GPUification avec OpenAcc

section efficace de capture d'électrons dans les supernovæ

Vincent C. Lafage

[1]D2I, Institut de Physique Nucléaire
Université Paris-Sud

21 juin 2018

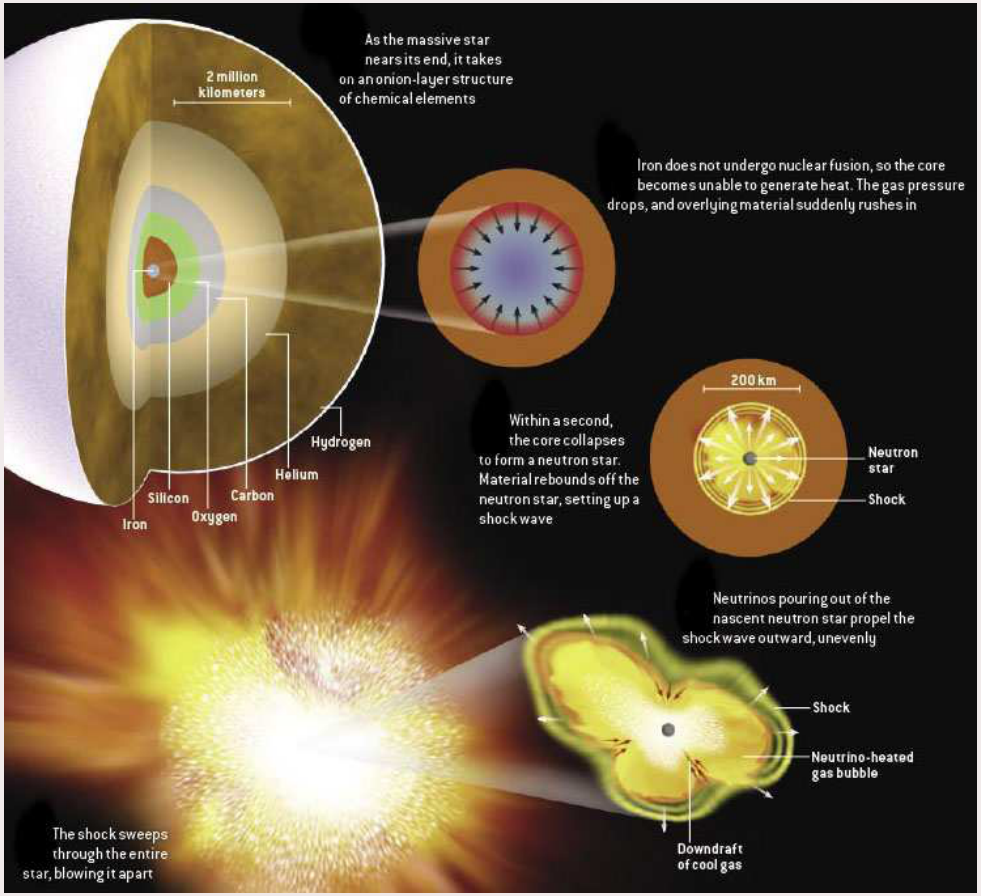
電子捕獲と超新星

IPNO の理論部
計算天体物理学
恒星内元素合成：
中性子星になるとき
$e^- + p \to n + \nu_e$?

$\Rightarrow$

$e^- + {}^{A}_{Z} \to {}^{A}_{Z-1} + \nu_e$

千原子核の種類
温度：[0.5, 5] MeV,
50 ステップ
$\Rightarrow$ CPU 千年！



$\Rightarrow$ スピードアップ $>$ **55**

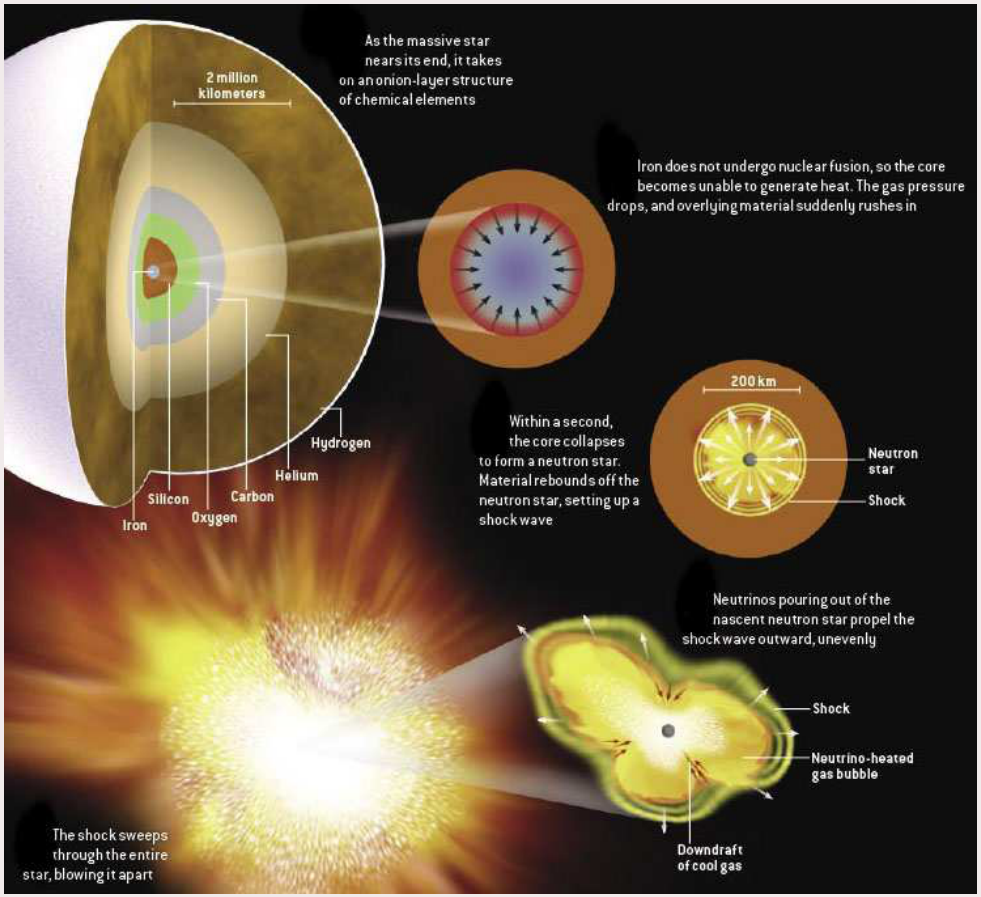
$\Rightarrow$ 後で、**CPU 20** 年だけ

逆原子核反応も大切：　$\nu_e + {}^{A}_{Z-1} \to {}^{A}_{Z} + e^-$

Capture d'électrons & SuperNovæ

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

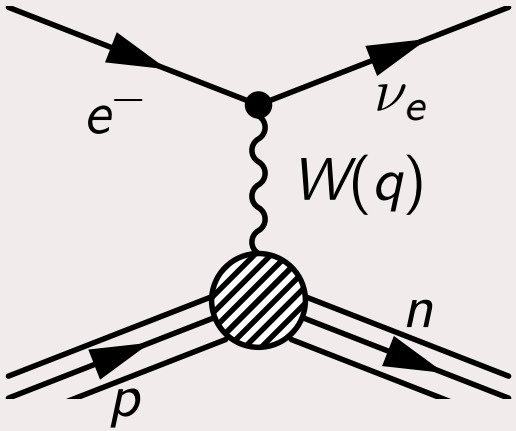
* *profiling*
* audit bibliothèques
* gestion mémoire
* vectorisation



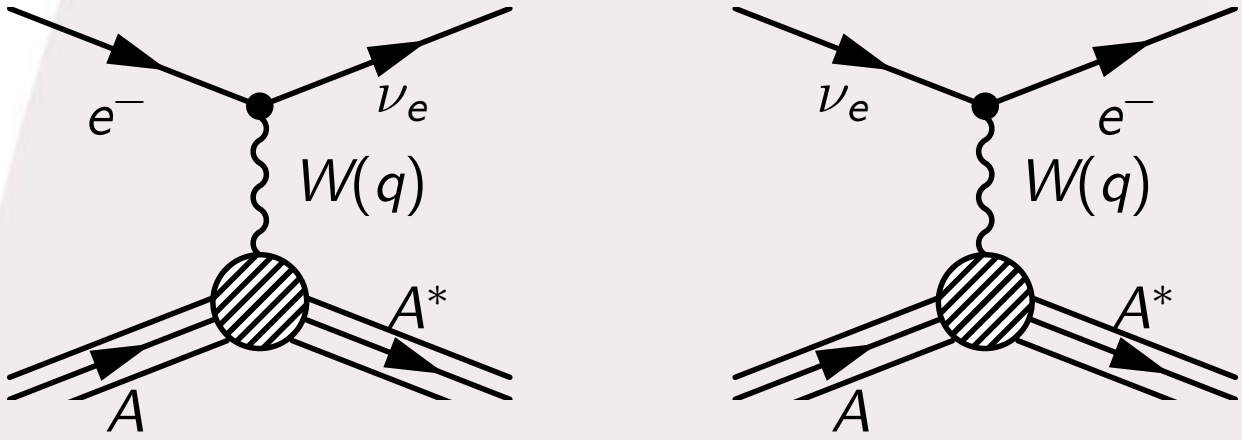
$\Rightarrow$ **facteur $>$ 55**

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

Processus



Processus effectif



$$< p_e, n_{pA}|\mathcal{M}|\cos\theta_\nu, n_{nA^*}, r; J^\pi; T > 0$$

$$< p_e, n_{pA}(J^\pi; T)|\mathcal{M}|\cos\theta_\nu, n_{nA^*}(J^\pi; T), r >$$

$$\forall J^\pi, T \quad \sigma \quad \propto \quad \int \mathrm{d}p_e \mathrm{d}\cos\theta_\nu \mathrm{d}r \sum_{n_{pA}, n_{nA^*}} |< p_e, n_{pA}|\mathcal{M}|\cos\theta_\nu, n_{nA^*}, r >|^2$$

Structure du programme

```
(... boucles sur J, P ...)                ! 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
            do i = 1, maxr                ! 168 mailles radiales de fonction d'onde
               ··· à fond de boucles, fonctions de bessel sphériques : > 92% du temps
```

$$j_\ell(qr)$$

Bessel Sphérique

$$
\begin{aligned}
j_0(z) &= \frac{\sin z}{z} \\
j_1(z) &= \frac{\sin z}{z^2} - \frac{\cos z}{z} \\
j_2(z) &= \left(\frac{3}{z^3} - \frac{1}{z}\right)\sin z - \frac{3}{z^2}\cos z \\
&\dots
\end{aligned}
$$

Plus une fonction a de zéros, plus elle est difficile à évaluer
Là, une infinité...

Benchmark de Bessel Sphérique

langage	algo	valeur	correct ?	-O3 (ns)
Mathematica		0.06596800707652196074158...		
C++	`g++ tr1::sph_bessel_L`	0.0659680070765219607 530	62	794
C++	`boost::sph_bessel`	0.06596800707652196 45	53	
C	`gsl_sf_bessel_jl` = ROOT	0.06596800707652196	56	1065
C	`NR sphbes` $\mu=10^{-1}$ rtpio+O3	0.06596800707652196	56	559
C++	`g++ tr1::sph_bessel`	0.0659680070765219 367	51	661
Fortran	`SPHBR`	0.06596800707652196 4	54	3240
Fortran	`SPHBES`	0.0659680070765219 51	52	747
Fortran	`SPHBES` $\mu=10^{-1\circ}$ rtpio+	0.06596800707652 67385 ???	52	669
Fortran	`SBESJY`	0.06596800707652 2006	50	439
Fortran	`SBESJH`	0.06596800707652196 4	54	3130
Fortran	`SBESJH`	0.06596800707652 2020	50	
Fortran	`SBESJ`	0.0659680070765219 51	52	440
C	`NR sphbes` $\mu=10^{-1\circ}$ rtpio+	0.06596800707652 67385	43	699
C	**NR sphbes** $\mu=10^{-1\circ}$	**0.06596800 511243080**	**25**	**699**

DSE Bessel sphérique

CUDA

$$j_n(z) = z^n \sum_{k=0} \frac{(\frac{1}{2}z^2)^k}{k!(2n+2k+1)!!}$$

```
module bessel
  use, intrinsic :: ISO_C_BINDING
  implicit none
  integer, parameter :: pr = c_double  ! precision
  integer, parameter :: &
       nmax = 19, &
       lmax = 10
  integer(kind=c_int), protected, bind (C, NAME = "lsize")    :: C_lsize   = lmax
  integer(kind=c_int), protected, bind (C, NAME = "nsize")    :: C_nsize   = nmax

  integer, parameter :: ep = 16
  real (ep), parameter :: &
       Qpi    = 4 * atan (1.0_ep), &
       Qlog2  = log (2.0_ep)

  real (c_double), dimension (0:lmax, 0:nmax), save, protected, target, bind (C, NAME = "coef
       finecoeff = reshape (source = real (exp (- (/ &
       ((log_gamma (n+1.0_ep) + &
         log_gamma (n+l+1.5_ep) + (2*n+l+1) * Qlog2 - log (Qpi) / 2.0_ep, l = 0, lmax), n = 0
       shape = shape (finecoeff))
  type (C_PTR), save, protected, bind (C, NAME = "coeffBessel0") :: coeffBesselPtr ! = C_LOC
...
```

Constantes

Constante	initiale	Actuelle		Δ	bits	incertitude	bits
TanCab	0.25	0.23125(87)		7.50e-02	3.7	3.76e-03	8.1
neutronmass	939.0	939.565 4133(58)	MeV	6.02e-04	10.7	2.24e-08	27.3
alphainv	137.0	137.035 999 139(31)		2.63e-04	11.9	3.21e-10	32.0
hbc	197.3	197.3269788(12)	MeV.fm	1.37e-04	12.8	2.23e-08	27.3
Vud	0.97419	0.97427(21)		8.21e-05	13.6	2.16e-04	12.2
dblMasse...	0.04823	0.0482264271(32)		7.41e-05	13.7	6.70e-08	24.8
Gf	1.16639e-11	1.166 3787e-11(6)	MeV ^{2}	2.23e-05	15.5	4.29e-05	20.9
protonmass	938.2796	938.272 0813(58)	MeV	8.05e-06	16.9	2.24e-08	27.3
emass	0.511003	0.5109989461(31)	MeV	7.97e-06	16.9	2.15e-08	27.3
clight	2.99792e23	2.99792458e23	fm.s ^{1}	1.53e-06	19.3		

Constantes 定数値

```
module physconst
  use, intrinsic :: ISO_C_BINDING
  implicit none
  real (kind=C_DOUBLE), parameter ::  & ! (2014 CODATA)
       pi           = 4 * atan (1.d0), & !  , Archimedes' constant
       sqrtpi       = sqrt (pi),       & ! √
       sq4pi        = sqrt (4 * pi),   & ! √(4 )
       Gf           = 1.1663787d-11,   & ! GF, Fermi constant ! 1.166 3787(6) × 10 ¹¹ MeV ²
       hbc          = 197.3269788d0,   & !                    ! 197.326 9788(12) MeV ⊠ fm
       alphainv     = 137.035999139d0, & ! 1/                 ! 137.035 999 139(31)
       protonmass   = 938.2720813d0,   & ! m                  ! 938.272 0813(58) MeV
       neutronmass  = 939.5654133d0,   & ! m                  ! 939.565 4133(58) MeV
       TanCab       = 0.23129d0,       & ! tan  Cab, Cabibbo angle => 0.22534/0.97427
       thetaC       = atan (TanCab),   & !  Cab, Cabibbo angle
       emass        = 0.5109989461d0,  & ! m                  ! 0.510 998 9461(31) MeV
       clight       = 2.99792458d23,   & !   speed of light in vacuum ! 2.99792458 × 10²³ fm ⊠ s
       Vud          = 0.97427d0,       & ! V , Cabibbo-Kobayashi-Maskawa up-down coupling matr
       gv           = 1.0d0,           & ! g
       Na           = 6.022140857d23,  & ! N , Avogadro constant ! 6.022 140 857(74) × 10²³  ⊠
       ...
       factco       = (Gf * cos (thetaC) * hbc)**2 * 1d16    ! factor for cross sections
       ...
  ! Error: PARAMETER attribute conflicts with BIND(C) attribute
  real(kind=C_DOUBLE), protected, bind (C, NAME = "hqc")   :: C_hqc   = hbc          ! 197.32
       ...
```

Success story

- Code de capture d'électrons
  9 kSLOC de `Fortran`, 5 kSLOC de `C(++)`
- Accélération avant $//^n$
  - facteur 12 (stockage plutôt que recalcul)
  - facteur **25** $\Rightarrow$ **55** au total
    - utiliser les bibliothèques standard (`nearbyint`)
    - Spherical Bessel Benchmark (15 codes, 9 algos)
    - vectorisation des boucles...
- amélioration de la précision

Méthode

Ces résultats sont le fruit d'une méthode :

$\Rightarrow$ on ne rentre pas dans 15 000 lignes de code comme dans un moulin : mise du code sous controle de version `svn`

- analyse statique `ftncheck`, `cppchecker`
  - branches mortes (procédures, variables)
  - métriques `sloccount`
- analyse dynamique "profiling" `gprof`
  - identification des goulets d'étranglements
  - optimisation de fond de boucles
- chasse aux problèmes de mémoire `valgrind`
- typographie, indentation, documentation `doxygen`
- pêche aux mauvaises pratiques numériques

  D. Goldberg, *What every computer scientist should know about floating point arithmetic*
  - constantification des constantes : extraction des constantes en dur, uniformisation : combien de valeurs de π distinctes ?
  - accélération (Horner, Richardson, stockage intermédiaire...)

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?

Expression //me

```
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))

  build10: do maj = Jmin, Jmax
    build11: do idxEnergy = 0, nbenergystep-1
      build12: do energyc = 1, nconf
        if (mask_kinematics (energyc, idxEnergy)) then
          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,
            end do build14
          end do build13
        end if
      end do build12
    end do build11
  end do build10

  Rad_proint1_array (:, :, :, :, :) = Rad_proint1_array (:, :, :, :, :) - Rad_proint2A_array
end subroutine radproint_array
```

Expression //^{me} matmul

```
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))

  build10: do maj = Jmin, Jmax
    build11: do idxEnergy = 0, nbenergystep-1
      build12: do energyc = 1, nconf
        build13: do k_simps = 0, n_simps
          build14: do pairc = 1, nconf
            Rad_proint1_array = reshape (matmul (transpose (wf1dwf2_array (1:maxr, 1:nconf))
                reshape (bess_f_array, (/maxr, (n_simps+1) * nconf * nbenergystep * (maxj+
                (/nconf, (n_simps+1), nconf, nbenergystep, (maxj+2)/))

          end do build14
        end do build13
      end do build12
    end do build11
  end do build10

  Rad_proint1_array (:, :, :, :, :) = Rad_proint1_array (:, :, :, :, :) - Rad_proint2A_array
end subroutine radproint_array
```

Expression //^{me} matmul

```
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))

  build10: do maj = Jmin, Jmax
    build11: do idxEnergy = 0, nbenergystep-1
      build12: do energyc = 1, nconf
        if (mask_kinematics (energyc, idxEnergy)) then
          build13: do k_simps = 0, n_simps
            build14: do pairc = 1, nconf
              Rad_proint1_array (1:nconf, 0:n_simps, energyc, idxEnergy, maj) = &
                   matmul (transpose (wf1dwf2_array (1:maxr, 1:nconf)), &
                   Bess_f_array (1:maxr, 0:n_simps, energyc, idxEnergy, maj))
            end do build14
          end do build13
        end if
      end do build12
    end do build11
  end do build10

  Rad_proint1_array (:, :, :, :, :) = Rad_proint1_array (:, :, :, :, :) - Rad_proint2A_array
end subroutine radproint_array
```

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))

  !$acc data present_or_create (mask_kinematics, Rad_proint2B_array, Rad_proint2A_array, Ra
  !$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,
            end do build14
          end do build13
        end if
      end do build12
    end do build11
  end do build10
  !$acc end parallel loop

  !$acc parallel
  Rad_proint1_array (:,:,:,:,:) = Rad_proint1_array (:,:,:,:,:) - Rad_proint2A_array
```

Speedup

	2013 Intel	r88 Intel	r153 F90 Intel	r242 new Intel	Intel ×	GNU	GNU ×	PGI	PGI ×	PGI openAcc
ftchrpa	39,31	39,31	39,31	39,31	39,14	83,19	83,15	69,35	69,27	69,41
eccalc	502,05	10,21	19,17	4,66	5,54	10,90	6,02	5,28	5,95	4,66
total	541	49,52	58,47	44,01	44,73	94,14	89,21	83,22	83,88	82,67
×eccalc		49,2		107,8	90,7		83,4		84,4	107,8
×eccalcr.				2,2	1,8		1,7		1,7	2,2

Précision

- simple, double, étendue, quadruple
- généricité Fortran fondée sur `REAL(KIND=`*global_parameter*`)`
- différentes performances selon le compilo
  reste à faire du coté GNU (pas de vectorisation)
- pas testé sous `OpenAcc`

Conclusion

- différentes précision (simple, double, étendue, quadruple)
- différentes performances selon le compilo
- pénalité d'abstraction de `Fortran+C` vers `Fortran90`

\+ Tesla M2090 $\overset{2011}{\Rightarrow}$ Tesla V100 $\times 7,7$

– `cc20`$\Rightarrow$`cc70` (no profiling)

– `matmul` !!! `cuBLAS` !!!

– mémoire... (seulement 5 GB c'est limite $\Rightarrow$ 16 GB)
gestion plus fine des régions mémoires

\+ nouvel `OpenMPification`