De JurassicFortran à Fortr'&Furious

Calculer sur CPU et GPU avec NVfortran et OpenACC

Vincent Lafage

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

2 mai 2024

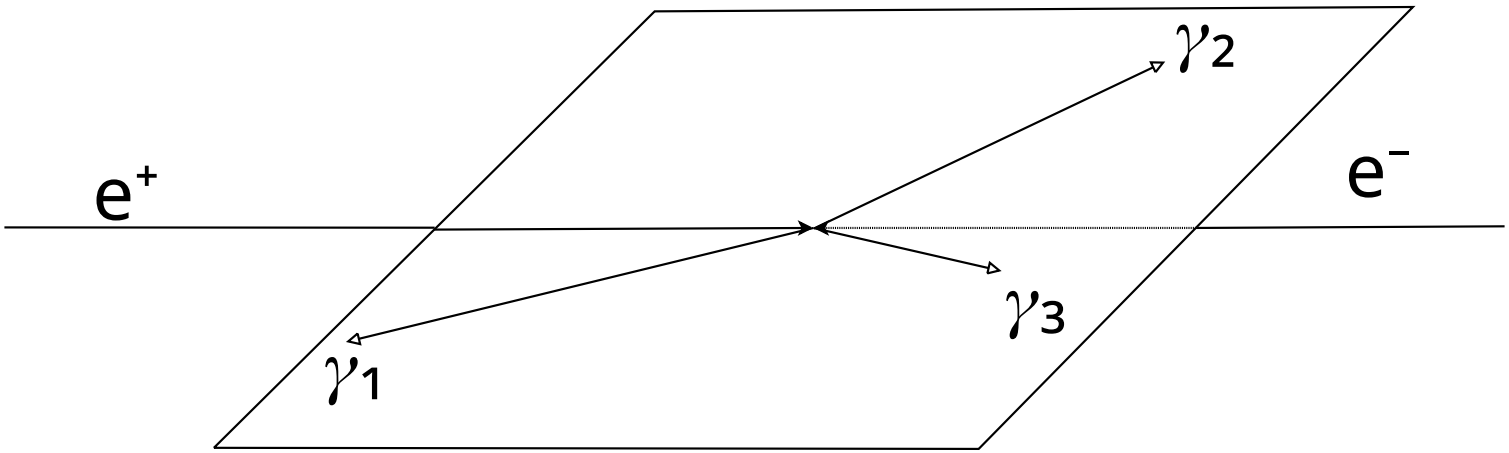
Fortran OO 2003

```
module test_class_module
   implicit none
   type, public :: Test
      integer, private, allocatable :: arr (:)
   contains
      final :: destructor
      procedure :: method
   end type

   interface Test
      procedure :: constructor
   end interface
   private :: constructor
contains
   function constructor (arr_size) result (this)
      type (Test) :: this
      integer, intent (in) :: arr_size
      write (*, *) 'Constructor works'
      allocate(this%arr (arr_size))
      this%arr = 0
   end function

   subroutine destructor (this)
      type (Test) :: this
      write (*, *) 'Destructor works'
      if (allocated (this%arr)) deallocate (this%arr)
   end subroutine

   pure integer function method (this)
     class (Test), intent (in) :: this
     method = sum (this%arr)
   end function method
end module
```

3 petits photons

École Informatique IN2P3 « Parallélisme sur Matériel Hétérogène »

https://indico.in2p3.fr/event/13126



« New physics with three photon events at LEP »,
M. Baillargeon, F. Boudjema, E. Chopin, V. Lafage,

Z.Phys. C71 (1996) 431–442, e-Print : arXiv:hep-ph/9506396v2

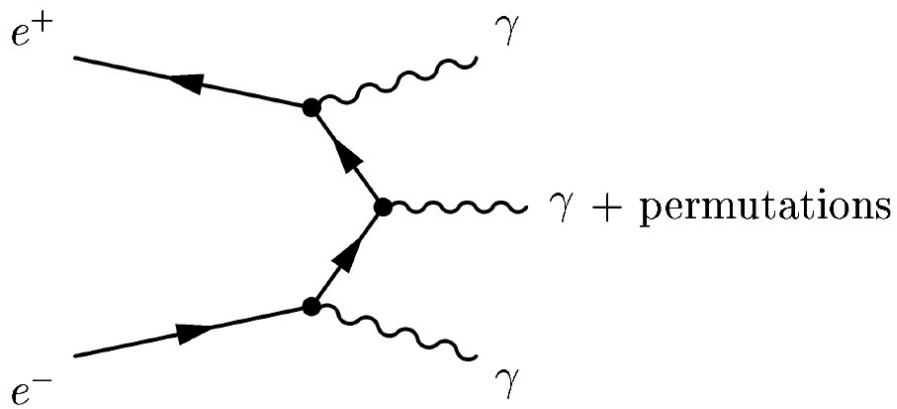
Section efficace

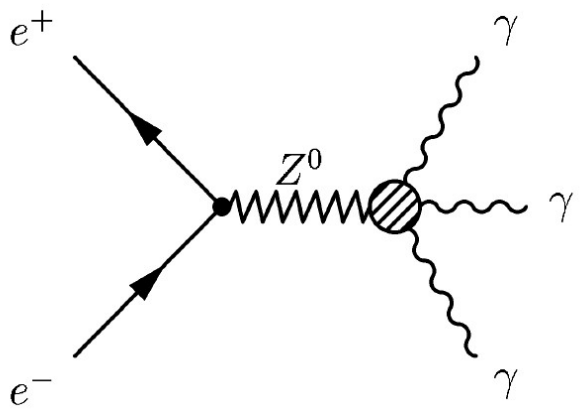
Comment la calculer ?

- d'autant plus grande qu'ils y a plus de configurations possibles pour les particules sortantes : somme sur l'espace des configurations, appelé ***espace des phases***

$$\sigma = \int |\mathcal{M}|^2 \times \delta^4(P - p_1 - \cdots - p_n) \prod_{i=1}^{n} \mathrm{d}^4 p_i \, \delta(p_i^2 - m_i^2) \, \theta(p_i^0) \qquad (1)$$

- ce qu'on somme (la section efficace différentielle) dépend de ***l'élément de matrice de transition*** $|\mathcal{M}|^2$ calculé à partir des ***graphes de*** **Feynman**





Intégration numérique : $\Rightarrow$ *Monte-Carlo*

Ici, la plus simple (pas de raffinement adaptatif) :

- générer un événement avec un certain poids,
  - générateur de nombres pseudo-aléatoires
  - méthode RAMBO
    « A new Monte Carlo treatment of multiparticle phase space at high energies »,

    R. Kleiss, W. J. Stirling, S. D. Ellis, Comput. Phys. Commun. 40, 359 (1986).
- voir s'il passe les coupures,
- calculer l'élément de matrice,
- pondérer,
- accumuler...

...et recommencer
Mais il y a deux types d'itérations :

1. celles qui dépendent de la précédente...
2. ... et les autres $\Rightarrow$ Monte-Carlo est ***délicieusement parallélisable***

(*embarassingly parallel*)

Main loop F77

```
      DO I=1,ITOT
         CALL RAMBO(INP,ETOT,MFIN,POUT,WTEV)
         WTEV=WTEV*NORM
         CALL TRI
         CALL PRECALC
         IF (.NOT.CUT(0.)) THEN
            CALL MATRX(ETOT)
            DO K=1,NRESUL
               SPM2DIF(K)=0.0d0
               DO L1=1,2
                  DO L2=1,2
                     DO L3=1,2
                        SPM2DIF(K)=SPM2DIF(K)+M2(L1,L2,L3,K)
                     ENDDO
                  ENDDO
               ENDDO
               SPM2(1,K)=SPM2(1,K)+SPM2DIF(K)
               VAR(1,K) =VAR(1,K) +SPM2DIF(K)**2
            ENDDO
            PROBA=
     &     CAA*SPM2DIF(1)
     &    +CBB*(BETAPLUS**2*SPM2DIF(2)
     &         +BETAMOINS**2*SPM2DIF(3))
     &                 /GZR**2*PROPAG
     &    +CAB*2*BETAPLUS*(ECARTPIC*SPM2DIF(4)
     &                  -SPM2DIF(5))
     &                 /GZR*PROPAG
            POIDS=PROBA*WTEV/4.D0
            SIGMA=SIGMA+POIDS
            VARIANCE=VARIANCE+POIDS**2
            NTOT=NTOT+1
         ENDIF
      ENDDO
```

Main loop F77+OpenMP

```
!$OMP PARALLEL  DEFAULT(FIRSTPRIVATE)
!$OMP& SHARED(SIGMA, VARIANCE, NTOT, /resfin/)
!$OMP& COPYIN(/angle/, /cutpar/, /param/, /spinor/, /PAWC/, /event/)
!$OMP DO REDUCTION (+:SIGMA, VARIANCE, NTOT, SPM2, VAR)
      DO I=1,ITOT
         CALL RAMBO(INP,ETOT,MFIN,POUT,WTEV)
         WTEV=WTEV*NORM
         CALL TRI
         CALL PRECALC
         IF (.NOT.CUT(0.)) THEN
            CALL MATRX(ETOT)
            DO K=1,NRESUL
               SPM2DIF(K)=0.0d0
               DO L1=1,2
                  DO L2=1,2
                     DO L3=1,2
                        SPM2DIF(K)=SPM2DIF(K)+M2(L1,L2,L3,K)
                     ENDDO
                  ENDDO
               ENDDO
               SPM2(1,K)=SPM2(1,K)+SPM2DIF(K)
               VAR(1,K) =VAR(1,K) +SPM2DIF(K)**2
            ENDDO
            PROBA=
     &     CAA*SPM2DIF(1)
     &    +CBB*(BETAPLUS**2*SPM2DIF(2)
     &         +BETAMOINS**2*SPM2DIF(3))
     &                /GZR**2*PROPAG
     &    +CAB*2*BETAPLUS*(ECARTPIC*SPM2DIF(4)
     &                 -SPM2DIF(5))
     &                /GZR*PROPAG
            POIDS=PROBA*WTEV/4.D0
            SIGMA=SIGMA+POIDS
            VARIANCE=VARIANCE+POIDS**2
            NTOT=NTOT+1
         ENDIF
      ENDDO
!$OMP END DO
!$OMP END PARALLEL
```

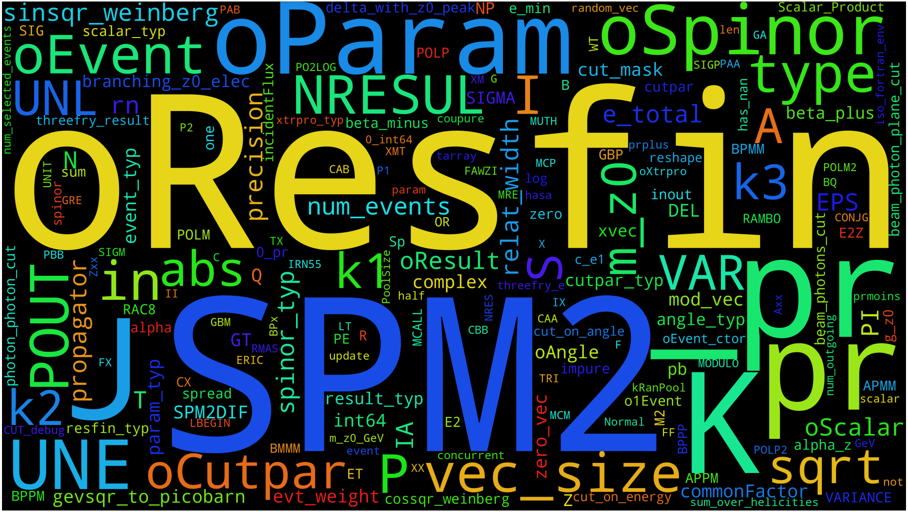
Main loop F77+OpenMP : results

Threads (8-core)	Durée (µs)	Durée CPU (µs)
1	0.519	0.533
2	0.277	0.538
3	0.190	0.564
4	0.144	0.563
5	0.113	0.594
6	0.100	0.596
7	0.088	0.609
8	0.077	0.605
9	0.081	0.667

Main loop F03

```
integration: do I=1, num_events
   call o1Event%RAMBO (num_outgoing, e_total, final_state_masses, evt_weight)
   evt_weight=evt_weight*cstVolume
   if (o1Event%cut_on_energy (oCutpar)) cycle integration
   call o1Event%TRI
   o1Spinor = spinor_typ (o1Event)
   o1Scalar = scalar_typ (o1Spinor)
   o1Angle  = angle_typ (o1Event, o1Scalar)
   if (.NOT. o1Angle%cut_on_angle (oCutpar)) then
      o1Result = result_typ (o1Spinor, oParam)
      oResfin%SPM2DIF (1:NRESUL) = o1Result%sum_over_helicities ()
      oResfin%SPM2 (1, 1:NRESUL) = oResfin%SPM2 (1, 1:NRESUL) + oResfin%SPM2DIF (1:NRESUL)
      oResfin%VAR  (1, 1:NRESUL) = oResfin%VAR  (1, 1:NRESUL) + oResfin%SPM2DIF (1:NRESUL)**2

      PROBA= &
           CAA*oResfin%SPM2DIF (1) &
           +CBB*(beta_plus**2*oResfin%SPM2DIF (2) &
           +beta_minus**2*oResfin%SPM2DIF (3)) &
           /relat_width**2*propagator &
           +CAB*2*beta_plus*(delta_with_z0_peak*oResfin%SPM2DIF (4) &
           -oResfin%SPM2DIF (5)) &
           /relat_width*propagator
      POIDS=PROBA*evt_weight/4._pr
      SIGMA=SIGMA+POIDS
      VARIANCE=VARIANCE+POIDS**2
      num_selected_events=num_selected_events+1
   end if
end do integration
```

wordcloud



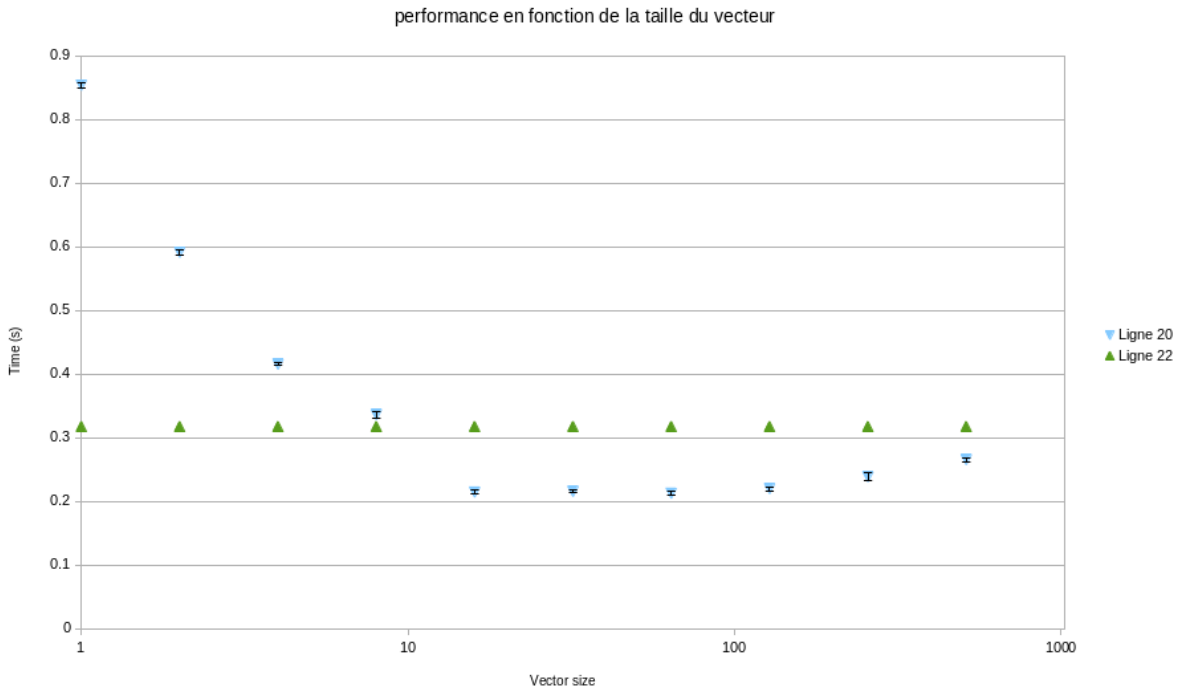
```
module ThreefryModule
  use iso_fortran_env, only: int64
  integer, parameter :: NW_THREEFRY = 2, N_ROUNDS_THREEFRY = 20
  integer(int64), parameter :: C240 = int (z'1BD11BDAA9FC1A22', int64), &
       MASK = huge(0_int64) - 1_int64, R_THREEFRY (0:*) = [16, 42, 12, 31, 16, 32, 24, 21]
  type, public :: threefry_key
     integer(int64), dimension(0:NW_THREEFRY) :: k
  end type threefry_key
contains
  pure function key_schedule_threefry(K, s) result(ksi)
    type (threefry_key), intent (in) :: K
    integer, intent (in) :: s
    integer (int64), dimension (0:1) :: ksi
    ksi(0:1) = [ iand (K%k  (mod (s,     NW_THREEFRY + 1)),      MASK), &
         iand ((K%k (mod (s + 1, NW_THREEFRY + 1)) + s), MASK) ]
  end function key_schedule_threefry

  pure function mix (e, r) result(v)
    integer (int64), intent (in) :: e (0:1), r
    integer (int64), dimension (0:1) :: v
    v(0:1) = [ ieor(e(0), ishftc(e(1), r)), ieor (e(1), ishftc(e(0), 64 - r)) ]
  end function mix

  pure function threefry (p, K0) result (v)
    integer(int64), intent(in) :: p(0:)
    type(threefry_key), intent(in) :: K0
    integer(int64), dimension (0:1) :: v, e, ksi
    type(threefry_key) :: K
    integer :: r

    K%k = [K0%k(0), K0%k(1), ieor(C240, ieor(K0%k(0), K0%k(0)))]
    v = p
    do r = 0, N_ROUNDS_THREEFRY - 1  ! Perform Threefish rounds
       if (mod(r, 4) == 0) then
          ksi = key_schedule_threefry(K, r / 4)
          e = [iand(v(0) + ksi(0), MASK), iand(v(1) + ksi(1), MASK)]
       else
          e = v
       end if
       v = mix (e, R_THREEFRY (mod (r, 8)))
    end do
    ksi = key_schedule_threefry(K, N_ROUNDS_THREEFRY / 4)
    v = iand (v + ksi, MASK)
  end function threefry
end module ThreefryModule
```

Main loop F03 : vectorisation

```
integration: do concurrent (I=1:num_vecs)
   rn = reshape ([ ((random_vec (6*((i - 1) * vec_size + k) + j), j=0, 5), k=0, vec_size - 1) ], shape (rn))
   call o1Event%RAMBO (rn, num_outgoing, e_total, evt_weight)
   evt_weight=evt_weight*cstVolume
   cut_mask = .not.o1Event%cut_on_energy (oCutpar)
   call o1Event%TRI
   o1Spinor = spinor_typ (o1Event)
   o1Scalar = scalar_typ (o1Spinor)
   o1Angle  = angle_typ (o1Event, o1Scalar)
   cut_mask = cut_mask .and.(.not.o1Angle%cut_on_angle (oCutpar))
      o1Result = result_typ (o1Spinor, oParam)
      call oResfin%update (o1Result, cut_mask)

      PROBA= sum ( &
           CAA*oResfin%SPM2DIF (:, 1) &
           +CBB*(beta_plus**2*oResfin%SPM2DIF (:, 2) &
           +beta_minus**2*oResfin%SPM2DIF (:, 3)) &
           /relat_width**2*propagator &
           +CAB*2*beta_plus*(delta_with_z0_peak*oResfin%SPM2DIF (:, 4) &
           -oResfin%SPM2DIF (:, 5)) &
           /relat_width*propagator, cut_mask)
      PROBASQ= sum (( &
           CAA*oResfin%SPM2DIF (:, 1) &
           +CBB*(beta_plus**2*oResfin%SPM2DIF (:, 2) &
           +beta_minus**2*oResfin%SPM2DIF (:, 3)) &
           /relat_width**2*propagator &
           +CAB*2*beta_plus*(delta_with_z0_peak*oResfin%SPM2DIF (:, 4) &
           -oResfin%SPM2DIF (:, 5)) &
           /relat_width*propagator)**2, cut_mask)
      POIDS=PROBA*evt_weight/4._pr
      SIGMA=SIGMA+POIDS
      VARIANCE=VARIANCE+PROBASQ*(evt_weight/4._pr)**2
      num_selected_events=num_selected_events+count(cut_mask)
end do integration
```

Main loop `F03` : autovectorisation

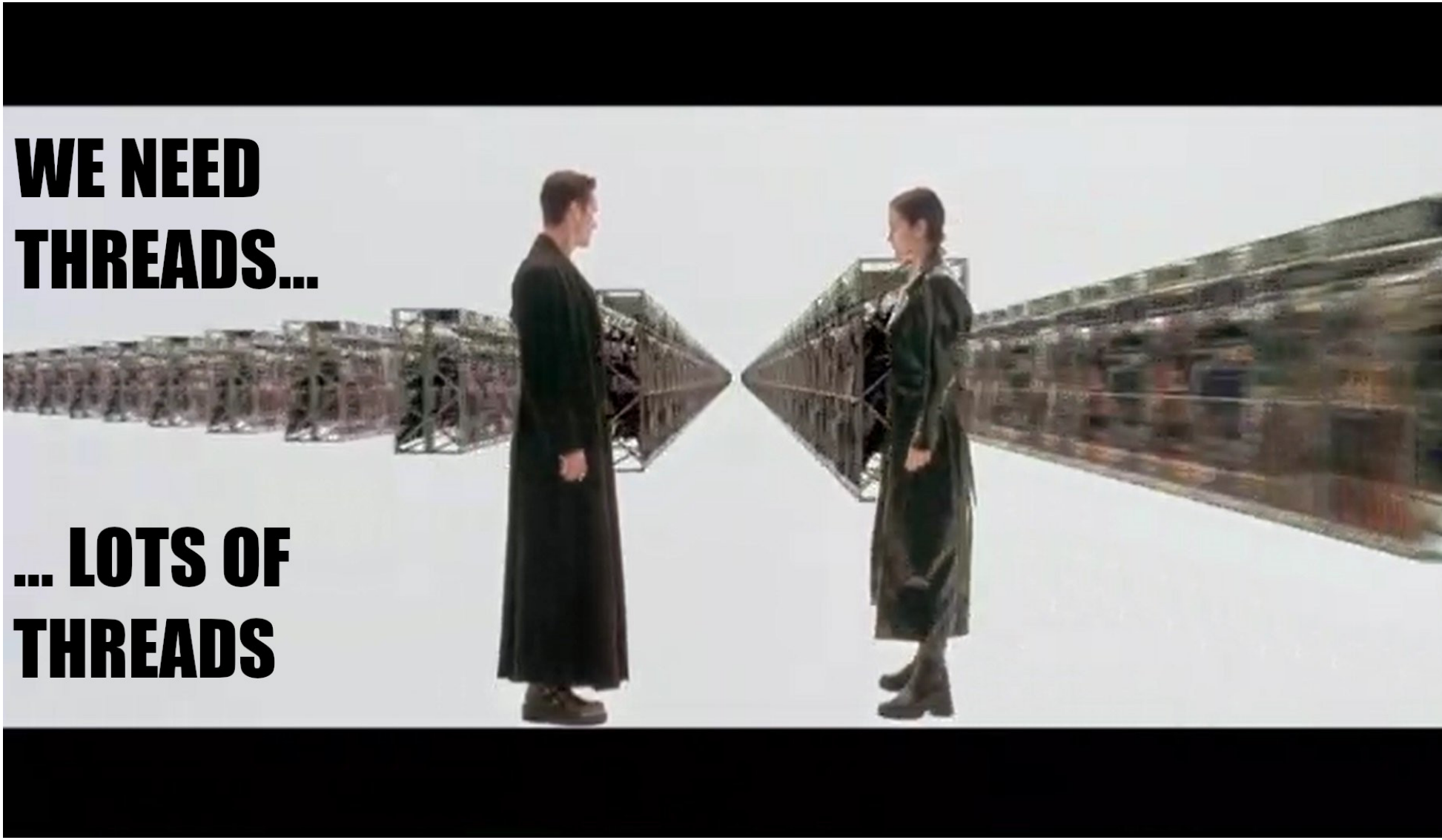
Taille	Durée (µs)	Incertitude (µs)
0	0.317	0.002
1	0.853	0.005
2	0.591	0.004
4	0.415	0.002
8	0.336	0.005
16	0.214	0.004
32	0.215	0.002
64	0.213	0.003
128	0.220	0.003
256	0.238	0.006
512	0.265	0.003
1024	0.263	0.004



avec `gfortran-12` sur une archi `avx-512f`

Main loop `F03` : autovectorisation

- pas terrible
- malgré le potentiel de `loop-unrolling`
- ... mais tous les calculs sont menés à terme
  même ceux qui vont être jeté par les cuts
- ... mais le calcul complexe de coeur de boucle n'est vraisemblablement pas vectorisé


WE NEED
THREADS...
... LOTS OF
THREADS

Main loop F03 : autoparallélisation ?

```
...
  use, intrinsic :: iso_fortran_env, only: pr => real64
  integer, parameter    :: NP = 3 !< Number of generated impulsions
  real (pr), parameter :: one = 1, & ! zero = 0, two = 2, half = one / two,
         TWOPI  = 2 * acos (-one), &
         PO2LOG = log (TWOPI / 4)
  integer    ::  K !> Dummy loop variables
  real (pr), dimension (2:NP), parameter :: Z = [ ( &
         ((k-1) * PO2LOG - log_gamma (real (k-1, pr)) - log (real (k-1, pr))), k = 2, NP) ]
...
```

```
$ nvfortran factorial.f90 && ./a.out
NVFORTRAN-S-0155-Intrinsic not supported in initialization: log_gamma (factorial.f90: 8)…
```

Main loop F03 : autoparallélisation ?

```
$ FC=nvfortran make -f Makefile.mk
nvfortran -g -Wall -Wextra -Werror -tp=native -C -march=native -mtune=native -mavx512f -acc autopar -Minfo=accel  -c -o obj/precision.o src/precision.f90
nvfortran-Warning-CUDA Fortran or OpenACC GPU targets disables -Mbounds
src/precision.f90:
nvfortran -g -Wall -Wextra -Werror -tp=native -C -march=native -mtune=native -mavx512f -acc autopar -Minfo=accel  -c -o obj/mod_vec.o src/mod_vec.f90
nvfortran-Warning-CUDA Fortran or OpenACC GPU targets disables -Mbounds
src/mod_vec.f90:
nvfortran -g -Wall -Wextra -Werror -tp=native -C -march=native -mtune=native -mavx512f -acc autopar -Minfo=accel  -c -o obj/cutpar.o src/cutpar.f90
nvfortran-Warning-CUDA Fortran or OpenACC GPU targets disables -Mbounds
src/cutpar.f90:
nvfortran -g -Wall -Wextra -Werror -tp=native -C -march=native -mtune=native -mavx512f -acc autopar -Minfo=accel  -c -o obj/param.o src/param.f90
nvfortran-Warning-CUDA Fortran or OpenACC GPU targets disables -Mbounds
src/param.f90:
nvfortran -g -Wall -Wextra -Werror -tp=native -C -march=native -mtune=native -mavx512f -acc autopar -Minfo=accel  -c -o obj/hasa.o src/hasa.f90
nvfortran-Warning-CUDA Fortran or OpenACC GPU targets disables -Mbounds
src/hasa.f90:
nvfortran -g -Wall -Wextra -Werror -tp=native -C -march=native -mtune=native -mavx512f -acc autopar -Minfo=accel  -c -o obj/event.o src/event.f90
nvfortran-Warning-CUDA Fortran or OpenACC GPU targets disables -Mbounds
src/event.f90:
NVFORTRAN-F-0000-Internal compiler error. module:new_dtype, dt nfd     160  (src/event.f90: 144)
NVFORTRAN/x86-64 Linux 24.3-0: compilation aborted
make: *** [Makefile.mk:73 : obj/event.o] Erreur 2
```

Main loop F03 : autoparallélisation...

```
ππ                                                                                1
  pure subroutine RAMBO (oEvent, RN0, N, ET, WT)                                  2
    !-----------------------------------------------------------------------------3
    !                                                                             4
    !                         RAMBO                                               5
    !                                                                             6
    !    RA(NDOM)  M(OMENTA)  B(EAUTIFULLY)  O(RGANIZED)                          7
    ! a.k.a RA(NDOM)  M(OMENTA)  BO(OSTER)                                        8
    ! ****** VERSION LIGHT & LIGHTSPEED ******                                   9
    !                                                                             10
    !    A DEMOCRATIC MULTI-PARTICLE PHASE SPACE GENERATOR                        11
    !    AUTHORS@D  S.D. ELLIS,  R. KLEISS,  W.J. STIRLING                        12
    !    THIS IS VERSION 1.0 -  WRITTEN BY R. KLEISS                              13
    !-----------------------------------------------------------------------------14
    use hasa, only: gRanVec, random_vec                                           15
    type (event_typ), intent (inout) :: oEvent !> Particle momenta (DIM=(4, 100))16
    real (pr), intent (in), dimension (1:vec_size, 1:12) :: RN0                   17
    integer, intent (in)    :: N  !> Number of particles (>1, IN THIS VERSION <101)18
    real (pr), intent (in)  :: ET !> Total centre-of-mass energy                  19
   !real (pr), intent (in), dimension (NP) :: XM   !> Particle masses             20
    real (pr), intent (out) :: WT !> Weight of the event                          21
    integer :: I !> Dummy loop variables                                         22
    real (pr), dimension (vec_size) :: C, S, F, RMAS, G, A, X, BQ                23
    real (pr) :: Q (vec_size, 4, NP), R (vec_size, 4), B (vec_size, 3)            24
!    integer, dimension (1:5) :: IWARN = [ 0, 0, 0, 0, 0 ]                       25
    real (pr), dimension (vec_size, 1:4) :: RN                                    26
    integer :: J !> Dummy loop variables                                         27
                                                                                  28
    ! GENERATE N MASSLESS MOMENTA IN INFINITE PHASE SPACE                         29
    do I = 1, N                                                                   30
```

Main loop F03 : autogpuification ?

```
$ FC=nvfortran make -f Makefile.mk
...
nvfortran -g -Wall -Wextra -Werror -tp=native -C -march=native -mtune=native -mavx512f -acc gpu -Minfo=accel  -c -o obj/event.o src/event.f90
nvfortran-Warning-CUDA Fortran or OpenACC GPU targets disables -Mbounds
src/event.f90:
NVFORTRAN-F-0000-Internal compiler error. module:new_dtype, dt nfd     160  (src/event.f90: 144)
NVFORTRAN/x86-64 Linux 24.3-0: compilation aborted
make: *** [Makefile.mk:73 : obj/event.o] Erreur 2
```

Capture d'électrons & supernovae

Groupe théorie IPNO

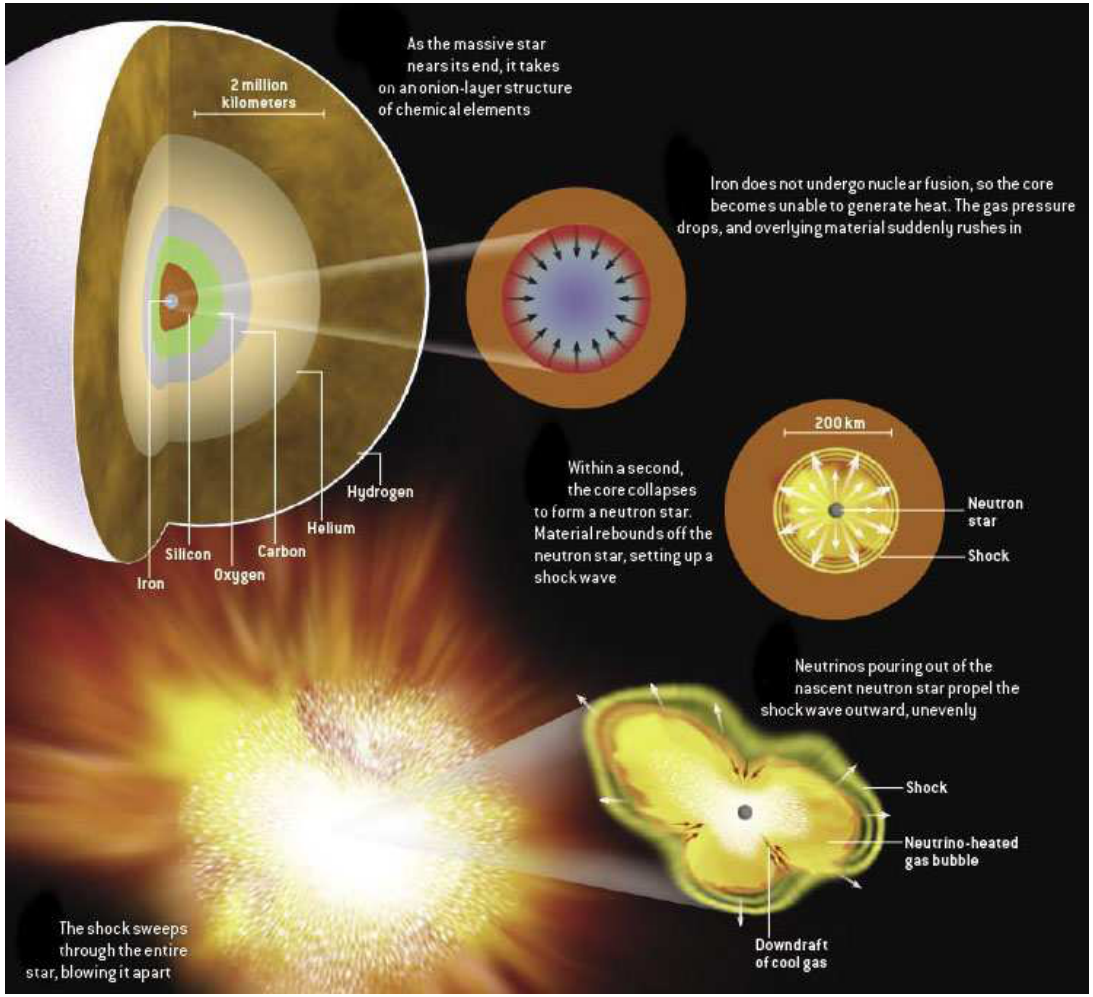
Simulation :

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 sphé□riques> 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 sphé?riques> 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
```

En guise de conclusion

- un peu déçu
- les bug sont remontés / en cours de remontage à NVidia
- ... mais l'autoparallélisation de GrayScott fonctionne
- ... mais l'autogpufication de GrayScott fonctionne
- l'exploration se poursuit sur le code de capture d'électron
  - ...autoparallélisation
  - ...autogpufication