Anciens codes, nouveaux programmes (?)

Thierry Dumont

Institut Camille Jordan
UMR CNRS 5208.

Canum 2014.

Faut il revisiter (toutes) les vieilles briques logicielles à la lumière de l'évolution du matériel et des outils et méthodes de programmation ?

Exemple des solveurs d'équations différentielles.

Faut il revisiter (toutes) les vieilles briques logicielles à la lumière de l'évolution du matériel et des outils et méthodes de programmation ?

Exemple des solveurs d'équations différentielles.

Travail réalisé dans le cadre de :

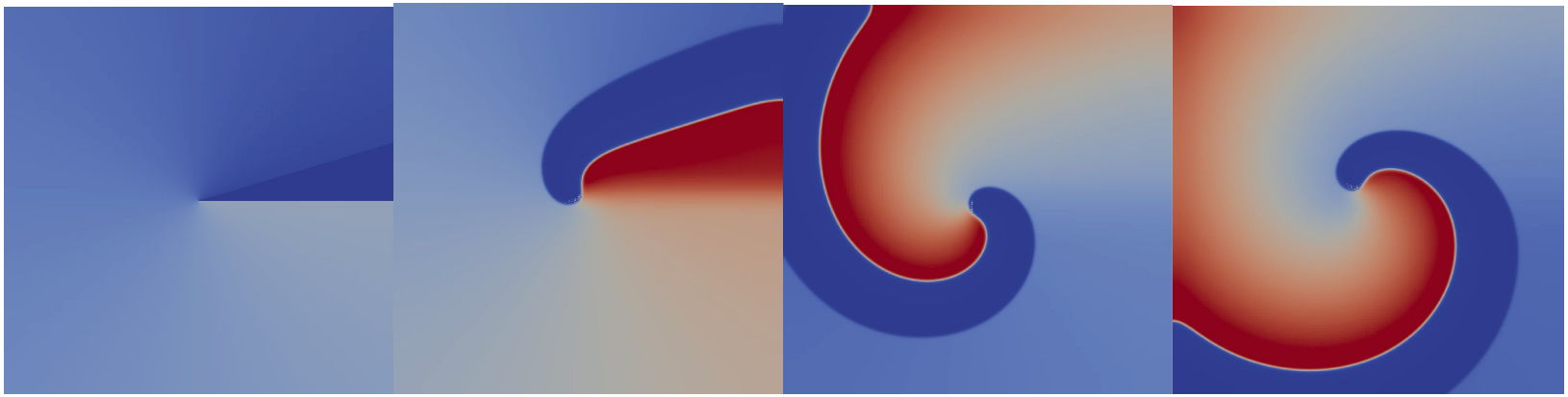
- ANR Sechelles.
- Peps Amies (collaboration avec Intel).

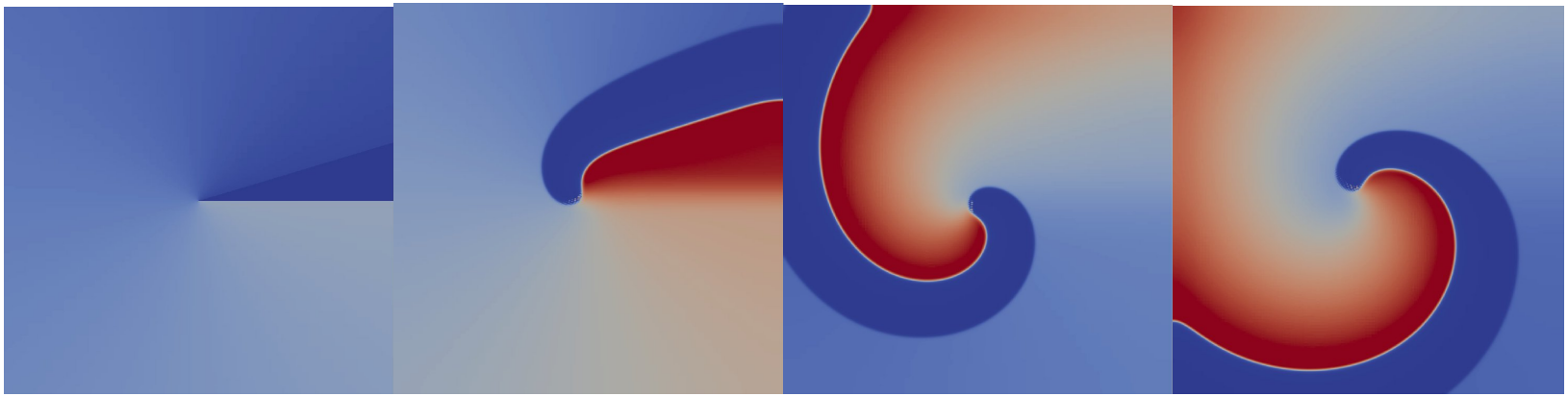
(Grands) systèmes de Réaction–Diffusion **raides** .

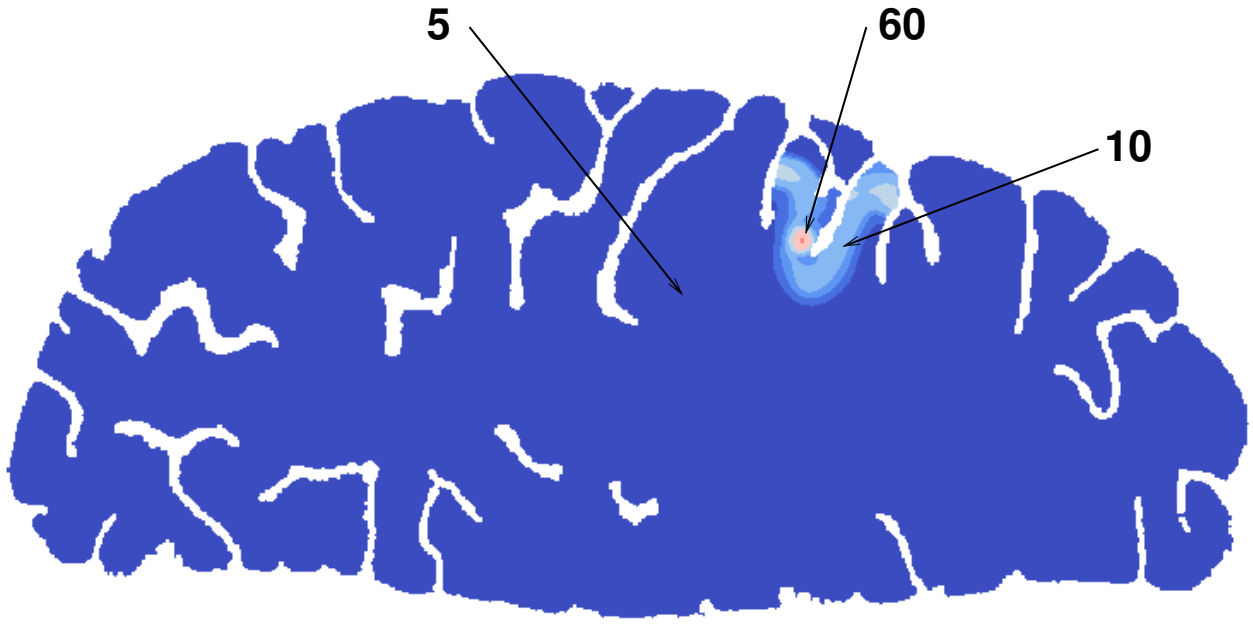
$$\frac{\partial u_i}{\partial t}(x,t) - \mathrm{div}(\varepsilon_i(x)\mathrm{grad}\ u_i(x,t)) = f_i(u_1(x,t),\ldots,u_m(x,t))$$

$$i = 1, m.$$

1. $m = 3,\ 20\ldots100\ldots$ (chimie simplifiée, AVC, chimie complexe), en dimension 2 ou 3.
2. Le système d'EDOs $du_i/dt = f_i(u_1,\ldots,u_m)$ est raide (échelles de temps dans un rapport $1/10^8$).
3. Il faut mailler finement !






5
60
10

Splitting, directions alternées

$$\frac{dU}{dt} = A_\varepsilon U + F(U), \tag{1}$$

$$\frac{dV}{dt} = A_\varepsilon V \quad \text{et} \quad \frac{dW}{dt} = F(W).$$

$D_{\Delta t} V_0$ et $R_{\Delta t} W_0$.

Splitting, directions alternées

$$\frac{dU}{dt} = A_\varepsilon U + F(U), \tag{1}$$

$$\frac{dV}{dt} = A_\varepsilon V \quad \text{et} \quad \frac{dW}{dt} = F(W).$$

$D_{\Delta t} V_0$ et $R_{\Delta t} W_0$.

Schéma de Strang :

$U_{n+1} = R_{\Delta t/2} \circ D_{\Delta t} \circ R_{\Delta t/2} U_n$.

Splitting, directions alternées

$$\frac{dU}{dt} = A_\varepsilon U + F(U), \tag{1}$$

$$\frac{dV}{dt} = A_\varepsilon V \quad \text{et} \quad \frac{dW}{dt} = F(W).$$

$D_{\Delta t} V_0$ et $R_{\Delta t} W_0$.

Schéma de Strang :

$U_{n+1} = R_{\Delta t/2} \circ D_{\Delta t} \circ R_{\Delta t/2} U_n$.

$R_{\Delta t/2} U_n$: résoudre un très grand nombre de systèmes d'EDOs raides, indépendants .

(Ce sont des problèmes de Cauchy).

On consulte la bible

(Hairer & Wanner : Solving Ordinary Differential Equations II, Stiff and Differential-Algebric Problems).

On consulte la bible

(Hairer & Wanner : Solving Ordinary Differential Equations II, Stiff and Differential-Algebric Problems).

Il faut des méthodes A-stables (et mieux L-stables), d'ordre > 2, donc implicites.

- Runge–Kutta diagonalement implicites.
- Méthodes de type Rosenbrock.
- Euler Implicite Extrapolé (Seulex). $\star\star$
- Runge–Kutta implicites : Radau5. $\star\star\star$

On teste les programmes des auteurs, sur des problèmes significatifs $\Rightarrow$ Radau5 l'emporte, de tous les points de vue (performances et précision).

On teste les programmes des auteurs, sur des problèmes significatifs => Radau5 l'emporte, de tous les points de vue (performances et précision).

Radau5

- Collocation aux 3 points de Radau => méthode RK implicite, d'ordre 5, L-stable.
- Estimateur d'erreur et contrôle du pas.
- Perte d'ordre (3) pour des problèmes très raides.

On teste les programmes des auteurs, sur des problèmes significatifs => Radau5 l'emporte, de tous les points de vue (performances et précision).

Radau5

- Collocation aux 3 points de Radau => méthode RK implicite, d'ordre 5, L-stable.
- Estimateur d'erreur et contrôle du pas.
- Perte d'ordre (3) pour des problèmes très raides.

Sur les problèmes qui nous intéressent, la résolution de la Réaction prend jusqu'à 90 % du temps calcul !

On teste les programmes des auteurs, sur des problèmes significatifs => Radau5 l'emporte, de tous les points de vue (performances et précision).

Radau5

- Collocation aux 3 points de Radau => méthode RK implicite, d'ordre 5, L-stable.
- Estimateur d'erreur et contrôle du pas.
- Perte d'ordre (3) pour des problèmes très raides.

Sur les problèmes qui nous intéressent, la résolution de la Réaction prend jusqu'à 90 % du temps calcul !

Le code utilisé est-il améliorable ?

Radau5 : comment ça marche.

1. Résolution d'un système non-linéaire de taille $3m$: Newton simplifié.
   Calcul de la matrice Jacobienne, exactement ou numériquement.
   A chaque itération :
   - Résoudre un système de taille m et un de taille $2m$.
   - Algèbre linéaire spécialisée (Jacobien plein ou bande, matrice de masse éventuelle).
2. Estimateur d'erreur (normes, tests, branchements).

Radau5 : comment ça marche.

1. Résolution d'un système non-linéaire de taille $3m$: Newton simplifié.
   Calcul de la matrice Jacobienne, exactement ou numériquement.
   A chaque itération :
   - Résoudre un système de taille m et un de taille $2m$.
   - Algèbre linéaire spécialisée (Jacobien plein ou bande, matrice de masse éventuelle).
2. Estimateur d'erreur (normes, tests, branchements).

Ensemble extrêmement astucieux et délicat !

Radau5 : comment ça marche.

1. Résolution d'un système non-linéaire de taille $3m$: Newton simplifié.
   Calcul de la matrice Jacobienne, exactement ou numériquement.
   A chaque itération :
   - Résoudre un système de taille m et un de taille $2m$.
   - Algèbre linéaire spécialisée (Jacobien plein ou bande, matrice de masse éventuelle).
2. Estimateur d'erreur (normes, tests, branchements).

Ensemble extrêmement astucieux et délicat !

Un an de travail !
Écrit à l'origine pour montrer que Radau5 ... n'est pas une bonne méthode !

E. Hairer, communication personnelle.

Radau5 : code

C'est du Fortran 77 ! Interface calquée sur LSODE.

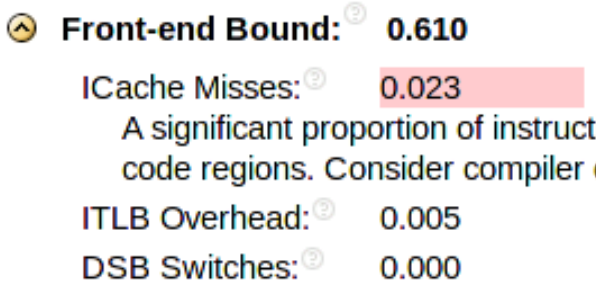
```
      SUBROUTINE RADAU5(N,FCN,X,Y,XEND,H,
     &                  RTOL,ATOL,ITOL,
     &                  JAC  ,IJAC,MLJAC,MUJAC,
     &                  MAS  ,IMAS,MLMAS,MUMAS,
     &                  SOLOUT,IOUT,
     &                  WORK,LWORK,IWORK,LIWORK,RPAR,IPA
```

1100 lignes de code, très dense.

Problèmes test :

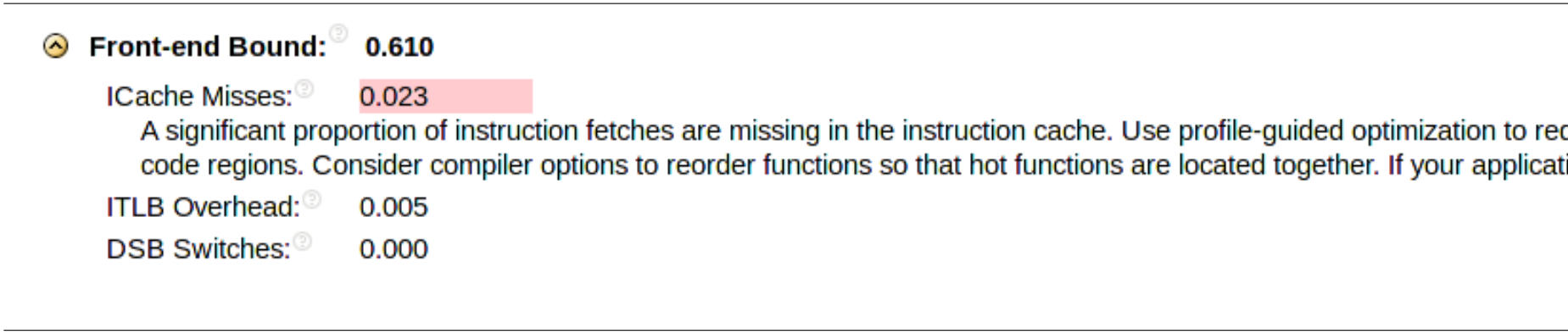
1. Oregonator : $m = 3$ équations, raide. Jacobien analytique ou numérique.
2. AVC : $m = 21$, f très complexe et raide (sigmoïdes). Jacobien numérique.
3. Réaction–Diffusion en dimension 1. KPP : $f(u) = ku^2(1 - u)$. $m = 1000$, peu raide. Jacobien analytique.

Mesures avec VTune.



On trouve aussi un CPI rate de 1.3.

Mesures avec VTune.



On trouve aussi un CPI rate de 1.3.

Exemple de construction pénalisante :

GOTO (1,2,3,4,5,6,7,55,55,55,11,12,13,14,15), IJOB

`IJOB` est un switch pour l'algèbre linéaire (plein, bande, Hessenberg...etc.).

Réécriture complète

Ingrédients :

- C++.
- *Templatisation* des objets : différents types de matrices etc.

```
template<class Fonct> class Radau5cc:
  private Matrices<(Fonct::n-Fonct::nsub)==1&&(Fonct::n-Fonct::nsup)==1,
   Fonct::Hessenberg, Fonct::n, Fonct::nsub, Fonct::nsup>,
  compat<(Fonct::n-Fonct::nsub)==1&&(Fonct::n-Fonct::nsup)==1,
 Fonct::Hessenberg>
```

- Description du problème (m, f, Df) sous forme de classe et d'objets fonctions inlinés.
- Alignements soigneux.
- Vectorisation.
- Objets constants là où c'est possible.

$=>$ minimisation des branchements, déroulements de boucles etc...

Réécriture complète

Jacobien numérique :

`Pour i=1 à m faire:`

$$DF_{*,i} = \varepsilon^{-1}(f(x + \varepsilon 1_i) - f(x))$$

Vectorisation de la boucle quand c'est possible.

La bibliothèque MKL a des versions vectorielles de certaines fonctions élémentaires.

Réécriture complète

Jacobien numérique :

`Pour i=1 à m faire:`

$$DF_{*,i} = \varepsilon^{-1}(f(x + \varepsilon 1_i) - f(x))$$

Vectorisation de la boucle quand c'est possible.

La bibliothèque MKL a des versions vectorielles de certaines fonctions élémentaires.

Qu'est-ce qu'on gagne ?

Qu'est-ce qu'on gagne ?

Très dépendant du problème.

1. « Petits » problèmes : Oregonator, raide, $m = 3$.
2. Problèmes avec 20 équations ou plus (AVC).
3. « Grands » problèmes (KPP : 1000 équations).

Qu'est-ce qu'on gagne ?

Très dépendant du problème.

1. « Petits » problèmes : Oregonator, raide, $m = 3$.
2. Problèmes avec 20 équations ou plus (AVC).
3. « Grands » problèmes (KPP : 1000 équations).

1. $T_{Radau5NG}/T_{Radau5} \simeq 0.8$.
2. Idem.
3. Un peu mieux...mais pas très réaliste !

Qu'est-ce qu'on gagne ?

Très dépendant du problème.

1. « Petits » problèmes : Oregonator, raide, $m = 3$.
2. Problèmes avec 20 équations ou plus (AVC).
3. « Grands » problèmes (KPP : 1000 équations).

1. $T_{Radau5NG}/T_{Radau5} \simeq 0.8$.
2. Idem.
3. Un peu mieux...mais pas très réaliste !

Beaucoup de travail pour un gain de 20%...

Qu'est-ce qu'on gagne ?

Très dépendant du problème.

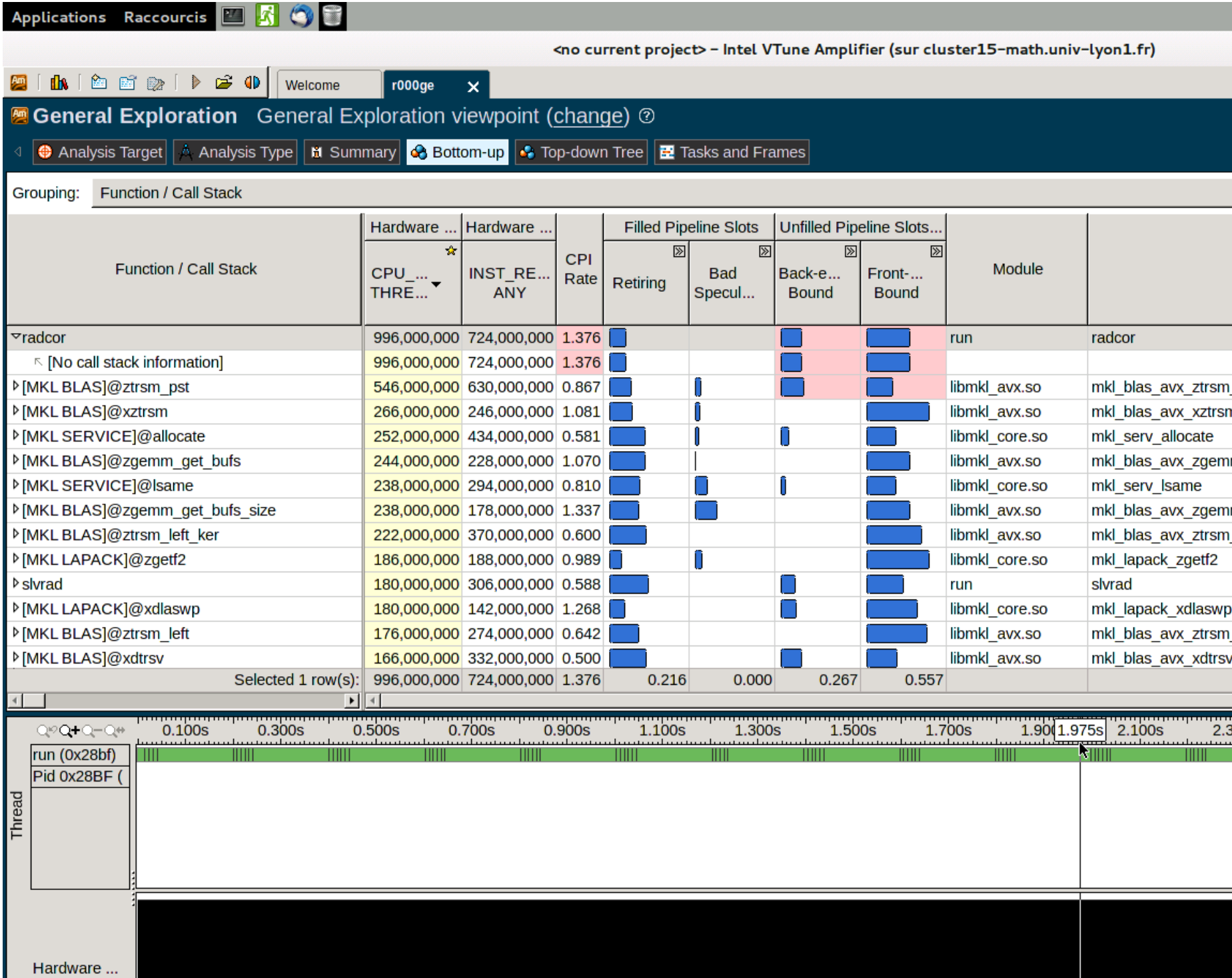
1. « Petits » problèmes : Oregonator, raide, $m = 3$.
2. Problèmes avec 20 équations ou plus (AVC).
3. « Grands » problèmes (KPP : 1000 équations).

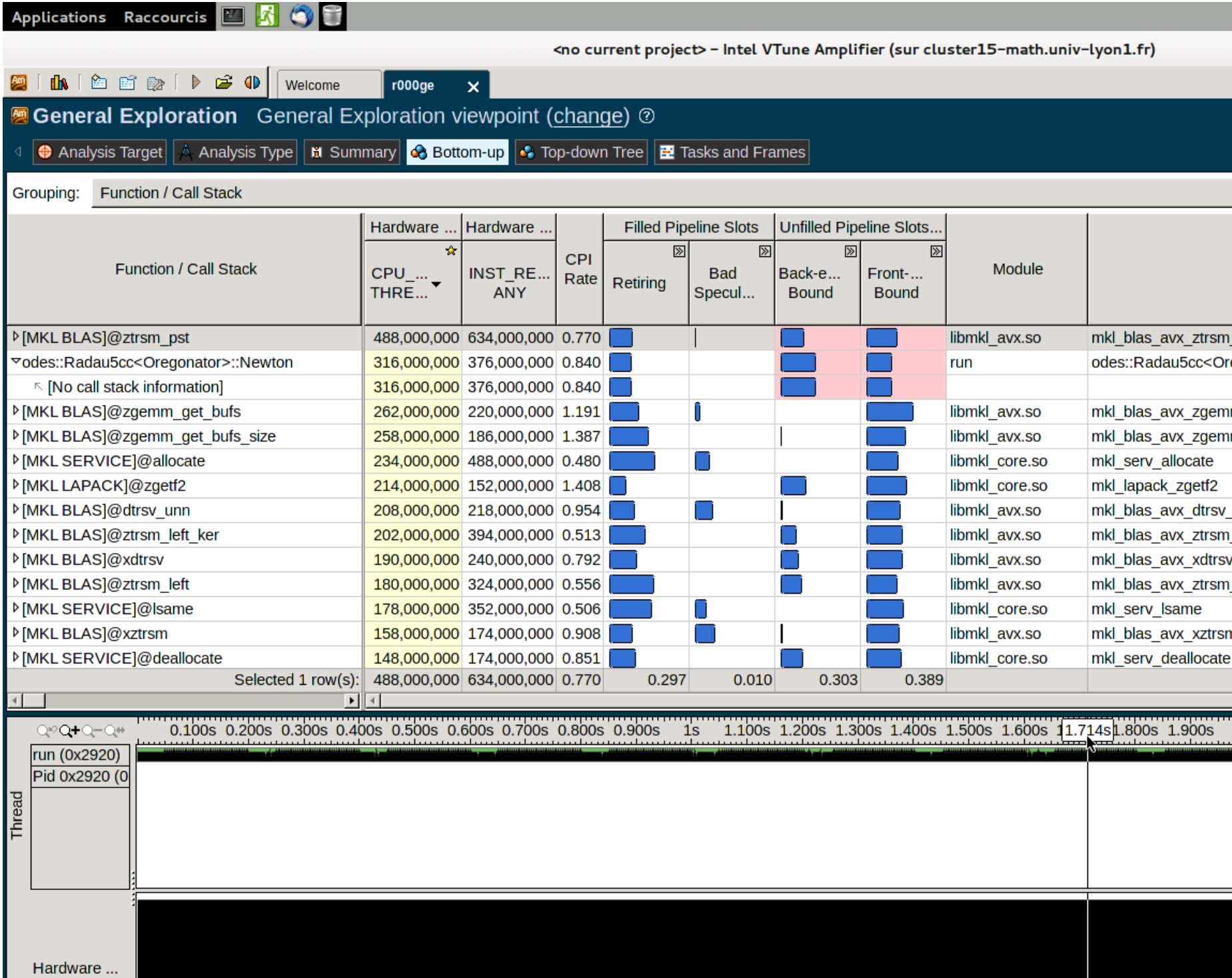
1. $T_{Radau5NG}/T_{Radau5} \simeq 0.8$.
2. Idem.
3. Un peu mieux...mais pas très réaliste !

Beaucoup de travail pour un gain de 20%...

Est-ce définitif ?

Mesures pour l'Oregonator.


Applications Raccourcis
<no current project> – Intel VTune Amplifier (sur cluster15-math.univ-lyon1.fr)
Welcome
r000ge
General Exploration
General Exploration viewpoint (change)
Analysis Target
Analysis Type
Summary
Bottom-up
Top-down Tree
Tasks and Frames
Grouping: Function / Call Stack
Function / Call Stack
Hardware ... CPU_... THRE...
Hardware ... INST_RE... ANY
CPI Rate
Filled Pipeline Slots
Retiring
Bad Specul...
Unfilled Pipeline Slots...
Back-e... Bound
Front-... Bound
Module
radcor 996,000,000 724,000,000 1.376 run radcor
[No call stack information] 996,000,000 724,000,000 1.376
[MKL BLAS]@ztrsm_pst 546,000,000 630,000,000 0.867 libmkl_avx.so mkl_blas_avx_ztrsm_
[MKL BLAS]@xztrsm 266,000,000 246,000,000 1.081 libmkl_avx.so mkl_blas_avx_xztrsm
[MKL SERVICE]@allocate 252,000,000 434,000,000 0.581 libmkl_core.so mkl_serv_allocate
[MKL BLAS]@zgemm_get_bufs 244,000,000 228,000,000 1.070 libmkl_avx.so mkl_blas_avx_zgemm
[MKL SERVICE]@lsame 238,000,000 294,000,000 0.810 libmkl_core.so mkl_serv_lsame
[MKL BLAS]@zgemm_get_bufs_size 238,000,000 178,000,000 1.337 libmkl_avx.so mkl_blas_avx_zgemm
[MKL BLAS]@ztrsm_left_ker 222,000,000 370,000,000 0.600 libmkl_avx.so mkl_blas_avx_ztrsm_
[MKL LAPACK]@zgetf2 186,000,000 188,000,000 0.989 libmkl_core.so mkl_lapack_zgetf2
slvrad 180,000,000 306,000,000 0.588 run slvrad
[MKL LAPACK]@xdlaswp 180,000,000 142,000,000 1.268 libmkl_core.so mkl_lapack_xdlaswp
[MKL BLAS]@ztrsm_left 176,000,000 274,000,000 0.642 libmkl_avx.so mkl_blas_avx_ztrsm_
[MKL BLAS]@xdtrsv 166,000,000 332,000,000 0.500 libmkl_avx.so mkl_blas_avx_xdtrsv
Selected 1 row(s): 996,000,000 724,000,000 1.376 0.216 0.000 0.267 0.557
0.100s 0.300s 0.500s 0.700s 0.900s 1.100s 1.300s 1.500s 1.700s 1.975s 2.100s
Thread
run (0x28bf)
Pid 0x28BF (
Hardware ...


Applications Raccourcis
<no current project> – Intel VTune Amplifier (sur cluster15-math.univ-lyon1.fr)
Welcome
r000ge
General Exploration
General Exploration viewpoint (change)
Analysis Target
Analysis Type
Summary
Bottom-up
Top-down Tree
Tasks and Frames
Grouping: Function / Call Stack
Function / Call Stack
Hardware ... CPU_... THRE...
Hardware ... INST_RE... ANY
CPI Rate
Filled Pipeline Slots
Retiring
Bad Specul...
Unfilled Pipeline Slots...
Back-e... Bound
Front-... Bound
Module
[MKL BLAS]@ztrsm_pst 488,000,000 634,000,000 0.770 libmkl_avx.so mkl_blas_avx_ztrsm_
odes::Radau5cc<Oregonator>::Newton 316,000,000 376,000,000 0.840 run odes::Radau5cc<Or
[No call stack information] 316,000,000 376,000,000 0.840
[MKL BLAS]@zgemm_get_bufs 262,000,000 220,000,000 1.191 libmkl_avx.so mkl_blas_avx_zgemm
[MKL BLAS]@zgemm_get_bufs_size 258,000,000 186,000,000 1.387 libmkl_avx.so mkl_blas_avx_zgemm
[MKL SERVICE]@allocate 234,000,000 488,000,000 0.480 libmkl_core.so mkl_serv_allocate
[MKL LAPACK]@zgetf2 214,000,000 152,000,000 1.408 libmkl_core.so mkl_lapack_zgetf2
[MKL BLAS]@dtrsv_unn 208,000,000 218,000,000 0.954 libmkl_avx.so mkl_blas_avx_dtrsv_
[MKL BLAS]@ztrsm_left_ker 202,000,000 394,000,000 0.513 libmkl_avx.so mkl_blas_avx_ztrsm_
[MKL BLAS]@xdtrsv 190,000,000 240,000,000 0.792 libmkl_avx.so mkl_blas_avx_xdtrsv
[MKL BLAS]@ztrsm_left 180,000,000 324,000,000 0.556 libmkl_avx.so mkl_blas_avx_ztrsm_
[MKL SERVICE]@lsame 178,000,000 352,000,000 0.506 libmkl_core.so mkl_serv_lsame
[MKL BLAS]@xztrsm 158,000,000 174,000,000 0.908 libmkl_avx.so mkl_blas_avx_xztrsm
[MKL SERVICE]@deallocate 148,000,000 174,000,000 0.851 libmkl_core.so mkl_serv_deallocate
Selected 1 row(s): 488,000,000 634,000,000 0.770 0.297 0.010 0.303 0.389
0.100s 0.200s 0.300s 0.400s 0.500s 0.600s 0.700s 0.800s 0.900s 1s 1.100s 1.200s 1.300s 1.400s 1.500s 1.600s 1.714s 1.800s 1.900s
Thread
run (0x2920)
Pid 0x2920 (0
Hardware ...