Introduction aux méthodes d'optimisation

Kamel Mazouzi

Mésocentre de calcul de Franche-Comté

UFC
UNIVERSITÉ DE FRANCHE-COMTÉ

mésocentre de calcul de franche-comté

Plan

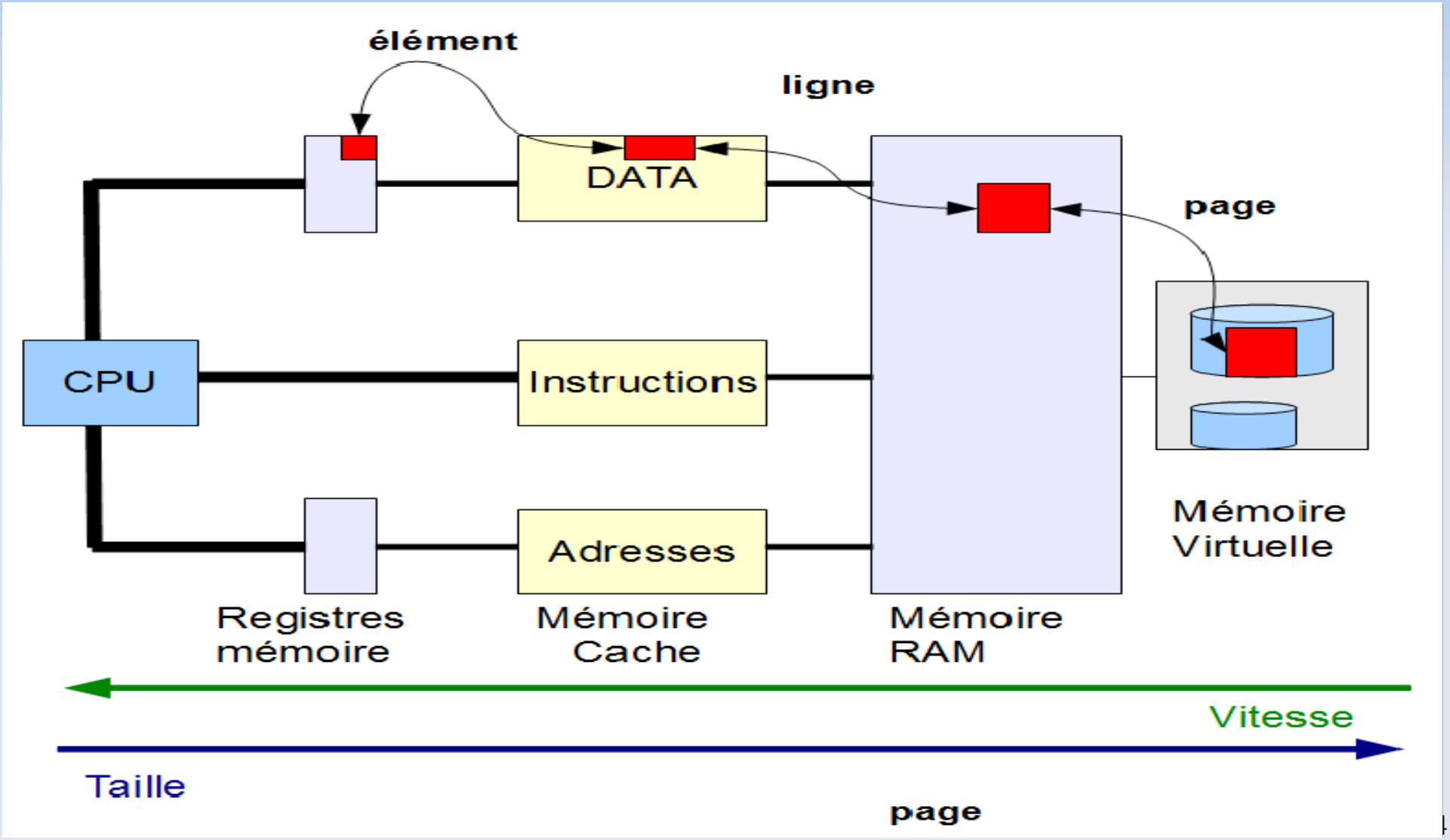
- Introduction
- Techniques d'optimisation
- Débogage
- Profiling
- Bibliothèques scientifiques

Introduction

- Pour optimiser un code de calcul, il faut tenir compte de l'architecture sur laquelle ce dernier sera exécuté
- Le facteur le plus pénalisant en terme de performances est le **manque de données** au niveau du **processeur**
  - les données nécessaires à la réalisation des opérations ne sont pas présentes au voisinage du processeur
  - Accès mémoires réguliers ⇨ **perte de temps (cycle processeur)**

Architecture de Von Neumann

Programme = instructions + données



Architecture de Van Newman : exemple

- Registres
  - Le processeur utilise uniquement les données dans les registres
  - Temps d'accès: 2ns
- Deux niveaux de caches (sur certaines machines 3 niveaux)
  - Cache primaire (L1)
    - Accès par bloc (ligne de cache): 32 bytes
    - Taille : 64KB
    - Temps d'accès: 2 à 3 cycles
  - Cache secondaire (L2)
    - Accès par bloc : 128 bytes
    - Taille : 4MB
    - Temps d'accès : 10 à 15 cycles
- Mémoire principale (taille 2.5GB/proc, latence: 220ns)

Techniques d'optimisation

Techniques d'optimisation

Plusieurs directions à suivre pour optimiser un code, par exemple :

- Choix des algorithmes
- Optimisations des boucles
- Optimisations mémoire
- Optimisations des E/S (entrées/sorties)
- Opérations en virgule flottante

Techniques d'optimisation

Plusieurs directions à suivre pour optimiser un code, par exemple :

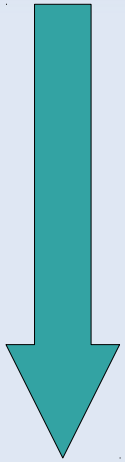
- **Choix des algorithmes**
- Optimisations des boucles
- Optimisations mémoire
- Optimisations des E/S (entrées/sorties)
- Utilisation des librairies scientifiques
- Opérations en virgule flottante

Choix des algorithmes

Calcul de la somme `S` des `n` premiers entiers naturels non nuls

$$\sum S, 0 < s \leq n$$

```
S = 1+2+3..+(n-1)+n
```



```
S = n*(n+1)/2
```

```
function (int borne)
 {
 int som=0;
  for(int i=1;i<=borne;borne++)
   {
    some =some + i;
   }
 return some;
}
```

Choix des algorithmes

Calcul d'une valeur approchée de π

$$t_0 = \frac{1}{\sqrt{3}}$$

$$t_{i+1} = \frac{\sqrt{t_i^2 + 1} - 1}{t_i}$$

deuxième forme

$$t_{i+1} = \frac{t_i}{\sqrt{t_i^2 + 1} + 1}$$

Valeur approchée

$$6.2^i + t_i \, i \to \infty$$

0	3.4641016151377543863	3.4641016151377543863
1	3.2153903091734710173	3.2153903091734723496
2	3.1596599420974940120	3.1596599420975006733
3	3.1460862151314012979	3.1460862151314352708
4	3.1427145996453136334	3.1427145996453689225
5	3.1418730499801259536	3.1418730499798241950
6	3.1416627470548084133	3.1416627470568494473
7	3.1416101765997805905	3.1416101766046906629
8	3.1415970343230776862	3.1415970343215275928
9	3.1415937488171150615	3.1415937487713536668
10	3.1415929278733740748	3.1415929273850979885
11	3.1415927256228504127	3.1415927220386148377
12	3.1415926717412858693	3.1415926707019992125
13	3.1415926189011456060	3.1415926578678454728
14	3.1415926717412858693	3.1415926546593073709
15	3.1415919358822321783	3.1415926538571730119
16	3.1415926717412858693	3.1415926536566394222
17	3.1415810075796233302	3.1415926536065061913
18	3.1415926717412858693	3.1415926535939728836
19	3.1414061547378810956	3.1415926535908393901
20	3.1405434924008406305	3.1415926535900560168
21	3.1400068646912273617	3.1415926535898608396
22	3.1349453756585929919	3.1415926535898122118
23	3.1400068646912273617	3.1415926535897995552
24	3.2245152435345525443	3.1415926535897968907
25		3.1415926535897962246
26		3.1415926535897962246
27		3.1415926535897962246
28		3.1415926535897962246
	The true value is	3.1415926535897932384626433832...

Techniques d'optimisation

Plusieurs directions à suivre pour optimiser un code, par exemple :

- **Choix des algorithmes**
- **Optimisations des boucles**
- Optimisations mémoire
- Optimisations des E/S (entrées/sorties)
- Opérations en virgule flottante

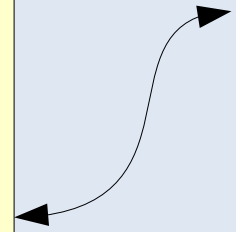
Optimisation des boucles

- En général, les boucles sont responsables d'une grande partie du temps d'exécution
- Pour gagner en performance, on peut appliquer les modifications suivantes :
  - changement de l'ordre des boucles
  - fusion de boucle
  - localité des données
  - ...

Changement de l'ordre des boucles

- Changer l'ordre des boucles imbriquées
- Echanger une boucle externe avec une interne
- Permet de vérifier le principe de localité : éviter à la mémoire cache de chercher des lignes de données d'une itération à une autre
- **Cette transformation dépend de la technique du cache utilisée et de la disposition du tableau dans la mémoire utilisée par le compilateur**

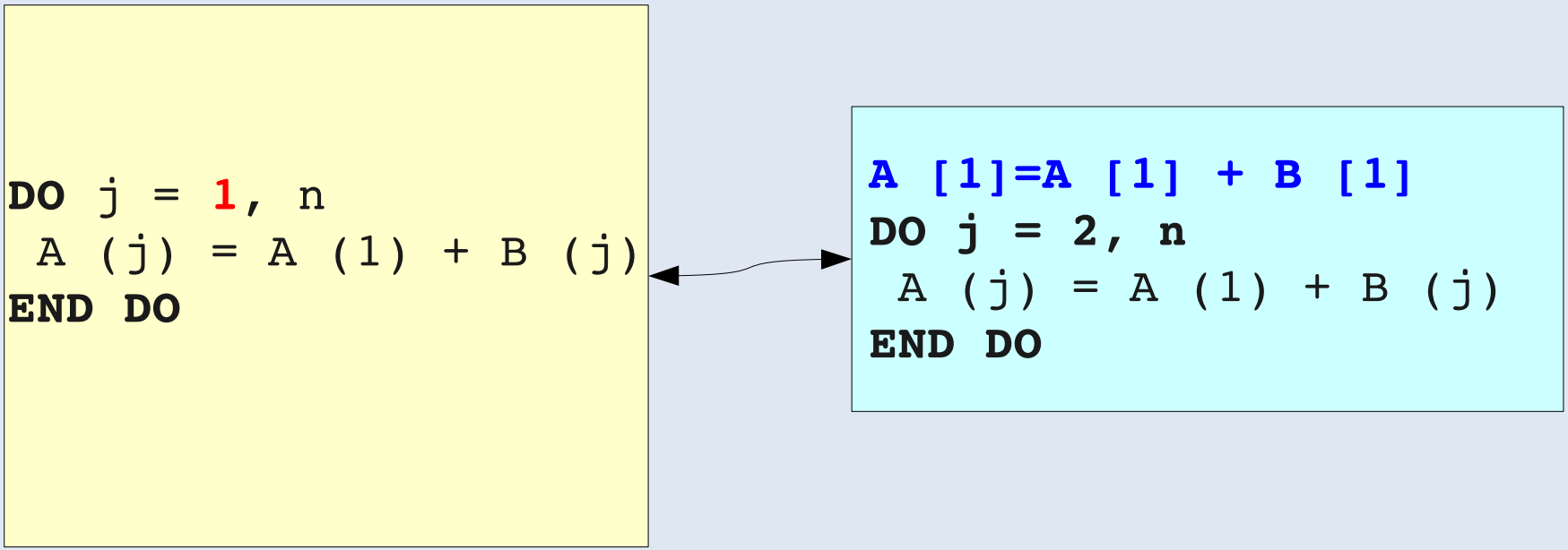
```
for (i=0;i<20;i++)
  for (j=0;j<10;j++)
     A [i,j]= i + j
```



```
for (j=0;j<10;j++)
  for (i=0;i<20;i++)
     A [i,j]= i + j
```

« Loop splitting »

- Division d'une boucle en plusieurs autres boucles ou bien la suppression de dépendance
- Un cas simple et spécial de cette technique consiste à supprimer une première itération complexe de la boucle et de la mettre à l'extérieur de cette dernière



Fusion de boucles

- Fusionne plusieurs boucles en une seule
- Optimise le temps d'exécution du programme

```
int i, a[100], b[100];
for (i = 0; i < 100; i++) {
  a[i] = 1;
}
for (i = 0; i < 100; i++) {
 b[i] = 2;   }
```

```
int i, a[100], b[100];
for (i = 0; i < 100; i++) {
  a[i] = 1;
  b[i] = 2;
}
```

« Loop Unwiding »

- Mise en place de prochaines affectations dans la même itération
- Augmenter le taux de « cache hit »

```
for (int x = 0; x < 100; x++)
{
  delete(x);
}
```

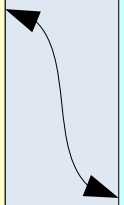
```
for (int x = 0; x < 100; x += 5)
{
 delete(x);
 delete(x+1);
 delete(x+2);
 delete(x+3);
 delete(x+4);
}
```

- Exécute seulement 20 boucles au lieu de 100
- Diminue le nombre d'instructions par un facteur de 1/5

« Loop Unswitching »

- Enlever une structure conditionnelle de l'intérieur de la boucle et de la mettre a l'extérieur
- Utile pour paralléliser une boucle

```
do i=1, 1000
  x[i] = x[i] + y[i];
  if (w) then
   y[i] = 0;
  endif

end do;
```

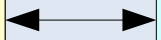


```
if (w) then
  do i=1,1000
    x[i] = x[i] + y[i];
    y[i] = 0;
  end do;
else
  do i=1,1000 do
    x[i] = x[i] + y[i];
  end do
endif;
```

Optimisation des boucles

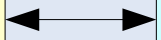
- Enlever les opérations invariantes, i.e. éviter un travail qui n'est pas nécessaire

```
DO j = 1, n
   i = 3
   work (j) = ...
END DO
```



```
i = 3
DO j = 1, n

   work (j) = ...
END DO
```

```
x=r*sin(a)*cos(b)
y=r*sin(a)*sin(b)
z=r*cos(a)
```



```
temp=r*sin(a)
x=temp*cos(b)
y=temp*sin(b)
z=r*cos(a)
```

Techniques d'optimisation

Plusieurs directions à suivre pour optimiser un code, par exemple :

- **Choix des algorithmes**
- **Optimisations des boucles**
- **Optimisations mémoire**
- Optimisations des E/S (entrées/sorties)
- Opération en virgule flottante

Optimisation mémoire

- Aligner les données en mémoire (plus rapide car minimise les accès mémoire)
- Optimiser les accès mémoire : accéder aux tableaux de la même façon qu'ils sont stockés. En Fortran les tableaux sont stockés par colonne, en C par ligne

```
DO i = 1, n

   DO j = 1, n
      A (i,j) = B (i,j) lent
      C (j,i) = D (j,i) rapide
   END DO
END DO
```

```
for (i=1; i<(n+1); i++)
{
    for (j=1; j<(n+1); j++)
    {
       A[i][j] = B[i][j];  rapide
       C[j][i] = D[j][i];  lent
    }
}
```

Techniques d'optimisation

Plusieurs directions à suivre pour optimiser un code, par exemple :

- **Choix des algorithmes**
- **Optimisations des boucles**
- **Optimisations mémoire**
- **Optimisations des E/S (entrées/sorties)**
- **Opération en virgule flottante**

Optimisations des E/S (entrées/sorties)

- Éviter les répertoires surpeuplés et utiliser des chemins d'accès courts : cela réduit les temps de recherche du répertoire
- Éviter les fichiers temporaires et garder les données en mémoire
- Rediriger les E/S du terminal (clavier/écran) car plus lentes que les E/S fichier
- Minimiser les E/S formatées
- Une E/S non formatée transfère les données sans les modifier : la représentation des données est la même que celle en mémoire
- Les E/S non formatées (fichier binaire) sont beaucoup plus rapides que les E/S formatées (facteur pouvant aller jusqu'à 6)

Techniques d'optimisation

Plusieurs directions à suivre pour optimiser un code, par exemple :

- **Choix des algorithmes**
- **Optimisations des boucles**
- **Optimisations mémoire**
- **Optimisations des E/S (entrées/sorties)**
- **Opération en virgule flottante**

Opérations en virgule flottante

- Les opérations qui coûtent le plus cher à effectuer sont les opérations en virgule flottante
- Elles sont effectuées par les unités arithmétiques (FPU)
- L'instruction de calcul de base est la multiplication / addition (**FMA** pour Floating-point Multiply and Add) en double précision
- Plusieurs FMA indépendantes qui se succèdent peuvent être pipelinées

Instruction	Simple précision (nombre de cycles horloge)	Double précision (nombre de cycles horloge)	Pipelinées ?
fma	6	6	oui
fdiv	32	32	non
fsqrt	38	38	non

Opérations en virgule flottante

- Il est important de minimiser la quantité des opérations coûteuses
- Éviter les conversions de type inutiles

```
x = value
DO i = 1, n
   y (i) = z (i) / x
END DO
```

```
x = 1.0_rp / value
DO i = 1, n
   y (i) = z (i) * x
END DO
```

```
REAL U (10)

DO i = 1, 10
   U (i) = 1
END DO
```

```
INTEGER,PARAMETER::rp= kind (1.0)
REAL(rp) U(10)

DO i = 1, 10
   U (i) = 1.0_rp
END DO
```

Opérations en virgule flottante

- Par défaut les constantes sont définies en simple précision en fortran
- Problématique si le programme est sensible à la précision

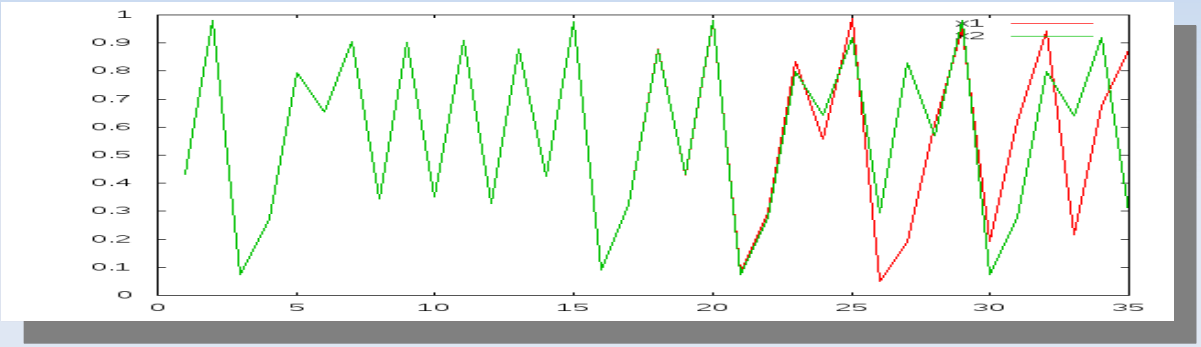
```
program main
double precision::a,b

a=0.123
b=0.123d0

write (*,*),a,b

end program main
```

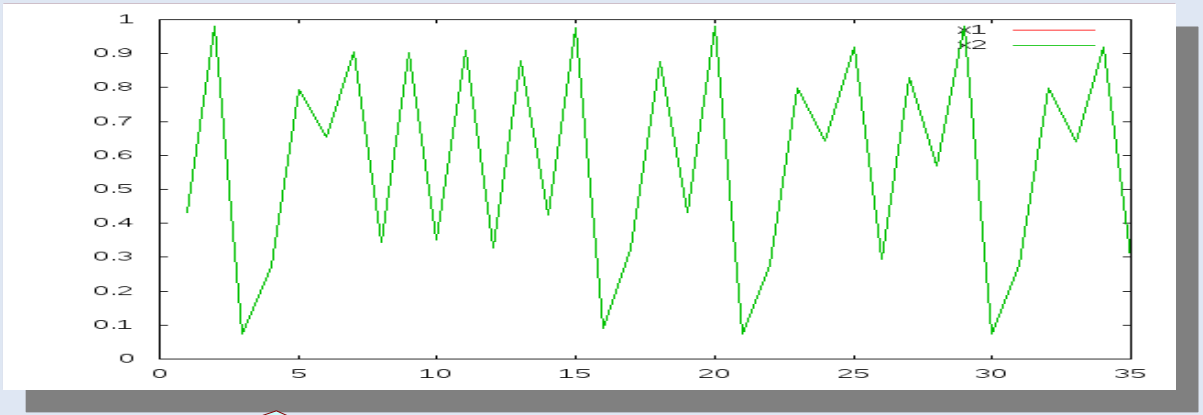
```
0.123000003397465    0.123000000000000
```

Opérations en virgule flottante : exemple

- Exemple d'une suite logistique : $X_{n+1} = 4 \cdot X_n \cdot (1 - x_n)$

```
program main
integer (kind=4)::i
double precision ::x1,x2
x1=0.123
x2=0.123d0
do i=1,35
x1=4.0d0*x1*(1.0d0-x1)
x2=4.0d0*x2*(1.0d0-x2)
write(*,*)i,x1,x2
end do
end program main
```



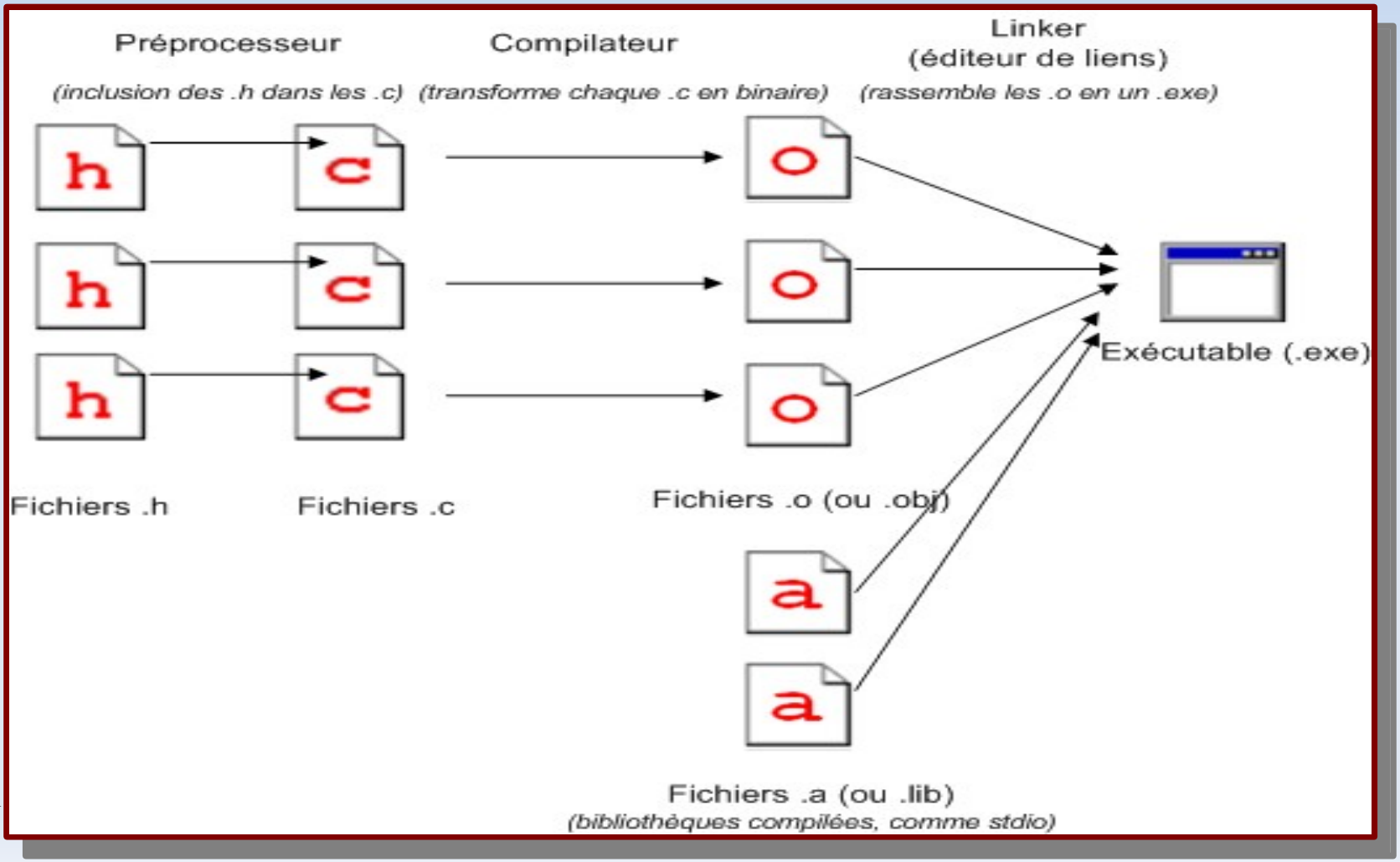


Le compilateur peut m'aider!
On force les real sur 8octets
ifort -r8

Compilation

Compilation

- **Compilation** : processus de transformation d'un code source en un code binaire



Compilateur d'Intel

- Programme séquentiel : ifort, icc
- Programme parallèle : mpif77 ou mpif90, mpiicc, mpiicpp

Option	Decription	Default
-c	Effectue uniquement la compilation. Un fichier objet (.o) est produit	OFF
-o name	Précise le nom de l'exécutable produit	OFF
O0,-O1,-O2,-O3	Spécifie le niveau d'optimisation	-O2
-Bdynamic	Les librairies sont liées avec le programme lors de l'exécution	ON
-Bstatic -static	Les librairies sont liées avec le programme à la compilation. (-Bstatic limite aux librairies de l'usager)	OFF

Compilateur d'Intel

Option	Description	Défaut
-g	Produit un fichier objet avec les symboles permettant l'utilisation d'un débogueur	OFF
-Idir	Ajoute le répertoire dir à la liste des répertoires contenant les fichiers modules et includes	OFF
-Ldir	Ajoute le répertoire dir à la liste des répertoires contenant des librairies. Les librairies sont recherchées dans ce répertoire en priorité	OFF
-w	Spécifie au compilateur de n'afficher aucun message d'avertissement	OFF

Compilateur GNU

- Programme séquentiel : gfortran, gcc
- Programme parallèle : mpif77 ou mpif90, mpic, mpicxx

Option	Description	Défaut
-c	Effectue uniquement la compilation. Un fichier objet (.o) est produit	OFF
-o name	Précise le nom de l'exécutable produit	OFF
O0,-O1,-O2,-O3	Spécifie le niveau d'optimisation	-O0
-arch=native -tune=native	Optimise le code généré pour la machine actuelle	OFF
-static	Les librairies sont liées avec le programme à la compilation	OFF

Compilateur GNU

Option	Decription	Default
-g		OFF
-Idir	Ajoute le répertoire dir à la liste des répertoires contenant les fichiers modules et includes	OFF
-Ldir	Ajoute le répertoire dir à la liste des répertoires contenant des librairies. Les librairies sont recherchées dans ce répertoire en priorité	OFF
-w	Spécifie au compilateur de n'afficher aucun message d'avertissement	OFF
-m64	Génère du code pour un environnement 64-bit.	OFF
-sse4	Active l'utilisation du jeu d'instructions vectorielles SSE version 4	OFF

Débogage

Débogage

- Objectif : éliminer les erreurs dans les programmes → **débogueur**
- Déboguer un programme consiste à l'arrêter sous certaines conditions pour examiner l'état de la pile d'appels et les valeurs stockées dans les variables
- L'utilisation d'un débogueur nécessite l'ajout de l'option **-g** à la compilation

Débogage : conseils

- Aérer le code de commentaires
- Indenter les lignes du code
- Aucune optimisation de code
- Afficher les messages de compilation

Débogage : gdb

```
gcc -g  program.c -o program.exe
```

```
gdb program.exe
```

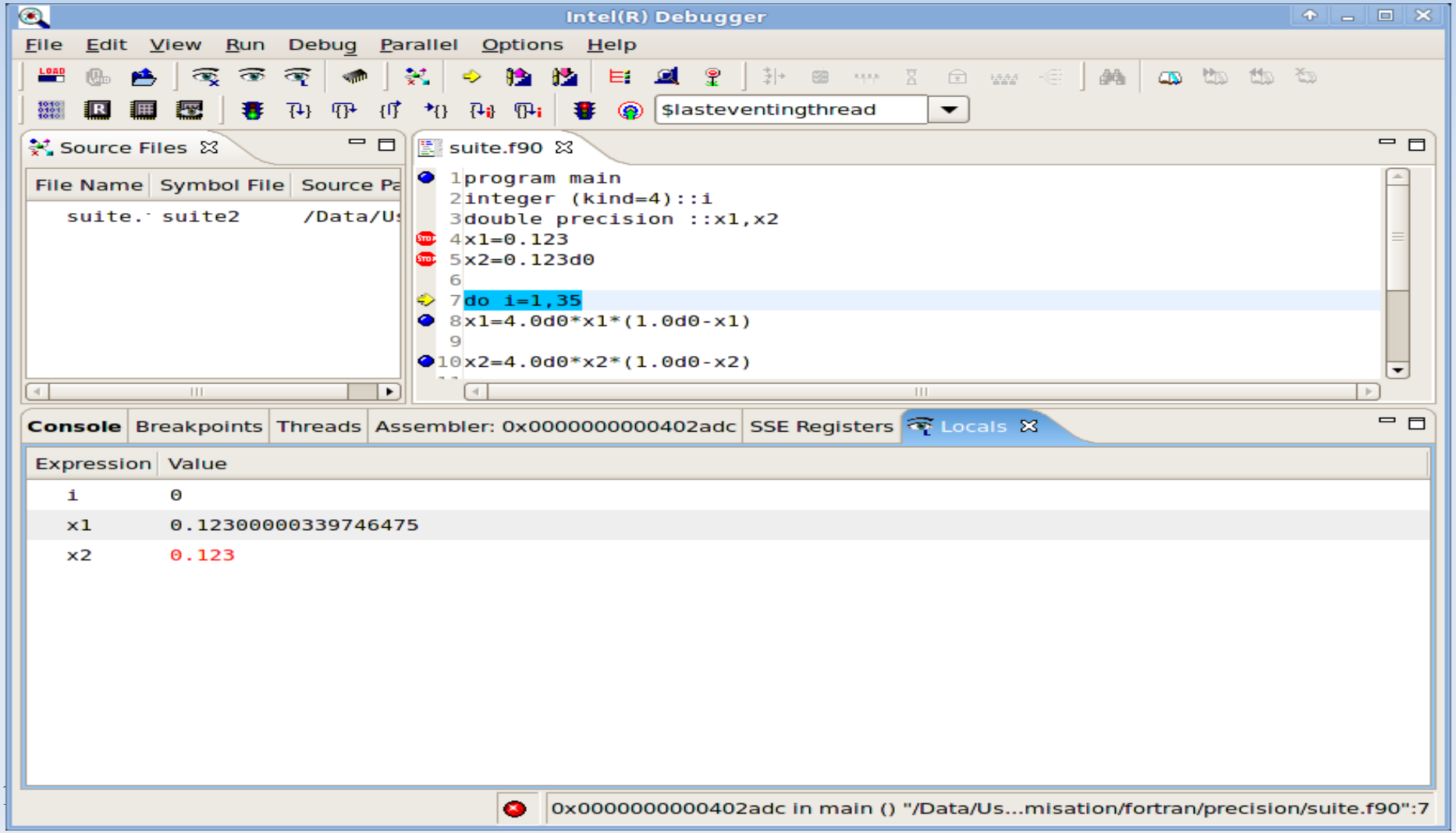
Commande	Description
r <param>	Exécute le programme : `r param1 param2`
l <ligne>	Liste le programme source : `l` Affiche uniquement la ligne <ligne> : `l 29` Si plusieurs fichiers sources : `l toto.c:29`
b <ligne>	b (Breakpoint) un point d'arrêt : `b 30. b main` Un point d'arrêt conditionnel: `b 3 Z > 92`
c	c (Continue) continue l'exécution après un arrêt

Débogage : gdb

Commande	Description
display var	Affiche le contenu d'une variable : `display toto`
printf	Affichage formaté des variables : `printf "X = %d, Y = %d\n",X,Y`
n	n (Next) exécute la ligne suivante du programme et PAUSE
s	s (Step) exécute la 1ere ligne de la fonction suivante et PAUSE
bt	bt (Backtrace) indique l'emplacement de l'erreur dans le programme

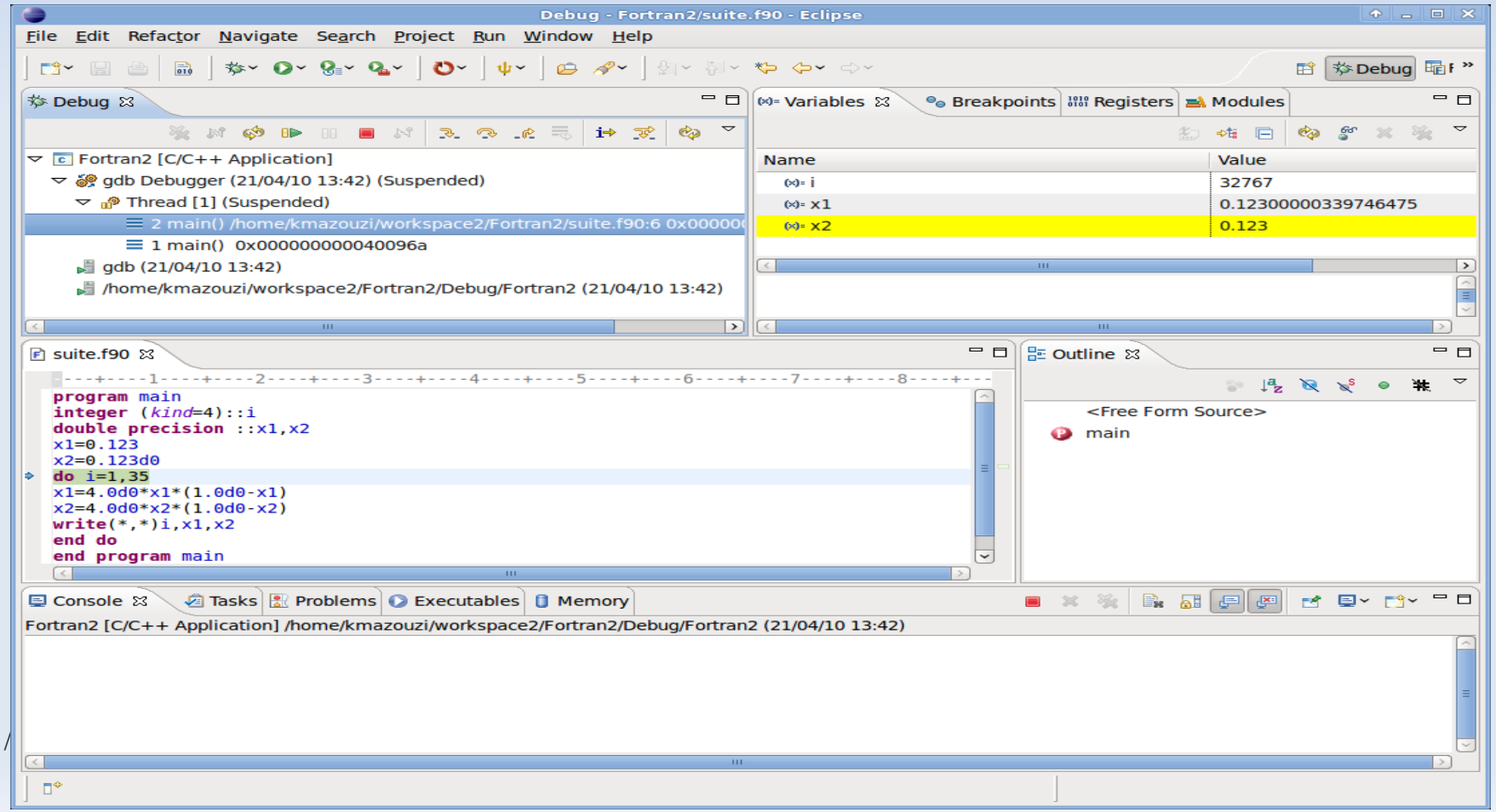
Débogage : idb

- Outil Intel
- Payant !



IDE Eclipse

- Environnement de développement similaire à **Visual Studio** et **Kdevelop**
- Personnalisable par un ensemble de plugins
- Fonctionnalités avancées : réfactoring, debug, ...
- **Gratuit !**



Profiling

Profiling

Objectif : déterminer les parties du code qui consomment le plus de temps CPU → **Profiling**

Trois étapes pour le «profiling» :

1. Instrumentation du programme
   - par le compilateur: **-p, –pg**
2. Exécution du programme instrumenté
   - fichier de données pour le profiler (mon.out,gmon.out,…)
3. Utilisation du profiler pour l'extraction et la lecture des résultats: **prof, gprof**

Profiling : exemple

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
36.9	92.23	92.23	87253	1.06	1.06	.saxpy [4]
29.9	167.06	74.83	45751	1.64	1.64	.pmv [5]
22.0	222.15	55.09	61502	0.90	0.90	.prodscal [6]
6.1	237.48	15.33	10000	1.53	3.17	.scdmb [7]
5.1	250.19	12.71	10000	1.27	21.85	.gradconj [3]
0.0	250.22	0.03				.__mcount [8]
0.0	250.25	0.03				.qincrement [9]
0.0	250.27	0.02	1	20.00	20.00	.fctfx [10]
0.0	250.28	0.01	6	1.67	1.67	.nrmerr [11]
0.0	250.28	0.00	20000	0.00	0.00	.fctft [12]
0.0	250.28	0.00	252	0.00	0.00	._sigsetmask [13]
0.0	250.28	0.00	170	0.00	0.00	thread_mutex_lock [14]

- La colonne **%** time indique le pourcentage du temps cpu total consommé par la routine courante.
- La colonne cumulative seconds représente la somme partielle des temps cpu du haut de la liste jusqu'à la routine courante
- La colonne self seconds représente le temps cpu de la routine courante
- La colonne calls indique le nombre d'appels de la routine courante
- La colonne self ms/call indique le temps cpu moyen (en milisecondes) consommé par appel à la routine courante, temps exclusif (ne contient pas la consommation des routines qu'elle appelle) uniquement
- La colonne total ms/call indique le temps total (inclusif + exclusif) consommé par appel à la routine courante

Bibliothèques scientifiques

Bibliothèques scientifiques

- Les bibliothèques scientifiques sont des ensembles de sous-programmes **testés, validés**, **optimisés**
  - portables, indépendantes de l'architecture des machines
  - supportent différents types de donnée : réel ou complexe, simple ou double précision
  - prennent en compte différents types de stockage : matrice bande, symétrique, hermitienne
  - Performantes : optimisées pour différentes machines
- Il est donc recommandé de les utiliser quand c'est possible
  - Cela permet de se consacrer uniquement aux nouveaux développements
- Exemple : BLAS, PBLAS, LAPACK, FFT, ...

Bibliothèques scientifiques « open source »

Acronyme	Utilitaire	Directe	Itératif	Non linéaire	Valeurs propres	Parallèle	Matrice
BLAS	X						quelconque
PBLAS	X					X	quelconque
BLACS	X					X	quelconque
LAPACK		X			X		pleine/ diag
ScaLAPACK		X			X	X	pleine/ diag
SuperLU		X					creuse
SuperLU_DIST		X				X	creuse
PETSc	X		X	X		X	creuse

Bibliothèques scientifiques : BLAS

- **BLAS** : **B**asic **L**inear **A**lgebra **S**ubprograms
- Ensemble de routines effectuant des opérations élémentaires d'algèbre linéaire de façon optimisée
- Plusieurs autres librairies y font référence
- Trois niveaux :
  - BLAS 1 : opérations de type vecteur-vecteur
    - $y \rightarrow y + \alpha\, x$
  - BLAS 2 : opérations de type vecteur-matrice
    - $y \rightarrow \beta\, y + \alpha\, A$
  - BLAS 3 : opérations de type matrice-matrice
    - $C \rightarrow \beta\, C + \alpha\, A\, B$

Bibliothèques scientifiques : exemple BLAS

BLAS LVL 1 : function sdot

```
!--a scalar product of two
--single-precision real vectors x and y

program dot_main
real x(10), y(10), sdot, res
integer n, incx, incy, i
external sdot
n = 5
incx = 2
incy = 1
do i = 1, 10
  x(i) = 2.0e0
  y(i) = 1.0e0
end do
res = sdot (n, x, incx, y, incy)
print*, `SDOT = `, res
end
```

```
SDOT = 10.000
```

BLAS LVL 2 : function sger

```
!--This routine performs a matrix-vector operation
!a :=  alpha*x*y' + a

program ger_main
real a(5,3), x(10), y(10), alpha
integer m, n, incx, incy, i, j, lda
m = 2
n = 3
lda = 5
incx = 2
incy = 1
alpha = 0.5
do i = 1, 10
  x(i) = 1.0
  y(i) = 1.0
end do
do i = 1, m
  do j = 1, n
    a(i,j) = j
  end do
end do
call sger (m, n, alpha, x, incx, y, incy, a, lda)
print*, `Matrix A: `
do i = 1, m
  print*, (a(i,j), j = 1, n)
end do
end
```

```
Matrix A:

1.50000 2.50000 3.50000

1.50000 2.50000 3.50000
```

Bibliothèques scientifiques : LAPACK

- LAPACK : Linear Algebra PACKage
- Cette librairie est une collection de routines écrites en Fortran 77
  - résolution séquentielle des systèmes d'équations linéaires
  - problèmes aux valeurs propres
  - factorisations de matrice

```
!-----Appel a LAPACK pour retrouver la solution V tq A*V=U

  INFO = 0
  NRHS = 1
  LDA  = N
  LDB  = N
  CALL DGESV (N, NRHS, A, LDA, IPIV, U, LDB, INFO)
```

Questions