Επιστημονικός Υπολογισμός

Ε.Γαλλόπουλος

ΤΜΗΥΠ, Π. Πατρών

Διάλεξη 6: 10 Νοεμβρίου 2017

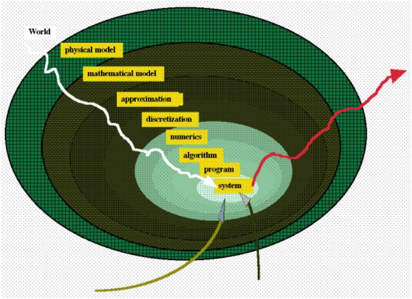
Περιεχόμενο

Υπενθύμιση

Στον ΕΥ:

συνήθως πλοηγούμε μεταξύ του Διακριτού, του Αριθμητικού και του Υπολογιστικού μοντέλου.

Τα μοντέλα βοηθούν σε προβλέψεις ταχύτητας και την ακρίβειας των υπολογισμών,



Αριθμητικό μοντέλο

Attractive mathematics does not protect one from the rigors of digital computation

[J.H. Wilkinson, "von Neumann Lecture", SIAM Meeting, Boston, 1970]

Στόχος: Μελέτη της επίδρασης της πεπερασμένης ακρίβειας στην αναπαράσταση αριθμών και στους υπολογισμούς

- Μοντέλο
- Είδη απώλειας πληροφορίας
- Διάδοση και συσσώρευση σφαλμάτων
- Πρόβλεψη και εκτίμηση σφαλμάτων
- Προσαρμογή υλοποιήσεων για να έχουμε ανεκτό σφάλμα

Δεν είναι και λίγα! Δείτε το άρθρο!

Παραδείγματα - φαινομενικά παράδοξα

Υπενθύμιση

$$
\begin{aligned}
10^{20} - 10 - 10^{20} + 20 &= 20 \\
10^{20} + 20 - 10^{20} - 10 &= -10 \\
-10 + 20 - 10^{20} + 10^{20} &= 0 \\
10^{20} - 10^{20} + 20 - 10 &= 10
\end{aligned}
$$

- Προσέξτε ότι $10^{20} > 2^{52}$ επομένως τα παραπάνω πρέπει να αναμένονται!
- (Ερώτηση:) Από τους 24 (=4!) τρόπους υπολογισμού παραπάνω, ποιοί επιστρέφουν σωστό αποτέλεσμα;

Listing 1: Ατέρμων βρόχος

```
d=0; while (d ~=1.0), d=d+0.1, end;
```

Floating point arithmetic is by nature inexact, and it is not difficult to misuse it so that the computed answers consist almost entirely of noise (D.Knuth, "The Art of Computer Programming", vol. 2).

Προσοχή

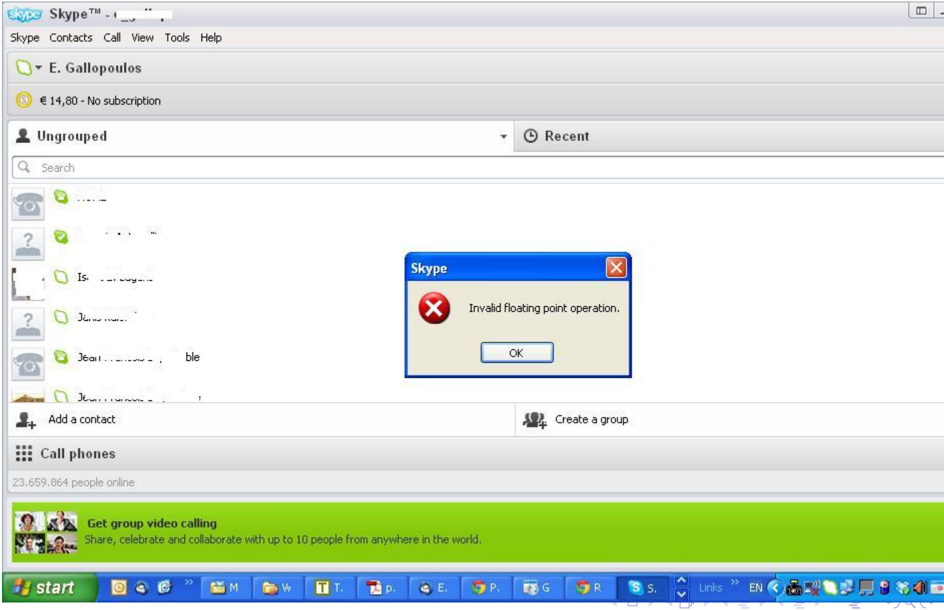
Αντί για τα γνωστά αξιώματα και πράξεις στους πραγματικούς αριθμούς[α΄] εκτελούνται υπολογισμοί που υλοποιούνται σε υλικό και λογισμικό επί κάποιων α.κ.υ. που είναι υποσύνολο των πραγματικών.

[α΄] Θα τα αναφέρουμε ως "θεϊκή αριθμητική".

Φαινόμενα

- στρογγύλευση (στρογγυλοποίηση, rounding)
- υπερχείλιση
- υποχείλιση ή βαθμιαία υποχείλιση

Ενοχλητικές επιπτώσεις !



προβληματικές περιπτώσεις!

An Ensemble Analysis of Forecast Errors Related To Floating Point Performance

Stephen J. Thomas*[1], Joshua P. Hacker[2],
Michel Desgagné[3], and Roland B. Stull[2]
[1]*SCD/NCAR, Boulder, Colorado*
[2]*University of British Columbia, Vancouver, Canada*
[3]*Recherche en prévision numérique, Environment Canada, Dorval, Canada*

Oct 25, 2001

Abstract

The dynamical core of the mesoscale compressible community (MC2) model is described. Ensemble forecast techniques for high resolution mesoscale simulations are applied to assess the impact of floating point optimization, math libraries and processor configuration on forecast accuracy. It is shown that the iterative solver in the dynamical core is most sensitive to processor configuration, but also shows weak sensitivity to the usage of fast math libraries and floating point instruction reordering. Semi-implicit pressure solver errors are amplified in the physical parameterization package, which is sensitive to small pressure differences and feed back to the dynamical solution. In this case, local rms spreads are around 1°C in temperature by the end of a 42 h forecast. We conclude that careful validation is required when changing computing platforms or introducing fast math libraries.

Εκνευριστικές περιπτώσεις!

At least insofar as the accuracy of published results is concerned, applied economics is a "poor relation" to theoretical economics. Theoretical economic results, generally speaking, are sounder than empirical economic results. The reason for this is simple: the process by which the researcher obtained the result is transparent and amenable to verification. Frequently referees check to make sure that theorems are correct and, if the referee has not vouchsafed every part of the article, the interested reader can do so. Not so with empirical economics, where the process of obtaining a result is far from transparent, and myriad details not described in the text can be found only in the code. Empirical economics, by actively discouraging replication, does not incorporate the self-correcting mechanism of the scientific method -- there is no process whereby bad results can be removed from the cumulative body of knowledge.

Previous studies by two of the present authors have found that, left to themselves, many economists (and econometricians) do not understand the difference between *algebraic* calculations and *numerical* calculations: Altman (2003) contains a number of excellent discussions of the issues. Perhaps the most frequent (and egregious) example is calculating the coefficient vector of an ordinary least squares regression. The algebraic formula, $\hat{b} = (X'X)^{-1}X'y$, is well-known to have poor numerical properties (McCullough and Vinod, 1999). It is nonetheless commonplace for GAUSS and Matlab code written by economists to implement the algebraic formula rather than the QR decomposition. But, without access to an author's code, how can a reader know how the article's results were obtained?

Καταστροφικές περιπτώσεις

Patriot missile failure, 1991
(28 deaths because of bad rounding)

Explosion of Ariane 5, 1996
(500M because of overflow)

Sinking of Sleipner A offshore platform, 1991
(700M because of inaccurate fem approximation)

- Vancouver stock exchange
  … and many others

$$$$$ LOSS IN EVALUATING NUMERICAL RELIABILITY

Σημαντικό για την καθημερινότητα (Boe17)!

practice

DOI:10.1145/2911981

Rounding errors are usually avoidable, and sometimes we can afford to avoid them.

BY HANS-J. BOEHM

Small-Data Computing: Correct Calculator Arithmetic

COMPUTERS COMMONLY PERFORM numerical computations using floating point arithmetic,[a] typically representing numbers as specified by the IEEE 754 standard. Numbers are represented as `m x b`^{e}, where `b` is base, `m` is a fixed bit length length fraction (*mantissa*), with an implicit "decimal point" on the left, and `e` is an exponent. For conventional IEEE "double precision" floating point, the base `b` is 2, and the mantissa `m` is 53 bits (approximately 16 decimal digits) long. For a hardware calculator, we might use `b` = 10, with a 12-digit mantissa.[b]

Floating-point representations are used pervasively, from large-scale scientific computing problems down to pocket calculators. They provide a great time-honored compromise between speed of computation and sufficient accuracy to usually provide meaningful results. A 53-bit mantissa can often provide 10 to 15 decimal digits of accuracy in the final result, and modern processors can often perform more than one floating-point operation per cycle.

But conventional floating point arithmetic remains a compromise. The results can be computed quickly, but they are only usually precise enough to be meaningful. Usually the precision loss from rounding to 53 bits is not noticeable, because we are usually computing on measured physical quantities that are far less accurate to start with, and usually well-designed algorithms do not compound these incoming measurement errors too much. But none of those "usually" qualifiers can be dropped, and algorithms are not always well designed.

Most of us are familiar with some of the programming hazards of floating point. We may have observed, for example, that the loop

```
for (x = 0.0; x != 10.0; x += 0.1) { … }
```

usually fails to terminate. But we are willing to deal with these issues, and write more careful code, in order to get high performance numerical computation.

But sometimes performance, at least in the conventional sense, really doesn't matter. Calculators, which normally target expressions with few operations, are probably the canonical example for this category of applications. That is particularly true when the calculator is really an application running on a smartphone with four 2GHz processor cores. This is also an environment in which users are unlikely to think much about formulating algorithms to optimize floating point precision properties.

Even for calculators, the hazards of floating point extend to more than a few digits off in the last place. For example, if we try to compute:

a Goldberg, D. What every computer scientist should know about floatingpoint arithmetic. *ACM Computing Surveys 23*, 1 (1991), 5–48.
b Cochran, D.S. Internal programming of the 9100A Calculator. *HP Journal*, Sept. 1968.

Πώς υπολογίζουμε την ευκλείδεια νόρμα?

Listing 2: Απλή εκδοχή

```
function [s]=norm2_naive(x);
n = length(x);
s = 0;
for i = 1:n, s = s+x(i)^2; end
s = sqrt(s);
% faster version
% s = sqrt(sum(x.^2));
```

Επιθυμητές ιδιότητες συνάρτησης ([Dem97])

1. Να υπολογίζει το αποτέλεσμα με ακρίβεια, δηλ. να είναι ορθά (σχεδόν) όλα τα ψηφία της απάντησης, εκτός αν το $\|x\|_2$ είναι (σχεδόν) εκτός του συνόλου των κανονικοποιημένων α.κ.υ. του συστήματος.
2. Να είναι (σχεδόν) όσο γρήγορο θα ήταν το απλό (αλλά μη αξιόπιστο) πρόγραμμα.
3. Να λειτουργεί αξιόπιστα ακόμα και εκτός αριθμητικής IEEE εκτός αν η θεωρητική τιμή είναι (σχεδόν) μεγαλύτερη του μέγιστου αναπαραστήσιμου αριθμού.

Αστοχίες της απλή εκδοχής

- Αν x=[sqrt(1.7977e+308),sqrt(1.7977e+308)] τότε

$$\mathrm{norm2\_naive(x)} \rightarrow \mathrm{Inf}$$

αντί για

$$1.896150381621835e+154$$

Αστοχίες της απλή εκδοχής

- Αν $x=[\mathrm{sqrt}(1.7977e+308),\mathrm{sqrt}(1.7977e+308)]$ τότε

$$\mathrm{norm2\_naive(x)} \to \mathrm{Inf}$$

  αντί για

$$1.896150381621835e+154$$

- Αν $x=[2.2251e\text{-}308]$ τότε

$$\mathrm{norm2\_naive(x)} \to 0$$

  αντί για

$$2.2251e\text{-}308$$

Αστοχίες της απλή εκδοχής

- Αν x=[sqrt(1.7977e+308),sqrt(1.7977e+308)] τότε

$$\text{norm2\_naive(x)} \rightarrow \text{Inf}$$

αντί για

$$1.896150381621835\text{e}+154$$

- Αν x=[2.2251e-308] τότε

$$\text{norm2\_naive(x)} \rightarrow 0$$

αντί για

$$2.2251\text{e-}308$$

ΠΡΟΣΟΧΗ: realmax = 1.7977e+308; realmin = 2.2251e-308. Αυτές οι τιμές δεν είναι οριακές για τη συνάρτηση!

Πώς μπορούμε να αποφύγουμε τις αστοχίες!

Παράδειγμα vnorm.m (MATLAB File Exchange)

File Exchange

from **Vector norm** by Winston Smith
Returns the vector norm for a specified dimension (e.g. row/col) of a matrix

vnorm(A,varargin)

```
function y = vnorm(A,varargin)
% VNORM - Return the vector norm along specified dimension of A
%
%   VNORM(A) returns the 2-norm along the first non-singleton
%   dimension of A
%   VNORM(A,dim) return the 2-norm along the dimension 'dim'
%   VNORM(A,dim,normtype) returns the norm specified by normtype
%   along the dimension 'dim'
%   VNORM(A,[],normtype) returns the norm specified by normtype along
%   the first non-singleton dimension of A
%
%   normtype may be one of {inf,-inf,positive integer}.
%   For a given vector, v, these norms are defined as
```

Παράδειγμα από vnorm.m (MATLAB File Exchange)

πιθανή υπερχείλιση

```
    end
end


if isempty(ntype)
    y = sqrt(sum( abs(A).^2 , dim) );
elseif ntype==1
    y = sum( abs(A) , dim );
elseif isinf(ntype)
    if ntype > 0
        y=max(abs(A), [], dim);
    else
        y=min(abs(A), [], dim);
    end
elseif ntype~=floor(ntype) || ntype<1
    error(['Norm type must be one of inf,-inf or a positive ' ...
           'integer']);
else
    y = (sum( abs(A).^ntype , dim) ).^(1/ntype);
end
```

Τροποποίηση

Ιδέα

$$\|x\|_2 = \xi_{max} \sqrt{\sum_{i=1}^{n} \underbrace{(\frac{\xi_i}{\xi_{max}})^2}_{\leq 1}}, \text{ όπου } \xi_{max} = \max(|x|)$$

```
function [s] = norm_2rat(x); % author: EG
n = length(x); s = 0; xmax = max(abs(x));
if (xmax==0), return; end
for i = 1:n, s = s+(x(i)/xmax)^2; end
s = xmax*sqrt(s);
```

Θεραπεία?

1. norm2_rat([sqrt(realmax),sqrt(realmax)]) = 1.8962e+154

ΠΡΟΣΟΧΗ εύρεση μεγίστου $\rightarrow$ 2 περάσματα από τα δεδομένα

Τροποποίηση

Ιδέα

$$\|x\|_2 = \xi_{\max} \sqrt{\sum_{i=1}^{n} \underbrace{(\frac{\xi_i}{\xi_{\max}})^2}_{\leq 1}}, \text{ όπου } \xi_{\max} = \max(|x|)$$

```
function [s] = norm_2rat(x); % author: EG
n = length(x); s = 0; xmax = max(abs(x));
if (xmax==0), return; end
for i = 1:n, s = s+(x(i)/xmax)^2; end
s = xmax*sqrt(s);
```

Θεραπεία?

1. norm2_rat([sqrt(realmax),sqrt(realmax)])= 1.8962e+154
2. norm2_rat(realmin)= 2.2251e-308

ΠΡΟΣΟΧΗ εύρεση μεγίστου → 2 περάσματα από τα δεδομένα

```
      DOUBLE PRECISION FUNCTION DNRM2 ( N, X, INCX )
      INTEGER                           INCX, N
      DOUBLE PRECISION                  X( * )
#   -- This version written on 25-October-1982. Modified on ...
    14-October-1993 Sven Hammarling, Nag Ltd.

      DOUBLE PRECISION        ONE           , ZERO
      PARAMETER             ( ONE = 1.0D+0, ZERO = 0.0D+0 )
      INTEGER                 IX
      DOUBLE PRECISION        ABSXI, NORM, SCALE, SSQ
      INTRINSIC               ABS, SQRT
#       .. Executable Statements ..
      IF( N<1 || INCX<1 )THEN
         NORM  = ZERO
      ELSE IF( N==1 )THEN
         NORM  = ABS( X( 1 ) )
      ELSE
         SCALE = ZERO
```

Κώδικας αναφοράς (Fortran)

```
        SSQ    = ONE
        DO 10, IX = 1, 1 + ( N - 1 )*INCX, INCX
           IF( X( IX ) !=ZERO )THEN
              ABSXI = ABS( X( IX ) )
              IF( SCALE<ABSXI )THEN
                 SSQ   = ONE   + SSQ*( SCALE/ABSXI )**2
                 SCALE = ABSXI
              ELSE
                 SSQ   = SSQ   +     ( ABSXI/SCALE )**2
              END IF
           END IF
   10   CONTINUE
        NORM  = SCALE * SQRT( SSQ )
     END IF
     DNRM2 = NORM
     RETURN
     END
```

Εκδοχή MATLAB

```
function s = dnrm2(n,x,incx) %MATLAB BLAS-1 by J.Burkardt
  if ( n < 1 | incx < 1 ), s = 0.0; % value = 0.0; ...
     /*correction by EG*/
  elseif ( n == 1 ), s = abs(x(1)); %value = abs ...
     (x(1));/*correction by EG*/
  else scale = 0.0; ssq = 1.0;
    for ix = 1 : incx : 1 + ( n - 1 )*incx
      if ( x(ix) ~= 0.0 )
        absxi = abs ( x(ix) );
        if ( scale < absxi )
          ssq = 1.0 + ssq * ( scale / absxi )^2;
          scale = absxi;
        else
          ssq = ssq + ( absxi / scale )^2;
        end
      end
    end
    s  = scale * sqrt( ssq );
  end
```

Σχόλια

- Έξυπνος τρόπος : υπολογίζει κάθε φορά αν το νέο στοιχείο είναι μεγαλύτερο ή μικρότερο του μέχρι τώρα μεγίστου και ανάλογα προσαρμόζει τον υπολογισμό.
- Προσαρμογή:
- ένα πέρασμα από τα δεδομένα.
- ΠΩΣ? Διαβάστε τον κώδικα και επιβεβαιώστε !

Σχόλια

- Έξυπνος τρόπος: υπολογίζει κάθε φορά αν το νέο στοιχείο είναι μεγαλύτερο ή μικρότερο του μέχρι τώρα μεγίστου και ανάλογα προσαρμόζει τον υπολογισμό.
- Προσαρμογή:
- ένα πέρασμα από τα δεδομένα.
- ΠΩΣ? Διαβάστε τον κώδικα και επιβεβαιώστε!
- Αστοχία: `dnrm2([1,Inf]) = NaN`

Περισσότερες ((απρόσμενες)) ιστορίες

```
>> floor(0.075/0.025)
ans = 2
>> floor(0.75/0.25)
ans = 3
% observation due to C. Bekas
```

Περισσότερες ((απρόσμενες)) ιστορίες

```
>> floor(0.075/0.025)
ans = 2
>> floor(0.75/0.25)
ans = 3
% observation due to C. Bekas
```

```
double v = 1E308;
double x = (v * v) / v;
printf("%g %d\n", x, x==v);
```

Προσοχή (Mon08)

- με `gcc .0.1` σε Linux εκτυπώνει 10^{308}.
- με την επιλογή `-ffloat-store` εκτυπώνει $+\infty$.

Τι λένε οι κατασκευαστές

Των Corden & Kreitzer, Intel Consistency of Floating-Point Results using the Intel Compiler or

Why doesn't my application always give the same answer?

Binary floating-point [FP] representations of most real numbers are inexact, and there is an inherent uncertainty in the result of most calculations involving floating-point numbers. Programmers of floating-point applications typically have the following objectives:

- Accuracy
  - Produce results that are "close" to the result of the exact calculation
    - Usually measured in fractional error, or sometimes "units in the last place" (ulp).
- Reproducibility
  - Produce consistent results:
    - From one run to the next;
    - From one set of build options to another;
    - From one compiler to another
    - From one processor or operating system to another
- Performance
  - Produce an application that runs as fast as possible

These objectives usually conflict! However, good programming practices and judicious use of compiler options allow you to control the tradeoffs.

Παράγοντες δυσκολίας

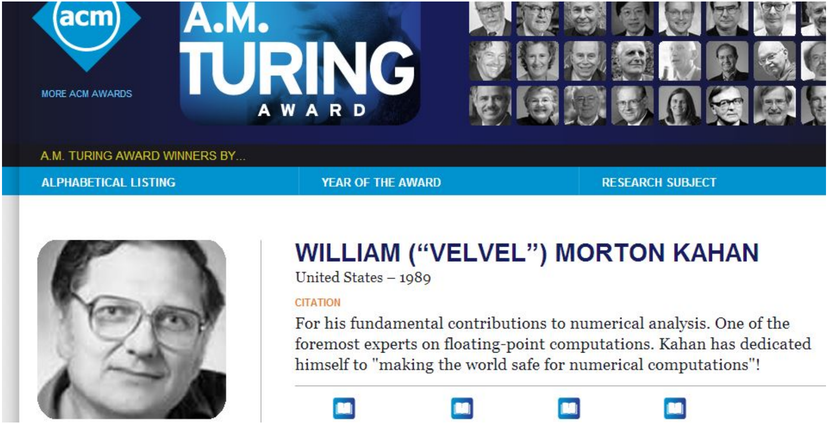
- Ακόμα και φαινομενικά αξιόπιστα προγράμματα χρειάζονται προσοχή ως προς την ορθότητα.

Παράγοντες δυσκολίας

- Ακόμα και φαινομενικά αξιόπιστα προγράμματα χρειάζονται προσοχή ως προς την ορθότητα.
- <u>ΠΡΟΣΟΧΗ</u>: Το ((σκηνικό)) περιλαμβάνει συνδυαστικά
  1. την αρχιτεκτονική (CPU, καταχωρητές και cache, μικροεντολές)
  2. το λογισμικό (γλώσσα και μεταφραστής, περιβάλλον χρόνου εκτέλεσης, αριθμητικές βιβλιοθήκες, πρόγραμμα)
  3. τον αλγόριθμο και την υλοποίησή του σε πρόγραμμα

<u>Παρατήρηση</u>: Η συγγραφή (ακόμα και) απλού αξιόπιστου κώδικα που είναι ταχύς και ακριβής είναι πολύπλοκη υπόθεση!

Πλαίσιο


acm
MORE ACM AWARDS
A.M. TURING AWARD
A.M. TURING AWARD WINNERS BY...
ALPHABETICAL LISTING
YEAR OF THE AWARD
RESEARCH SUBJECT
WILLIAM ("VELVEL") MORTON KAHAN
United States – 1989
CITATION
For his fundamental contributions to numerical analysis. One of the foremost experts on floating-point computations. Kahan has dedicated himself to "making the world safe for numerical computations"!

Πλαίσιο

FLOATING-POINT ARITHMETIC

AT THE MERCY OF

COMPILER WRITERS.

W. Kahan
Univ. of Calif.
Berkeley

FLOATING-POINT ARITHMETIC

SHORTCUTS

for Hardware Designers

TO AVOID.

W. Kahan
Univ. of Calif.
Berkeley

Mathematics Written in Sand　　　　Version of 22 Nov. 1983

MATHEMATICS WRITTEN IN SAND -
the hp-15C, Intel 8087, etc.

W. Kahan,
University of California @ Berkeley

This paper was presented at the Joint Statistical Meeting of the American Statistical Association with ENAR, WNAR, IMS and SSC held in Toronto, Canada, August 15-18, 1983. Then the paper appeared in pp. 12-26 of the 1983 Statistical Computing Section of the Proceedings of the American Statistical Association. It had been typeset on an IBM PC and printed on an EPSON FX-80 at draft speed with an unreadable type-font of the author's devising, and then photo-reduced. The paper is reproduced here unaltered but for type fonts, pagination, and an appended Contents page.

ABSTRACT: Simplicity is a Virtue; yet we continue to cram ever more complicated circuits ever more densely into silicon chips, hoping all the while that their internal complexity will promote simplicity of use. This paper exhibits how well that hope has been fulfilled by several inexpensive devices widely used nowadays for numerical computation. One of them is the Hewlett-Packard hp-15C programmable shirt-

Αριθμοί κινητής υποδιαστολής (υπενθύμιση)

Τι είναι Ψηφιακή αναπαράσταση πραγματικών αριθμών με πεπερασμένο πλήθος ψηφίων (π.χ. 32 ή 64) ως προς κάποια βάση β (συνήθως 2) στα οποία αποθηκεύονται πρόσημο, εκθέτης και ουρά. Επειδη ο εκθέτης δεν είναι σταθερός, δίνεται η δυνατότητα αναπαράστασης μεγάλου εύρους τιμών.

Τριμερής κώδικας για κάθε αριθμό:

1. πρόσημο s (0 αν θετικός, 1 αν αρνητικός)
2. εκθέτης $e = (a_1 a_2 \ldots a_k)_2$ (πολωμένος κατά P)
3. ουρά $(b_0 b_1 b_2 \ldots b_{t-1})_2$

Τότε

$$x = (-1)^s \times 2^{e-P} \times \left(b_0 + b_1 \beta^{-1} + \cdots + b_{t-1} \beta^{-(t-1)}\right)$$

Συμβολίζουμε το σύστημα των α.κ.υ. ως $\mathcal{F}(\beta, t, e_{\min}, e_{\max})$

Οι συνηθισμένοι α.κ.υ. είναι πάντα ρητοί!

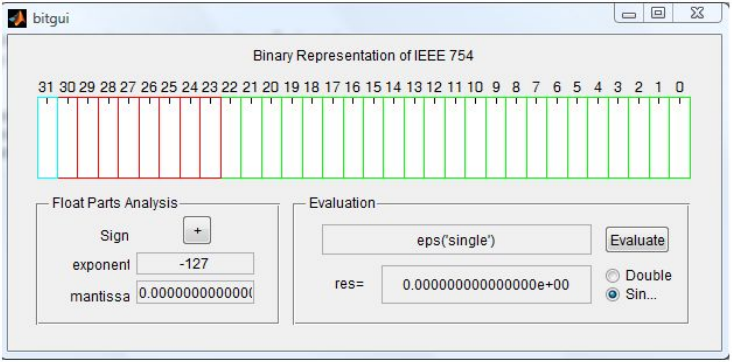
Κανονικοποίηση

- Για να αποφύγουμε την πολλαπλή αναπαράσταση του ίδιου αριθμού που επιτρέπει η ((Επιστημονική Γραφή Αριθμών)), $a \times 10^e$,
- π.χ. .. ότι το 350 μπορεί να γραφεί ως 3.5×10^2, ή 35×10^1, ή 350×10^0, ...
- χρησιμοποιείται κανονικοποιημένη (normalized) αναπαράσταση: π.χ. επιλέγεται $1 \leq |a| < 10$.
- Με βάση 2, υποχρεώνουμε την ουρά a να είναι $1 \leq |a| < 2$. Τότε με βάση τα προηγούμενα

$$x = (-1)^s \times 2^{e-P} \times \left(1 + b_1 2^{-1} + \cdots + b_{t-1} 2^{-(t-1)}\right)$$

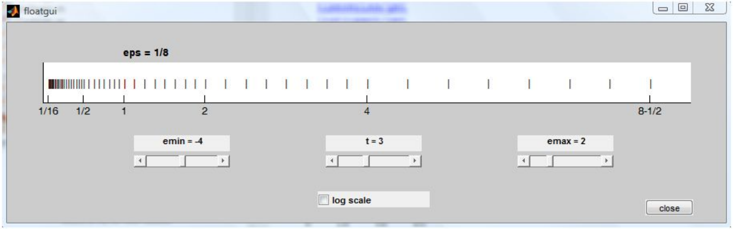
- Αν χρησιμοποιούμε κανονικοποιημένη αναπαράσταση, το μέγεθος του εκθέτη e καθορίζει άμεσα την τάξη μεγέθους του αριθμού.

Το "δικό μας" bitgui



Σχήμα: Το bitgui.m που αναπτύχθηκε στα πλαίσια του ΕΥ1 από τον Γ. Καλοφωλιά (Saarbrucken).

Κατανομή α.κ.υ.



Σχήμα: Οι α.κ.υ. είναι ανισοκατανεμημένοι στον άξονα των πραγματικών. Το παραπάνω προέρχεται από τη συνάρτηση flotgui.m που περιέχονται στη βιβλιοθήκη NCM.

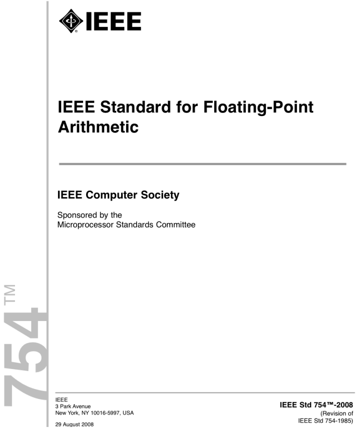
- Η απόσταση διαδοχικών α.κ.υ. που έχουν τον ίδιο εκθέτη, π.χ. $d(e) = (m+1)2^e - m2^e$, είναι σταθερή.
- Η απόσταση μεταξύ διαδοχικών α.κ.υ. διπλασιάζεται κάθε φορά που ο εκθέτης αυξάνει κατά 1: $(m+1)2^{e+1} - m2^{e+1} = d(e+1) = 2d(e)$

Ευχή του 1984

> *... The simplest and best, though harder to attain, solution to the problem of environmental parameters is to* ***standardize floating-point hardware****, so that the values of the parameters become universal constants. (W. Miller,* The Engineering of Numerical Software*, 1984.)*

Πραγματοποίηση το 1985

Μια από τις μεγαλύτερες επιτυχίες στην επιστήμη και τεχνολογία των υπολογιστών ήταν η υιοθέτηση του προτύπου IEEE για την α.κ.υ.



IEEE

IEEE Standard for Floating-Point Arithmetic

IEEE Computer Society

Sponsored by the
Microprocessor Standards Committee

754™

IEEE
3 Park Avenue
New York, NY 10016-5997, USA

29 August 2008

IEEE Std 754™-2008
(Revision of
IEEE Std 754-1985)

Χαρακτηριστικά προτύπου IEEE 754

είδη αριθμών: πεπερασμένα σύνολα δυαδικών και δεκαδικών α.κ.υ. Συμπεριλαμβάνονται ((προσημασμένο μηδενικό)), ((προσημασμένο άπειρο)), ((υποκανονικοποιημένοι αριθμοί), και η ((τιμή)) not a number (NaN) για ((αόριστα αποτελέσματα)).

αλγόριθμοι στρογγύλευσης: μέθοδοι για την στρογγύλευση αριθμών κατά τις αριθμητικές πράξεις και τις μετατροπές.

πράξεις: αριθμητικές και άλλες πράξεις σε αριθμητικά δεδομένα

διαχείριση εξαιρέσεων: επισήμανση ιδιαίτερων καταστάσεων (διαίρεση με 0, υπερχείλιση, κ.λπ.)

συστάσεις: για διαχείριση εξαιρέσεων, υπολογισμό εκφράσεων, υπολογισμό ιδιαίτερων συναρτήσεων (π.χ. τριγωνομετρικών), κ.λπ.

format μετατροπών: δυαδικές κωδικοποιήσεις για τη διευκόλυνση μεταφορών α.κ.υ.

Χρήσιμες αναφορές: [Mon08], [M+10] [Gol91]

Είδη α.κ.υ. στο πρότυπο IEEE-754

	single	single-ext	double	double-ext	quad-precision
μήκος α.κ.υ.	32	≥ 43	64	80	128
e_{max}	+127	1023	+ 1023	+16383	+ 16383
e_{min}	-126	1022	-1022	-16382	-16382
πόλωση	+127	+1023	+1023	+16383	+16383
bits m t	24	≥ 32	53	≥ 64	113
bits s	1	1	1	1	1
bits e	8	11	11	15	15

Βαθμιαία υποχείλιση και υποκανονικοποιημένοι αριθμοί

Το πρότυπο IEEE επιτρέπει το αποτέλεσμα πράξεων μεταξύ α.κ.υ. να είναι α.κ.υ. που είναι μικρότεροι του ελάχιστου κανονικοποιημένου.

Υποκανονικοποιημένοι αριθμοί (subnormal numbers) Τότε το (κρυφό bit) είναι 0. Η κωδικοποίηση αυτών των αριθμών έχει 0 σε όλες τις θέσεις του εκθέτη.

Έτσι αξιοποιούνται όλα τα bits μετά την υποδιαστολή της ουράς όταν η απόλυτη τιμή τοου αποτέλεσματος της πράξης είναι μικρότερη του $2^{e_{\min}}$.

Ελάχιστα

κανονικοποιημένος $2^{e_{\min}}$, π.χ. $2.2251\text{e-}308$ σε διπλή ακρίβεια IEEE

υποκανονικοποιημένος $2^{e_{\min}-t+1}$, π.χ. $4.9407\text{e-}324$ σε διπλή ακρίβεια IEEE. Διαίρεση με όποιο $y > 1$ επιστρέφει 0.

Στη MATLAB

Listing 3: Παραδείγματα (για οικονομία έχουμε αφαιρέσει το `ans`)

```
format hex;
realmin = 0010000000000000
realmin/2 = 0008000000000000  %               ()
realmin/2^52 =  0000000000000001
realmin/2^53 =  0000000000000000
```

Ειδικές τιμές και σύμβολα

(Πέραν των μη κανονικοποιημένων αριθμών) το πρότυπο της IEEE προβλέπει την αναπαράσταση των παρακάτω ειδικών τιμών. Οι τιμές αυτές ανήκουν στο σύνολο F και μπορούμε να κάνουμε πράξεις με αυτές.

-0 υπάρχει επειδή η ουρά έχει αναπαράσταση σεσημασμένου προσήμου

$\pm\infty$ όπως και το σύστημα των πραγματικών αριθμών, είναι ανάγκη να επαυξήσουμε με σύμβολα για το + και - άπειρο

$\mathtt{NaN}$ Not a Number χρησιμοποιείται για την αναπαράσταση οποιουδήποτε αόριστου αποτελέσματος, όπως $0/0, \infty \times 0$.

Εξαιρέσεις

Εξαίρεση	Παράδειγμα	Αποτέλεσμα
invalid op	$0/0, 0 \times \mathbf{Inf}$	`NaN`
overflow		$\pm$`Inf`, $\pm$`realmax`
divide by 0		$\pm$`Inf`
underflow		± 0, $\pm$`realmin`, subnormal
inexact	$\mathrm{fl}(x \oplus y) \neq x \oplus y$	στρογγύλευση

όπου `realmin`, `realmax` είναι ο ελάχιστος κανονικοποιημένος και ο μέγιστος α.κ.υ. αντίστοιχα για τν υπό συζήτηση δεδομένη αναπαράσταση.

Προσοχή $\mathrm{NaN} == \mathrm{NaN} \rightarrow 0$

Κωδικοποίηση για ΜΟΝΗ ακρίβεια στο πρότυπο IEEE

$\pm$	$a_1 a_2 \ldots a_8$	$b_1 b_2 \ldots b_{23}$

Αν ο εκθέτης είναι	τότε η τιμή είναι
$(00000000)_2 = (0)_{10}$	$\pm(0.b_1 b_2 \ldots b_{23})_2 \times 2^{-126}$ (υποκανονικοποιημένος)
$(00000000)_2 = (0)_{10}$	$\pm(0.0\ldots0)_2 \times 2^{-126} = \pm 0$
$(00000001)_2 = (1)_{10}$	$\pm(1.b_1 b_2 \ldots b_{23})_2 \times 2^{-126}$
$(00000010)_2 = (2)_{10}$	$\pm(1.b_1 b_2 \ldots b_{23})_2 \times 2^{-125}$
$\vdots$	$\vdots$
$(01111111)_2 = (127)_{10}$	$\pm(1.b_1 b_2 \ldots b_{23})_2 \times 2^{0}$
$(10000000)_2 = (128)_{10}$	$\pm(1.b_1 b_2 \ldots b_{23})_2 \times 2^{1}$
$\vdots$	$\vdots$
$(11111110)_2 = (254)_{10}$	$\pm(1.b_1 b_2 \ldots b_{23})_2 \times 2^{127}$
$(11111111)_2 = (255)_{10}$	$\pm\infty$ αν $b_1 = \cdots = b_{23} = 0$, αλλιώς $\mathtt{NaN}$

Στρογγύλευση

Έστω F το σύνολο των α.κ.υ. που αναπαρίστανται στο $\mathcal{F}$. Σημαντικό ρόλο παίζει η συνάρτηση στρογγύλευσης

$$\mathrm{fl} : \mathbb{R} \rightarrow F$$

που για κάθε $z \in \mathbb{R}$ επιστρέφει κάποια τιμή $\mathrm{fl}(z) \in F$ σύμφωνα με κάποια προσυμφωνημένη μέθοδο στρογγύλευσης για την οποία πρέπει να ισχύουν οι εξής ιδιότητες:

- Αν $z \in F \Rightarrow \mathrm{fl}(z) = z$.
- Αν $z_1 \leq z_2$ τότε $\mathrm{fl}(z_1) \leq \mathrm{fl}(z_2)$.

Περιπτώσεις

- Το z είναι δεδομένο που θα αποτελέσει είσοδο στο πρόγραμμα.
- Το z είναι αποτέλεσμα πράξης μεταξύ δύο α.κ.υ.

Επίσης έστω ότι $G \supset F$

$$G := \{x \in \mathbb{R} \ : m \leq |x| \leq M\} \cup \{0\} \subset \mathbb{R}\,,$$

όπου $m \in F$ είναι ο ελάχιστος μη μηδενικός και $M \in F$ ο μέγιστος θετικός (κανονικοποιημένος) α.κ.υ. στο $\mathcal{F}$. Τότε ισχύει ένα από τα εξής:

- $z \in F$ άρα $\mathrm{fl}(z) = z$.
- $z \in G$ και $z \notin F$ άρα $\mathrm{fl}(z) \neq z$ (στρογγύλευση, η διαφορά $|\mathrm{fl}(z) - z|$ είναι το απόλυτο σφάλμα στρογγύλευσης).
- $z \notin G$ και $|\mathrm{fl}(z)| > M$ υπερχείλιση, το αποτέλεσμα $\in (-\infty, -M, M, \infty)$, εξαρτάται από τη μέθοδο στρογγύλευσης.
- $z \notin G$ και $|\mathrm{fl}(z)| < m$ υποχείλιση, $0 < |\mathrm{fl}(z)| < m$. Το πρότυπο IEEE περιέχει υποκανονικοποιημένους αριθμούς και υποστηρίζεται η βαθμιαία υποχείλιση.

> *Round-to-nearest mode:* In this mode, the representable value nearest to the infinitely precise result shall be delivered; if the two nearest representable values are equally near, the one with its least significant bit equal to zero shall be delivered.

Στις στρατηγικές στρογγύλευσης ενός $z \notin F$, έχουν σημασία ο πλησιέστερος α.κ.υ. μικρότερος του z και ο πλησιέστερος α.κ.υ. μεγαλύτερος του z. Έστω ότι τους συμβολίζουμε ως $z_-, z_+ \in F$ αντίστοιχα.

Στρογγύλευση προς το πλησιέστερο άρτιο

Θέτουμε $\mathrm{fl}(z) = \arg\min_{y\in\{z_-,z_+\}} |y - z|$ και αν $z - z_- = z_+ - y$ τότε θέτουμε $\mathrm{fl}(z)$ το ένα από τα z_-, z_+ που έχει στην ουρά του το 0 ως τελευταίο bit.

Συμβολισμός Το σύμβολο $\arg\min_{x\in X} f(x)$[1] είναι η τιμή ή το σύνολο τιμών $x \in X$ στις οποίες ελαχιστοποιείται η $f(x)$. Π.χ. αν $f(x) = x^2 + 1$ τότε $\min_{x\in\mathbb{R}} f(x) = 1$ ενώ $\arg\min_{x\in X} f(x) = 0$.

[1] Προέρχεται από το argument of the mininimum και προφέρεται (άργκ-μίν).

Υπάρχουν άλλοι τρόποι στρογγύλευσης?

<u>ΒΕΒΑΙΩΣ:</u> στο πρότυπο IEEE προβλέπονται 5 τρόποι που είναι χρήσιμοι σε ορισμένες περιπτώσεις[2]

1. Προς τον πλησιέστερο, ισοπαλίες προς ζυγό (default)
2. Προς τον πλησιέστερο, ισοπαλίες μακρύτερα από το 0
3. Προς το 0 (αποκοπή)
4. Προς το ∞
5. Προς το $-\infty$

Δεν προσφέρεται πάντα εύκολη λογισμική υποστήριξη. Στη MATLAB υπάρχει τρόπος αλλά είναι ((κρυμμένος)).

Listing 4: Και όμως - η ((μυστική)) εντολή feature

```
>> feature('setround',n) % SET n TO 0.5 (round to nearest ...
    even), 0, + OR - Inf (round to ....)
```

[2] π.χ. στην ανάλυση διαστημάτων - δείτε επόμενη διάλεξη.

Ορισμός

Το έψιλον της μηχανής είναι η απόσταση του 1.0 από τον αμέσως μεγαλύτερο α.κ.υ.

Σημ. Στην IEEE-754: single precision $\epsilon_M \approx 1.1921e-007$; double precision $\epsilon_M \approx 2.2204e-016$.
Στη MATLAB δείτε τις εντολές `eps` και `eps('single')`.

Ενδιαφέροντα φαινόμενα:

Είσοδος	`format short.`	`format hex`
$1+\text{eps}/2$	1	3ff0000000000000
$1+\text{eps}$	1.0000	3ff0000000000001
$1+\text{eps}/2+\text{eps}$	1.0000	3ff0000000000001
$1+\text{eps}+\text{eps}/2$	1.0000	3ff0000000000002
$1+2*\text{eps}$	1.0000	3ff0000000000002
$1+2*\text{eps}+\text{eps}/2+\text{eps}/4$	1.0000	3ff0000000000002

Ορισμός

Το έψιλον της μηχανής είναι η απόσταση του 1.0 από τον αμέσως μεγαλύτερο α.κ.υ.

Σημ. Στην IEEE-754: single precision $\epsilon_M \approx 1.1921e-007$; double precision $\epsilon_M \approx 2.2204e-016$.
Στη MATLAB δείτε τις εντολές `eps` και `eps('single')`.
Ενδιαφέροντα φαινόμενα: feature('setround', Inf)

Είσοδος	`format short.`	`format hex`
$1+\mathtt{eps}/2$	1.0000	3ff0000000000001
$1+\mathtt{eps}$	1.0000	3ff0000000000001
$1+\mathtt{eps}/2+\mathtt{eps}$	1.0000	3ff0000000000002
$1+\mathtt{eps}+\mathtt{eps}/2$	1.0000	3ff0000000000002
$1+2*\mathtt{eps}$	1.0000	3ff0000000000002
$1+2*\mathtt{eps}+\mathtt{eps}/2+\mathtt{eps}/4$	1.0000	3ff0000000000004

Παρατηρήσεις

Για τις μετρήσεις σφαλμάτων βολεύει να τα εκφράζουμε ως πολλαπλάσια κάποιας μονάδας μέτρησης του σφάλματος στρογγύλευσης.

Μονάδα στρογγύλευσης (unit roundoff)

Η μονάδα στρογγύλευσης είναι το μέγιστο δυνατό σχετικό σφάλμα για την επιλεγμένη στρατηγική στρογγύλευσης.

Παράδειγμα: Όταν χρησιμοποιείται στρογγύλευση προς το πλησιέστερο, τότε είναι $\mathbf{u} := \max_{z\neq 0} \frac{|z-\mathrm{fl}(z)|}{|z|} = \frac{\beta^{1-t}}{2}$.

IEEE double: $\mathbf{u} = 1.1102e-016$; IEEE single: $\mathbf{u} = 5.9605e-008$.

Θεμελιώδης αρχή

Αρχή ακριβούς στρογγύλευσης (ΑΑΣ)

Αν $\tilde{\odot}$ είναι η υλοποίηση της αριθμητικής πράξης $\odot$, τότε αν $x, y \in F$ ισχύει ότι $x \tilde{\odot} y = \mathrm{fl}(x \odot y) \in F$.

- Το υπολογισμένο αποτέλεσμα είναι ακριβώς ίδιο με το να εκτελούνταν η πράξη με ((θεϊκή)) αριθμητική και μετά να εφαρμοζόταν στρογγύλευση.
- Η υλοποίηση δεν είναι απλή! Πέρασαν δεκαετίες μέχρι να βρεθεί οικονομικός και ορθός τρόπος υλοποίησης και τους κατασκευαστές να την υιοθετήσουν.
- Φαινόταν ότι θα απαιτούνταν πολύ μεγάλοι καταχωρητές
- αλλά τελικά 3 έξτρα ψηφία αρκούν (guard digit, rounding digit, sticky bit)!

ΠΡΟΣΟΧΗ στο double rounding πολλών επεξεργαστών.

Οι κατασκευαστές γενικά ακολουθούν το πρότυπο IEEE-754 εδώ και καιρό αλλά ενίοτε έχουμε παραβάσεις (ακόμα και πρόσφατα)...

GPU Floating-Point Paranoia

Karl E. Hillesland
University of North Carolina at Chapel Hill *

Anselmo Lastra
University of North Carolina at Chapel Hill *

1 Introduction

Up until the late eighties, each computer vendor was left to develop their own conventions for floating-point computation as they saw fit. As a result, programmers needed to familiarize themselves with the peculiarities of each system in order to write effective software and evaluate numerical error. In 1987, a standard was established for floating-point computation to alleviate this problem, and CPU vendors now design to this standard [IEEE 1987].

Today there is an interest in the use of graphics processing units, or GPUs, for non-graphics applications such as scientific computing. GPUs have floating-point representations similar to, and sometimes matching, the IEEE standard. However, we have found that GPUs do not adhere to IEEE standards for floating-point operations, nor do they give the information necessary to establish bounds on error for these operations. Another complication is that this behavior seems to be in a constant state of flux due to the depen-

Operation	R300/arbfp	NV30/fp30
Addition	[-1.000, 0.000]	[-1.000, 0.000]
Subtraction	[-1.000, 1.000]	[-0.750, 0.750]
Multiplication	[-0.989, 0.125]	[-0.782, 0.625]
Division	[-2.869, 0.094]	[-1.199, 1.375]

Table 1: Floating-Point Error in ULPs (Units in Last Place). Note that the R300 has a 16 bit significand, whereas the NV30 has 23 bits. Therefore one ULP on an R300 is equivalent to 2^7 ULPs on an NV30. Division is implemented by a combination of reciprical and multiply on these systems. Cg version 1.2.1. ATI driver 6.14.10.6444. NVIDIA driver 56.72.

Schryer [Schryer 1981]. By testing all combinations of these numbers, we include all the test cases in Paranoia, as well as cases that push the limits of round-off error and cases where the most work must be performed, such as extensive carry propagation. Table 1 gives results for some example systems.

Hans-J Boehm.
Small-data computing.
Communications of the ACM, 60(8):44–49, July 2017.

J.W. Demmel.
Applied Numerical Linear Algebra.
SIAM, Philadelphia, 1997.

D. Goldberg.
What every computer scientist should know about floating point arithmetic.
ACM Comput. Surveys, pages 5–48, 1991.

J.-M. Muller et al.
Handbook of Floating-Point Arithmetic.
Birkhäuser Boston, 2010.

David Monniaux.
The pitfalls of verifying floating-point computations.
ACM Trans. Program. Lang. Syst., 30(3):12:1–12:41, May 2008.