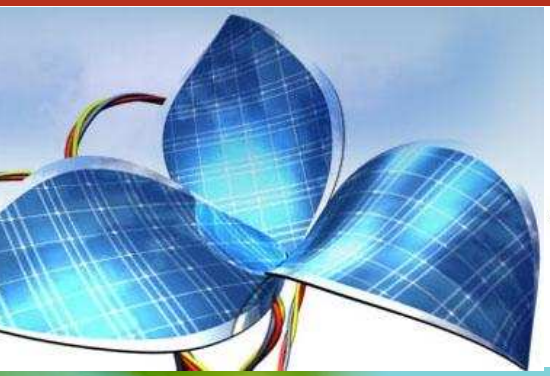
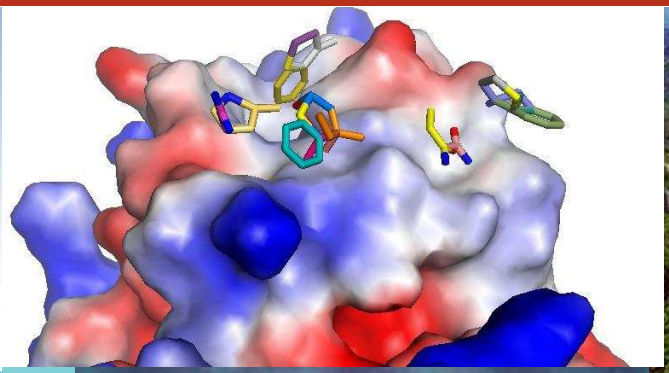
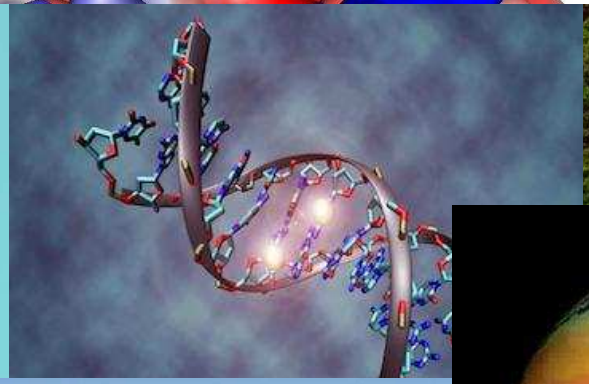
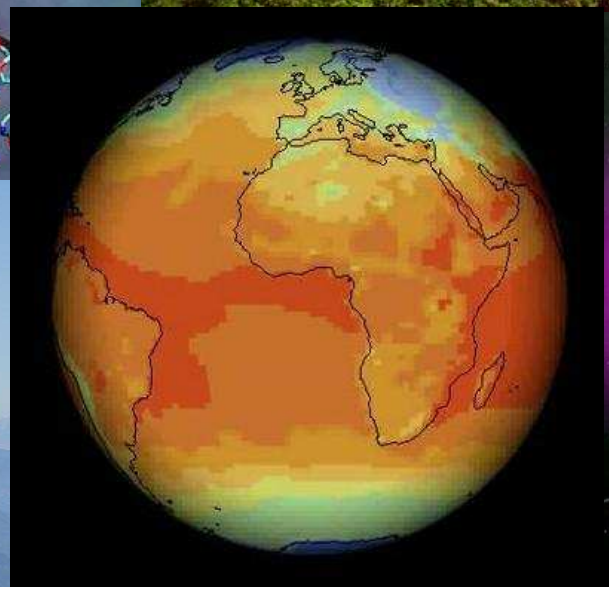
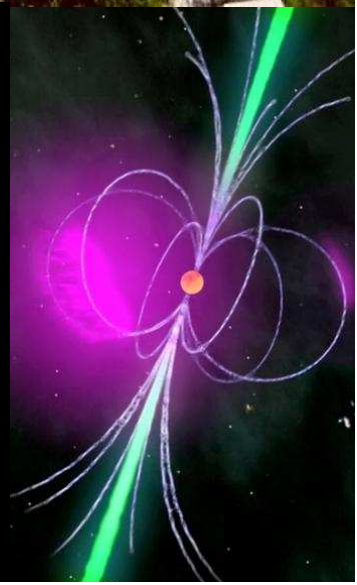
Défis des grandes infrastructures de calcul

Arnaud Legrand

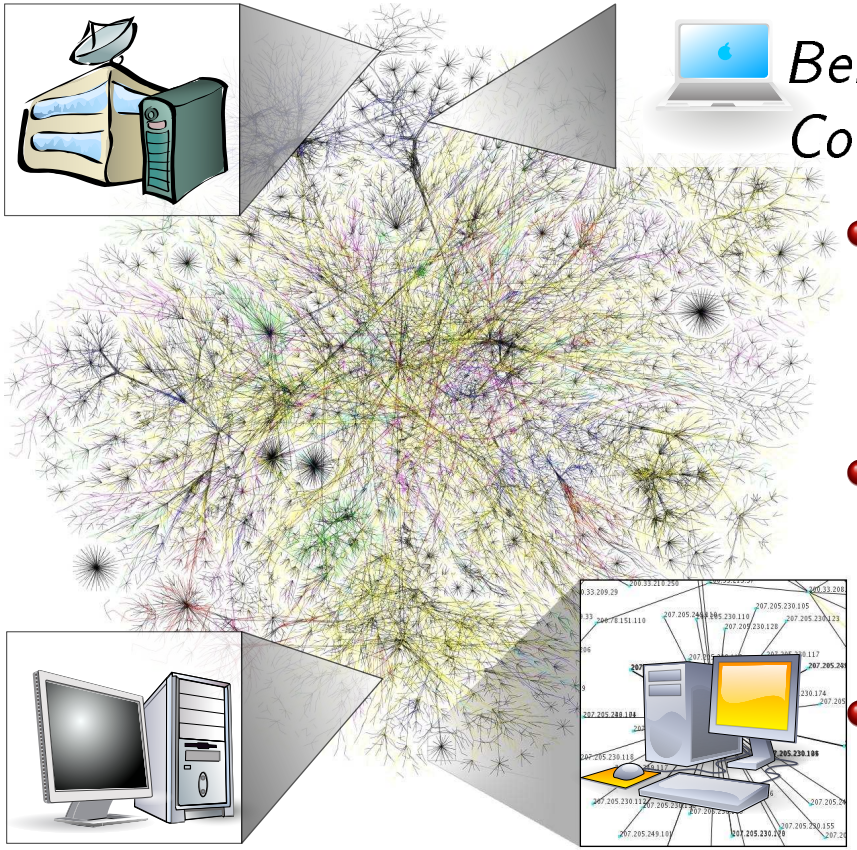
Quel est le point commun entre . . . ?

Eau potable, panneaux solaires, médicaments contre Ebola, le SIDA ou le cancer, l'évolution du climat, le para-pente, la recherche d'extra-terrestres, les pulsars, . . .

Le calcul bénévole



Berkeley Open Infrastructure for Network Computing (BOINC):

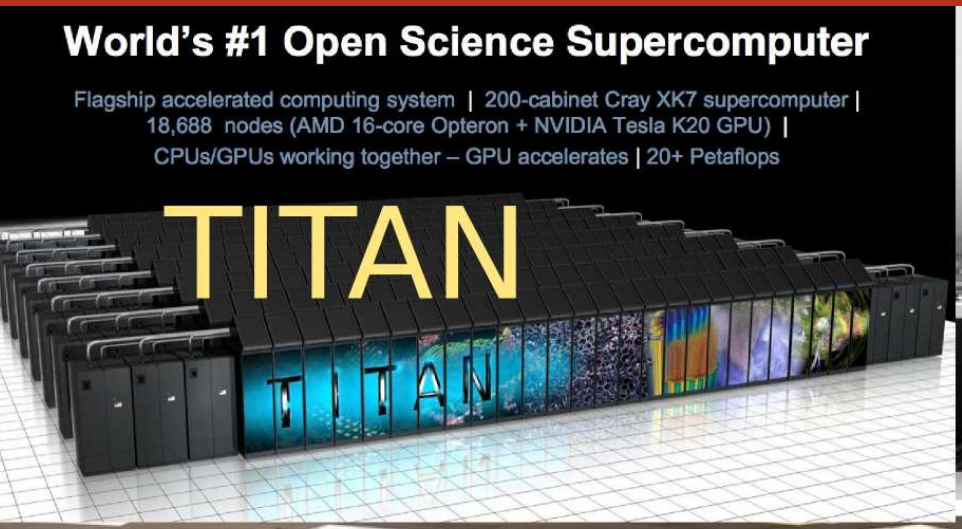
- Environ 238 000 volontaires actifs sur plus de 420 800 ordinateurs

  (mais aussi tablettes, téléphones, ...)
- La puissance de calcul moyenne sur 24 heures de l'ensemble des participants atteint 6 722 PetaFlops
- Hétérogène, dynamique, des ordinateurs pas forcément fiables, ...

Today the computer is just as important a tool for chemists as the test tube. Simulations are so realistic that they predict the outcome of traditional experiments

– Comité Nobel (Chimie), 2013

Les supercalculateurs



- 100,000 à 1,000,000 de coeurs, des accélérateurs (GPU, Xeon Phi), un réseau d'interconnection ultra-rapide
- Une course entre les pays (Top500)

Le cloud


Facebook
Microsoft
Amazon
Google
North Atlantic Ocean
South Atlantic Ocean
Indian Ocean
Canada
United States
Mexico
Brazil
Argentina
Chile
Peru
Bolivia
Venezuela
Colombia
Russia
Kazakhstan
Mongolia
China
Afghanistan
Pakistan
India
Iran
Iraq
Saudi Arabia
Sudan
Nigeria
Ethiopia
Kenya
Tanzania
Angola
Namibia
Botswana
South Africa
Indonesia
Papua New Guinea
Australia
Active Cab
Future Cab

Une évolution vertigineuse

Des architectures parallèles, hybrides et de très grande taille pour répondre aux besoins de calcul et aux contraintes énergétiques

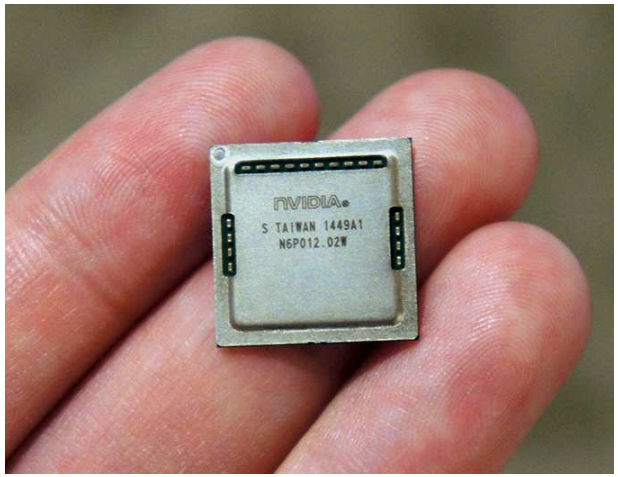
1996

ASCI Red
1 Teraflop
9298 Pentium II
1 000 Flops/W

2009

ATI Radeon
2.4 Teraflop
1600 Stream Processors
1 600 000 Flops/W

2015



Nvidia Tegra X1
1 Teraflop
8-core ARM CPU
667 000 000 Flops/W

Mon téléphone est aussi puissant qu'un super-calculateur d'il y a 20 ans

Une évolution vertigineuse

Des architectures parallèles, hybrides et de très grande taille pour répondre aux besoins de calcul et aux contraintes énergétiques

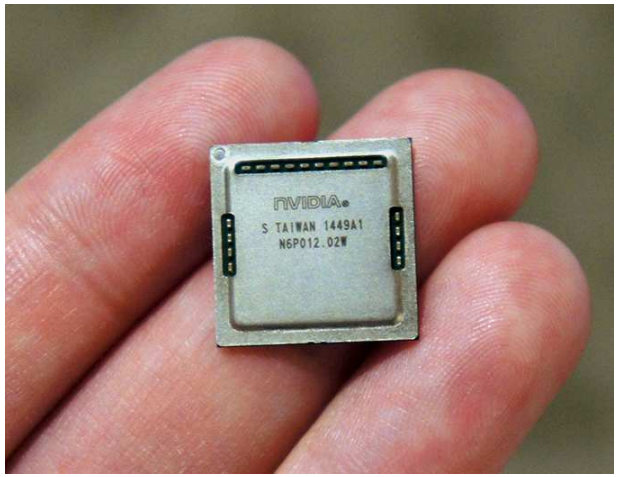
1996

ASCI Red
1 Teraflop
9298 Pentium II
1 000 Flops/W

2009

ATI Radeon
2.4 Teraflop
1600 Stream Processors
1 600 000 Flops/W

2015



Nvidia Tegra X1
1 Teraflop
8-core ARM CPU
667 000 000 Flops/W

Mon téléphone est aussi puissant qu'un super-calculateur d'il y a 20 ans

Mais Sequoia, c'est **6 millions de threads** s'échangeant des données...

Comment programme-t-on un tel monstre ?

Résolution d'un système triangulaire (à la main)

$$\begin{cases} x+ & y+ & z+ & t = & 6 \\ & y- & 3z- & t = & 5 \\ & & 6z+ & t = & -4 \\ & & & 4t = & 8 \end{cases}$$

Résolution d'un système triangulaire (à la main)

$$\begin{cases} x+ & y+ & z+ & t = & 6 \\ & y- & 3z- & t = & 5 \\ & & 6z+ & t = & -4 \\ & & & 4t = & 8 \end{cases} \qquad \begin{cases} \\ \\ \\ t = 8/4 \end{cases}$$

Résolution d'un système triangulaire (à la main)

$$\begin{cases} x+ & y+ & z+ & t= & 6 \\ & y- & 3z- & t= & 5 \\ & & 6z+ & t= & -4 \\ & & & 4t= & 8 \end{cases} \qquad \begin{cases} \\ \\ \\ t= & 2 \end{cases}$$

Résolution d'un système triangulaire (à la main)

$$\begin{cases} x+ & y+ & z+ & t= & 6 \\ & y- & 3z- & t= & 5 \\ & & 6z+ & t= & -4 \\ & & & 4t= & 8 \end{cases} \qquad \begin{cases} & & & & \\ & & & & \\ & & 6z+ & t= & -4 \\ & & & t= & 2 \end{cases}$$

Résolution d'un système triangulaire (à la main)

$$\begin{cases} x+ & y+ & z+ & t= & 6 \\ & y- & 3z- & t= & 5 \\ & & 6z+ & t= & -4 \\ & & & 4t= & 8 \end{cases} \qquad \begin{cases} & & & & \\ & & & & \\ & & 6z+ & 2= & -4 \\ & & & t= & 2 \end{cases}$$

Résolution d'un système triangulaire (à la main)

$$\begin{cases} x+ & y+ & z+ & t= & 6 \\ & y- & 3z- & t= & 5 \\ & & 6z+ & t= & -4 \\ & & & 4t= & 8 \end{cases} \qquad \begin{cases} & & & & \\ & & & & \\ & & 6z= & -4-2 \\ & & t= & 2 \end{cases}$$

Résolution d'un système triangulaire (à la main)

$$\begin{cases} x+ & y+ & z+ & t= & 6 \\ & y- & 3z- & t= & 5 \\ & & 6z+ & t= & -4 \\ & & & 4t= & 8 \end{cases} \qquad \begin{cases} & & & & \\ & & & & \\ & & z= & -6/6 \\ & & t= & 2 \end{cases}$$

Résolution d'un système triangulaire (à la main)

$$\begin{cases} x+ & y+ & z+ & t= & 6 \\ & y- & 3z- & t= & 5 \\ & & 6z+ & t= & -4 \\ & & & 4t= & 8 \end{cases} \qquad \begin{cases} & & & & \\ & & & & \\ & & & z= & -1 \\ & & & t= & 2 \end{cases}$$

Résolution d'un système triangulaire (à la main)

$$\begin{cases} x+ & y+ & z+ & t= & 6 \\ & y- & 3z- & t= & 5 \\ & & 6z+ & t= & -4 \\ & & & 4t= & 8 \end{cases}$$

$$\begin{cases} & y- & 3z- & t= & 5 \\ & & & z= & -1 \\ & & & t= & 2 \end{cases}$$

Résolution d'un système triangulaire (à la main)

$$\begin{cases} x+ & y+ & z+ & t = & 6 \\ & y- & 3z- & t = & 5 \\ & & 6z+ & t = & -4 \\ & & & 4t = & 8 \end{cases}$$

$$\begin{cases} & & y+(3-2) = & 5 \\ & & z = & -1 \\ & & t = & 2 \end{cases}$$

Résolution d'un système triangulaire (à la main)

$$\begin{cases} x+ & y+ & z+ & t = & 6 \\ & y- & 3z- & t = & 5 \\ & & 6z+ & t = & -4 \\ & & & 4t = & 8 \end{cases}$$

$$\begin{cases} & y+ & 1 & = & 5 \\ & & z & = & -1 \\ & & t & = & 2 \end{cases}$$

Résolution d'un système triangulaire (à la main)

$$\begin{cases} x+ & y+ & z+ & t = & 6 \\ & y- & 3z- & t = & 5 \\ & & 6z+ & t = & -4 \\ & & & 4t = & 8 \end{cases} \qquad \begin{cases} & & & y = (5-1)/1 \\ & & & z = -1 \\ & & & t = \ \ 2 \end{cases}$$

Résolution d'un système triangulaire (à la main)

$$\begin{cases} x+ & y+ & z+ & t= & 6 \\ & y- & 3z- & t= & 5 \\ & & 6z+ & t= & -4 \\ & & & 4t= & 8 \end{cases} \qquad \begin{cases} & & & y= & 4 \\ & & & z= & -1 \\ & & & t= & 2 \end{cases}$$

Résolution d'un système triangulaire (à la main)

$$\begin{cases} x+ & y+ & z+ & t = & 6 \\ & y- & 3z- & t = & 5 \\ & & 6z+ & t = & -4 \\ & & & 4t = & 8 \end{cases}$$

$$\begin{cases} x+ & y+ & z+ & t = & 6 \\ & & & y = & 4 \\ & & & z = & -1 \\ & & & t = & 2 \end{cases}$$

Résolution d'un système triangulaire (à la main)

$$\begin{cases} x+ y+ z+ t = 6 \\ y- 3z- t = 5 \\ 6z+ t = -4 \\ 4t = 8 \end{cases}$$

$$\begin{cases} x+(4- 1+2) = 6 \\ y = 4 \\ z = -1 \\ t = 2 \end{cases}$$

Résolution d'un système triangulaire (à la main)

$$\begin{cases} x+ & y+ & z+ & t = & 6 \\ & y- & 3z- & t = & 5 \\ & & 6z+ & t = & -4 \\ & & & 4t = & 8 \end{cases} \qquad \begin{cases} x+ & 5 & & = & 6 \\ & & y & = & 4 \\ & & z & = & -1 \\ & & t & = & 2 \end{cases}$$

Résolution d'un système triangulaire (à la main)

$$\begin{cases} x+ & y+ & z+ & t = & 6 \\ & y- & 3z- & t = & 5 \\ & & 6z+ & t = & -4 \\ & & & 4t = & 8 \end{cases} \qquad \begin{cases} x = (6-5)/1 \\ y = 4 \\ z = -1 \\ t = 2 \end{cases}$$

Résolution d'un système triangulaire (à la main)

$$\begin{cases} x+ & y+ & z+ & t = & 6 \\ & y- & 3z- & t = & 5 \\ & & 6z+ & t = & -4 \\ & & & 4t = & 8 \end{cases} \qquad \begin{cases} x = & 1 \\ y = & 4 \\ z = & -1 \\ t = & 2 \end{cases}$$

Les principales étapes:

- on part du bas
- on somme des produits horizontalement
- on divise

Résolution d'un système triangulaire (en python)

(sans utiliser `np.linalg.solve(A,b)`, bien sûr ☺)

$$\begin{aligned} x+ y+ z+ t &= 6 \\ y- 3z- t &= 5 \\ 6z+ t &= -4 \\ 4t &= 8 \end{aligned}$$

```
1  import numpy as np
2  A = np.array([[1, 1, 1, 1], [0, 1, -3, -1],
3        [0, 0, 6, 1] , [0, 0, 0, 4]], float)
4  b = np.array([6, 5, -4, 8], float)
5
6  n = len(b)
7  x = np.zeros(n, float)
8  for i in reversed(range(0,n)): # en partant du bas
9    S = 0
10   for j in range(i+1,n):
11      S = S + A[i][j] * x[j]     # la somme
12   x[i] = (b[i] - S) / A[i][i]   # la division
13 print(x)
```

```
1  [ 1.  4. -1.  2.]
```

Tel quel, ce programme est intrinsèquement séquentiel (à cause de S)

Résolution d'un système triangulaire

```
1 for i in reversed(range(0,n)):        # en partant du bas
2   S = np.dot(A[i][i+1:n],x[i+1:n])    # la somme
3   x[i] = (b[i] - S) / A[i][i]         # la division
```

Cette version-ci est bien plus rapide que la précédente car:

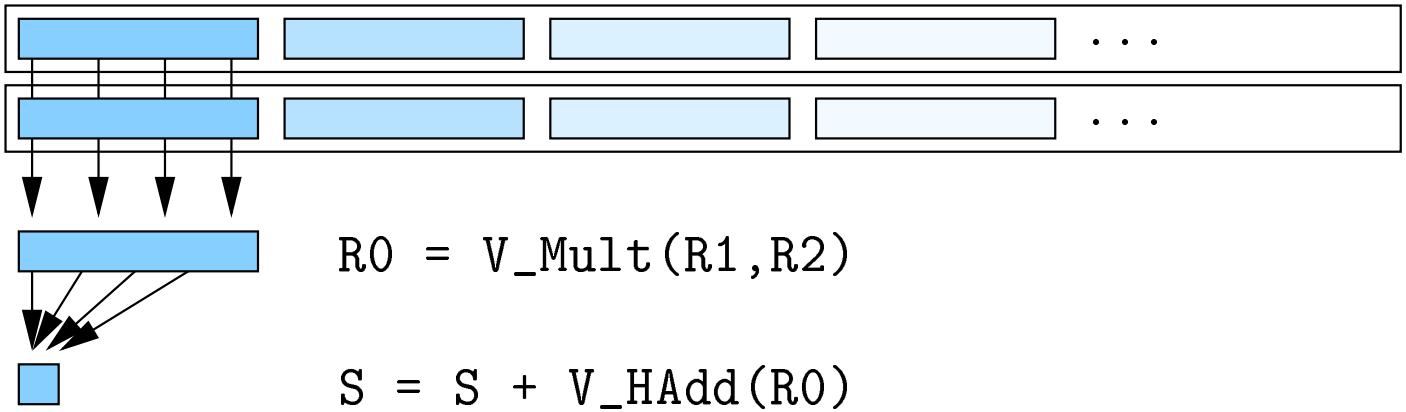
- pas d'interprétation de la boucle interne, de vérification des bornes, ...

Résolution d'un système triangulaire

```
1 for i in reversed(range(0,n)):        # en partant du bas
2   S = np.dot(A[i][i+1:n],x[i+1:n])    # la somme
3   x[i] = (b[i] - S) / A[i][i]         # la division
```

Cette version-ci est bien plus rapide que la précédente car:

- pas d'interprétation de la boucle interne, de vérification des bornes, ...
- la somme est *vectorisable* (MMX, SSE, *GPU*,...)
- la somme est *parallélisable* (threads, multicore)
- la somme est *distribuable* (réseau, MPI)

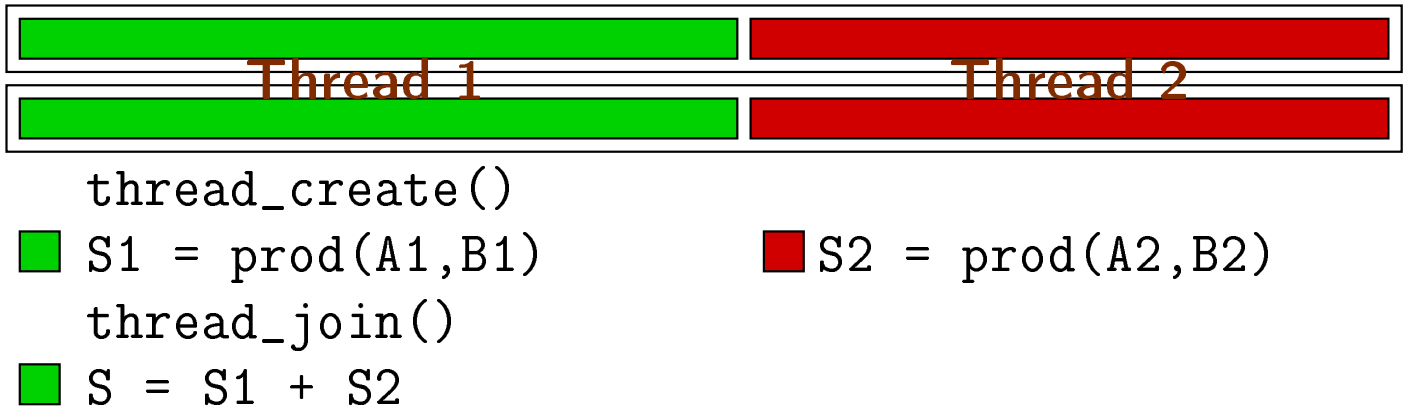


Résolution d'un système triangulaire

```
1 for i in reversed(range(0,n)):        # en partant du bas
2   S = np.dot(A[i][i+1:n],x[i+1:n])    # la somme
3   x[i] = (b[i] - S) / A[i][i]         # la division
```

Cette version-ci est bien plus rapide que la précédente car:

- pas d'interprétation de la boucle interne, de vérification des bornes, ...
- la somme est *vectorisable* (MMX, SSE, *GPU*,...)
- la somme est *parallélisable* (threads, multicore)
- la somme est *distribuable* (réseau, MPI)

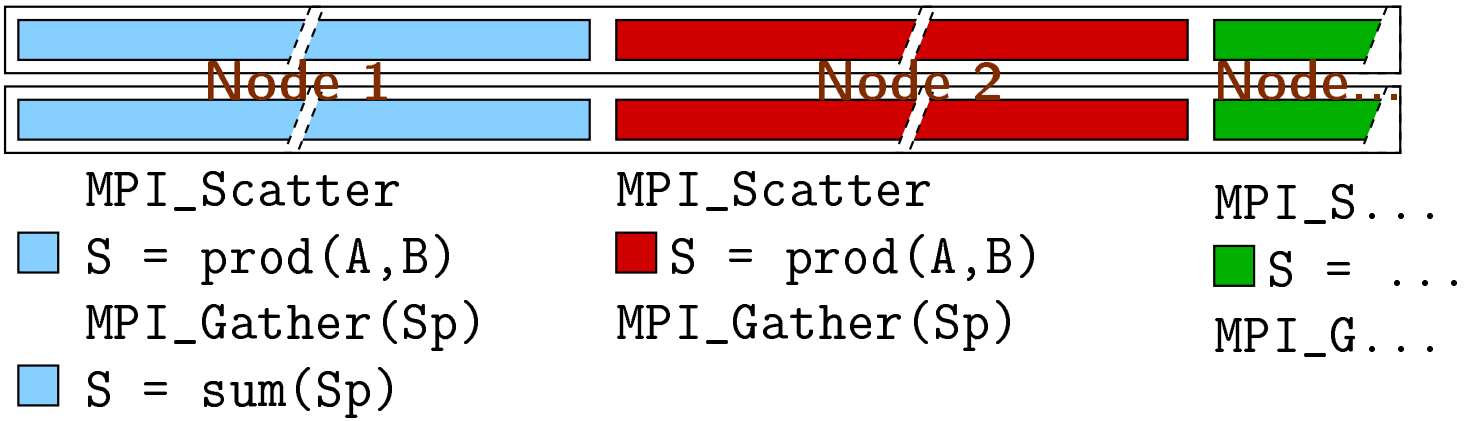


Résolution d'un système triangulaire

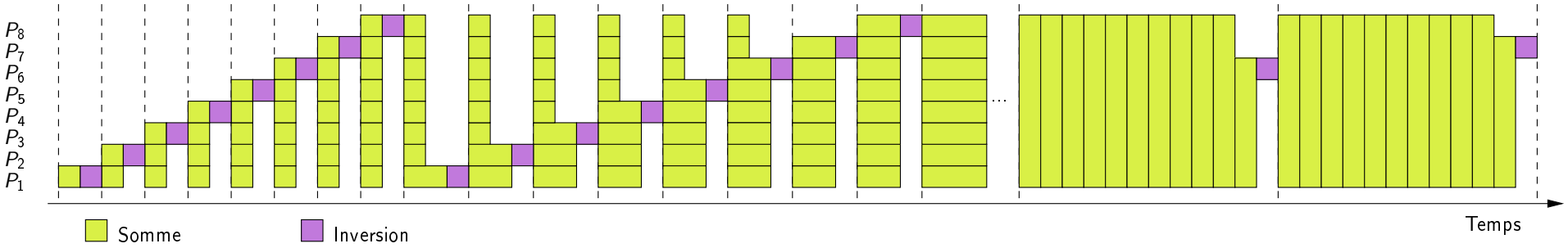
```
1 for i in reversed(range(0,n)):        # en partant du bas
2   S = np.dot(A[i][i+1:n],x[i+1:n])    # la somme
3   x[i] = (b[i] - S) / A[i][i]         # la division
```

Cette version-ci est bien plus rapide que la précédente car:

- pas d'interprétation de la boucle interne, de vérification des bornes, ...
- la somme est *vectorisable* (MMX, SSE, *GPU*,...)
- la somme est *parallélisable* (threads, multicore)
- la somme est *distribuable* (réseau, MPI)



Est-ce satisfaisant ?



Non!

- Les processeurs sont largement inactifs (sauf vers à la "fin")
- On communique très souvent des données toutes petites
- Le moindre retard d'un processeur ralenti tout le monde

Il faut tout réorganiser:

- changer la granularité
- supprimer les synchronisations

Un peu de restructuration

$$\begin{cases} x+ & y+ & z+ & t = & 6 \\ & y- & 3z- & t = & 5 \\ & & 6z+ & t = & -4 \\ & & & 4t = & 8 \end{cases}$$

Substituons les variables dans chacune des lignes (sommes) dès que possible

Un peu de restructuration

$$\begin{cases} x+ & y+ & z+ & 2 = & 6 \\ & y- & 3z- & 2 = & 5 \\ & & 6z+ & 2 = & -4 \\ & & & 4t = & 8 \end{cases}$$

Substituons les variables dans chacune des lignes (sommes) dès que possible (on travaille donc "verticalement")

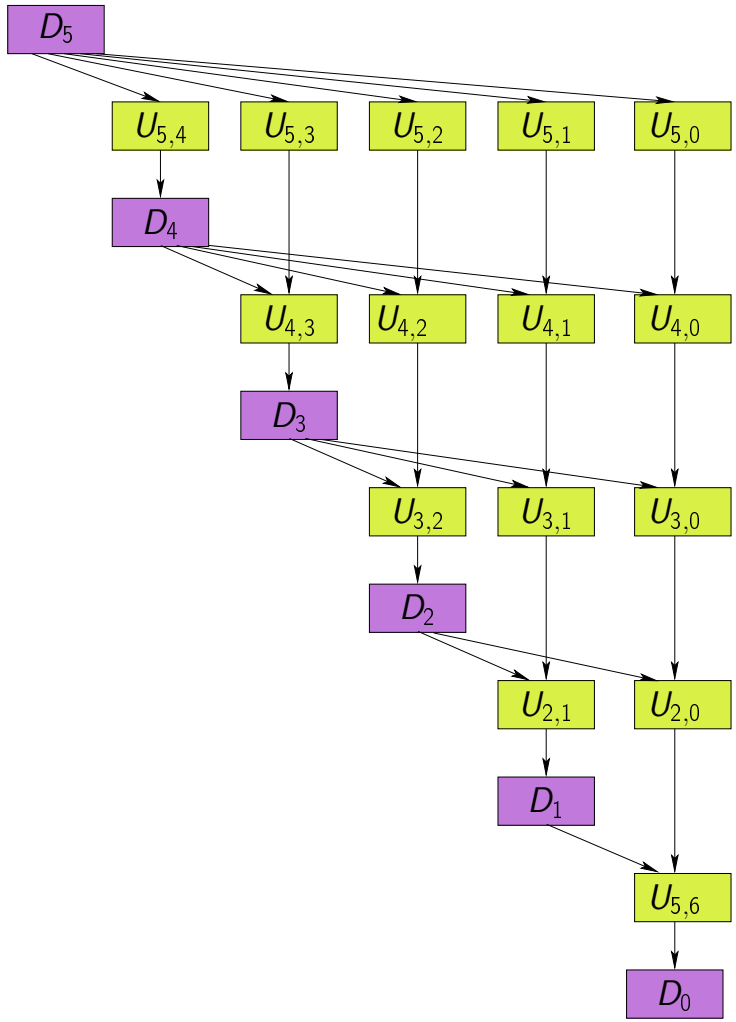
```
1  import numpy as np
2  A = np.array([[1, 1, 1, 1], [0, 1, -3, -1],
3        [0, 0, 6, 1] , [0, 0, 0, 4]], float)
4  b = np.array([6, 5, -4, 8], float)
5
6  n = len(b)
7  x = np.zeros(n, float)
8  S = np.zeros(n, float)
9  for j in reversed(range(0,n)):   # en partant du bas
10   x[j] = (b[j] - S[j]) / A[j][j]    # la division              (D_j)
11   for i in range(0,j):              # vraie boucle parallele
12      S[i] = S[i] + A[i][j] * x[j]   # mise a jour            (U_i,j)
```

. . . pour obtenir des graphes de tâches

Si I_1 écrit z et I_2 lit/écrit z, il faut faire I_1 et I_2 dans le bon ordre [Bernstein66]

Les données définissent des dépendances entre les instructions/tâches

```
1  import numpy as np
2  A = np.array([[1, 1, 1, 1], [0, 1, -3, -1],
3       [0, 0, 6, 1] , [0, 0, 0, 4]], float)
4  b = np.array([6, 5, -4, 8], float)
5
6  n = len(b)
7  x = np.zeros(n, float)
8  S = np.zeros(n, float)
9  for j in reversed(range(0,n)):  # en partant du bas
10   x[j] = (b[j] - S[j]) / A[j][j]   # la division          (D_j)
11   for i in range(0,j):             # vraie boucle parallele
12     S[i] = S[i] + A[i][j] * x[j]   # mise a jour          (U_i,j)
```

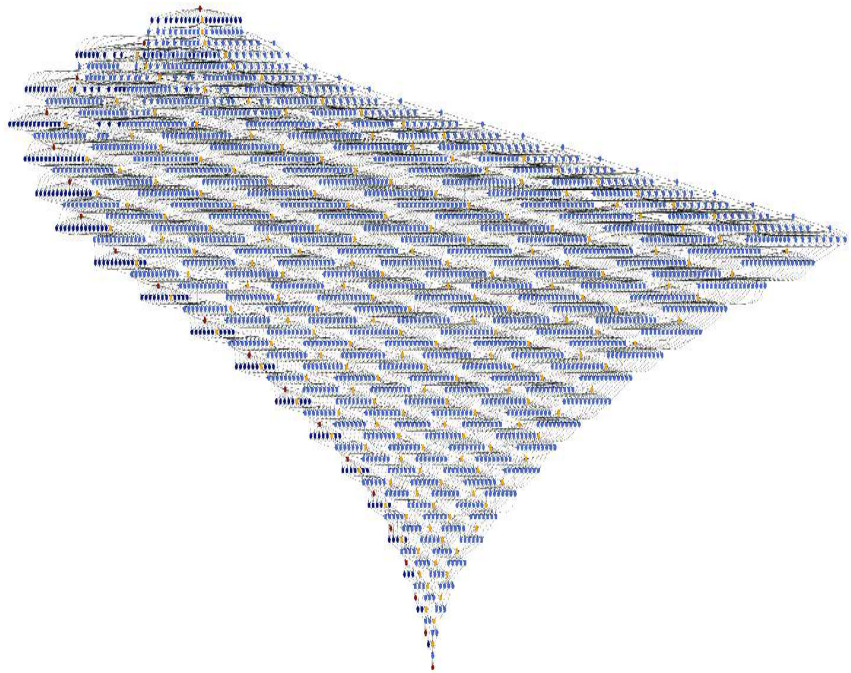


. . . pour obtenir des graphes de tâches

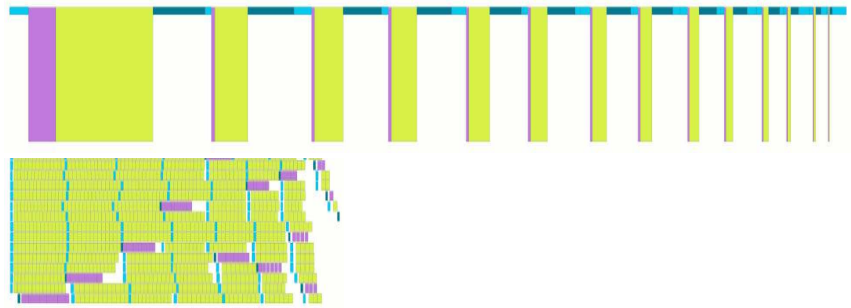
Si I_1 écrit z et I_2 lit/écrit z, il faut faire I_1 et I_2 dans le bon ordre [Bernstein66]

Les données définissent des dépendances entre les instructions/tâches

```
1  import numpy as np
2  A = np.array([[1, 1, 1, 1], [0, 1, -3, -1],
3       [0, 0, 6, 1] , [0, 0, 0, 4]], float)
4  b = np.array([6, 5, -4, 8], float)
5
6  n = len(b)
7  x = np.zeros(n, float)
8  S = np.zeros(n, float)
9  for j in reversed(range(0,n)):  # en partant du bas
10   x[j] = (b[j] - S[j]) / A[j][j]   # la division          (D_j)
11   for i in range(0,j):             # vraie boucle parallele
12     S[i] = S[i] + A[i][j] * x[j]   # mise a jour          (U_i,j)
```



- adaptation de la granularité
- des versions optimisées selon les ressources (CPU/GPU/auto-tuning)
- équilibrage de charge dynamique
- des performances plus portables



Des infrastructures omni-présentes et en perpétuelle évolution

Notre société repose (parfois sans le savoir) sur de gigantesques infrastructures de calcul.

Comment concevoir/utiliser/comprendre de telles infrastructures ?

- Programmation (portabilité, performance, évolution technologique)
- Tolérance aux pannes
- Consommation énergétique
- Partage équitable des ressources
- Modélisation/analyse/évaluation/expérimentation

Problèmatiques similaires dans:

- Réseaux (sans fils, de capteurs, d'objets connectés...)
- Vélos en libre service, co-voiturage, transport, smart-grids, ...