De JurassicFortran à Fortr'&Furious

Vincent Lafage

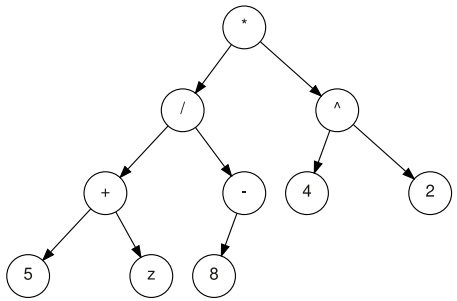
[1]IJCLab, Laboratoire de Physique des 2 Infinis Irène Joliot-Curie
Université Paris-Saclay

21 mars 2024

UrFortran

- 1954 : *Automatic Coding System for the IBM 704* dated October 15, 1956, first presented in February 1957 and first delivered in April 1957
- FORmula TRANslator : langage de haut niveau gérant la première abstraction, l'abstraction des expressions :

Le compilateur transforme les expressions en arbre d'évaluation



- pas le plus ancien compilateur, mais presque, et en tous cas le plus ancien à subsister
- le plus ancien compilateur optimiseur
- format fixe (cartes perforées)
- orienté calcul et pas système (relativement haut niveau = abstraction vis-à-vis du modèle mémoire)
- peu de types, économie de moyen, déclarations implicites

Historique

- 1966, *a.k.a.* `FORTRAN IV` première standardisation (ANSI X3.9-1966)
  (paléo FORTRAN)
- *« … a 1968 journal article by the authors of BASIC already described FORTRAN as "old-fashioned"… »*
- 1977 (ANSI X3.9-1978 FORTRAN, ISO 1539 :1980) influence de la programmation structurée :
  (archéo FORTRAN)
  paradigme **procédural** langage de haut niveau gérant la deuxième abstraction, l'abstraction du flux de contrôle
- 1989 `f2c` premier traducteur open source de `FORTRAN` vers `C`
- 1988 ? Rendez-vous manqué avec l'histoire
- 1991 `FORTRAN` ⇒ `Fortran`
  `Fortran 90` : langage de haut niveau gérant la troisième abstraction, l'abstraction des données
  paradigme **modulaire** + gestion des types dérivés `struct/record`
- *Free Format*
- 1995, Feb 17 : `g77` premier compilateur open source de `FORTRAN`
- 1997 `Fortran 95` : allocation dynamique
- early 2000—2006 : `g95` premier compilateur open source de `Fortran`
- 2003, Jan—2005, Apr 20 : `gfortran` fork de `g95`

Historique synoptique

	66	77	90	95	03	08	18/23	26

Procedural		ELSE
		ENDIF
		ENDDO
		DO WHILE
			SELECT CASE			BLOCK		

Modular	COMMON	INCLUDE	MODULE TYPE					
Inheritance					EXTENDS			
Dynamic			ALLOCATE POINTER		*			
Functional	passing subprogram argument		RECURSIVE	PURE ELEMENTAL			SIMPLE	
Concurrent			array exp	FORALL		DO CONCURRENT CONTIGUOUS		
Distributed						coarray	TEAM	
Generic	preprocessing	intrinsic						templat

Aspect sociolinguistique

- Style (supposé) de FORTRAN :
  - format fixe
  - pas d'indentation
  - ALL CAPS !
  - identifiant trop courts (6 caractères)
  - typage implicite
  - répétitions des déclarations et des `COMMON`
  - fonctions trop longues
  - `GOTO` à outrance, `GOTO` arithmétique, code spaggheti

Is all Fortran legacy ? SPAG

```
      IBON=0
      IF(KON)35,19,35
19    IF(NSQ-56)24,22,24
22    IF(LSQ-46)5,28,5
24    IF(NSQ-55)29,27,29
27    IF(LSQ-45)5,28,5
28    IBON=2
      GOTO 5
29    IF(LSQ-32)30,31,30
30    IF(LSQ-39)39,31,39
31    IBON=-5
      GOTO 5
39    IF(LSQ-35)52,51,52
52    IF(LSQ-36)5,51,5
51    IBON=10
      GOTO 5
35    IF(MARK(NMOVE))36,37,37
36    IBON=-5
      GOTO 5
37    IBON=5
5     end
```

⇒

```
ibon = 0
if (kon.ne.0) then
   if (mark(nmove).lt.0) then
      ibon = -5
   else
      ibon = 5
   endif
elseif (nsq.ne.56) then
   if (nsq.ne.55) then
      if (lsq.eq.32) then
         ibon = -5
      elseif (lsq.eq.39) then
         ibon = -5
      elseif (lsq.eq.35) then
         ibon = 10
      elseif (lsq.eq.36) then
         ibon = 10
      endif
   elseif (lsq.eq.45) then
      ibon = 2
   endif
elseif (lsq.eq.46) then
   ibon = 2
endif
end
```

Is all Fortran legacy ? SPAG

```
      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
```

⇒

```
module c_eg1
  implicit none
  private
  integer, public :: LENgth, NCHar, DONe
end module c_eg1

subroutine obact (Todo)
  use c_eg1
  implicit none
  integer, parameter :: BASE = 10
  integer, intent (inout) :: Todo
  integer :: act

  do while (Todo /= 0)
     act = mod (Todo, BASE)
     Todo = Todo / BASE
     select case (act)
     case (1, 4, 7, 8, 9)
        call badact (act)
        exit
     case (2)

…

     case default
        cycle
     end select
     DONe = DONe + 1
     call resync
  enddo
end subroutine obact
```

Is all Fortran legacy ? une recette

- tout déclarer (`implicit none`)
- *unspagghetiffy* (`SPAG`)
- convertir chaque `COMMON` en `include`
- convertir en *free format* (utiliser `convert` de M Metcalf)
  `https://dl.acm.org/doi/pdf/10.1145/192115.192139`
  `https://groups.google.com/g/comp.lang.fortran/c/C4Rth_-66is?pli=1`
- convertir chaque `include` en `module`
- convertir les variables statiques initialisées (`save`) constantes en `parameter`
- déclarer les `intent` de tous les paramètres de procédure
- déclarer `pure` toutes les procédures possibles
- déclarer `elemental` toutes les procédures possibles
- généricriser le `kind` des variables flottantes
- utiliser la forme générique de toutes les fonctions
- convertir les vieilles boucles `do ... enddo` en
  - appel de fonctions `elemental`
  - boucle `forall`
  - boucle `do concurrent`
- utiliser des fonctions plus appropriées :
  - `dot_product`, `norm2`
  - `matmul`
  - `reshape`, `transpose`
- ...

Fortran Unit Test ? pFUnit

```
!> @author Εὐκλείδης
!> @brief
!> General algorithm for factorising an integer into its prime factors
module divisor_mod
  use, intrinsic :: iso_fortran_env
  implicit none
contains
!> @brief
!> Extract the list of prime divisor of the input argument
  pure function divisor_list (n)
    implicit none
    integer, intent (in) :: n !< @param[in] n
    integer :: remainder  !< decreasing during the factorisation process
    integer :: divisor    !< increasing during the factorisation process
    integer, allocatable, dimension (:) :: divisor_list

    remainder = n
    divisor = 2 ! Starting the decomposition with first prime
    divisor_list = [ integer :: ] ! initialize to an empty vector (of integer)
    do while (remainder > 1)
       if (modulo (remainder, divisor) == 0) then
          divisor_list = [ divisor_list, divisor ]
          remainder = remainder / divisor
       else
          divisor = divisor + 1
       end if
    end do
  end function divisor_list
end module divisor_mod
```

Fortran Unit Test ? pFUnit

```
module test_divisor
   use divisor_mod
   use funit
   implicit none

contains
   @test
   subroutine test_assert_equal()
     implicit none

      @assertEqual(divisor_list (1), [ integer :: ], message="NoDivisorForOldMen")
      @assertEqual(divisor_list (2), [ 2 ], message="FirstDivisorEver")
      @assertEqual(divisor_list (3), [ 3 ], message="SecondDivisor")
      @assertEqual(divisor_list (4), [ 2, 2 ])
      @assertEqual(divisor_list (5), [ 5 ], message="ThirdPrime")
      @assertEqual(divisor_list (6), [ 2, 3 ])
      @assertEqual(divisor_list (7), [ 7 ], message="FourthPrime")
      @assertEqual(divisor_list (8), [ 2, 2, 2 ])
      @assertEqual(divisor_list (9), [ 3, 3 ])

   end subroutine test_assert_equal

end module test_divisor
```

Fortran OO ?

```
TYPE :: POINT ! A base type
  REAL :: X, Y
END TYPE POINT

TYPE, EXTENDS(POINT) :: COLOR_POINT ! An extension of TYPE(POINT)
  INTEGER :: COLOR                  ! Components X and Y, and component name
END TYPE COLOR_POINT                ! POINT, inherited from parent
```

Missing In `Fortran`

- héritage multiple ⇒ Facade Pattern
- first-class functions ⇒ `real, external, pointer :: f_ptr.`
  Alternative : `procedure (f), pointer :: f_ptr`
  `https://gcc.gnu.org/onlinedocs/gfortran/Working-with-C-Pointers.html`
- gestion d'exception (au-delà des exceptions IEEE 754)
- `assert`
- généricité (polymorphisme paramétrique) ⇒ `Fortran202Y`
  `https://github.com/j3-fortran/generics/tree/main`
- `const` ? (≠ `parameter` ∼ `consteval`) ⇒ …, `intent (in) ::` ou …, `protected ::`
- `C` binding to variadic functions
- coroutines
- …

Paradigme procédural/structuré

- GOTO label … label:
- IF (…) THEN… ELSE… ENDIF
-
```
SELECT CASE (selector-expression)
CASE (label-list-1)
   statements-1
  .............
CASE (label-list-n)
   statements-n
CASE DEFAULT
   statements-DEFAULT
END SELECT
```
- DO variable = from, to{, step} … ENDDO
- DO … ENDDO
- DO WHILE (condition) … ENDDO
- EXIT {construct-name}
- CYCLE {construct-name}
- CONTINUE
- Fortran 2008

```
{label:} BLOCK
           ...lines of code...
         END BLOCK
```

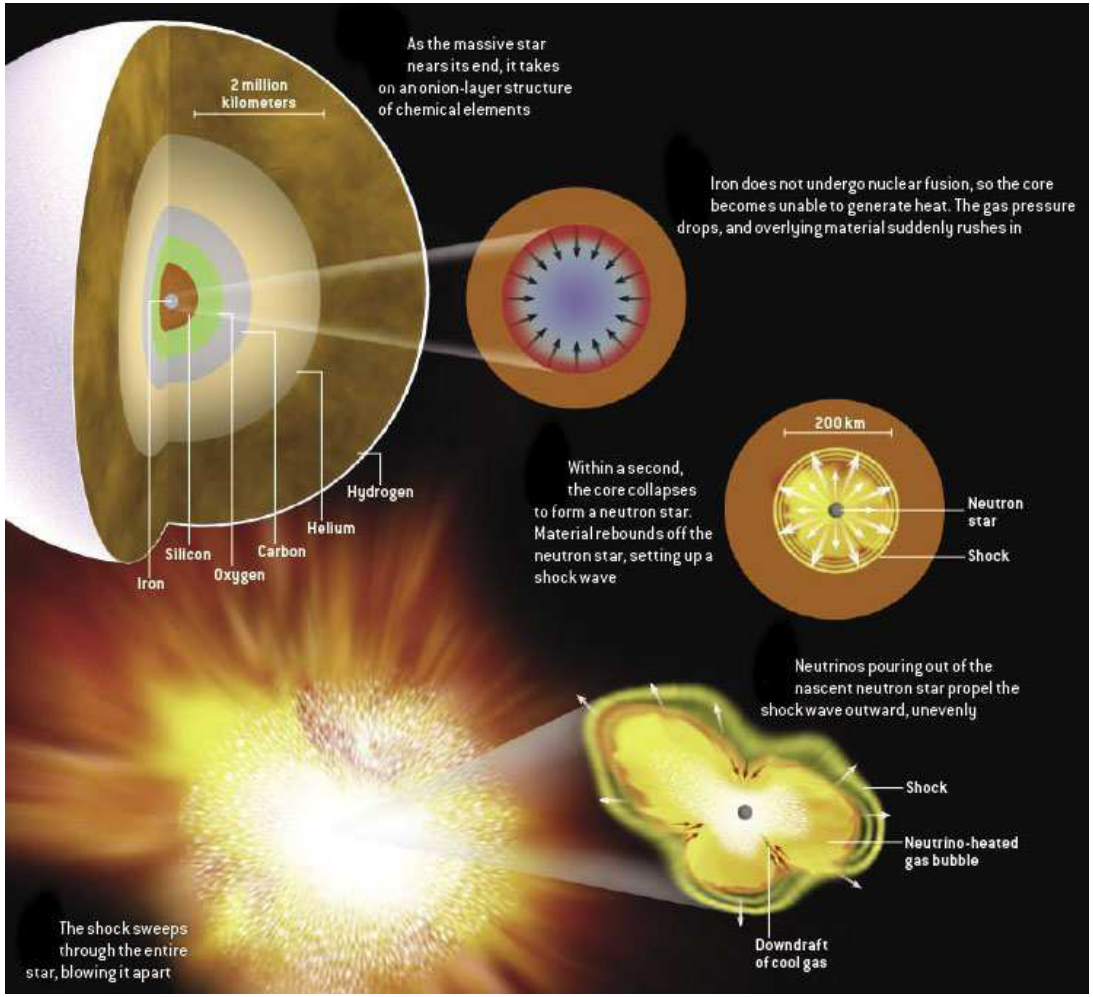
Harnessing Array la chasse aux boucles est ouverte

- opérations sur tableaux : $+$, $-$, $*$ (HADAMARD), / (*element-wise*)
- applications de fonctions sur tableaux (mapping)
  - les fonctions intrinsèques
  - les fonctions `ELEMENTAL` : `PURE` + automap
- réduction de tableaux : `SUM`, `PRODUCT(ARRAY, DIM, MASK)`
- `DOT_PRODUCT(VECTOR1, VECTOR2)`
- `MATMUL`
- `RESHAPE`
- `WHERE (mask) … [ELSEWHERE (mask) …] [ELSEWHERE …] END WHERE`
- …
- `FORALL (variable = from:to{:step})`
- `DO CONCURRENT (variable = from:to{:step})`

Capture d'électrons & supernovae

Groupe théorie IPNO
Simulation :
1000 types de noyaux
50 valeurs de T
$\Rightarrow$ 1000 ans de CPU

* *profiling*
* audit bibliothèques
* gestion mémoire
* vectorisation



$\Rightarrow$ **facteur $>$ 55**

$\Rightarrow$ **20 ans de CPU après accélération**

Structure du programme

```
(... boucles sur J, Π ...)                    ! 6
(... calcul de structure nucléaire selon un modèle RPA : ...)
(... occupations des niveaux d'énergies, fonctions d'ondes wf / dwf ...)
do idxenergy0 = 0, nbenergystep - 1    ! 150 énergies electron
   do energyc = 1, nconf               ! 153 / 405 / 540 états d'énergie du noyau
      do k_simps = 0, n_simps          ! 31 angles electron-neutrino
         do pairc = 1, nconf           ! 153 / 405 / 540 états d'énergie du noyau T < 1 MeV
                                       ! 182 / 476 / 637 états d'énergie du noyau T = 2 MeV
                                       ! 232 / 609 / 821 états d'énergie du noyau T = 3 MeV
                                       ! 386 / 1041 / 1448 états d'énergie du noyau T = 4 MeV
            do i = 1, maxr             ! 168 mailles radiales de fonction d'onde…
               à fond de boucles, fonctions de bessel ?sphériques> 92% du temps
```

$$j_\ell(qr)$$

nid de 5 boucles indépendantes

$\Longrightarrow$ paradis des parallélistes

entre 121 millions et 11 milliards d'itérations seulement pour les 4 boucles internes

Parallelisation ?

La parallelisation a l'air facile ⇒ OpenMP

```
(... boucles sur J, P ...)                ! 6
!$OMP PARALLEL PRIVATE (Eelectron, ecsum) SHARED (Energy_electron, sigma)
!$    & COPYIN (/energyidx/, /rpamix/, /bwf/, /occupa/, /qval/, /bdiam/,
!$    & /bqwf/, /bwu2/, /bnri/, /bnrir/, /bwu1/, /bwu1r/, /blecp1/,
!$    & /blecp2/, /bptl/)
do idxenergy0 = 0, nbenergystep - 1    ! 150 énergies electron
   do energyc = 1, nconf               ! 153 / 405 / 540 états d'énergie du noyau
      do k_simps = 0, n_simps          ! 31 angles electron-neutrino
         do pairc = 1, nconf           ! 153 / 405 / 540 états d'énergie du noyau
            do i = 1, maxr             ! 168 mailles radiales de fonction d'onde…
                à fond de boucles, fonctions de bessel ?sphériques> 92% du temps
```

⇒ segmentation fault :(

IDRIS ⇒ convertissez tout dans un seul langage
Fortran 90,validation des résultats, ajout de OpenMP

⇒segmentation fault :(

* Faut-il simplifier le code ?
* Dur avec des threads, facile avec des process
* Threads pas si équivalents en durée…
* Transformer le code pour exprimer une transformée rapide de Bessel Sphérique *à la* FFT ?
* intégration numérique reposant des produits de matrice ⇒ utilisons le GPU
  CUDA, OpenCL, OpenAcc?

OpenACC

```
subroutine radproint_array (j0)
  implicit none
  integer, intent (in) :: j0
  integer :: k_simps, pairc, idxenergy, energyc, maj ! dummy indices
  integer :: Jmin, Jmax
  Jmin = max (j0-1, 0)
  Jmax = j0+1

  allocate (Rad_proint1_array  (1:nconf, 0:n_simps, 1:nconf, 0:nbenergystep-1, Jmin:Jmax))

  !$acc data present_or_create (mask_kinematics, Rad_proint2B_array, Rad_proint2A_array, Rad_pro
  !$acc                        wf12_array, Bess_f_array, wf12inv_array, wf1dwf2_array)

  !$acc parallel loop collapse (3)
  build10: do maj = Jmin, Jmax
    build11: do idxEnergy = 0, nbenergystep-1
      build12: do energyc = 1, nconf
        if (mask_kinematics (energyc, idxEnergy)) then
  !$acc loop vector independent collapse (2)
          build13: do k_simps = 0, n_simps
            build14: do pairc = 1, nconf
              Rad_proint1_array (pairc, k_simps, energyc, idxEnergy, maj) = &
              sum (wf1dwf2_array (1:maxr, pairc) * Bess_f_array (1:maxr, k_simps, energyc, idxEn
            end do build14
          end do build13
        end if
      end do build12
    end do build11
  end do build10
  !$acc end parallel loop

  !$acc parallel
```

Configuration

https://gitlab.in2p3.fr/lafage/GrayScottFortranTuto

```
&GRAYSCOTT
 DT= 1.0,                  ! time step
 DIFFUSIVITY_U=  0.1,      ! ur: diffusion rate of u
 DIFFUSIVITY_V=  0.05,     ! vr: diffusion rate of v
 FEED_RATE=  0.014,        ! rf
 KILL_RATE=  0.054,        ! rk
 /
```

https://gitlab.in2p3.fr/lafage/GrayScottFortranTuto