Généricité en Fortran

application à l'écriture de code multiprécision

Vincent Lafage

[1]IJCLab, Laboratoire de Physique des 2 Infinis Irène Joliot-Curie
Université Paris-Saclay

7 janvier 2020

Préambule : FORTRAN

```
      SUBROUTINE OBACT(TODO)
      INTEGER TODO,DONE,IP,BASE
      COMMON /EG1/N,L,DONE
      PARAMETER (BASE=10)
13    IF(TODO.EQ.0) GO TO 12
      I=MOD(TODO,BASE)
      TODO=TODO/BASE
      GO TO(62,42,43,62,404,45,62,62,62),I
      GO TO 13
42    CALL COPY
      GO TO 127
43    CALL MOVE
      GO TO 144
404   N=-N
44    CALL DELETE
      GO TO 127
45    CALL PRINT
      GO TO 144
62    CALL BADACT(I)
      GO TO 12
127   L=L+N
144   DONE=DONE+1
      CALL RESYNC
      GO TO 13
12    RETURN
      END
```

Préambule : Fortran moderne

- paléo-FORTRAN (IV, 66), archéo-FORTRAN (77)
  fixed-format, identifiant court, COMMON blocks, GOTO arithmétique, implicit typing, chaînes Hollerith, EQUIVALENCE…
  tout en majuscule ?
- Fortran 90 & 95 :
  - *free-format (l'arbre qui cache la forêt)*
  - modularité
  - pointeurs et allocations dynamiques
  - récursivité
  - expressions fonctionnelles (fonctions pure)
  - expressions tableaux à la MATLAB (fonctions elemental)
    $\Rightarrow$ vectorisation
- Fortran 03 : programmation orientée objet, interopérabilité avec C
- Fortran 08 : parallélisme *co-array*
- Fortran 18 : plus d'interopérabilité, plus de parallélisme

Préambule : `Fortran` moderne

- pas de collections génériques
- pas de `FTL`

Comment faire à l'époque du *data processing* ?

Généricité

- *Generic*, *template*, *patterns*...
- les langages statiques (et d'autant plus qu'ils sont fortement typés)
  - \+ de la sécurité
  - − l'obligation de répéter les mêmes algorithmes sur des types différents

```
subroutine swap_integer (Left, Right)
  implicit none
  integer, intent (inout) :: Left, Right
  integer :: V

  V = Left
  Left = Right
  Right = V
end subroutine swap_integer
```

```
subroutine swap_logical (Left, Right)
  implicit none
  logical, intent (inout) :: Left, Right
  logical :: V

  V = Left
  Left = Right
  Right = V
end subroutine swap_logical
```

Généricité

- **DRY : Don't Repeat Yourself !**

$\Longrightarrow$ programmation générique ou paramétrique :
on écrit du code dont le type est un paramètre

```
subroutine swap_TYPE (Left, Right)
  implicit none
  TYPE, intent (inout) :: Left, Right
  TYPE :: V

  V = Left
  Left = Right
  Right = V
end subroutine swap_TYPE
```

¿ duplication de code par substitution dans un patron

1. par simple substitution de text *a.k.a.* ***preprocessing***
   comme une macro ?
2. à la charge du compilateur
   meilleure intégration et vérification de cohérence

Généricité

- Quelle portée pour les paramètres ?

  une fonction ? un module ? tout le programme ?
- Le retour des infâmes macros ?
- Quid de la généricité... en `Fortran`

La ``généricité´´ de Fortran (1)

`sqrt` est un moyen légal de se référer à

- `sqrt` `real(kind=4)` $\mapsto$ `real(kind=4)`
- `dsqrt` `real(kind=8)` $\mapsto$ `real(kind=8)`
- `csqrt` `complex(kind=4)` $\mapsto$ `complex(kind=4)`
- `zsqrt` `complex(kind=8)` $\mapsto$ `complex(kind=8)`

$\Rightarrow$ généricité des intrinsèques

La ``généricité´´ de Fortran (2)

```
subroutine swap_integer (Left, Right)
  implicit none
  integer, intent (inout) :: Left, Right
  integer :: V

  V = Left
  Left = Right
  Right = V
end subroutine swap_integer
```

```
subroutine swap_logical (Left, Right)
  implicit none
  logical, intent (inout) :: Left, Right
  logical :: V

  V = Left
  Left = Right
  Right = V
end subroutine swap_logical
```

```
module swap_module
  implicit none

  interface swap
     module procedure :: swap_logical, swap_integer
  end interface swap

contains
  include 'swap_integer.f90'
  include 'swap_logical.f90'
end module swap_module
```

$\Rightarrow$ généricité des interfaces

La ``généricité´´ de `Fortran` (3)

toujours plus fort

$\Rightarrow$ généricité des fonctions tableaux

`min`, `max`, `sum`, `maxloc`, *etc*.

$\Rightarrow$ généricité des profils de tableaux, pour les intrinsèques

La ``généricité´´ de Fortran (4)

en somme

- concluant pour les *intrinsics*

  (se prêtent à des changements de précision, mais aussi de types, et surtout de rang)

- mais au mieux de la **surcharge (*overloading*)**, alias **polymorphisme *ad hoc*** ou faible

  (généricité d'interface, mais pas de type générique, de module générique, ni d'implementation générique)

  en fait, c'est adapté au type de problèmes de Fortran notamment l'implémentation de fonctions à divers degrés de précision

- alors pourquoi pas le polymorphisme fort, alias héritage ? après tout, c'est possible avec Fortran 2003

  $\Longrightarrow$ overkill : dynamique, pointeurs, ruine de la performance...

- on veut du **polymorphisme *paramétrique***

  (type paramétrique, module paramétrique)

- *polymorphisme* : fournir une interface unique à des entités pouvant avoir différents types.

Programmation générique en Fortran ?

$\Longrightarrow$ à l'ancienne, avec le préprocesseur

- `include` dans les usages depuis 77 et dans le standard du langage depuis 90
  étape de preprocessing par le compilateur
- cpp depuis des lustres, mais `fpp` pour éviter les soucis de caractères spécifiques
- `coco` (COnditional COmpilation) in `Fortran` 95 standard (ISO/IEC 1539-3 :1998) : Fortran like preprocessing directive en fait, c'est adapté au type de problèmes de `Fortran` notamment l'implémentation de fonctions à divers degrés de précision
- `f90ppr`, `Forpedo`, `Fpx3`, `PyF95++`, `PreForM.py`, `Fypp`, `ufpp`

Q Which pre-processor should I use for my Modern Fortran project ?

A1 **None**, just stick to the Fortran standard (**the safe bet**)

A2 If you need **conditional compilation only**, take **fpp** as it is used by the majority of the Fortran projects (**principle of least surprise**)

A3 At some point, you may need **meta-programming** capabilities, so let's investigate further...

Significance Loss ?

- Un cas classique : avec une issue évidente
- ...et sans issue apparente
- Qu'est-ce qu'on perd ?
- Peut-on le retrouver ? et si oui, Comment ?
- Quid de la généricité...en `Fortran`

Cas de calculs à risque

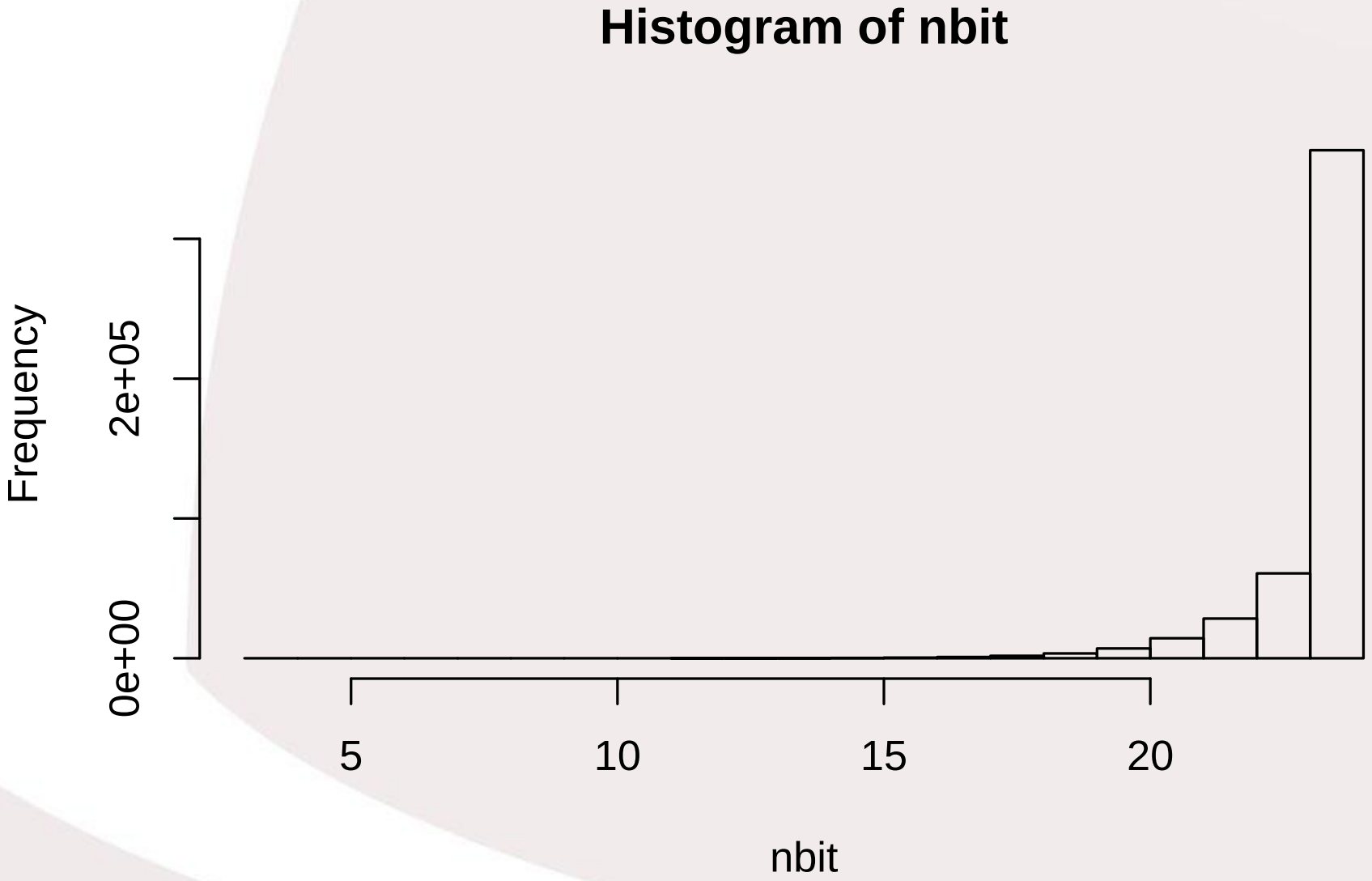
- différence de carrés
- discriminant et solutions de l'équation du second degré
- calcul de variance
- somme, produit scalaire, convolution, FFT
- évaluation de polynômes
- aire d'un triangle, volume d'un tétraèdre
- minimisation d'une fonction
- ...

Catastrophic Cancellation ?

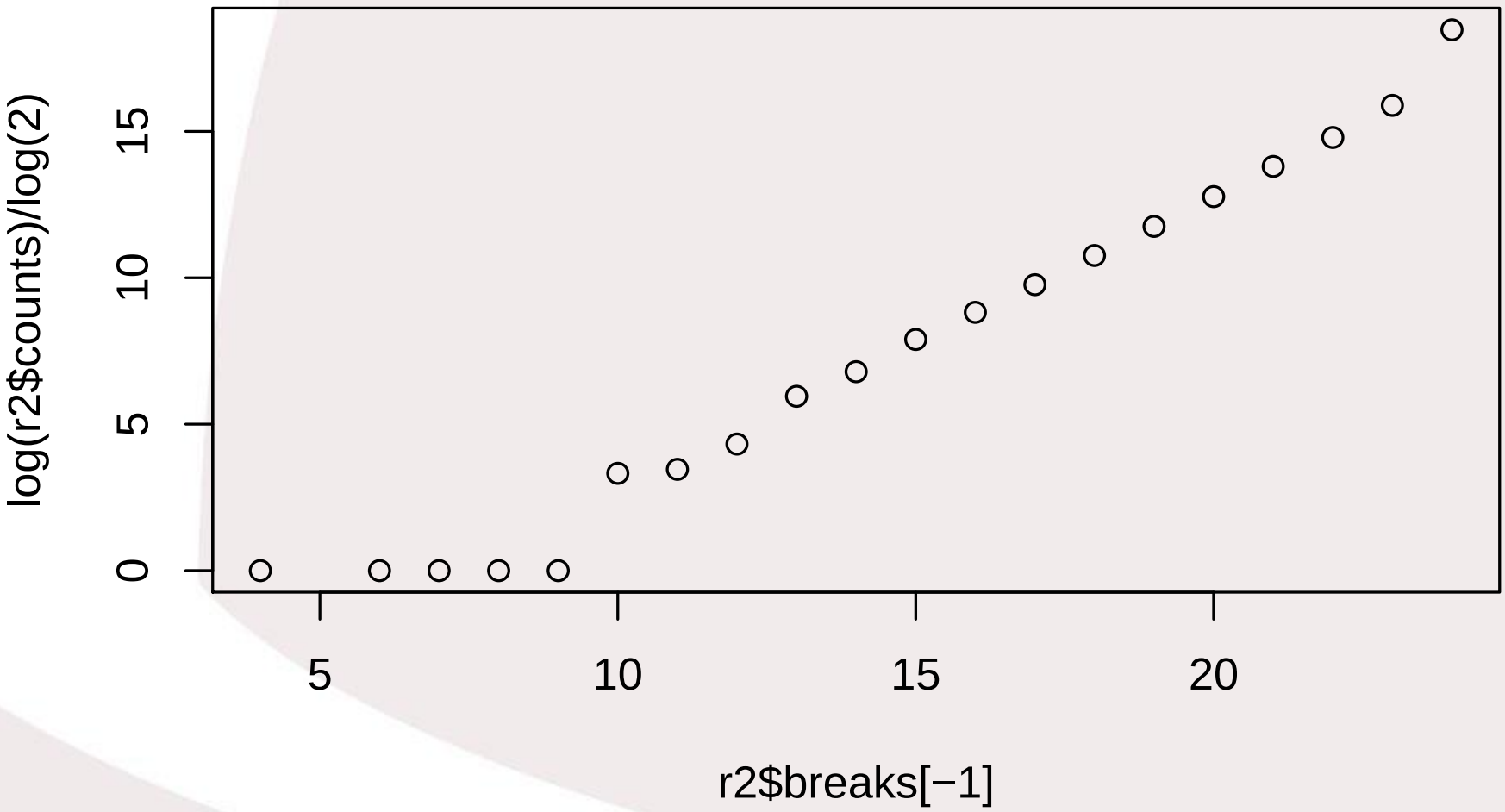
- $a^2 - b^2$ ne doit pas se calculer $\delta_0 =$ `a**2 - b**2`
- ...mais $\delta =$ `(a - b) * (a + b)`
- c'est vraiment grave ? $\Longrightarrow$ mesurons
- comptons l'erreur relative entre les deux, exprimée en bit

$$\ln_2 \frac{\delta_0 - \delta}{\delta}$$

Distribution de l'erreur relative



Distribution log de l'erreur relative



Catastrophic Cancellation ?

$ax^2 + 2bx + c = 0$ a pour discriminant réduit

- $\Delta = b^2 - ac$ n'est pas factorisable
- ...pourtant il y a un possibilité de compensation calamiteuse.
- peut-on faire mieux que calculer `b**2 - a*c` ?

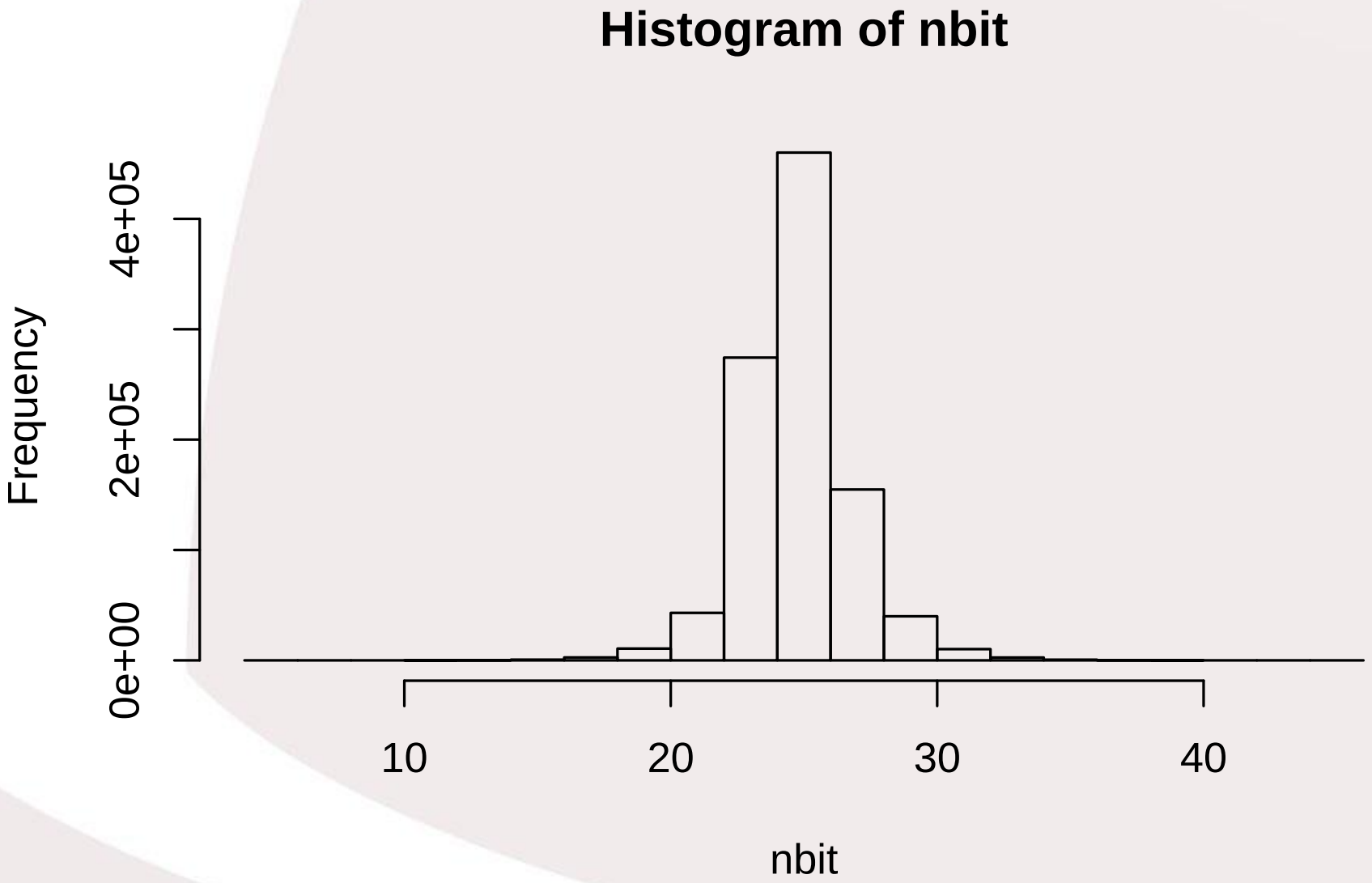
ab vs. a*b

- a*b $\neq ab$
- a*b $=$ rnd$(ab) = a \otimes b$
- *EFT= Error Free Transform*
- $ab = a \otimes b + \texttt{fma}(a, b, -a \otimes b)$
- `fma` procède à la multiplication exacte des 2 premiers arguments
  (il suffit d'un accumulateur à mantisse double)
  avant d'ajouter le dernier terme

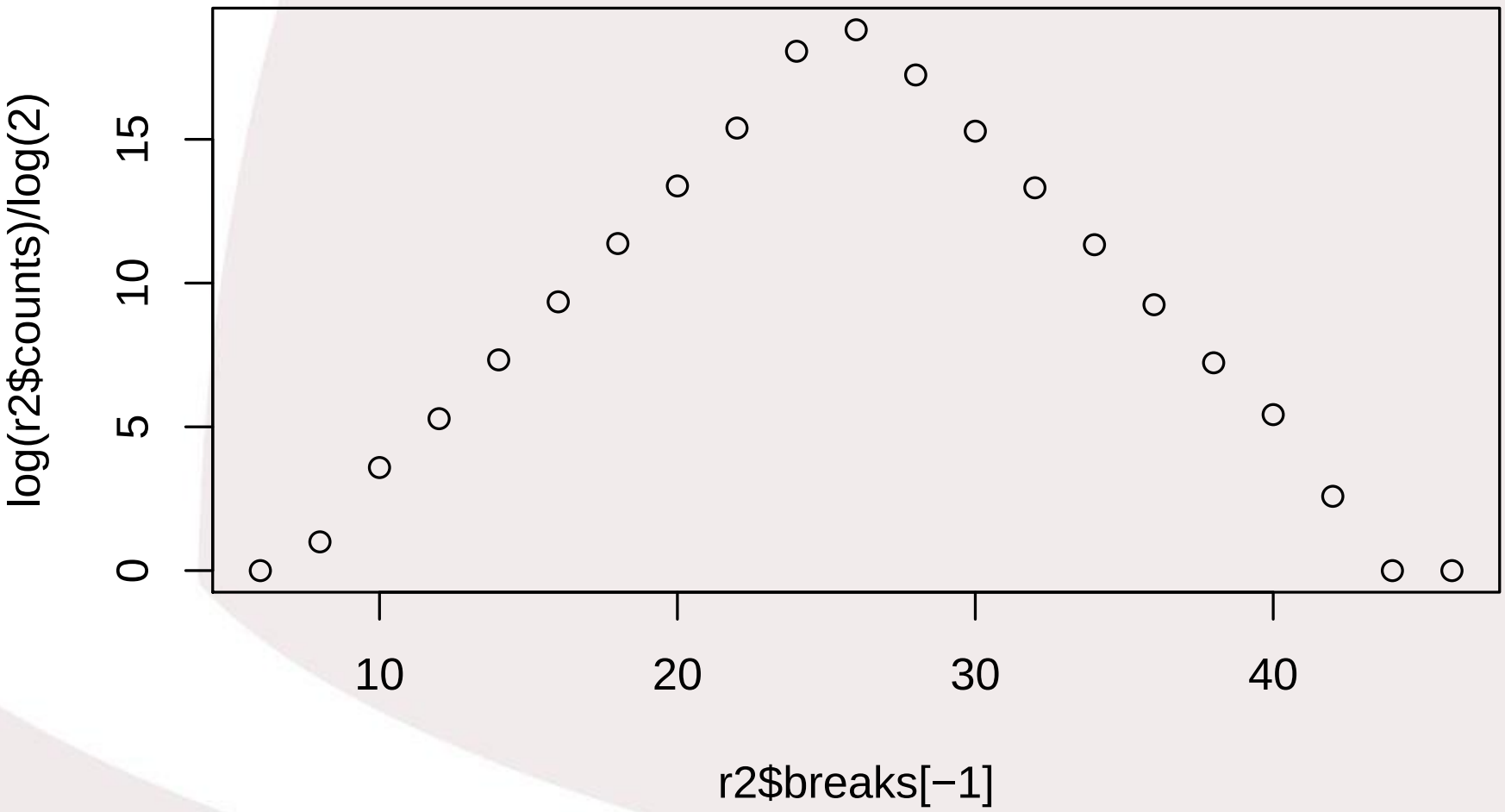
ab vs. a*b

- $\Delta = a^2 - b^2 =$
  $(a \otimes a - b \otimes b) + (\mathtt{fma}(a, a, -a \otimes a) - \mathtt{fma}(b, b, -b \otimes b))$
- erreur
  $= (a \otimes a - b \otimes b) + (\mathtt{fma}(a - b, a + b, -(a \otimes a - b \otimes b))$

Distribution de l'erreur relative



Distribution log de l'erreur relative



Module à interface générique

```
module mod_fma
  use, intrinsic :: iso_c_binding, only: C_FLOAT, C_DOUBLE, C_LONG_DOUBLE, C_FLOAT128
  use, intrinsic :: iso_fortran_env, only: REAL32, REAL64, REAL128
  implicit none

  interface fma_C
     pure function fmad (a, b, c) bind (c, name="fma")
       import c_double
       real (c_double) :: fmad
       real (c_double), value, intent (in) :: a, b, c
     end function fmad

     pure function fmaf (a, b, c) bind (c, name="fmaf")
       import c_float
       real (c_float) :: fmaf
       real (c_float), value, intent (in) :: a, b, c
     end function fmaf

     pure function fmal (a, b, c) bind (c, name="fmal")
       import c_long_double
       real (c_long_double) :: fmal
       real (c_long_double), value, intent (in) :: a, b, c
     end function fmal

     pure function fmaq (a, b, c) bind (c, name="fmaq")
       import c_float128
       real (c_float128) :: fmaq
       real (c_float128), value, intent (in) :: a, b, c
     end function fmaq
  end interface fma_C
...
```

... et ``thick binding´´

```
...
  interface fma
     module procedure fmad_e, fmaf_e, fmal_e, fmaq_e
  end interface fma

contains
  elemental function fmad_e (a, b, c)
    implicit none
    real (c_double) :: fmad_e
    real (c_double), value, intent (in) :: a, b, c
    fmad_e = fmad (a, b, c)
  end function fmad_e

  elemental function fmaf_e (a, b, c)
    implicit none
    real (c_float) :: fmaf_e
    real (c_float), value, intent (in) :: a, b, c
    fmaf_e = fmaf (a, b, c)
  end function fmaf_e

  elemental function fmal_e (a, b, c)
    implicit none
    real (c_long_double) :: fmal_e
    real (c_long_double), value, intent (in) :: a, b, c
    fmal_e = fmal (a, b, c)
  end function fmal_e

  elemental function fmaq_e (a, b, c)
    implicit none
    real (c_float128) :: fmaq_e
    real (c_float128), value, intent (in) :: a, b, c
    fmaq_e = fmaq (a, b, c)
  end function fmaq_e
end module mod_fma
```

module générique ?

```
module mod_square_diff
  use, intrinsic :: iso_fortran_env, only: REAL32, REAL64, REAL128
  use, intrinsic :: iso_c_binding, only: C_FLOAT, C_DOUBLE, C_LONG_DOUBLE, C_FLOAT128
  implicit none
  private
  public :: square_diff
  interface square_diff
     module procedure square_diff_sgl, &
          square_diff_dbl, &
          square_diff_ext, &
          square_diff_qdl
  end interface square_diff
contains
#define PR REAL32
#define SQUARE_DIFF_TYPE square_diff_sgl
#include "generic_square_diff.f90"
#undef PR
#undef SQUARE_DIFF_TYPE

#define PR REAL64
#define SQUARE_DIFF_TYPE square_diff_dbl
#include "generic_square_diff.f90"
#undef PR
#undef SQUARE_DIFF_TYPE

#define PR c_long_double
#define SQUARE_DIFF_TYPE square_diff_ext
#include "generic_square_diff.f90"
#undef PR
#undef SQUARE_DIFF_TYPE

#define PR REAL128
#define SQUARE_DIFF_TYPE square_diff_qdl
#include "generic_square_diff.f90"
#undef PR
#undef SQUARE_DIFF_TYPE
end module mod_square_diff
```

Mise en œuvre par *template*

```
impure subroutine SQUARE_DIFF_TYPE (a, b, delta, delta0, delta1)
  use mod_fma
  implicit none
  real (PR), intent (in) :: a, b
  real (kind=PR), intent (out) :: delta, delta0, delta1
  real (kind=PR) :: p, q, r, dp, dq, dr
  real (kind=PR) :: s, d, ds, dd, deltadelta

  p  = a * a
  dp = fma (a, a, -p)
  q  = b * b
  dq = fma (b, b, -q)
  delta0 = (p - q)
  delta1 = (dp - dq)
  delta = delta0 + delta1 ! (p - q) + (dp - dq)
  deltadelta = (delta -delta0) -delta1

  s  = a + b
  d  = a - b
  r  = s * d
  dr = fma (s, d, -r) ! compensation for the product

  ds = (s - a) - b    ! Kahan summation for Sum
  dd = (d - a) + b    ! Kahan summation for Difference

  write (*, *) 'Sum    ', s
  write (*, *) 'Diff   ', d
  write (*, *) ' Sum   ', ds
  write (*, *) ' Diff  ', dd
  write (*, *) 'w/ fma'
  write (*, *) 'a      ', a
  write (*, *) 'b      ', b
  write (*, *) 'p=a²   ', p
  write (*, *) 'q=b²   ', q
  write (*, *) ' p     ', dp
  write (*, *) ' q     ', dq
  write (*, *) '       ', deltadelta
  write (*, *) '(p-q)  ', (delta0 - p) + q
```

Bibliographie

Jason R. Blevins.
A generic linked list implementation in fortran 95.
SIGPLAN Fortran Forum, 28(3) :2–7, December 2009.

David Car and Michael List.
Pyf95++ : A templating capability for the fortran 95/2003 language.
SIGPLAN Fortran Forum, 29(1) :2–20, April 2010.

Aleksandar Donev.
Genericity in extending fortran.
SIGPLAN Fortran Forum, 23(1) :2–13, April 2004.

Martin Erwig and Zhe Fu.
Parametric fortran – a program generator for customized generic fortran extensions.
In Bharat Jayaraman, editor, *Practical Aspects of Declarative Languages*, pages 209–223, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

Martin Erwig and Zhe Fu.
Parametric fortran–a program generator for customized generic fortran extensions.
In *International Symposium on Practical Aspects of Declarative Languages*, pages 209–223. Springer, 06 2004.

Martin Erwig, Zhe Fu, and Ben Pflaum.
Generic programming in fortran.
In *Proceedings of the 2006 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-based Program Manipulation*, PEPM '06, pages 130–139, New York, NY, USA, 2006. ACM.
http://web.engr.oregonstate.edu/~erwig/papers/GenericFortran_PEPM06.pdf.

Martin Erwig, Zhe Fu, and Ben Pflaum.
Parametric fortran : Program generation in scientific computing.
J. Softw. Maint. Evol., 19(3) :155–182, May 2007.
https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.346?casa_token=SBB8S47m71gAAAAA:AUEWMX4c73eedVbURVZYwB-fu4zyEZkG5mP_NmgCVaP7iK6tZWImwmgKa5J4d3RdenS2RvU_VqMNLuiCRA.

Arjen Markus.
Generic programming in fortran 90.
SIGPLAN Fortran Forum, 20(3) :20–23, December 2001.

Drew McCormack.
Generic programming in fortran with forpedo.
SIGPLAN Fortran Forum, 24(2) :18–29, August 2005.

Bharat Jayaraman (eds.) Paul Hudak (auth.).
Practical Aspects of Declarative Languages : 6th International Symposium, PADL 2004, Dallas, TX, USA, June 18-19, 2004. Proceedings.
Lecture Notes in Computer Science 3057. Springer-Verlag Berlin Heidelberg, 1 edition, 2004.

Van Snyder.
Constructive uses for include.
SIGPLAN Fortran Forum, 20(2) :2–4, September 2001.

Conclusion

- Pas de solution parfaite
- item le plus demandé sur la liste du comité Fortran (WG5)
- pas avant `Fortran 202Y`...