Επιστημονικός Υπολογισμός

Ε.Γαλλόπουλος

ΤΜΗΥΠ, Π. Πατρών

Διάλεξη 3: 18 Οκτωβρίου 2017

Περιεχόμενα

Υπενθύμιση: Απλό υπολογιστικό μοντέλο

Χαρακτηριστικά

- Επεξεργαστής και αρχιτεκτονική Load/Store
- αρχείο καταχωρητών
- κρυφή μνήμη ενός επιπέδου K θέσεων με write back
- κύρια μνήμη M θέσεων
- κόστη: Load, Store, πράξεις α.κ.υ.
- Κάθε πράξη αριθμ.κ.υ. στοιχίζει $\tau_{\text{αριθ}}$
- load από μνήμη στον επεξεργαστή σε χρόνο $\tau_{\text{μετ}}$
- load από κρυφή μνήμη στον επεξεργαστή σε $\tau_{\text{μετ}}^{(0)}$
- store από κρυφή μνήμη η επεξεργαστή σε $\tau_{\text{μετ}}$
- $\tau_{\text{μετ}}^{(0)} \approx 0$

Δείκτες

What we can't measure we can't improve (D. Patterson)

- Ω αριθμός πράξεων α.κ.υ.
- Φ αριθμός μεταφορών μεταξύ κύριας μνήμης και καταχωρητών ή κρυφής μνήμης
- Φ_{min} ελάχιστος αριθμός μεταφορών αλγορίθμου αν διαθέταμε απεριόριστη μνήμη σε όλα τα επίπεδα
- $T_{\alpha\rho\theta}$ χρόνος που αναλώνεται για αριθμητικές πράξεις α.κ.υ.
- $T_{\mu\epsilon\tau}$ χρόνος που αναλώνεται για μεταφορές α.κ.υ.

Υποθέτουμε ότι ο χρόνος εκτέλεσης μιας υλοποίησης μπορεί να εκτιμηθεί από τον τύπο

$$T = T_{\alpha\rho\theta} + T_{\mu\epsilon\tau}$$

και ότι έχουμε μόνον 2 επίπεδα μνήμης: Κύρια μνήμη και register - cache file.

Παρατηρήσεις

Θέτουμε

$\mu := \frac{\Phi}{\Omega}$	μεταφορές ανά αριθμητική πράξη για τη συγκεκριμένη υλοποίηση (θέμα λογισμικού)
$\tau_{\text{αριθ}}$	χρόνος για 1 αριθμ. πράξη (θέμα υλικού)
$\tau_{\text{μετ}}$	χρόνος για 1 μεταφορά (θέμα υλικού)

Εμπειρική παρατήρηση και υπόθεση εργασίας:

Οι μεταφορές είναι πολύ πιο ακριβές από τις αριθμητικές πράξεις

$$\tau_{\text{μετ}} \gg \tau_{\text{αριθ}}$$

Επακόλουθα

Ξαναγράφουμε

$$\begin{aligned} T &= T_{\text{αρθ}} + T_{\text{μετ}} \\ &= \tau_{\text{αρθ}}\Omega + \tau_{\text{μετ}}\Phi, \\ &= T_{\text{αρθ}}\left(1 + \mu\frac{\tau_{\text{μετ}}}{\tau_{\text{αρθ}}}\right), \end{aligned}$$

Δείκτες

Αν διερευνήσουμε όλες τις υλοποιήσεις του αλγορίθμου, θα υπάρχει κάποια που απαιτεί το μικρότερο κόστος: Γράφουμε για το αντίστοιχο μ,

$$\mu_{\text{best}} = \arg \min_{[\Omega,\Phi]\in \text{σύνολο υλοποιήσεων}} T(\mu)$$

Προσοχή: Δεν έχουμε ακόμα μιλήσει για την ακρίβεια!

Τι λέει η θεωρία? Παιχνίδι με τους μπλέ και κόκκινους βόλους (HK81)

STOC(Milwaukee 1981),326-333.

In this paper, the *red-blue pebble game* is proposed to model the input-output complexity of algorithms. Using the pebble game formulation, a number of lower bound results for the I/O requirement are proven. For example, it is shown that to perform the n-point FFT or the ordinary nxn matrix multiplication algorithm with O(S) memory, at least $\Omega(n \log n/\log S)$ or $\Omega(n^3/\sqrt{S})$, respectively, time is needed for the I/O. Similar results are obtained for algorithms for several other problems. All of the lower bounds presented are the best possible in the sense that they are achievable by certain decomposition schemes.

Results of this paper may provide insight into the difficult task of balancing I/O and computation in special-purpose system designs. For example, for the n-point FFT, the lower bound on I/O time implies that an S-point device achieving a speed-up ratio of order log S over the conventional O(n log n) time implementation is all one can hope for.

I/O COMPLEXITY: THE RED-BLUE PEBBLE GAME

Hong, Jia-Wei and H. T. Kung

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Red-Blue pebble game[1]

... the only solution is to limit the volume of data movement to/from memory by enhancing data reuse in registers and higher levels of the cache.

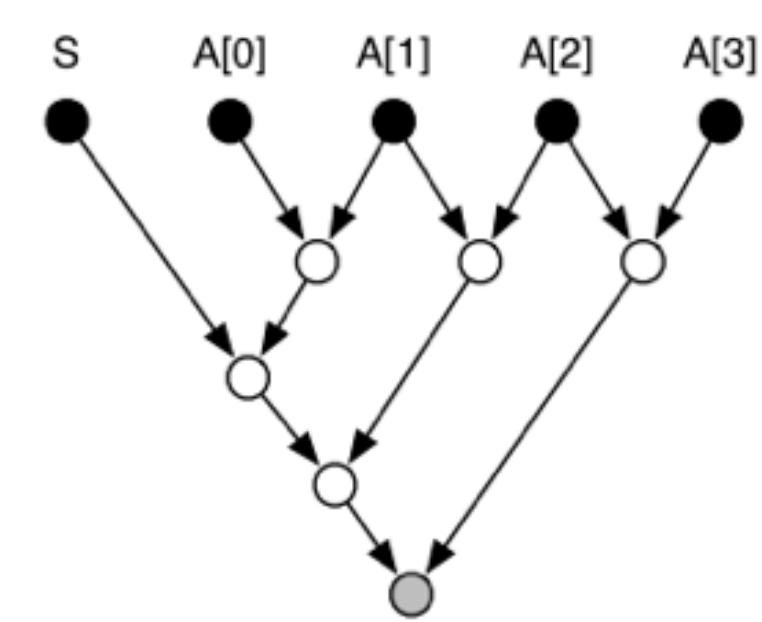
Given a CDAG (computational DAG) $C = (I, V, E, O)$ such that any vertex with no incoming (resp. outgoing) edge is an element of$I$ red pebbles and an arbitrary number of blue pebbles, with a blue pebble on each input vertex. A complete game is any sequence of steps using the following rules that results in a final state with blue pebbles on all output vertices:

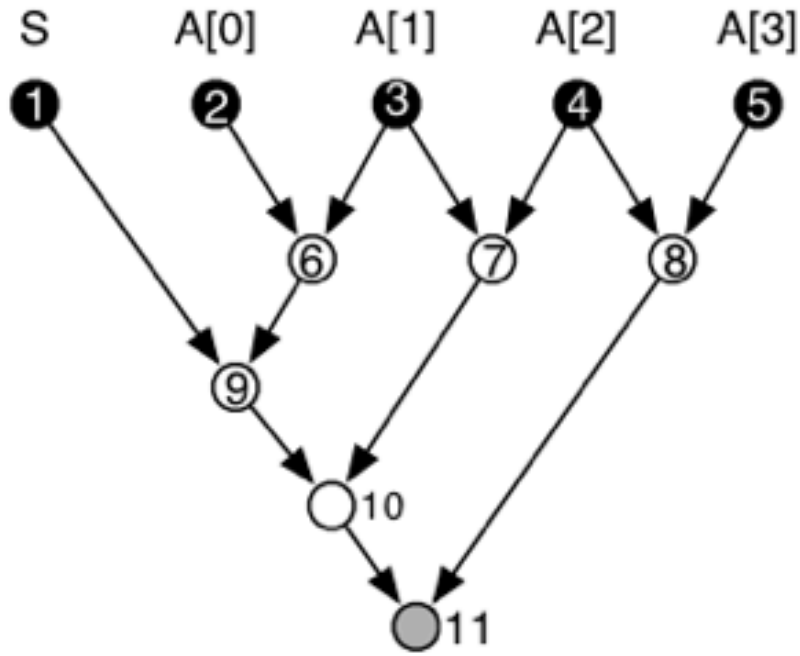
- **R1 (Input)** A red pebble may be placed on any vertex that has a blue pebble (load from slow to fast memory),
- **R2 (Output)** A blue pebble may be placed on any vertex that has a red pebble (store from fast to slow memory),
- **R3 (Compute)** If all immediate predecessors of a vertex of $V \setminus I$ have red pebbles, a red pebble may be placed on that vertex (execution or "firing" of operation),
- **R4 (Delete)** A red pebble may be removed from any vertex (reuse storage).

[1] Περιγραφή από (ERP+14)

Παράδειγμα (ERP$^+$14)

```
for (i = 1; i < 4; ++i)
  S += A[i-1] + A[i];
```





Σχήμα: CDAG and schedule for its complete execution with $S = 3$

Roofline model (WWP09)

DOI:10.1145/1498765.1498785

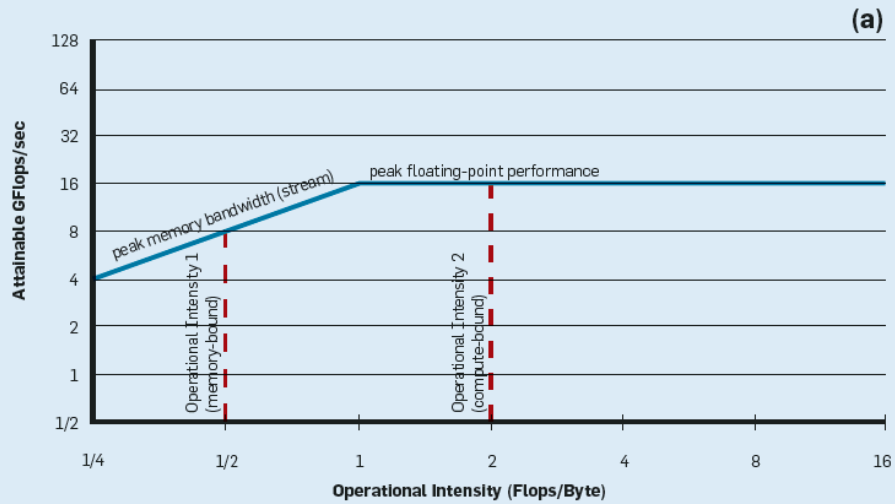
The Roofline model offers insight on how to improve the performance of software and hardware.

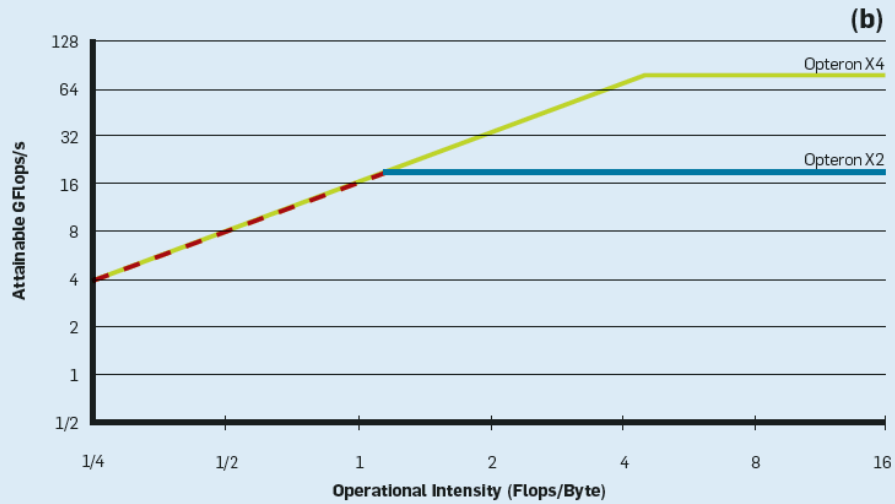
BY SAMUEL WILLIAMS, ANDREW WATERMAN, AND DAVID PATTERSON

Roofline: An Insightful Visual Performance Model for Multicore Architectures

Roofline model

Figure 1: Roofline model for (a) AMD Opteron X2 and (b) Opteron X2 vs. Opteron X4.





Σχόλια

Από όλες τις υλοποιήσεις, θα προτιμούσαμε εκείνη που είναι ταχύτερη, ακριβέστερη και με το μικρότερο κόστος.

- There is no free lunch: η ταυτόχρονη ικανοποίηση των παραπάνω δεν είναι δυνατή $\Rightarrow$ δύσκολη βελτιστοποίηση!
- Σχηματικά μπορεί $T_{\alpha\rho\theta} = (1/\mathrm{ΑΚΡΙΒΕΙΑ})$
- Αντιστάθμιση και trade-offs ώστε να επιτευχθεί ικανοποιητικός συνδυασμός.

Πώς μετράμε την επίδοση (Mflop/s)?

Θεωρητικά Εκτίμηση από το Ω και το Φ (πλήθος μεταφορών - περισσότερα σε λίγο)

Πρακτικά Προσεκτική μέτρηση του ((χρόνου επίλυσης)) και του ((πλήθους πράξεων)).

Ζητήματα:

- Χρειάζεται υποστήριξη (εργαλεία υλικού και λογισμικού),
- Χρειάζονται πολλές μετρήσεις και εξαγωγή στατιστικών στοιχείων από αυτές π.χ. μέσος όρος και απόκλιση
- συνήθως απορρίπτονται οι εξωφρενικές τιμές (outliers) ή διερευνάται γιατί εμφανίζονται.

Case study: Μελετάμε την επίδοση διαφορετικών υλοποιήσεων MATLAB της πράξης MV.

Listing 1: MV ανά γραμμές

```
function [y] = ...
    mulrLOOPS(A,x);
[m,n]=size(A);
for i=1:m, y(i) = 0; ...
    end; % y <- 0
for i = 1:m
    for j = 1:n
        y(i) = y(i) + ...
            A(i,j)*x(j);
    end
end
```

Listing 2: MV ανά στήλες

```
function [y] = ...
    mulcLOOPS(A,x);
[m,n]=size(A);
for i=1:m, y(i) = 0; ...
    end; % y <- 0
for j = 1:n
    for i = 1:m
        y(i) = y(i) + ...
            A(i,j)*x(j);
    end
end
```

Παραδείγματα

Listing 3: MV ανά γραμμές

```
function [y] = mulr(A,x);
[m,n]=size(A); y = ...
    zeros(m,1);
for i=1:m
    y(i) = A(i,:)*x;
end
```

Listing 4: MV ανά στήλες

```
function [y] = mulc(A,x);
[m,n]=size(A);
y= A(:,1)*x(1);
for j=2:n
    y = y + A(:,j)*x(j);
end
```


Listing 5: Ενδογενής

```
function [y] = mulmv(A,x);
y = A*x; % y = mtimes(A,x);
```

Σχετικά με τις μετρήσεις

Σύστημα Intel Atom N270 @ 1.6 GHz, RAM 1.48 GB, caches: level 1, 32KB; level 2, 512KB cache (write-back). Οι πληροφορίες μέσω του προγράμματος `winaudit.exe`.

Λογισμικό Windows XP Pro, MATLAB 7.11 (R2010b)

Μετρήσεις Λήφθηκαν με τις εντολές `tic`, `toc`.

Listing 6: script χρονομέτρησης για mtimes

```
n = 2000; A = rand(n);
x = rand(n,1);
total=40; jstart=100;
jend = n; jstep = 100; mv_types = {'mulrLOOPS','mulcLOOPS','mulr', 'mulc', 'mtimes'};
tmv=zeros(length((jstart:jstep:jend)),7);
tmv(:,1:2) = ((jstart:jstep:jend)',(2*(jstart:jstep:jend).^2)');
for mcount = 1:length(mv_types)
jcount=0;
for j= jstart:jstep:jend
    jcount = jcount+1;
    Aj=A(1:j,1:j); xj = x(1:j);
    tsum =0;
    for itimes=1:total
        tic; feval(mv_types{mcount},Aj,xj); %y = mtimes(Aj,xj); %y=mulr(Aj,xj); %y=mulc(Aj,xj);
        tsum = tsum + toc;
    end
    tmv(jcount,mcount+2)=tsum/total;
end
end
```

Περί `feval` της MATLAB

feval

Evaluate function

Syntax

```
[y1,...,yN] = feval(fun,x1,...,xM)
```

Description

`[y1,...,yN] = feval(fun,x1,...,xM)` evaluates a function using its name or its handle, and using the input arguments `x1,...,xM`.

The `feval` function follows the same scoping and precedence rules as calling a function handle directly. For more information, see Create Function Handle.

Examples

Evaluate Function with Function Name as Character Vector

Round the value of `pi` to the nearest integer using the name of the function.

```
fun = 'round';
x1 = pi;
y = feval(fun,x1)
```

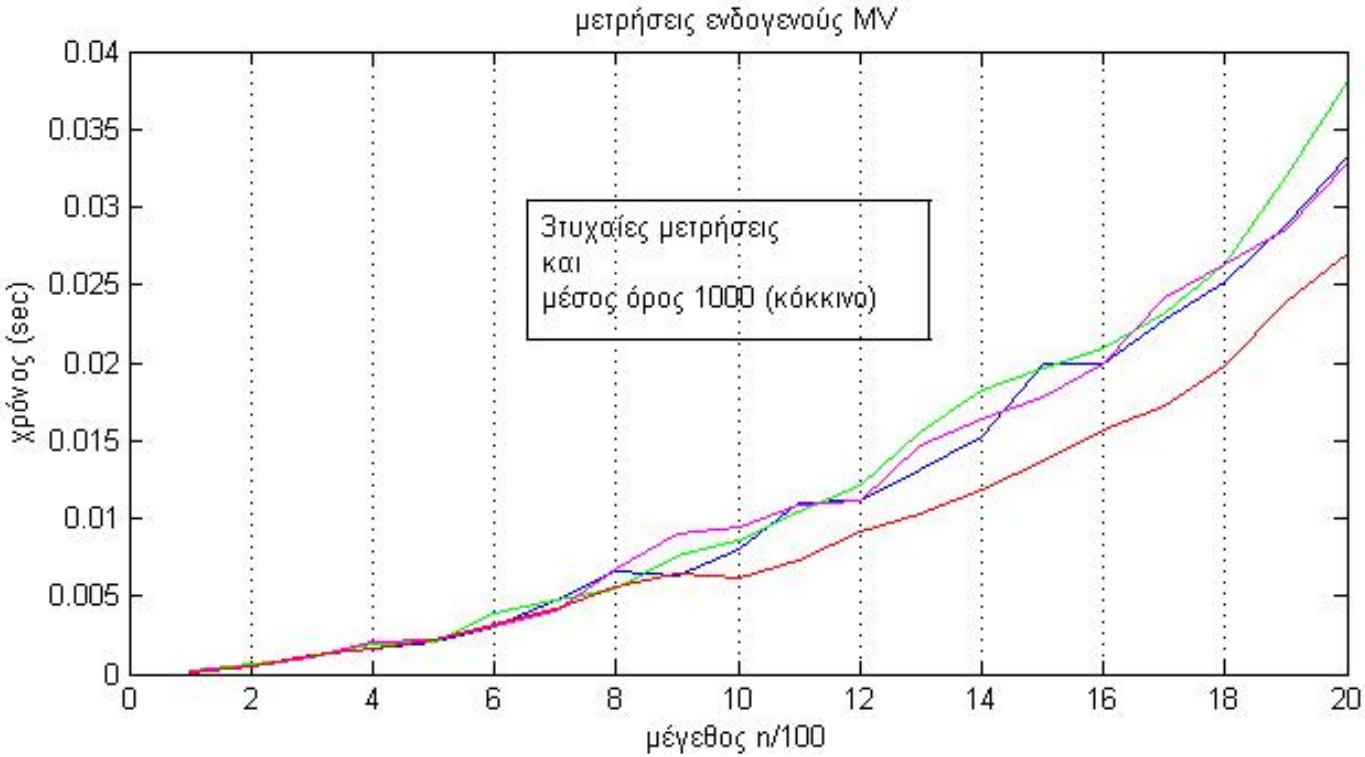
```
y = 3
```

Round the value of `pi` to two digits to the right of the decimal point.

```
x2 = 2;
y = feval(fun,x1,x2)
```

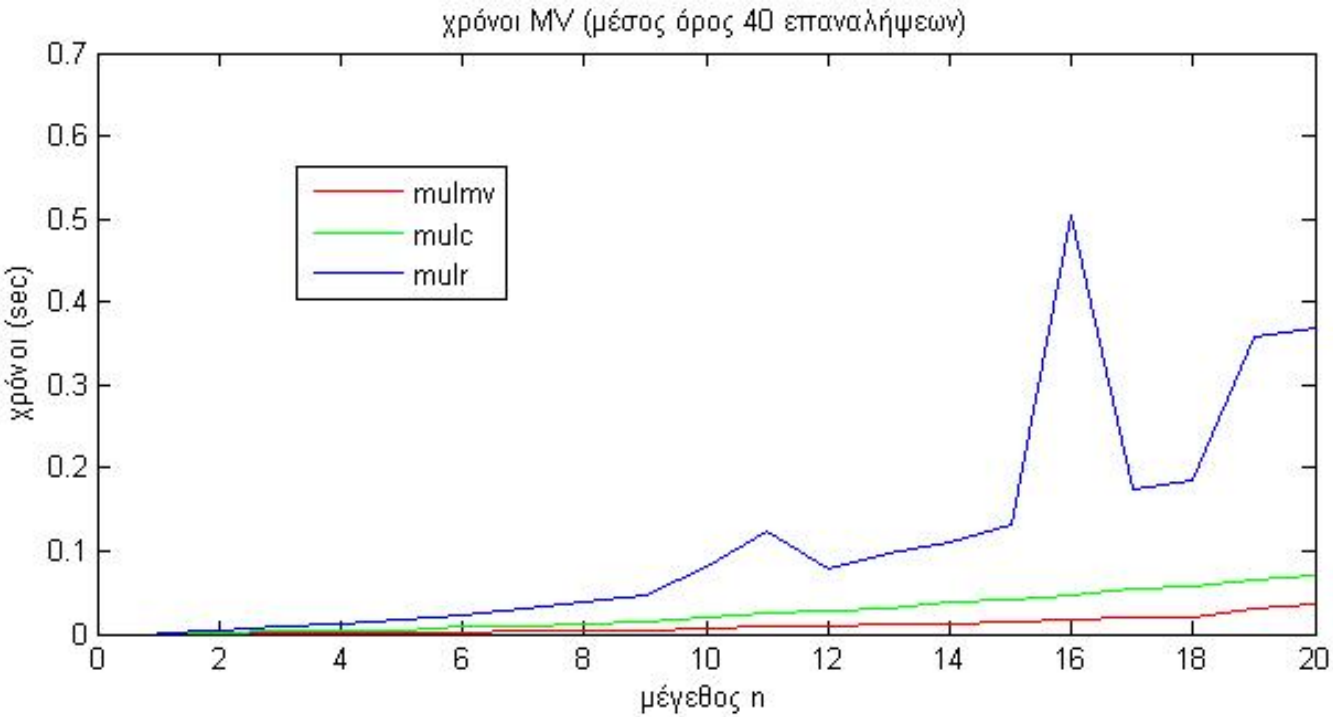
```
y = 3.1400
```

Αποτελέσματα μετρήσεων



Σχήμα: Χρόνοι εκτέλεσης ενδογενούς mtimes (εμείς την έχουμε πακετάρει εντός της mulmv) για $\mathrm{n{=}100{:}10{:}2000}$. Παρουσιάζονται 3 τυχαίες μετρήσεις και ο μέσος όρος 1000 μετρήσεων

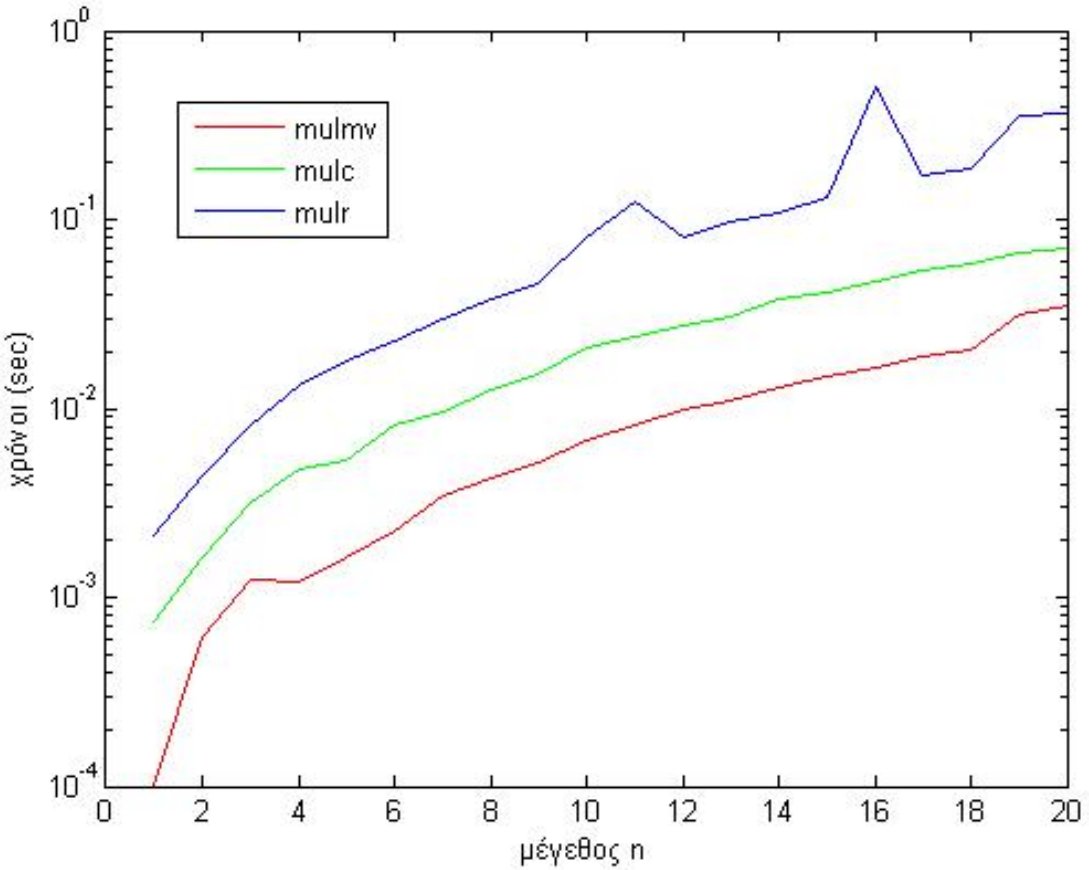
Ενδεικτικές μετρήσεις



Σχήμα: Μέσοι χρόνοι εκτέλεσης των $\mathrm{mulmv(mtimes)}$, mult, mulc για $\mathrm{n{=}100{:}10{:}2000}$.

Ενδεικτικές μετρήσεις

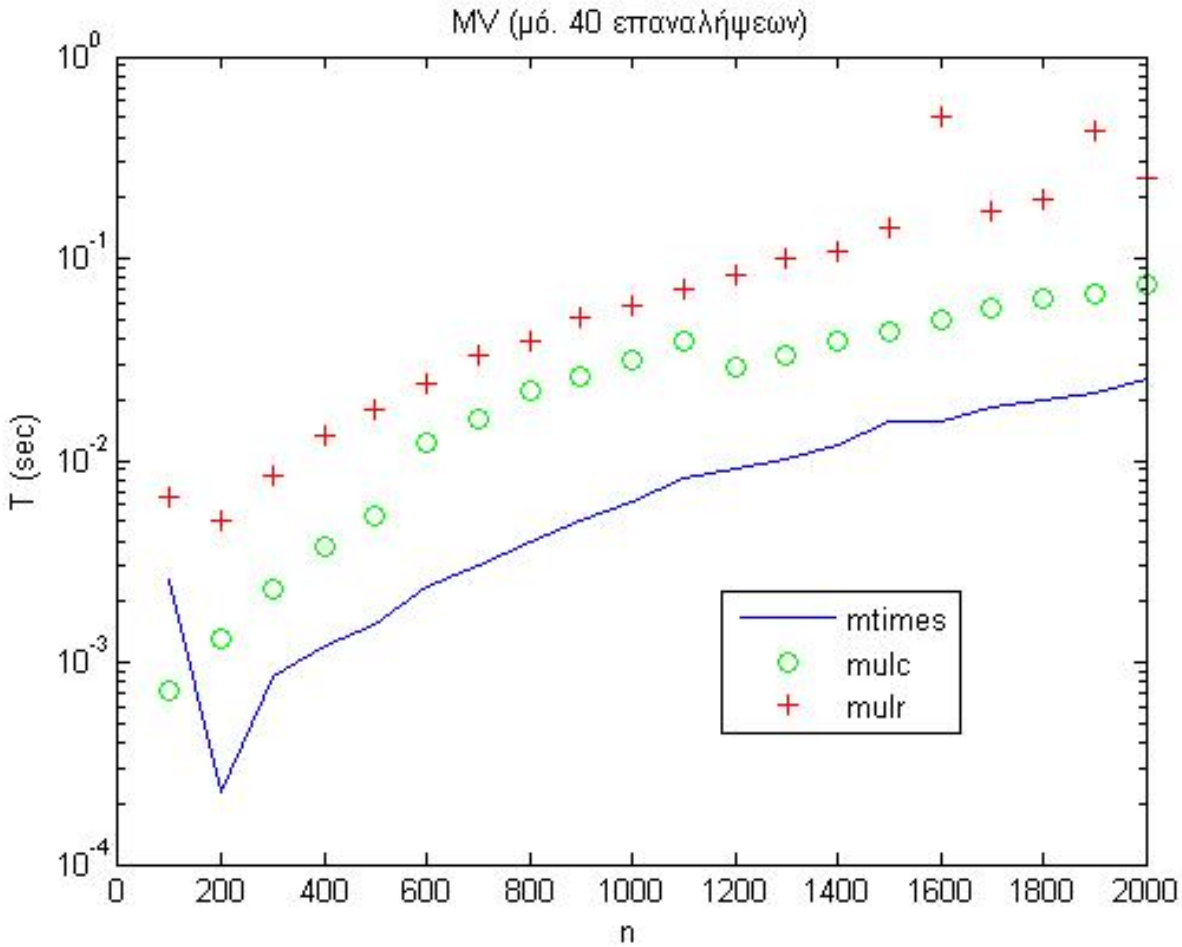
Ίδια δεδομένα σε λογαριθμική κλίμακα



Σχήμα: Μέσοι χρόνοι εκτέλεσης των $\mathrm{mulmv(mtimes)}$, mult, mulc για $\mathrm{n{=}100{:}10{:}2000}$

Προσέξτε ότι η λογαριθμική κλίμακα για το χρόνο αναδεικνύει πολύ καλύτερα τις διαφορές στην επίδοση για όλα τα μεγέθη *n*.

Επανάληψη μετρήσεων (μέσος όρος διάρκειας)



Σχήμα: Επανάληψη με λιγότερες μετρήσεις $\mathrm{n}{=}100{:}100{:}2000$. Οι μετρήσεις των $\mathrm{mulc},\ \mathrm{mulr}$ σημειώνονται με στίγμα αντί συνεχή γραμμή. Έχουμε μεγαλύτερη πιστότητα ίσως όμως λιγότερη ευκρίνεια.

Υπενθύμιση: Μονάδα ρυθμού επίδοσης

- ο ρυθμός εκτέλεσης πράξεων κινητής υποδιαστολής ...
- πλήθος πράξεων α.κ.υ. ανά μονάδα χρόνου

$$\mathrm{Mflop/s}(\mu\acute{\epsilon}\gamma\kappa\alpha\varphi\lambda o\pi\varsigma\ \pi\epsilon\rho\ \sigma\acute{\epsilon}\kappa o\nu\tau)$$

Με την αύξηση της υπολογιστικής ισχύος, η μονάδα αυτή αρχίζει να αντικαθίσταται από το Gflop/s που αντιστοιχεί σε 1 δισεκατομμύριο πράξεις το δευτερόλεπτο.

- πόσες πράξεις α.κ.υ. εκτελούνται ανά μονάδα χρόνου? (auto: απόσταση/χρόνος $\longrightarrow$ ταχύτητα)
- ΠΡΟΣΟΧΗ: δεν αρκεί για να αξιολογήσουμε τους χρόνους εκτέλεσης δύο αλγορίθμων.

Υπενθύμιση: Μονάδα ρυθμού επίδοσης

- ο ρυθμός εκτέλεσης πράξεων κινητής υποδιαστολής ...
- πλήθος πράξεων α.κ.υ. ανά μονάδα χρόνου

$$\text{Mflop/s}(\text{μέγκαφλοπς περ σέκοντ})$$

Με την αύξηση της υπολογιστικής ισχύος, η μονάδα αυτή αρχίζει να αντικαθίσταται από το Gflop/s που αντιστοιχεί σε 1 δισεκατομμύριο πράξεις το δευτερόλεπτο.

- πόσες πράξεις α.κ.υ. εκτελούνται ανά μονάδα χρόνου? (auto: απόσταση/χρόνος $\longrightarrow$ ταχύτητα)
- ΠΡΟΣΟΧΗ: δεν αρκεί για να αξιολογήσουμε τους χρόνους εκτέλεσης δύο αλγορίθμων.

Π.χ. ένα θεμελιώδες συμπέρασμα που θα προκύψει σε επόμενο κεφάλαιο είναι ότι στις σύγχρονες αρχιτεκτονικές, με έξυπνο προγραμματισμό, ο πολλαπλασιασμός μητρώων εκτελείται με πολύ υψηλότερο ρυθμό Mflop/s από τον πολλαπλασιασμό μητρώου με διάνυσμα. Προφανώς βέβαια, ο χρόνος εκτέλεσης ενός πολλαπλασιασμού δύο μητρώων μεγέθους $n \times n$ είναι μεγαλύτερος από τον χρόνο πολλαπλασιασμού ενός μητρώου του ίδιου μεγέθους επί διάνυσμα. Αν όμως προγραμματίσετε τον πολλαπλασιασμό μητρώων ως βρόχο που εκτελεί n πολλαπλασιασμούς μητρώου-διανύσματος, ο συνολικός χρόνος εκτέλεσης θα είναι πολύ μεγαλύτερος.

Μετρήσεις Ω και ρυθμού εκτέλεσης

Operations	n	Ω	Mflop/s
calling PAPI flops	200	2	0.15
dot product	200	413	13.73
matrix vector	200	82053	252.12
random matrix	200	139967	67.12
chol(a)	200	3201127	789.27
lu(a)	200	5493443	829.53
x=a\y	200	6228144	742.98
condest(a)	200	7126555	173.63
qr(a)	200	13236723	1033.10
matrix multiply	200	16000012	1280.42
inv(a)	200	17398916	853.39
svd(a)	200	27039244	685.65
cond(a)	200	27000896	763.26
hess(a)	200	30180072	1063.27
eig(a)	200	82578728	680.60
[u,s,v]=svd(a)	200	138280160	691.18
pinv(a)	200	170228800	764.50
s=gsvd(a)	200	303512192	765.81
[x,e]=eig(a)	200	198741216	753.79
[u,v,x,c,s]=gsvd(a,b)	200	319475232	789.67

Οι μετρήσεις έγιναν χρησιμοποιώντας παλαιά έκδοση της MATLAB και το σύστημα PAPI για την καταμέτρηση των Ω (flops).

Ειδικές περιπτώσεις τιμών Gflop/s

peak GFlop/s ανώτατη τιμή για το υπολογιστικό σύστημα (από τον κατασκευαστή)

μέγιστη για πρόγραμμα με $\Omega/(8\Phi)$ πράξεις ανά byte μεταφοράς (operational intensity) και μέγιστο bandwidth από μνήμη B *bytes*$/$*sec*

$$\min\left\{\text{peak}\textit{GFlop}/s,\ \frac{\Omega}{8\Phi}\times B\right\}$$

- για τη μέτρηση της ακριβούς πραγματικής τιμής χρειάζεται προσεκτική μέτρηση των Ω, Φ στο υπολογιστικό σύστημα.
- απαιτείται υποδομή υλικού και λογισμικού για την ανίχνευση των τιμών (monitors)
- κατάλληλη ενοργάνωση του προγράμματος (program instrumentation)
- προσεκτική δειγματοληψία

Ποιος χρόνος?

user time χρόνος εκτέλεσης του προγράμματος

system time χρόνος συστήματος για υποστήριξη εκτέλεσης (kernel mode)

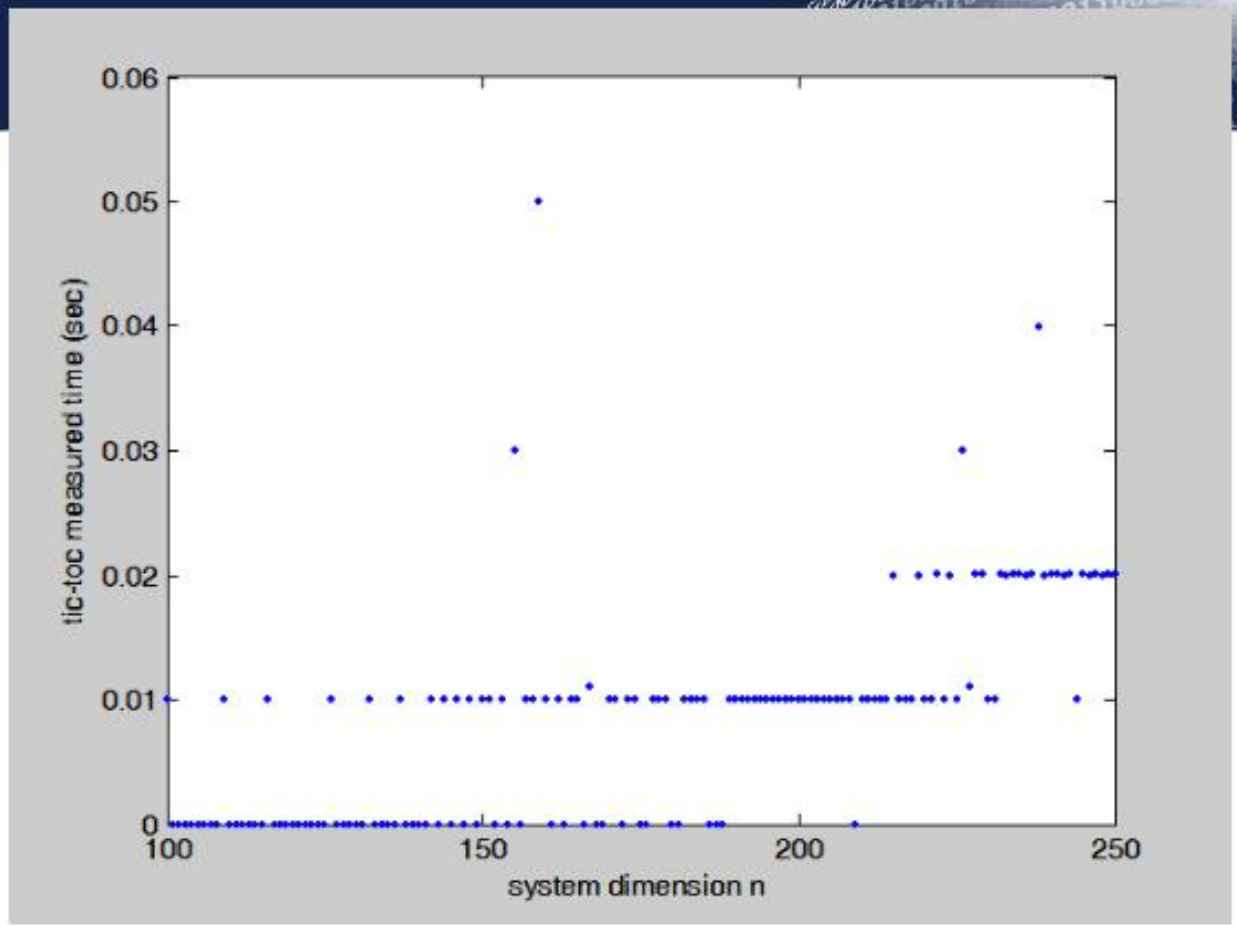
cpu time user time + system time χρόνος που αναλώνει το CPU για εκτέλεση διαδικασιών που οφείλονται στο πρόγραμμα. Δεν συμπεριλαμβάνεται ο χρόνος που το process ήταν switched out οπότε δεν συμπεριλαμβάνεται ο χρόνος για I/O.

wall-clock time (elapsed time) το χρονόμετρο - Συμπεριλαμβάνει χρόνο αναμονής για μεταφορές, καθυστερήσεις για μεταφορές από μνήμη και I/O, άλλες διαδικασίες συστήματος.

Ανίχνευση από MATLAB

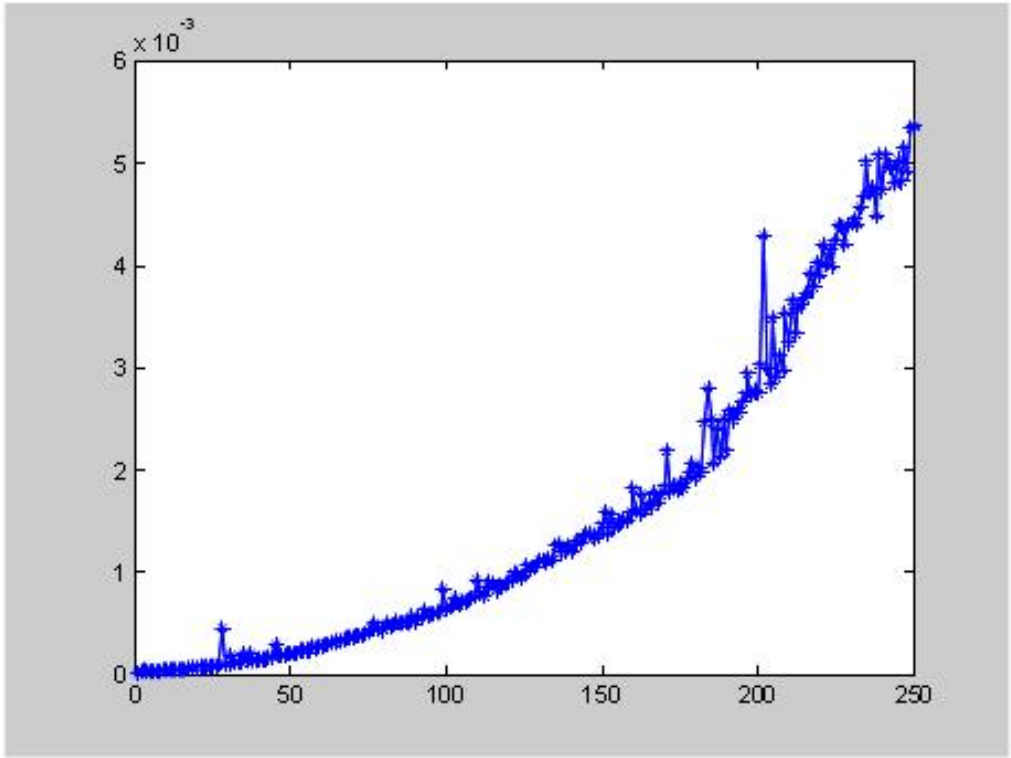
- tic , toc, clock, cputime, etime
- clock
- profile
- flops - ΑΚΥΡΟ -
- .... βλ.
  http://galton.uchicago.edu/ lekheng/courses/302/flo
- timeit.m ... επίσημη συνάρτηση από το MATLAB R2013b και μετά
- ... δείτε Measuring MATLAB Performance του Bill McKeeman, Mathworks, 2008.

Αποτελέσματα μετρήσεων

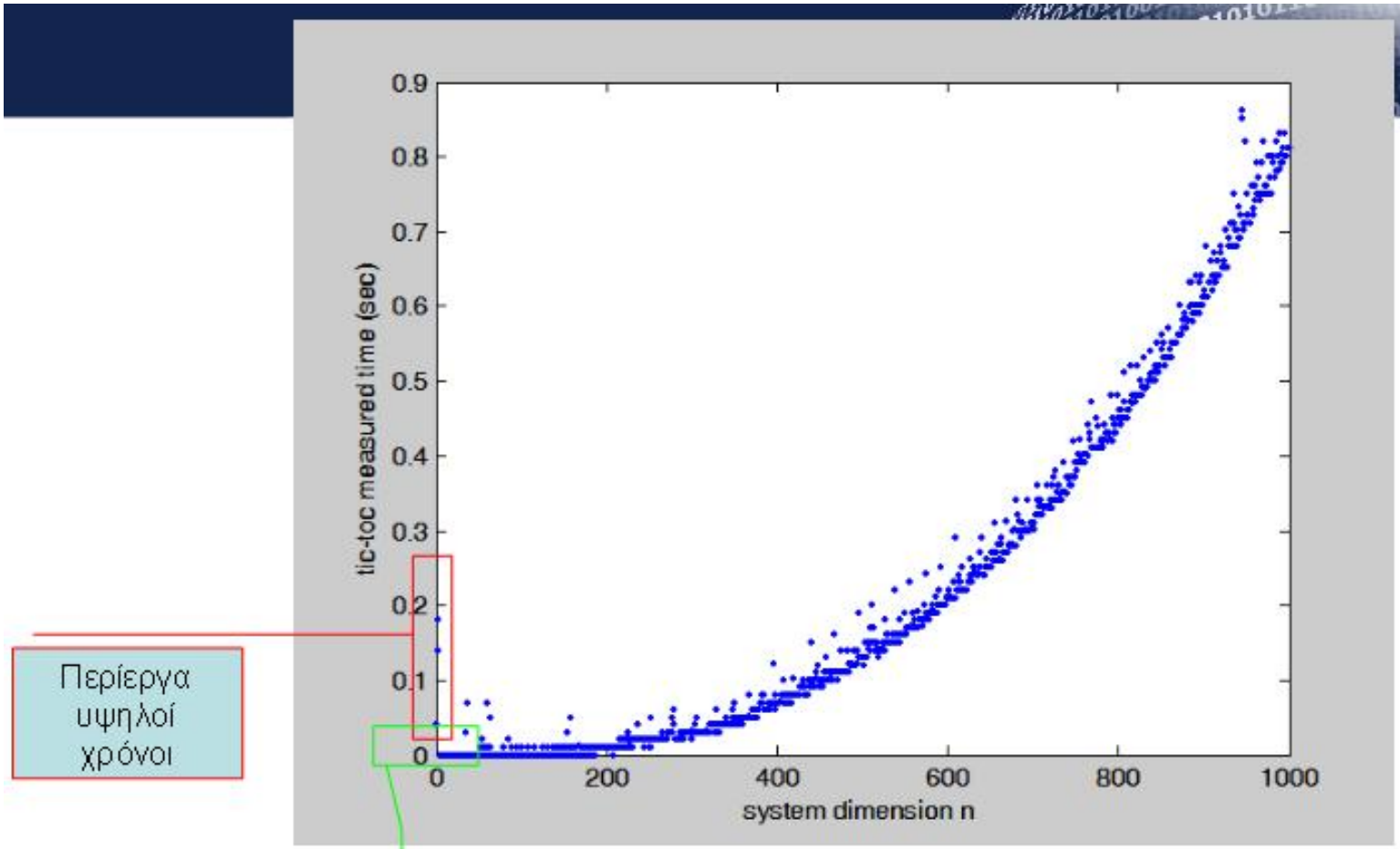


Χρόνος επίλυσης “Ax=b” για διάσταση n με tic-toc σε MATLAB 6.5

..... Παρατηρήστε ότι οι καταγεγραμμένοι χρόνοι είναι πολλαπλάσιοι του 0.01 που επιβεβαιώνει την ευκρίνεια του χρονομετρητή σε d=10 ms.



Χρόνος επίλυσης “Ax=b” για διάσταση n με tic-toc σε MATLAB 7.5
.Παρατηρήστε τις διαφορές με πριν λόγω της μεγαλύτερης διακριτότητας του tic-toc



Οι χρόνοι σε αρκετές περιπτώσεις είναι «απρόσμενοι»

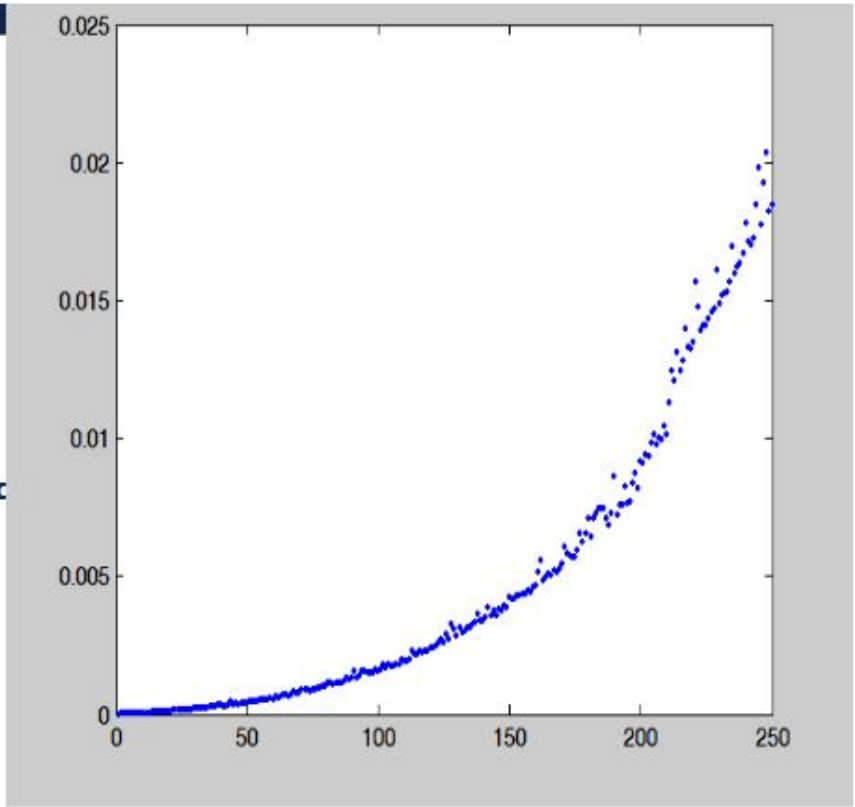
... μηδενικοί

... πολύ μεγάλοι

... μη μονοτονικοί ($n>m \nrightarrow t(n)>t(m)$)

Μετρήσεις

- **βελτίωση αξιοπιστίας:**
  - **Χρονομετρητές μεγαλύτερης ευκρίνειας**
    - **Όχι πάντα διαθέσιμοι**
  - **Χρονομέτρηση υπό ιδανικές συνθήκες:**
    - **π.χ. αποκλειστική χρήση (single user mode) και χωρίς να μεσολαβούν «αναταράξεις», π.χ. Να ξεκινούν ή να καλούνται ασχετα processes, services, ...**
  - **ΑΡΧΕΣ**
    - **«Η πρώτη φορά δεν μετρά»**
    - **«Μια φορά δεν φθάνει»**



Χρόνος επίλυσης “Ax=b” για διάσταση n με tic-toc σε MATLAB 6.5
..... Μετά από 500 επαναλήψεις και κανονικοποίηση

```
for n = 1:250, A = rand(n,n); b = rand(n,1); tic; for j=1:500, x = A\b; end; t(n) = toc; end; plot(t/500,'.')
```

Σχετικά με τη μείωση του θορύβου στις μετρήσεις

- Συνάρτηση της διακριτότητας του χρονομετρητή
- αφαίρεση θορύβου αρχικοποίησης
- επηρεάζει η θέση των στοιχείων σε σχέση με την cache στην έναρξη των επαναλήψεων
- ... μπορεί να χρειαστεί cache flushing για να μην έχουμε ((τεχνητά καλές τιμές))
- Χρειάζονται επαναλήψεις και εξαγωγή στατιστικών στοιχείων από τα δείγματα (ελάχιστη, μέγιστη και μέση τιμή, μέσος όρος, αρμονικός μ.ό.)
- Πόσες επαναλήψεις?
- Για προγράμματα μικρής διάρκειας, μπορεί να χρειάζονται αρκετές επαναλήψεις για αξιόπιστη μέτρηση

Πληροφορίες για `tic` από Mathworks

```
%TIC Start a stopwatch timer.
%TIC and TOC functions work together to measure elapsed time.
%TIC, by itself, saves the current time that TOC uses later
%to measure the time elapsed between the two.
%TSTART = TIC saves the time to an output argument, TSTART.
%The numeric value of TSTART is only useful as
%an input argument for a subsequent call to TOC.
%Ex: min and average time to compute sum of Bessel functions.
REPS = 1000; minTime = Inf; nsum = 10;
tic;
for i=1:REPS
 tstart = tic;
 sum = 0; for j=1:nsum, sum = sum + besselj(j,REPS); end
 telapsed = toc(tstart);
 minTime = min(telapsed,minTime);
end
averageTime = toc/REPS;
```

Θορυβώδης αρχικοποίηση

```
>> a=1;b=2;
>> tic;a+b;toc
Elapsed time is 0.000006 seconds.
>> tic;a+b;toc
Elapsed time is 0.000002 seconds.
>> tic;a+b;toc
Elapsed time is 0.000002 seconds.
>> tic;a+b;toc
Elapsed time is 0.000002 seconds.
```

Διακύμανση μετρήσεων

```
>> a=rand(10,1);b=rand(10,1);
>> c=zeros(10,1);
>> tic;c=a+b;toc
Elapsed time is 0.000023 seconds.
>> tic;c=a+b;toc
Elapsed time is 0.000020 seconds.
>> tic;c=a+b;toc
Elapsed time is 0.000019 seconds.
>> tic;c=a+b;toc
Elapsed time is 0.000023 seconds.
>> tic;c=a+b;toc
Elapsed time is 0.000028 seconds.
>> tic;c=a+b;toc
Elapsed time is 0.000020 seconds.
>> tic;c=a+b;toc
Elapsed time is 0.000021 seconds.
>> tic;c=a+b;toc
Elapsed time is 0.000028 seconds.
>> tic;c=a+b;toc
Elapsed time is 0.000043 seconds.
```

Σχετικά με τη συνάρτηση `timeit.m`

Περιγραφή

T = TIMEIT(F) measures the time (in seconds) required to run F, which is a function handle. TIMEIT handles automatically the usual benchmarking procedures of "warming up" F, figuring out how many times to repeat F in a timing loop, etc. TIMEIT uses a median to form a reasonably robust time estimate. ....

UPDATED 31-Dec-2008: More accurate when timing very fast functions; warns you when the reported time might be affected by time-measurement overhead; calls F fewer times when F takes more than a few seconds to run.

MATLAB release MATLAB 7.5 (R2007b)

Εξέλιξη του timeit.m

MATLAB CENTRAL

Search: Blogs

File Exchange | Answers | Newsgroup | Link Exchange | Blogs | Trendy | Cody | Contest | MathWorks.com

Blogs

Steve on Image Processing

September 30th, 2013

timeit makes it into MATLAB

This is my first blog post with "Published with MATLAB R2013b" at the bottom. The latest MATLAB release shipped earlier in September. And, for the first time in a while, a function that I wrote has made it into MATLAB.

Back in 2008, I spent some time trying to incorporate performance benchmarking lessons I learned from Cleve Moler and Bill McKeeman into a general-purpose function for accurately measuring the "typical" running time of MATLAB functions or expressions. The result was `timeit`, which I submitted to the File Exchange.

Well, I'm happy to say that `timeit` has made it into MATLAB in the R2013b release! The MATLAB development team took the code from the File Exchange, made some minor refinements, did some additional testing, wrote some nice doc, and got it into the release. (Thanks, everyone!)

Here are a couple of examples. The first one is just MATLAB, but the second one requires Image Processing Toolbox. (Note that it's helpful to know a little about anonymous functions in order to use `timeit`.)

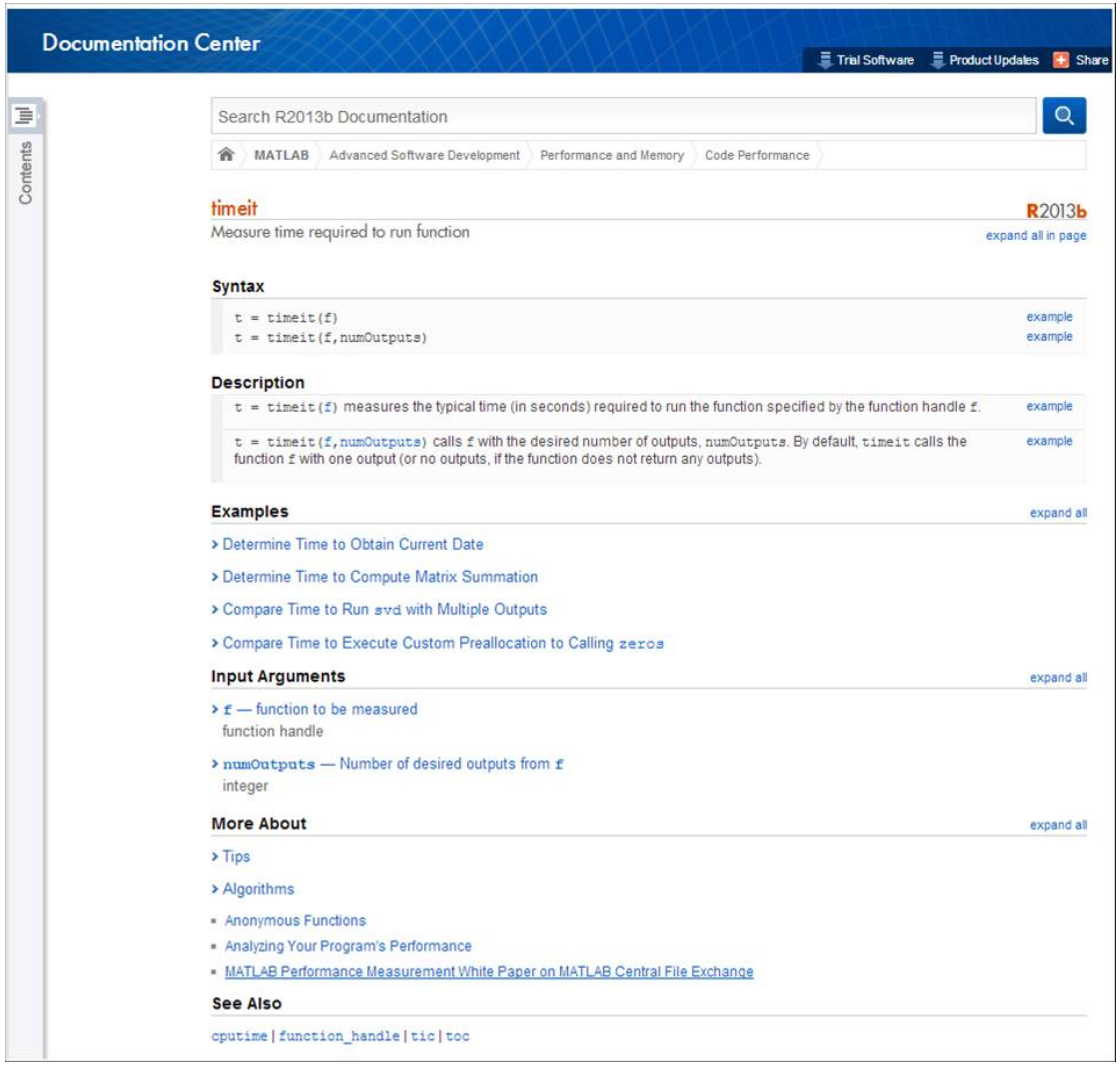
How much time does it take to compute sum(A.' .* B, 1), where A is 12000-by-400 and B is 400-by-12000?

MathWo

About

Steve Eddins is a sof development manage MATLAB and image processing areas at MathWorks. Steve coauthored Digital Im Processing Using MA He writes here about processing concepts, algorithm implementa and MATLAB.

timeit.m επίσημη συνάρτηση του MATLAB R2013b



Documentation Center

Trial Software | Product Updates | Share

Search R2013b Documentation

MATLAB > Advanced Software Development > Performance and Memory > Code Performance

Contents

timeit R2013b

Measure time required to run function

expand all in page

Syntax

```
t = timeit(f)
t = timeit(f,numOutputs)
```

example

example

Description

`t = timeit(f)` measures the typical time (in seconds) required to run the function specified by the function handle `f`. example

`t = timeit(f,numOutputs)` calls `f` with the desired number of outputs, `numOutputs`. By default, `timeit` calls the function `f` with one output (or no outputs, if the function does not return any outputs). example

Examples expand all

> Determine Time to Obtain Current Date

> Determine Time to Compute Matrix Summation

> Compare Time to Run `svd` with Multiple Outputs

> Compare Time to Execute Custom Preallocation to Calling `zeros`

Input Arguments expand all

> `f` — function to be measured
function handle

> `numOutputs` — Number of desired outputs from `f`
integer

More About expand all

> Tips

> Algorithms

- Anonymous Functions
- Analyzing Your Program's Performance
- MATLAB Performance Measurement White Paper on MATLAB Central File Exchange

See Also

`cputime` | `function_handle` | `tic` | `toc`

octave.pdf

`tic ()` [Built-in Function]
`toc ()` [Built-in Function]

Set or check a wall-clock timer. Calling `tic` without an output argument sets the timer. Subsequent calls to `toc` return the number of seconds since the timer was set. For example,

```
tic ();
# many computations later...
elapsed_time = toc ();
```

will set the variable `elapsed_time` to the number of seconds since the most recent call to the function `tic`.

If called with one output argument then this function returns a scalar of type `uint64` and the wall-clock timer is not started.

```
t = tic; sleep (5); (double (tic ()) - double (t)) * 1e-6
     ⇒ 5
```

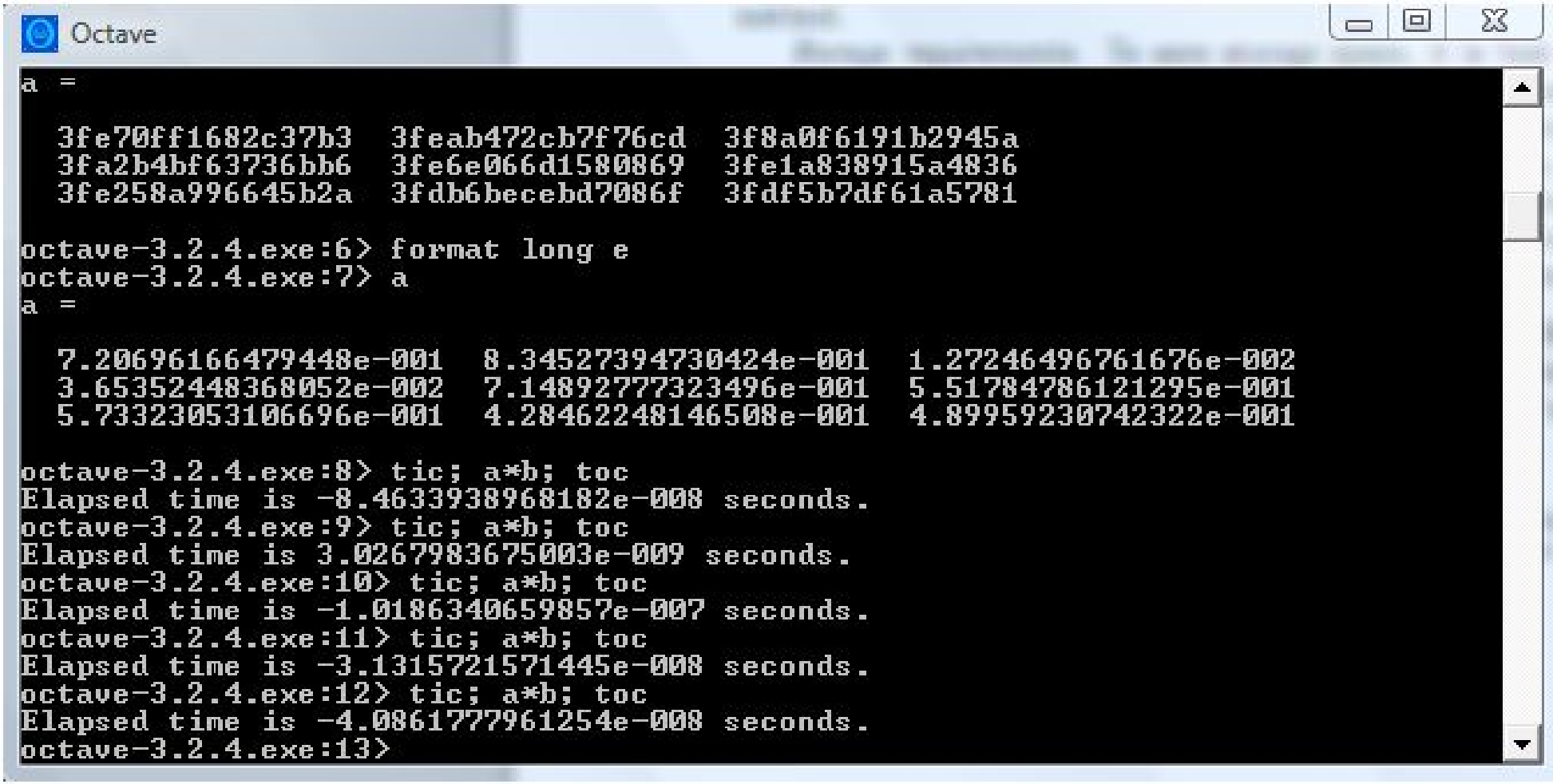
Nested timing with `tic` and `toc` is not supported. Therefore `toc` will always return the elapsed time from the most recent call to `tic`.

If you are more interested in the CPU time that your process used, you should use the `cputime` function instead. The `tic` and `toc` functions report the actual wall clock time that elapsed between the calls. This may include time spent processing other jobs or doing nothing at all. For example,

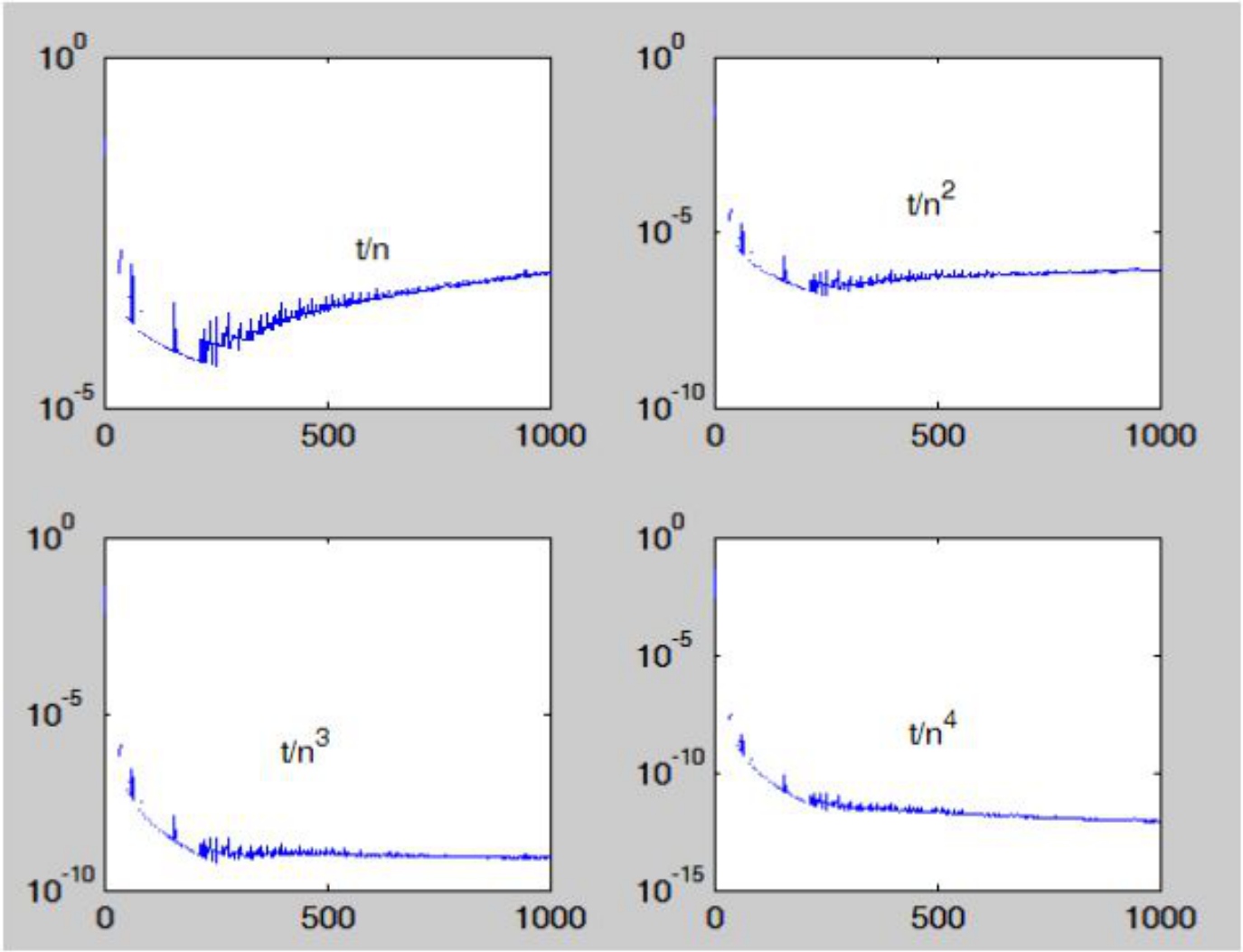
```
tic (); sleep (5); toc ()
     ⇒ 5
t = cputime (); sleep (5); cputime () - t
     ⇒ 0
```

(This example also illustrates that the CPU timer may have a fairly coarse resolution.)

tictoc στην Octave



Χρόνος εκτέλεσης ως συνάρτηση του μεγέθους για πειραματική ένδειξη πολυπλοκότητας?



Δυσκολίες κατά την Intel

> 5. Avoid thinking you know what the performance issues are. This overarching injunction has been a fundamental axiom of performance optimization for years. In a textbook on optimization, it would be first in this list of things to avoid, not the last. However, only after looking at the list of traps posed by today's processors can it be seen how much truer this injunction is now than it has ever been. Processors today are so complex that performance snags can occur in places that even experienced developers would never consider.

When Performance Really Counts: 5 Things to Do, 5 To Avoid: http://www.intel.com/

Δυσκολίες κατά την Intel

> Thank you for sharing your knowledge Dr. Bandwidth, it shows how 'simply' things as the major feature of the leading CPU are deceptively complicated and shrouded in corporative secrecy as if the goal is to prevent the plebs of reaching these speeds.

Comment by blog-user Georgi M. on 10/2014; cf. `https://goo.gl/VbygJj`

Venmugil Elango, Fabrice Rastello, Louis-Noël Pouchet, J Ramanujam, and P Sadayappan.
On Characterizing the Data Access Complexity of Programs.
arXiv.org, November 2014.

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.

S. Williams, A. Waterman, and D. Patterson.
Roofline: An insightful visual performance model for multicore architectures.
Commun. ACM, 52(4):65–76, April 2009.