

serge.petiton @ lflf.fr  
@ imia.fr  
@ gmail.com

$a + b$        $m_1 2^{e_1} + m_2 2^{e_2}$   
floating point arithmetic

## Super computers

1 teraflop (64 bits)  
gigaflop

www.netlib.net program

L library of parallel computing

www.misaflop.org

www.tops500.org

Forth => fortran language

TITAN 360 => 1<sup>st</sup> machine with compiler

(L+Q-1) Start-up time

bitonic

Pipeline arithmetic.

  
cos addit' 2 vect R<sup>n</sup>

2 GHz  $\Rightarrow 10^9$  opérations par seconde  
complexité algébrique (nombre d'opér.)  
guste une addition

$$d(n) = \frac{Cx(n)}{t(n)} = \frac{n \times 1}{(L-1)\tau + n\tau} = \frac{1}{\frac{(L-1)\tau + n\tau}{n}} = \frac{1}{\frac{\tau}{n} + \frac{L-1}{n}}$$

Temps min  
nb d'op. élémentaires nécessaires pour faire cette add en pipeline flottante

debit infini

Solu P:  $y \in \mathbb{C} \Rightarrow$  but it produces float (too much at some point)

$(x+y) \times z$

$$d(n) = \frac{2^n}{(L+Q-1)\tau + n\tau} \approx \frac{2}{\tau} = \text{dos}$$

Q doesn't really count

$N_{max} \Rightarrow$  capacité max de l'ordi

TIPI  
Fortham

goofy

processors -> contains cores.



MPI

Message passing interface

OpenMP

More adapted for shared memory

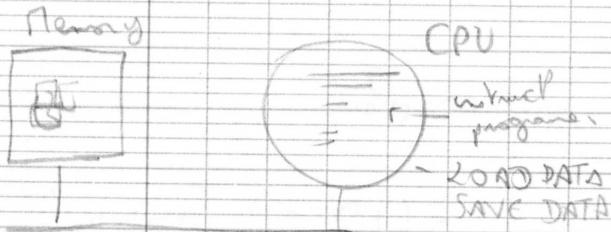
TOP500 => June 2011

the best place

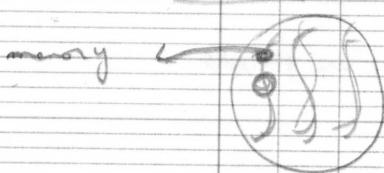
Cloud computing



Von Neumann



(Core)



memory

multithread cores

core  $\approx 1000$  threads/core

has to wait for  
If a cell has to think another one,  
then the computer asks it to  
do something else

Granularity number of op between 2 synchronisation command

SIMD

Single

Instruction

Multiples

Data

1 instruction but a lot of data

ex add 1000 vectors from  $R^{100}$   
=> only an addition

MIMD

Multiple

Several / lots instruction

Single  
Single program

SPMD

MSPMD => multiple SPMD

GPU / has GPU  $\Rightarrow$  Prog  $\neq$ , communication  $\neq$

TFP  $\neq$  Cluster

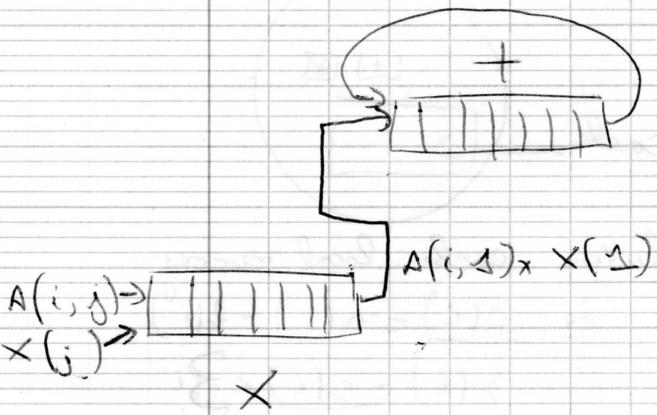
Large number of  
processor, and  
small number of core

cluster of cores with  
accelerators (GPU)

Scalar product with vectors

Forward 90

$$t(:) = A(i,:) \cdot x(1)$$



But i have nothing to add  
 $t(1) = A(i,1) \times x(1)$  with yet.  
I have to wait for  $t(2)$ .

then i have  $t(1)$  &  $t(2)$

then  $t(3)$  has to wait for  $t(4)$  no  $\emptyset$

then i have  $t(3)$  &  $t(4)$

1<sup>st</sup> step : fixed

$$S=0$$

do  $j = 1, n$

$$S = S + A(i,j) \times x(j)$$

end do

$$t(i)$$

2<sup>nd</sup> step  $\rightarrow$  add pipeline.

Auto chain / connect To keep adding partial sums  
We assume the vector  $x$  is large so  $t(1) + t(2)$  enters the  
pipeline, to be added with (for example)  $t(7)$  etc.

$\Rightarrow$  When we don't have many partial sum left  
Then we leave the vector computing & just do a regular addition

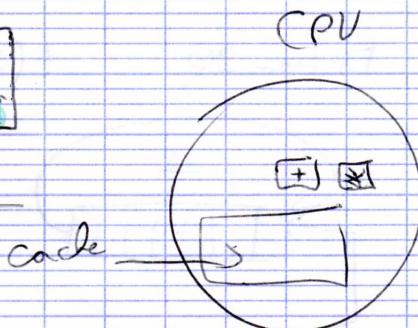
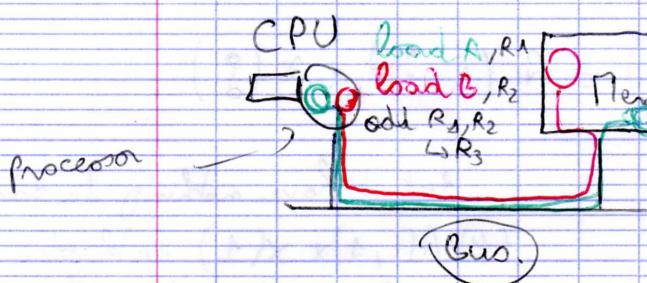
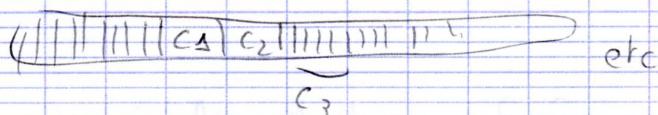
$$d_{\text{max}} \geq 2 d_{\text{dia}} \geq 6 d_{\text{red}}$$

hypothesis / not proved

How is a matrix stored?

A ( )

In Fortran  $\otimes$  matrix is stored by columns



→ lots of travelling between memory & CPU compared to the computing time.

## Solution

cache: local memory

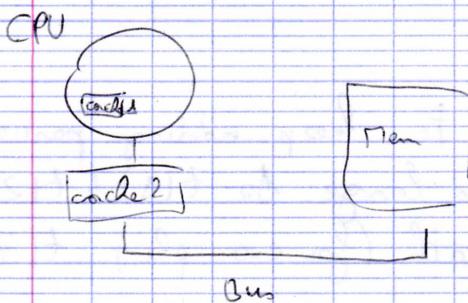
$$c(i) = \alpha(i) + \ell(i)$$

$$g(i) = c(i) \times \beta^i$$

$$d(i) = c(i) + v(i)$$

$c(i)$  is used a lot; let's keep it close to us, in a "cache" inside the CPU.

Usually proc. have 2 levels of cache, there is another  
~~and~~ one outside the proc, between it & the memory

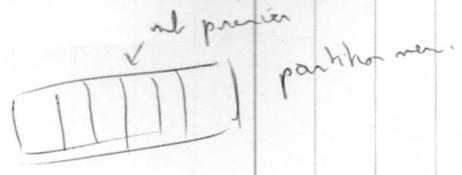


Pre-Jeudi Je commence à récupérer des valons et à les faire venir dès que possible, afin d'avoir le moins de temps possible à attendre quant à l'heure où j'aurai besoin d'eux.

## "Pre-Engagement"

base default

Le processeur n'a pas sous le cache la  
bonne info dans le cache pour continuer  
Il doit tout calculer les calculs pour aller chercher la valeur  
de la réponse.



besoin 1<sup>re</sup> ligne

P1 scat

X	X	X	1
C <sub>1</sub>	C <sub>2</sub>	C <sub>3</sub>	

l'ordi prend ces valeurs là pour son produit.

Mais il va fetch les valeurs de l'ordre et par exemple avoir assez de place pour seulement 2 colonnes dans le cache.

⇒ P1, à chaque valeur j'ai un page défaut.

Ruse Stocker la matrice par lignes (déjà fait en C)

Ou Faire l'op. avec la transposée.

Deux algo

A<sub>1</sub> → complexité  $C \times 1$

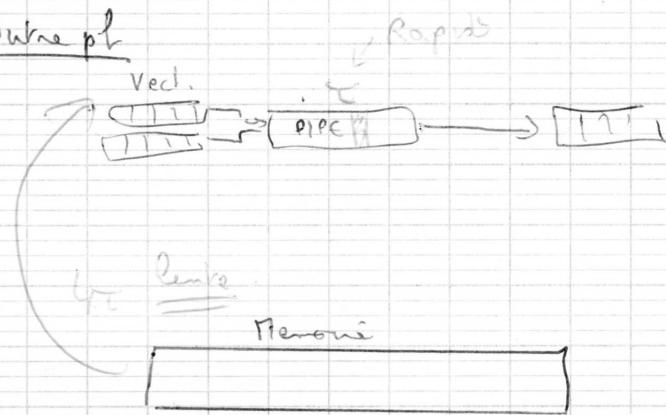
algorithme faites par le programmeur ( $x + \text{inc}$ )

et  $CX_1 < CX_2$

A<sub>2</sub> → 11 CX<sub>2</sub>

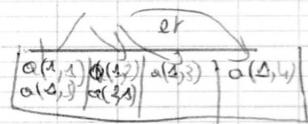
Le meilleur n'est pas faire sur A<sub>1</sub> (peut-être qu'il va ruiner le cache) et au final A<sub>2</sub> gagne

Autre ph



Autre Découper mémories en sous-mém.

et repartir les données dans l'ordre de classe de ses sous-mémo pour précher régulièrement et laisser le temps "4T" à chaque mem avant de la recontacter.



NB Si mat<sub>4</sub>(m) on tombe

$$Y = A(Ax + x) + x$$

Cel le niveau des calculs et de la mémoire  
dans les temps de lecture de données.

### Gauss triangulaire

do  $k=1, n-1$

do  $i=k+1, n$

$$a(i, k) = a(i, k) / a(k, k)$$

do  $j=k+1, n$

$$\rightarrow a(i, j) = a(i, j) - a(i, k) * a(k, j)$$

end do.

end do

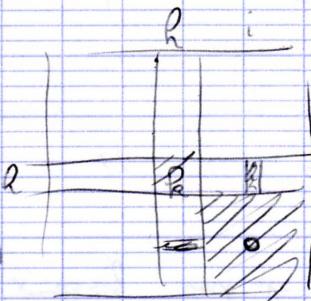
end do

$a, b$  fixé

on boucle qui à gauche

la borne où on  
peut faire en  
vecteur purgé

$a(i, j)$   
 $a(i, k)$   
 $a(k, j)$



$$a(i, k+1:n) = a(i, k+1:n) + \dots - \cancel{a(i, k)} \cancel{a(k, k)}$$

$$x(i) = a x(i) + b \Rightarrow \text{do } k, \text{ pipeline } \overbrace{x}^{\frac{x}{a}} \times \overbrace{a}^{\cancel{a}} \rightarrow \overbrace{b}^{\cancel{b}}$$

$$\text{Plus tard: } x(i) = a x(i+1) + b.$$

Regardons do  $i=2, n-1$

$$x(i) = a x(i+1) + b.$$

$$\begin{aligned}x(2) &= \alpha x(3) + b \\x(3) &= \alpha x(4) + b \\&\vdots\end{aligned}$$

Pas de pb si on inscrit les  $x(i)$  de l'ordre.  
 Si on peut écrire  $x(2)$  on en a + facile  
 "ancien valeur de  $x(3)$ "  
 $\downarrow$   
 $x(2)$  passe,  $x(3)$  suit et vient  $\alpha x(3) + b$   
 puis  $x(3)$  arrive, et est passé  $x(3) = \alpha x(4) + b$

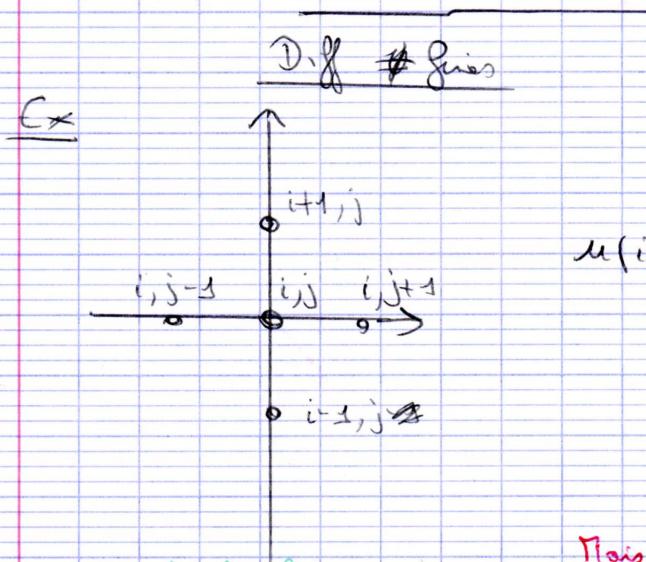
Comment faire ça en vecteur ?

$$x(2:n-1) = x(3:m) + b$$

Si on avait fait  $x(i+1) = \alpha x(i) + b$   
 le problème

$$\left. \begin{aligned}x(2) &= \alpha x(1) + b \\x(3) &= \alpha x(2) + b \\x(4) &= \dots\end{aligned}\right) \quad \text{dépendance entre les termes}$$

On doit attendre l'échutée de  $x(2)$  avant de faire la lecture.



$$\begin{aligned}u(i,j) &= \alpha u(i,j) + b u(i,j-1) \\&+ c u(i,j+1) + d u(i-1,j) \\&+ 2 u(i-1,j)\end{aligned}$$

y'a des dépendances.

⚠ Sinon, on a qu'à stocker le nouveau  $u(i,j)$  ailleurs  $[v(i,j)]$

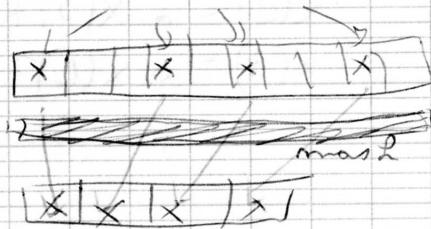
Mais on est bien sûr que les "nouvelles valeurs" de  $u(i,j-1)$  et  $u(i-1,j)$ , calculées avant, sont mieux que les anciennes.

## Gather & Scatter

[Gather]

2 vecs (remove)

volum interests



[Scatter]

Inverse.

Virtual proc. geometry reduction with add  
spread with add

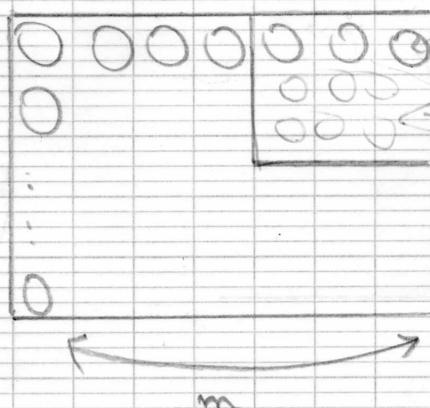
NEWS communication  
all to all; etc

Virtual processor

A real physical processor  
is made of "open" virtual one

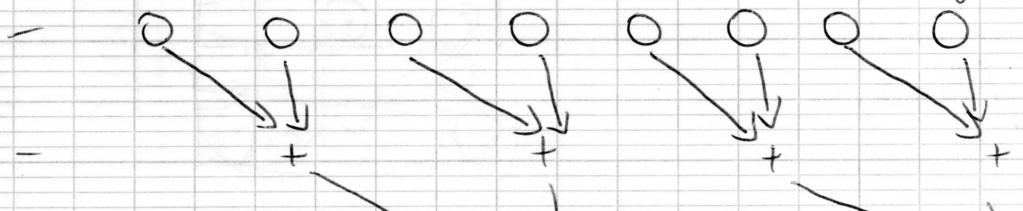
$\Phi_A$

↳ sub virtual proc.  
↑ 2 physical



p proc

$\Sigma_1 \quad \Sigma_2 \quad \Sigma_3 \quad \Sigma_4 \quad \Sigma_5 \quad \Sigma_6 \quad \Sigma_7 \quad \Sigma_8$



$\log_2 p$

étapes/point

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

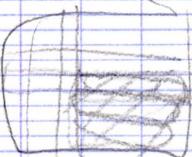
donc  $a_{ij} \neq$

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

A part tout

$$(0 \times 0 + 0 \times 0 + 0 \times 0)$$

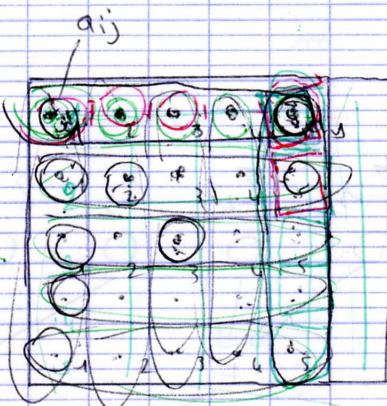
$$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 \end{pmatrix}$$



( $Ax + xc$ )

l'opération x partout

$AD + x$



$$\begin{aligned} *a_{ij} \\ *x_j \\ *0_{ij} \\ *c_{ij} \end{aligned}$$

J'entre les  $a_{ij}$  partout.

Je mets plein de fois mon vecteur en ligne \*

$\Rightarrow$  Je fais  $n^2$  multiplication en // je pose  $c_{ij} = a_{ij} \times x_i$

$\Rightarrow$  Je dois sommer les  $c_{ij}$  sur j.

$\Rightarrow$  REDUCTION ou SPREAD.  $\Rightarrow$

(Ici ça fait un spread pour additionner ( $Ax + xc$ ) + x)

Je stocke ça dans la dernière colonne  $*d_i = \sum_j c_{ij}$   
et je copie ça sur toutes les colonnes

J'ai donc  $*x_j$  en ligne et  $*d_i$  en colonne.

Je dois faire  $d_i + x_i \Rightarrow$  Il ne suffit d'ajouter de et  $x_i$   
grâce aux processus diagonaux.

G

$C = a \times x$  ① une op en //

SPREAD with add 3 loops in operation

G(i=j)

$Z = d + x$  ② //

Gauss m/1

do  $k=1, n-1$

doall  $i=k+1, n$

$$a(i,k) = a(i,k) / a(k,k)$$

end all

doall  $j=k+1, n$  and  $i=k+1, j$

$$a(i,j) = a(i,j) - a(i,k) * a(k,j)$$

end doall

end do

$k$

dimension

diag it.

2 ↓  
 $h$


a. a<sub>1</sub> a<sub>2</sub> a<sub>3</sub> a<sub>4</sub> a<sub>5</sub>  
C a<sub>1</sub> a<sub>2</sub> a<sub>3</sub> a<sub>4</sub> a<sub>5</sub>

\* spread  $a \left( G_{(i=k, j=h)} \right)$   
① \* dim 1 → f  $\left( G_{(l, l+1:n)} \right)$

\* ② C = a/f  $\left( G_{(k, h+1:n)} \right)$

We can get rid of  $f$  by spreading using

\* Spread  $\left( G_{(k, h+1:n)} \right)$

\* dim 1 → d  $\left( G_{(k+1:n, k+1:n)} \right)$

\* spread c  $\left( G_{(k, h+1:n)} \right)$

dim 2 → F  $\left( G_{(k+1:n, k+1:n)} \right)$

\*  $a = a - c * F \left( G_{(k+1:n, k+1:n)} \right)$

Pour certaines méthodes, on peut avoir un algo eff. en vectorielle  
mais pour autant plus efficace en parallèle,

(2.)

K50

base do  $k = 1, n-1$   
 do  $j = k+1, n$   
 $t(j) = a(k, j) / a(k, k)$   
 do  $j = k+1, n$   
 do  $i = k+1, n$

$$a(i, j) = a(i, j) - a(i, k) \times t(j)$$

↳ do  $h = 1, n-1$

$$t(k+1:n) = a(h, k+1:n) / a(h, h)$$

do  $j = h+1, n$

$$a(h+1:n, j) = a(h+1:n, j) - a(h+1:n, h) * t(j)$$

end do

end do

==

K50

base

do  $h = 1, n-1$

do  $i = h+1, n$

$$t(i) = a(i, h) / a(h, h)$$

do  $j = h+1:n$

$$a(i, j) = a(i, j) - a(h, j) \cancel{+} \cancel{a(h, h)} t(i)$$

↳ do  $h = 1, n-1$

$$t(k+1:n) = a(h+1:n, h) / a(h, h)$$

do  $i = h+1, n$

$$a(i, h+1:n) = a(i, h+1:n)$$

$$- a(h, h+1:n) * t(i)$$

end do

end do

(2 3 4)

### Coch Substitution

Réarranger  $Ax = b$  avec A triang. sup

Program seq

$$\begin{pmatrix} \dots & a_{ij} \\ 0 & \dots \end{pmatrix} \begin{pmatrix} x_i \\ \vdots \end{pmatrix} = \begin{pmatrix} b_i \\ \vdots \end{pmatrix}$$

do R n-1, 1, -1

s=0

on R<sub>0,n+1</sub>  $\rightarrow$  do j M, R+1, -1  
 $S = S + A(R, j) \times x(j)$

end do

$$x(R) = (b(R) - S) / A(R, R)$$

end do

$$x_R = \frac{b_n}{a_{n,n}}$$

$$a_{n-n+1}x_{n-1} + a_{n-n}x_n = b_n$$

$$\hookrightarrow x_{n-1} = \frac{b_n - a_{n-n}x_n}{a_{n-n+1}}$$

$$x(R) = \frac{b(R) - S}{A(R, R)}$$

pt scal

=> Comment faire du vecteur ?

on peut le faire

On a un pt. scal, ~~etape par etape~~ en vecteur.

↳ Reprenons tout le truc

$$\begin{pmatrix} \times & \times & \times & | & \times \\ \times & \times & \times & | & \times \\ \times & \times & \times & | & \times \\ \times & \times & \times & | & \times \\ \times & \times & \times & | & \times \end{pmatrix} \begin{pmatrix} \times \\ \times \\ \times \\ \times \\ \times \end{pmatrix} = \begin{pmatrix} X \\ X \\ X \\ X \\ X \end{pmatrix}$$

À un moment donné, ils vont tous être multiplié par sr.

Faisons le.

pipeline  
add et mult  
pour les scal

je veux faire  $Ax$

matr vect  
 $\times$

i) Algo seq

$$\begin{pmatrix} 1 & 1 & 1 \\ c_1 & c_2 & c_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

$$c_1 := c_1 x_1$$

$$c_2 := c_1 + c_2 x_2$$

$$c_3 := c_2 + c_3 x_3$$

$$\sum A_{ij} x_j \quad \text{For } j = 1..n \quad \left\{ \begin{array}{l} \text{for } i = 1..m \\ \quad y[i] = A[i, 1] \times x[1] \\ \quad \dots \\ \quad \text{end } i \end{array} \right. \quad \begin{array}{l} \Rightarrow 2 \text{ vecteur} \\ \Rightarrow 2 \text{ operat} \end{array} \quad \left. \begin{array}{l} \text{Stock's des vecteur} \\ \dots \end{array} \right\}$$

$$\text{For } i = 1..n$$

$$y[i] = y[i] + a[ij] \times x[j];$$

end i

end j

### Vectoriel

Fonction

$$y(:) = A(:, 1) \cdot X(1);$$

do  $j = 2, m$

$$y(:) = y(:) + A(:, j) \cdot X(j)$$