcal{K},\Omega,\Phi] = [O(n_1 n_2), 2n_1 n_2, 2n_1 n_2 + n_1 + n_2] \Rightarrow \mu_{\min} = 1 + \frac{1}{2n_1} + \frac{1}{2n_2}.$$

Ανανέωση 1ης τάξης (rank-1 update)

Listing 9: Pseudo-MATLAB BLAS-2 with L/S

```
%        rank-1 update
% C <- C + a*b'
for j = 1:n2
    LOAD(b(j))
    for i=1:n1
        LOAD(C(i,j), a(i))
        C(i,j) = C(i,j) + a(i)*b(j)
        STORE(C(i,j))
    end
end
```

$$[\mathcal{K}, \Omega, \Phi] = [O(1), 2n_1n_2, 3n_1n_2 + n_2] \Rightarrow \mu = \frac{3}{2} + \frac{1}{2n_1}$$

Ανανέωση 1ης τάξης (rank-1 update)

Listing 10: Pseudo-MATLAB BLAS-2 with L/S

```
%        rank-1 update
% C <- C + a*b'
LOAD(a(1:n1))
 for j = 1:n2
     LOAD(b(j))
      for i=1:n1
          LOAD(C(i,j))
          C(i,j) = C(i,j) + a(i)*b(j)
          STORE(C(i,j))
      end
 end
```

$$[\mathcal{K},\Omega,\Phi]=[O(n_1),2n_1n_2,\underbrace{2n_1n_2+n_1+n_2}_{\Phi_{\min}}]\Rightarrow\mu=1+\frac{1}{2n_1}+\frac{1}{2n_2}=\mu_{\min}$$

Παρατηρήσεις

ΥΠΕΡ

- Πετύχαμε υλοποίηση με $\mu_{\min}$ και $= O(n_1) < O(n_1 n_2)$
- Θα μπορούσαμε το ίδιο με $= O(n_2) < O(n_1 n_2)$
- Επομένως μπορούμε να πετύχουμε $\mu_{\min}$ με $= O(\min(n_1, n_2))$.

ΚΑΤΑ

Χρειάζεται κρυφή μνήμη μεγέθους $O(n_1)$ ή $O(n_2)$.

<u>ΠΡΟΚΛΗΣΗ</u>

Να ξεπεράσουμε την (ανεπιθύμητη) εξάρτηση του μεγέθους της cache από τις διαστάσεις του προβλήματος?

Τεμαχισμός (σε πλοκάδες)

$$\begin{pmatrix} C_{11} & \cdots & C_{1,k_2} \\ C_{21} & \ddots & \vdots \\ \vdots & C_{I,J} & \vdots \\ \vdots & \ddots & \vdots \\ C_{k_1,1} & \cdots & C_{k_1,k_2} \end{pmatrix} = \begin{pmatrix} C_{11} & \cdots & C_{1,k_2} \\ C_{21} & \ddots & \vdots \\ \vdots & C_{I,J} & \vdots \\ \vdots & \ddots & \vdots \\ C_{k_1,1} & \cdots & C_{k_1,k_2} \end{pmatrix} + \begin{pmatrix} a_1 \\ \vdots \\ a_I \\ \vdots \\ a_{k_1} \end{pmatrix} \begin{pmatrix} b_1 & \cdots & b_{k_2} \end{pmatrix}$$

Αν υποθέσουμε ότι για $j = 1, 2, 3$:

$$n_j = \underbrace{(\text{πλοκάδες στη διάσταση } j)}_{k_j} \underbrace{(\text{μέγεθος πλοκάδας στη διάσταση } j)}_{m_j}$$

Με μαθηματική γραφή (το b_J είναι γραμμή), για $I = 1, \dots, k_1; J = 1, \dots, k_2$:

$$\boxed{C_{IJ} = C_{IJ} + a_I b_J, \quad C_{IJ} \in \mathbb{R}^{m_1 \times m_2}, a_I \in \mathbb{R}^{m_1}, b_j \in \mathbb{R}^{m_2}}$$

Στην πράξη

Με τις παραπάνω διαστάσεις, έχουμε τις ακολουθες αντιστοιχίες μεταξύ αλγεβρικών συμβόλων για υπομητρώα και στοιχεία των πινάκων που αποθηκεύονται τα C, a, b:

Υπομητρώο/πλοκάδα	αντιστοιχεί στα στοιχεία	μέγεθος
C_{IJ},	C((I-1)*m1+1:I*m1,(J-1)*m2+1: J*m2)	$m_1 \times m_2$
a_I	a((I-1)*m1+1:I*m1)	m_1
b_J	b((J-1)*m2+1:J*m2)	m_2

Παρατηρήσεις:

1. Ο τεμαχισμός μπορεί να είναι ανισομερής, αρκεί να είναι συμβατές οι διαστάσεις
2. Χρησιμοποιούμε μαθηματικά σύμβολα και υποδείκτες για τα μαθηματικά
3. Χρησιμοποιούμε αντίστοιχα σύμβολα σε διαφορετική γραμματοσειρά αλλά με παρενθέσεις και χωρίς υποδείκτες για τα δεδομένα

Ανανέωση 1ης τάξης (rank-1 update)

Listing 11: Pseudo-MATLAB BLAS-2

```
%          rank-1  update
% C <- C + a*b'
 for  J=1:k2
      for  I=1:k1
%
           C((I-1)*m1+1:I*m1,(J-1)*m2+1:  J*m2) = ...
                      C((I-1)*m1+1:I*m1,(J-1)*m2+1:  J*m2) +...
                          a((I-1)*m1+1:I*m1)*(b((J-1)*m2+1:J*m2))
      end
end
```

Ανανέωση 1ης τάξης (rank-1 update)

Listing 12: Pseudo-MATLAB BLAS-2 with L/S

```
for I=1:k1
    LOAD(a((I-1)*m1+1:I*m1));
    for J=1:k2
        C((I-1)*m1+1:I*m1,(J-1)*m2+1: J*m2) = ...
                    C((I-1)*m1+1:I*m1,(J-1)*m2+1: J*m2) +...
                        a((I-1)*m1+1:I*m1)*b((J-1)*m2+1:J*m2);
    end
end
```

$$\Phi = ((2m_1m_2 + m_2)k_2 + m_1)k_1 = 2n_1n_2 + n_1 + n_2k_1$$

$$\mu = 1 + \frac{1}{2m_1} + \frac{1}{2n_2} \geq \mu_{\text{min}}.$$

Δίνονται $\mathcal{K}, n_1, n_2$ και αναζητούμε τεμαχισμό που ελαχιστοποιεί το μ.

1. Προσέξτε: μεγαλύτερη πλοκάδα του a στην cache συνεπάγεται λιγότερες επαναλήψεις.
2. ... είναι λογικό να επιλέξουμε το μέγιστο m_1 που επιτρέπεται από την cache, άρα

$$\min_{m_1 \leq \mathcal{K}} \mu = 1 + \frac{1}{2\mathcal{K}} + \frac{1}{2n_2}$$

3. Αν $n_1 \leq \mathcal{K}$ τότε θέτουμε $m_1 = n_1$ και κατά συνέπεια $\min_{m_1 \leq \mathcal{K}} \mu = \mu_{\min}$.

Ξαναγράφουμε ...

.... με κλήση σε ((συνάρτηση)) ger για ανανέωση τάξης-1 μεγέθους $m_1 \times m_2$:

Listing 13: Pseudo-MATLAB BLAS-2 block with L/S

```
for J=1:k2
    for I=1:k1
        LOAD(a((I-1)*m1+1:I*m1));
        C((I-1)*m1+1:I*m1,(J-1)*m2+1: J*m2) = ...
            ger(C((I-1)*m1+1:I*m1,(J-1)*m2+1: J*m2),...
              a((I-1)*m1+1:I*m1),b((J-1)*m2+1:J*m2));
    end
end
```

Listing 14: Pseudo-MATLAB BLAS-2 block with L/S

```
function C = ger(C,a,b);
for j = 1:n2
    LOAD(b(j)) %        a in cache
     for i=1:n1
         LOAD(C(i,j))
         C(i,j) = C(i,j) + a(i)*b(j)
         STORE(C(i,j))
     end
end
```

Παρατηρήσεις

Ο τεμαχισμός σε πλοκάδες (πλοκαδοποίηση - blocking) είναι κλειδί για την συγγραφή αποδοτικού κώδικα για ιεραρχική μνήμη.

- Ο καλύτερος τεμαχισμός εξαρτάται από το μέγεθος της κρυφής μνήμης και των καταχωρητών (cache aware).
- Η ελαχιστοποίηση βασίστηκε σε μια βέλτιστη υλοποίηση όμοιου αλλά μικρότερου προβλήματος με διαστάσεις που επιλέξαμε εμείς.
- Για να βρούμε την καλύτερη υλοποίηση πρέπει να αξιολογήσουμε εναλλακτικές εμφωλεύσεις των βρόχων.

Πολλαπλασιασμός μητρώων

Αναφερόμαστε στην πράξη

$$C \leftarrow AB, \text{ όπου } A, B \in \mathbb{R}^{n\times n} \text{ ή και } C \leftarrow C + AB$$

- Εν συντομία MM για Matrix-Matrix multiplication (και για τις δύο εκδοχές!)
- Ο κλασικός αλγόριθμος χρειάζεται $2n^3 - n^2$ πράξεις.
- Αλγόριθμος με *υποκυβική* πολυπλοκότητα, $O(n^{\log_2 7})$, οφείλεται στον .
  $(\log_2 7 = 2.81)$: Σημαντική ανακάλυψη (breakthrough) του Strassen (Str69)
- Ταχύτερος γνωστός αλγόριθμος χρειάζεται $O(n^{2.376})$ πράξεις
  (Coppersmith,Winograd'90) (CW90)

Ανοικτό ερευνητικό ερώτημα:

Ποιά ειναι η ελάχιστη τιμή ω ώστε να αρκούν $O(n^{\omega+\epsilon})$ πολλαπλασιασμοί βαθμωτών για να εκτελεστεί η MM για οποιοδήποτε $\epsilon > 0$. (προφανώς $\omega \geq 2$).

Πολλαπλασιασμός γενικών μητρώων (αδόμητων, πυκνών)

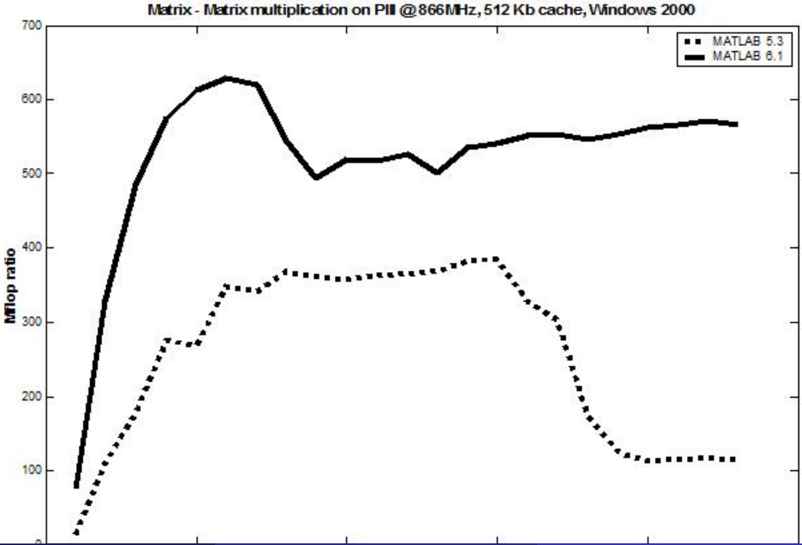
$$C = C + AB, \quad C \in \mathbf{R}^{n_1 \times n_2}, A \in \mathbf{R}^{n_1 \times n_3}, B \in \mathbf{R}^{n_3 \times n_2}.$$

Listing 15: Pseudo-MATLAB BLAS-3 block with L/S

```
 function  [C]  = mulmm(C,A,B)
LOAD (C,A,B)
 for  ?=1:n?
      for  ?=1:n?
           for  ?=1:n?
                C(i,j) = C(i,j)+A(i,k)*B(k,j);
           end
      end
 end
STORE(C)
```

$$\begin{aligned}[\mathcal{K}, \Omega, \Phi_{\min}] &= [O(n_1n_2 + n_1n_3 + n_2n_3), 2n_1n_2n_3, 2n_1n_2 + (n_1 + n_2)n_3] \\ \Rightarrow \mu_{\min} &= \frac{1}{n_3} + \frac{1}{2n_1} + \frac{1}{2n_2}.\end{aligned}$$

Γιατί μας ενδιαφέρει?



Γιατί μας ενδιαφέρει?

Πολλές εφαρμογές περιέχουν (μερικές φορές σε λανθάνουσα μορφή) πράξεις τύπου BLAS-3

Συμπεράσματα και βασικά αποτελέσματα

Αν $n_1 = n_2 = n_3 = n$, η ελάχιστη τιμή $\mu_{\text{min}} = O(\frac{1}{n})$ είναι πολύ μικρότερη από τις τιμές που συναντήσαμε ως τώρα και κάνει το πρόβλημα πολύ πιο ενδιαφέρον.

Υπόσχεση: Ο πολλαπλασιασμός μητρώων φαίνεται να προσφέρει την μεγαλύτερη δυνάμει τοπικότητα μεταξύ των αλγορίθμων που συναντήσαμε ως τώρα! (Θυμηθείτε τα Mflop/s). Το ερώτημα είναι "πώς μπορούμε να τον υλοποιήσουμε για να τον αξιοποιήσουμε?"

Θεωρητικό κάτω φράγμα: Το κάτω φράγμα του κόστους μετακινήσεων σε σύστημα με cache $\mathcal{K}$ α.κ.υ. για τον πολλαπλασιασμό μητρώων με $\Omega = \Omega(n^3)$ είναι $\Phi = \Theta(n^3/\sqrt{\mathcal{K}})$. (Αποδεικνυεται μέσω του red-blue pebble game των Hong και Kung (HK81)).

Πρακτικά: Χρησιμοποιώντας blocking για την καλύτερη υλοποίηση του πολλαπλασιασμού με $\Omega = \Omega(n^3)$ μπορούμε να πετύχουμε $\Phi = \Omega(n^3/\sqrt{\mathcal{K}})$.

Listing 16: MATLAB BLAS-3

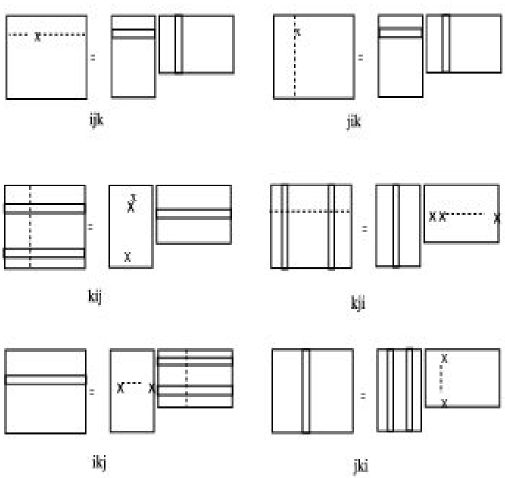
```
LOAD (C,A,B)
 for ?=1:n?
      for ?=1:n?
           for ?=1:n?
                C(i,j) = C(i,j)+A(i,k)*B(k,j);
           end
      end
 end
STORE(C)
```

Παρατήρηση: Υπάρχουν 6 ($= 3!$) διαφορετικές δυνατές δυνατότητες εμφώλευσης. Συμβολίζονται συνήθως με την τριπλέτα των δεικτών γραμμένη με τη σειρά που γίνεται η εμφώλευση: i δεικτοδοτεί τις γραμμές των C, A, j τις γραμμές των C, B και k τις στήλες του A και γραμμές του .

Ερμηνεία εμφωλεύσεων (GL96)

σειρά βρόχου	εσωτερικός βρόχος	μεσαίος βρόχος	τρόπος προσπέλασης
ijk	`DOT`	`MV/GAXPY` βάσει `DOT`	*A* κ. στήλ., *B* κ. γραμμ.
jik	`DOT`	`MV/GAXPY` βάσει `DOT`	*B* κ. στήλ., *A* κ. γραμμ.
ikj	`_AXPY`	`MV/GAXPY` βάσει `GAXPY`	*B* κ. γραμμές
jki	`_AXPY`	`MV/GAXPY` βάσει `GAXPY`	*A* κ. στήλες
kij	`_AXPY`	`GER`	*B* κ. γραμμές
kji	`_AXPY`	`GER`	*A* κ. στήλες

Σύνοψη ιεραρχίας



Παράδειγμα: πολλαπλασιασμός μητρώων με εμφώλευση *ijk*

Listing 17: MATLAB BLAS-3

```
for i=1:n1
    for j=1:n2
        s = 0;
        for k=1:n3
            s = s+A(i,k)*B(k,j);
        end
        C(i,j)=s;
    end
end
```

Σύσταση: Δοκιμάστε να ενθέσετε LOAD, STORE και εξετάστε τα προκύπτοντα $\mathcal{K}$, Φ_{min}.

- Στο μοντέλο μας (και σε συστήματα με ιεραρχική μνήμη) επιλέγουμε υλοποιήσεις που έχουν για βάση πολλαπλασιασμούς υπομητρώων/πλοκάδων.
- Οδηγούμαστε έτσι σε αλγορίθμους με τριπλά εμφωλευμένα βρόχους όπου η εσωτερική έκφραση είναι και αυτή πράξη ανανέωσης υπομητρώου.

Πολλαπλασιασμός μητρώων σε πλοκάδες

Listing 18: MATLAB BLAS-3: blocked MM with IKJ outer

```
% !TeX encoding = ISO-8859-7
%
for I=1:k1
 for K=1:k3
  LOAD(A((I-1)*m1+1:I*m1,(K-1)*m3+1:K*m3))
   for J=1:k2
               LOAD(C((I-1)*m1+1:I*m1,(J-1)*m2+1:J*m2),...
                    B((K-1)*m3+1:K*m3,(J-1)*m2+1:J*m2))
        C((I-1)*m1+1:I*m1,(J-1)*m2+1:J*m2) = ...
                    C((I-1)*m1+1:I*m1,(J-1)*m2+1:J*m2) +...
                        A((I-1)*m1+1:I*m1,(K-1)*m3+1:K*m3)*...
                            B((K-1)*m3+1:K*m3,(J-1)*m2+1:J*m2)
                STORE(C((I-1)*m1+1:I*m1,(J-1)*m2+1:J*m2))
     end
   end
end
```

Ανάλυση

- Τα m_1, m_2, m_3 επελέγησαν έτσι ώστε ο "μικρός" πολλαπλασιασμός MM, $Z \leftarrow Z + XY$ για $Z \in \mathbb{R}^{m_1 \times m_2}$, $X \in \mathbb{R}^{m_1 \times m_3}$, $Y \in \mathbb{R}^{m_3 \times m_2}$ να εκτελείται με βέλτιστο αριθμό μεταφορών $\Phi_{\text{min}} = 2m_1 m_2 + m_1 m_3 + m_2 m_3$, οπότε

$$\Phi = n_1 n_3 + n_1 n_2 n_3 \left(\frac{1}{m_1} + \frac{2}{m_3}\right),$$

- Αν θέσουμε $\beta = m_1 = m_2 = m_3$ όπου $3\beta^2 \leq \mathcal{K}$ και τα μητρώα είναι τετραγωνικά, τότε $\Phi = (\frac{3}{\beta})n^3 + n^2$ δηλ.

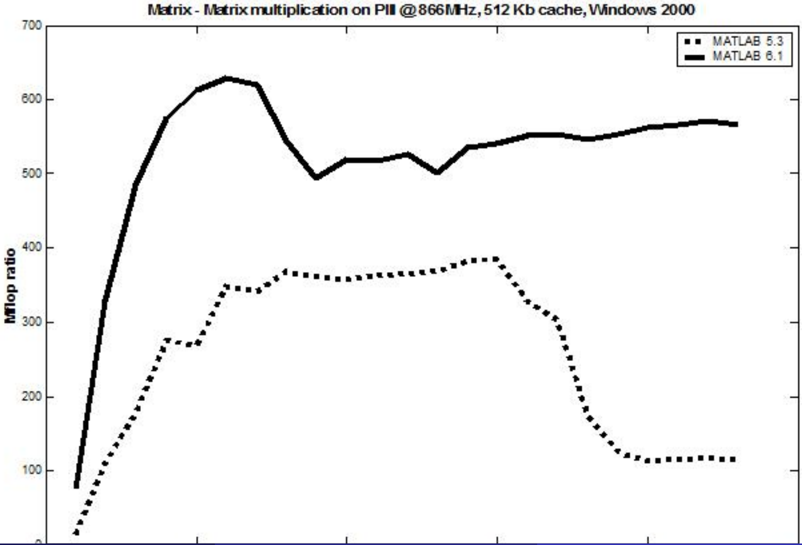
$$\Phi = \Omega\left(\frac{n^3}{\sqrt{\mathcal{K}}}\right) \text{ όπως πραναγγείλαμε νωρίτερα,}$$

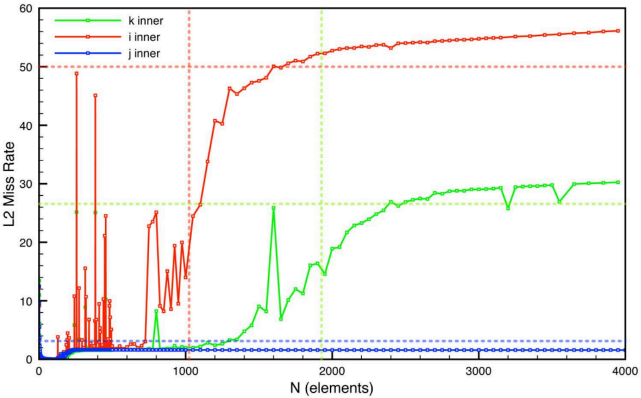
και

$$[\mathcal{K}, \Omega, \Phi] = [3\beta^2, 2n^3, \frac{3n^3}{\beta} + n^2].$$

- Προσεκτικότερη ανάλυση (όχι εδώ) δείχνει ότι επιλέγοντας β έτσι ώστε $m_1 m_3 + m_3 = \beta^2 + \beta \leq \mathcal{K}$, πετυχαίνουμε παρόμοιο Φ.

Μερικές οπτικοποιήσεις ≈ 2005

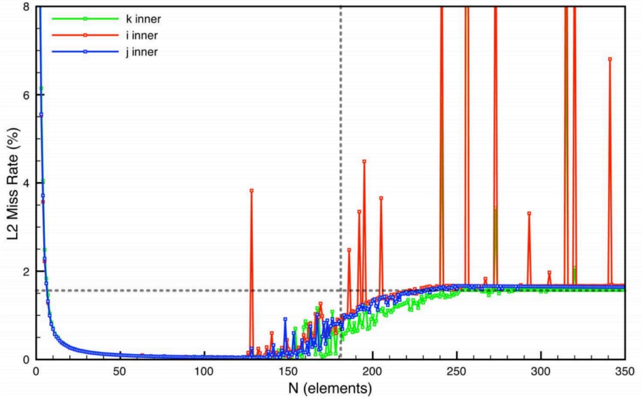




Σχήμα: cache miss ratio στο Intel Itanium 2: 256KB L2, line size 128 bytes, 64 bit arith.

Σύσταση για όσους ενδιαφέρονται: Διαβάστε (τις πρώτες 7 σελ.) της (KP08).

Μερικές οπτικοποιήσεις (KP08)



Σχήμα: cache miss ratio στο Intel Itanium 2: 256KB L2, line size 128 bytes, 64 bit arith.

Σύσταση για όσους ενδιαφέρονται: Διαβάστε (τις πρώτες 7 σελ.) της (KP08).

BLAS: Basic Linear Algebra Subprograms

Από τη Wikipedia:

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. They were first published in 1979, and are used to build larger packages such as LAPACK. Heavily used in high-performance computing, highly optimized implementations of the BLAS interface have been developed by hardware vendors ...

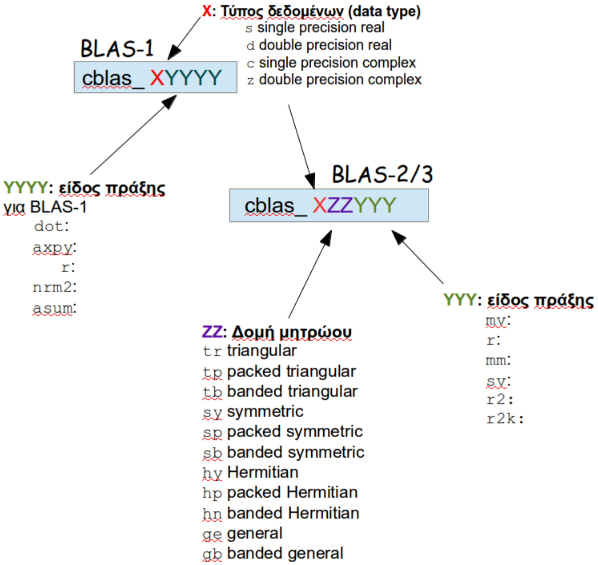
Τρία επίπεδα: Εν συντομία BLAS-1, BLAS-2, BLAS-3. Γενικά, αναφέρονται σε πράξεις μεταξύ αλγεβρικών αντικειμένων (μητρώων, διανυσμάτων) στα οποία μπορεί να υφίστανται 3 διαστάσεις συνολικά (οι βαθμωτοί δεν λαμβάνονται υπόψη)

- BLAS-1: 1 διάστ. > 1, δηλ. πράξεις μεταξύ διανυσμάτων (π.χ. DOT, _AXPY).
- BLAS-2: 2 διαστ. > 1, δηλ. πράξεις μεταξύ μητρώων, διανυσμάτων (π.χ. MV, ανανέωση τάξης-1)..
- BLAS-3: 3 διαστ. > 1, δηλ. πράξεις μεταξύ μητρώων (π.χ. MM).

BLAS: Basic Linear Algebra Subprograms

- API για βασικές πράξεις της ΑΓΑ. Περιγράφονται η ονοματολογία, ο τρόπος κλήσης και οι πράξεις, όχι όμως η υλοποίηση.
- Συνοπτική παρουσίαση http://www.netlib.org/lapack/lug/node145.html
- Υλοποιήσεις:
  - Κώδικες αναφοράς Fortran http://netlib.org/blas/
  - ATLAS Automaticaly Tuned Linear Algebra Subroutines για C, Fortran
  - GotoBLAS
  - MKL BLAS
  - OpenBLAS
  - και πολλές άλλες ...

Ονοματολογία BLAS



D. Coppersmith and S. Winograd.
Matrix multiplication via arithmetic progressions.
J. Symb. Comput., 9(3):251ÔøΩ280, 1990.

G.H. Golub and C.F. Van Loan.
Matrix Computations.
The Johns Hopkins University Press, Baltimore, 3d edition, 1996.

J.-W. Hong and H.T. Kung.
I/O complexity: The red-blue pebble game.
In *Proc. Thirteenth Annual ACM Symposium on Theory of Computing*, STOC '81, pages 326–333, New York, NY, USA, 1981. ACM.

M Kulkarni and K Pingali.
An Experimental Study of Self-Optimizing Dense Linear Algebra Software.
Proc. IEEE, 96(5):832–848, April 2008.

V. Strassen.
Gaussian elimination is not optimal.
Numer. Math., 13:354–356, 1969.