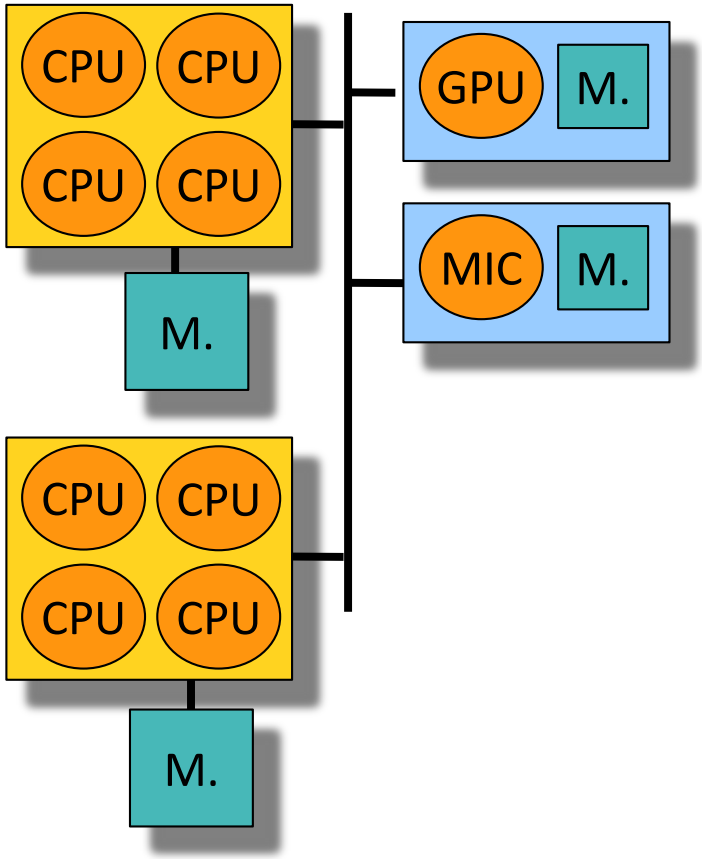
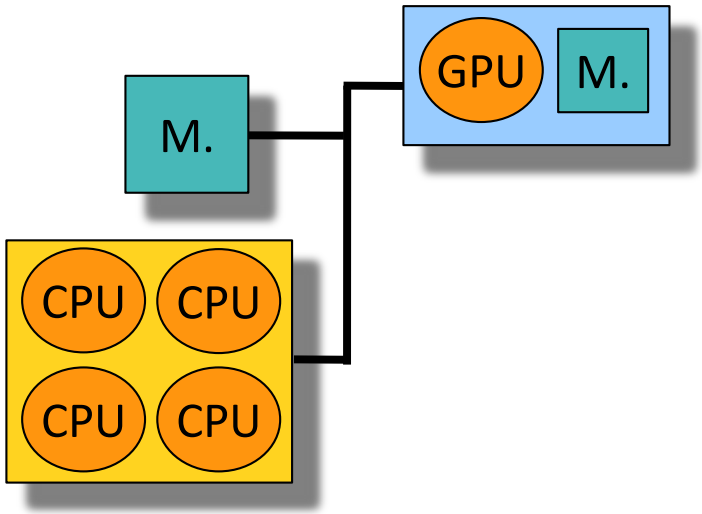
# Plan du cours

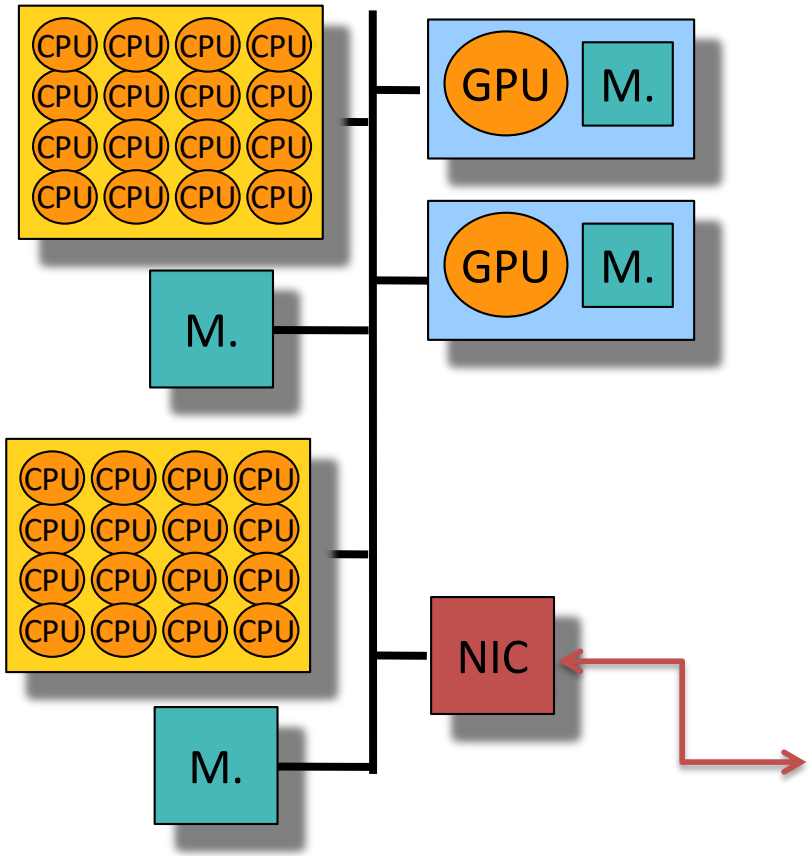
- Cette introduction
- Approche mémoire partagée
  - programmation (pthreads, OpenMP)
  - architecture (pipeline, cache, SMP, NUMA)
- Calcul sur accélérateurs
  - architecture des GPU / MIC
  - programmation (OpenCL)



# Est-ce important de savoir écrire des programmes parallèles ? De savoir programmer un GPU ?

Téléphone, PC portable,…

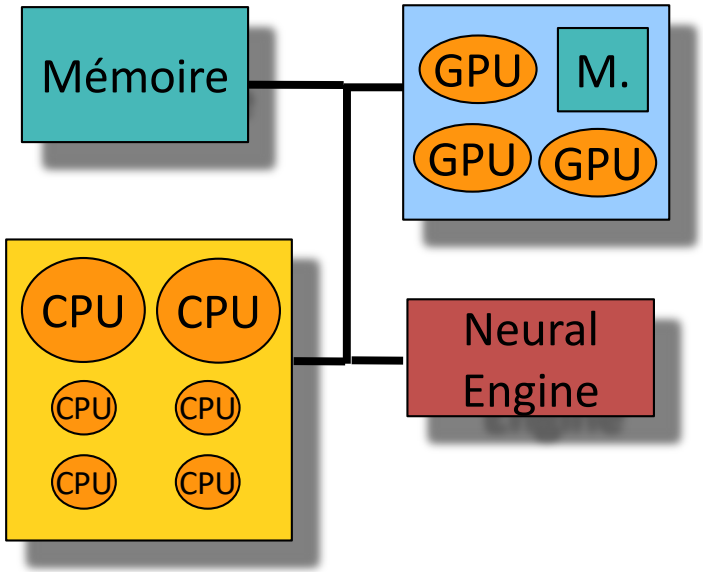


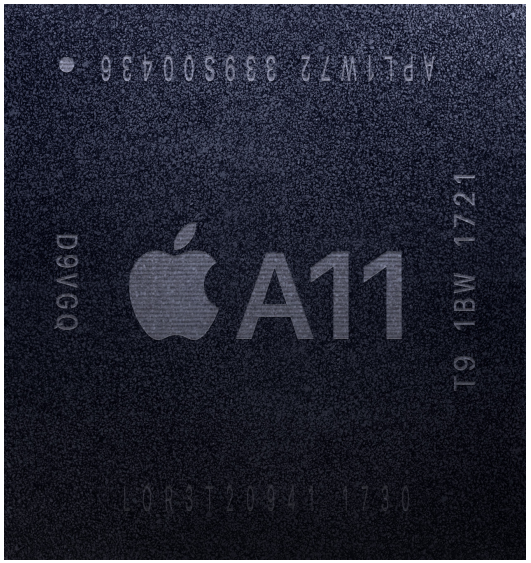




…, PC « gamer », serveur de calcul.

# Le parallélisme dans les téléphones

## Apple A11 (processeur des iphones 8 et X)

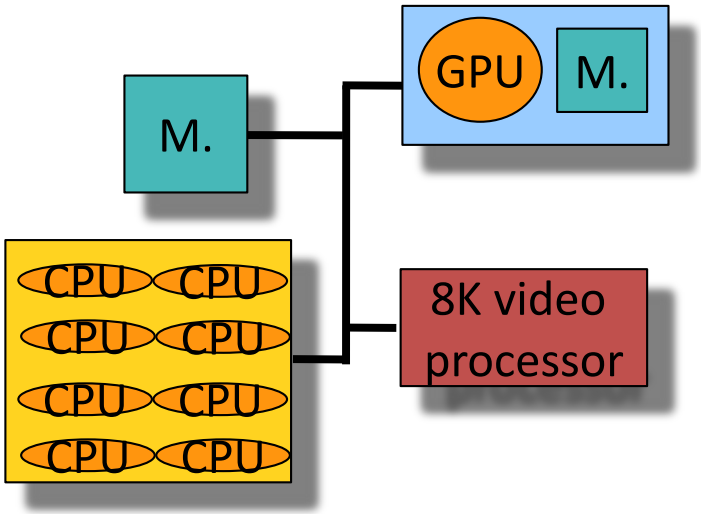



La puce Apple A11 intègre :

- 2 cœurs généralistes puissants
- 4 cœurs généralistes « faible consommation »
- 1 circuit spécialisé pour le deep-learning
- 1 GPU

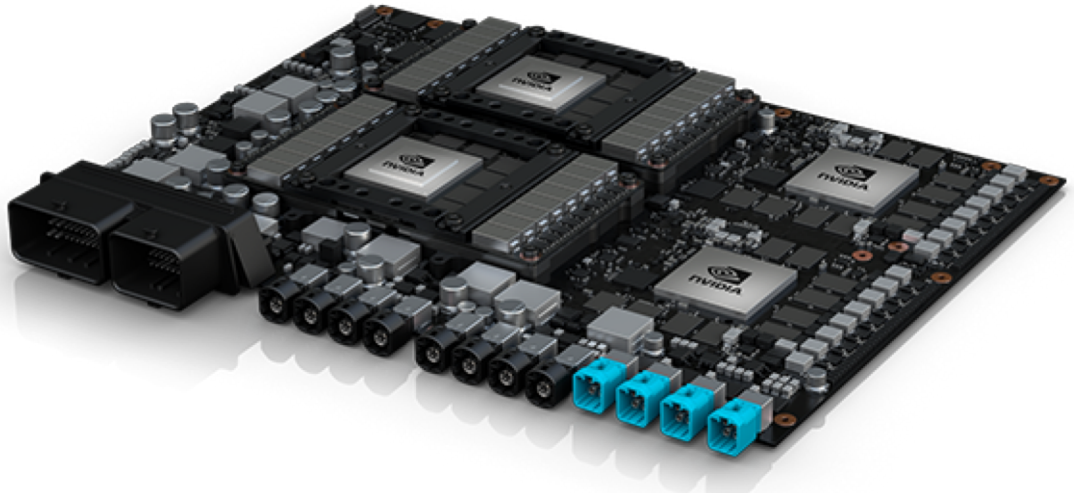
# Le parallélisme au service de la conduite automatique





La puce Nvidia Xavier Intègre :

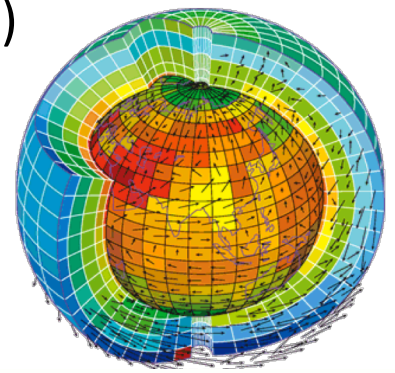
- 8 cœurs généralistes
- 1 circuit video 8K HDR
- 1 GPU de 512 cœurs pour le deep-learning



*Un ordinateur embarqué composé de 2 Xavier*

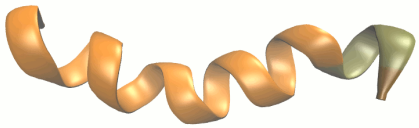
# Pourquoi calculer en parallèle ?

- Traiter des *gros* problèmes (Simulation scientifique, big data)
- Gagner du temps
  - Gagner en précision (simulation, jeu vidéo, apprentissage,…)
  - Faire plus de tests (cryptanalyse)
  - Prendre des décisions plus vite que les autres (finance)

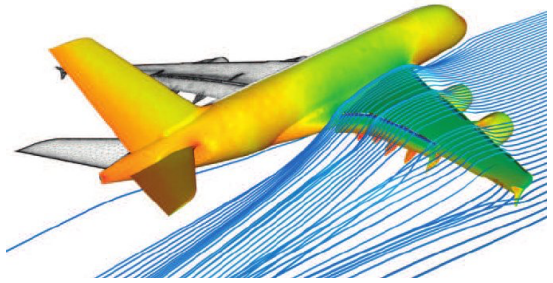


*La simulation haute performance est devenue*

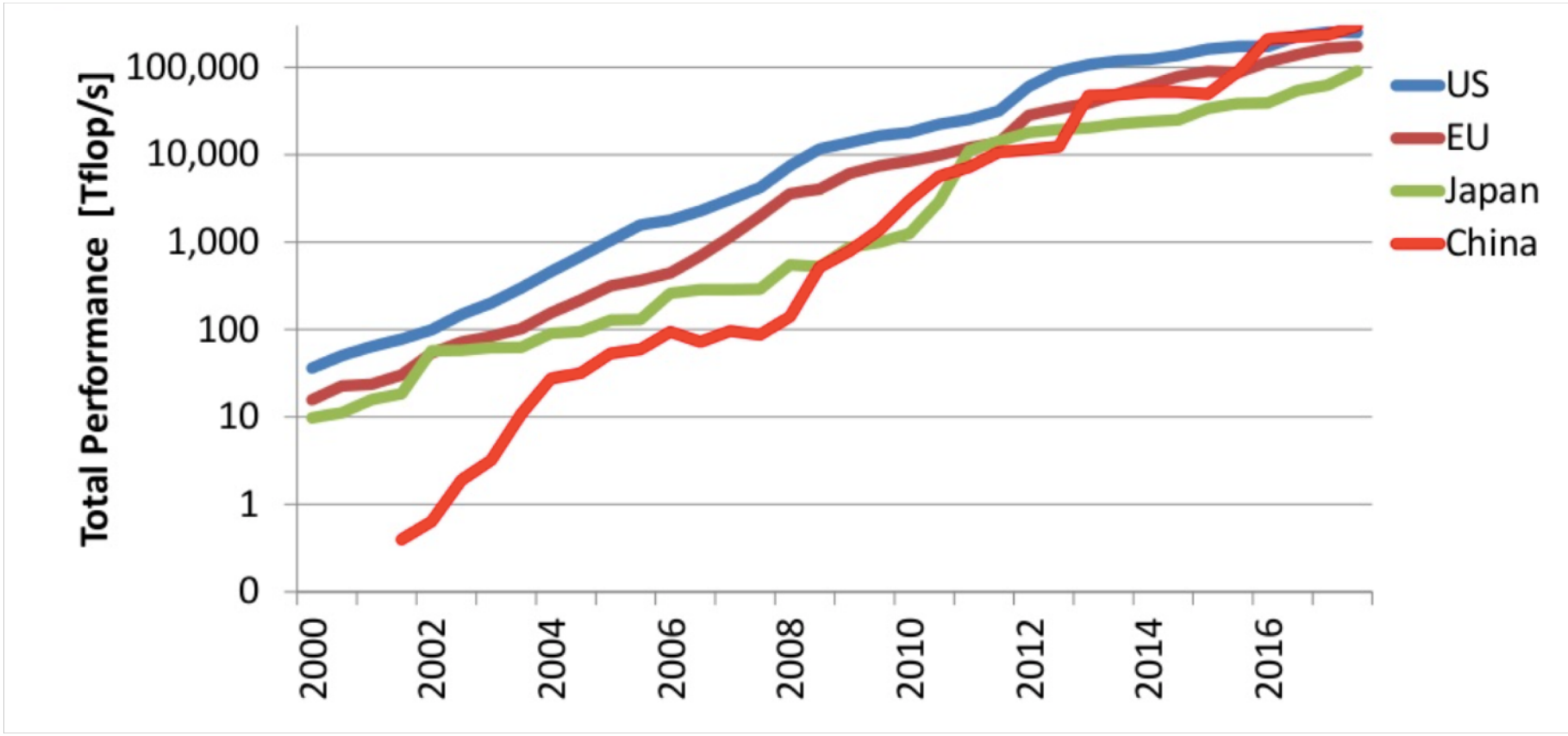
- Le *troisième pivot* de la science :
  - expérimentation / modélisation mathématique / simulation
- Incontournable dans les grands groupes industriels :
  - Simulation des phénomènes physiques / chimiques
  - Maquette numérique pour la conception et la fabrication



La démocratisation de la simulation haute performance est une des clés de notre avenir industriel.



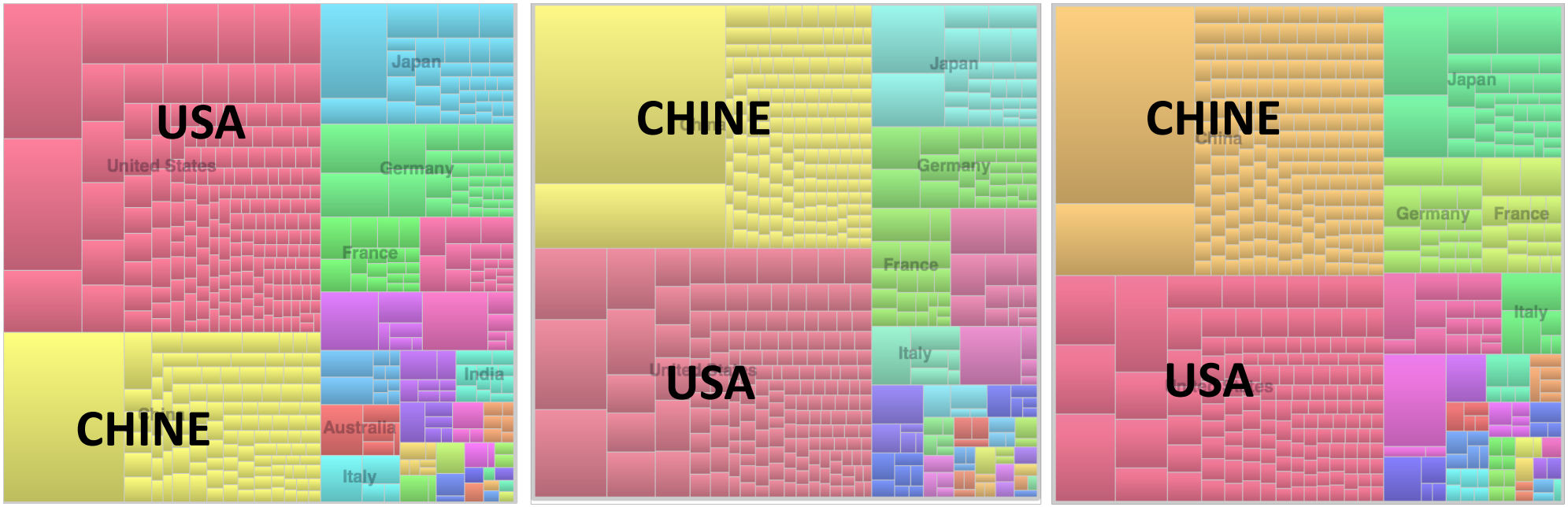
# Compétition internationale




*Puissance de calcul cumulée par « pays »*

*Basé le TOP 500 des plus puissantes plateformes de calcul connues*

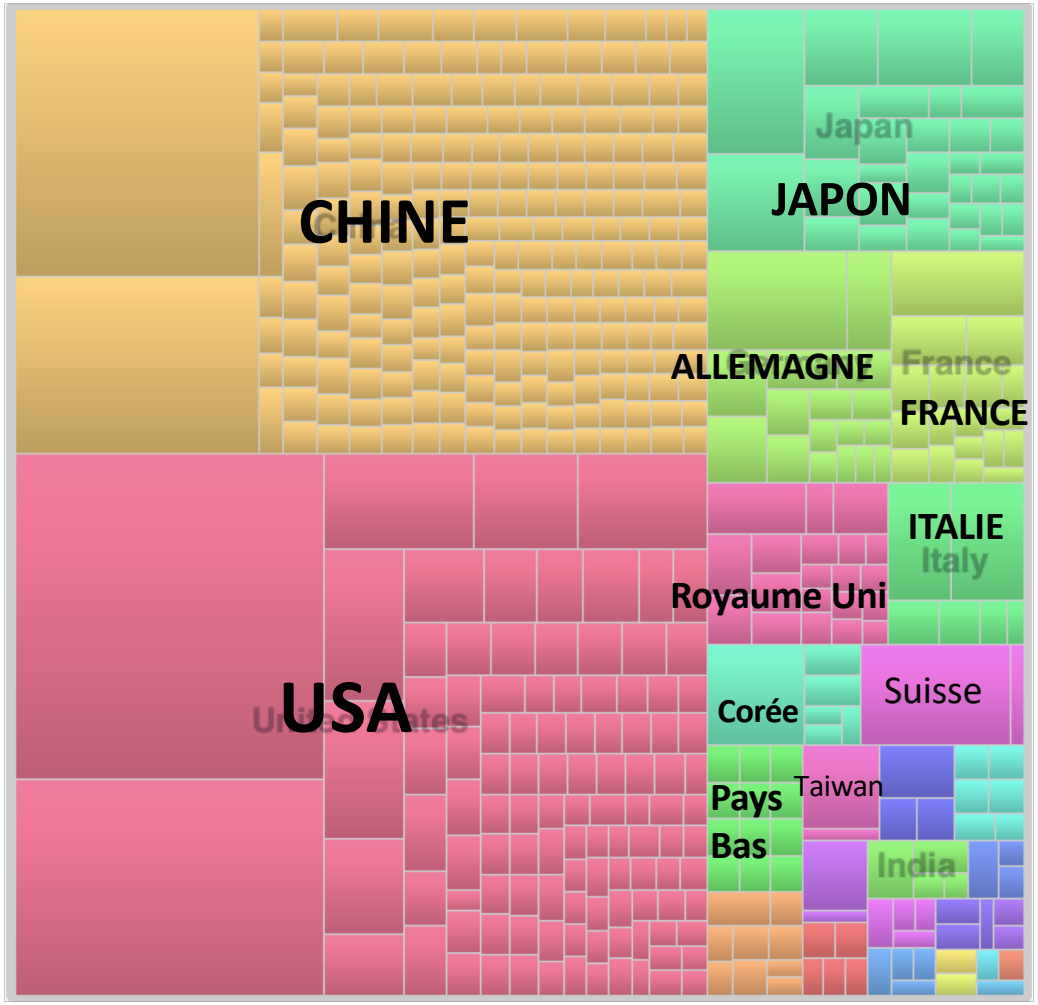
# Évolution du top 500




2015 2016 2017

L’aire de chaque rectangle est proportionnel à la puissance relative d’une machine du top 500

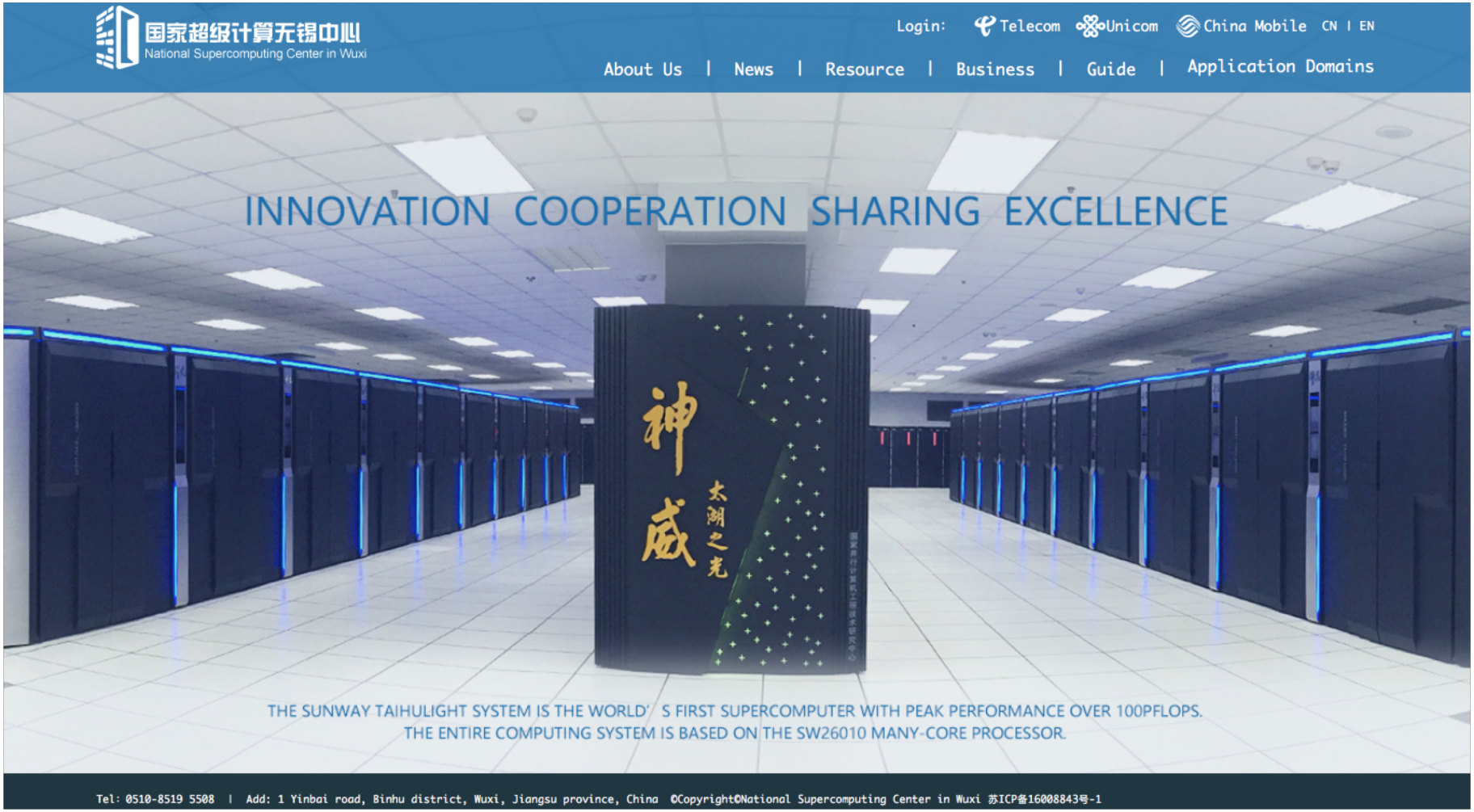
# TOP 500 novembre 2018



# TOP 1
## des plateformes de calcul recensées


- Summit (Oak Ridge National Lab – U.S. Department Of Energy)
- 2 397 824 cœurs – 145 . $10^{15}$ instructions / seconde – 10 GW
- 2 processeurs de 22 cœurs + 6 GPU V100 par nœud – 4356 nœuds
- Biologie – chimie – géologie – ingénierie - physique - informatique

# Top 2 des plateformes de calcul recensées
# Sunway TaihuLight



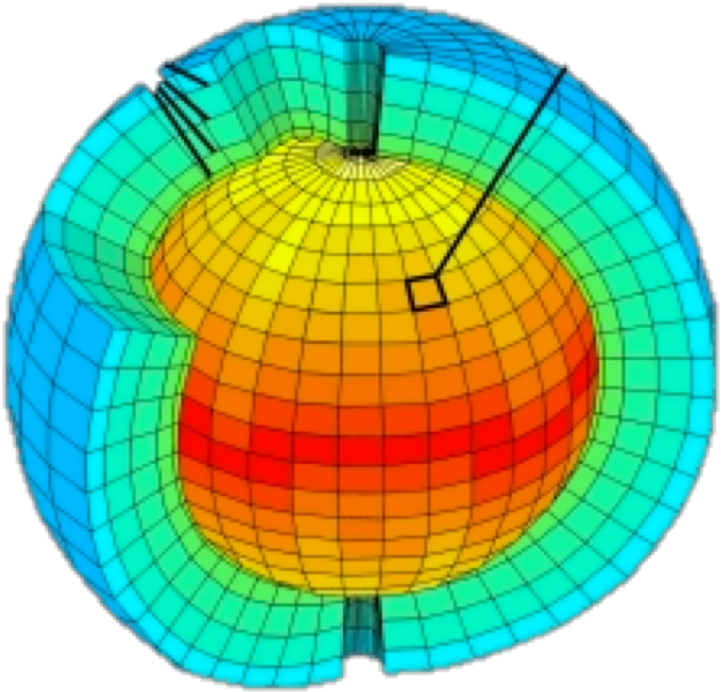

10 649 600 coeurs - 96 014 Tflop/s – 1 310 720 GB - 15 371 kW

# Modélisation météorologique

## Un exemple de simulation numérique

Objectif : calculer humidité, pression, température et vitesse du vent en fonction de x,y,z et t.

Résolution d'un système dynamique *impossible* à résoudre *formellement*

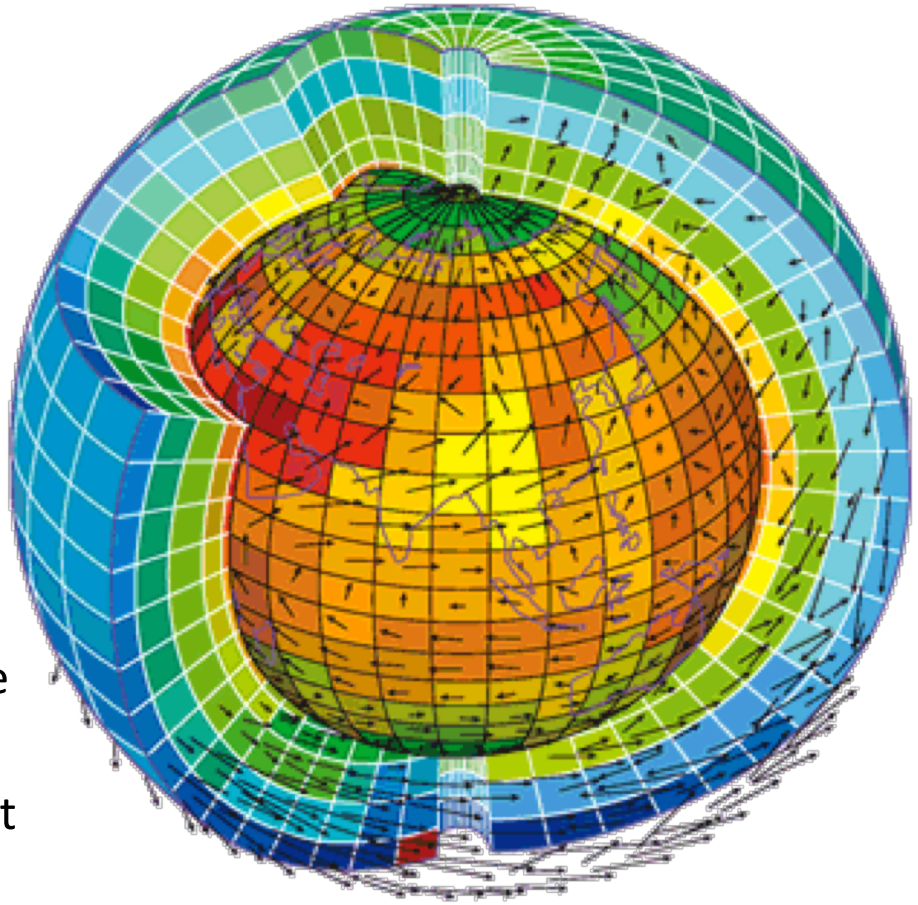


Simulation numérique déterministe :

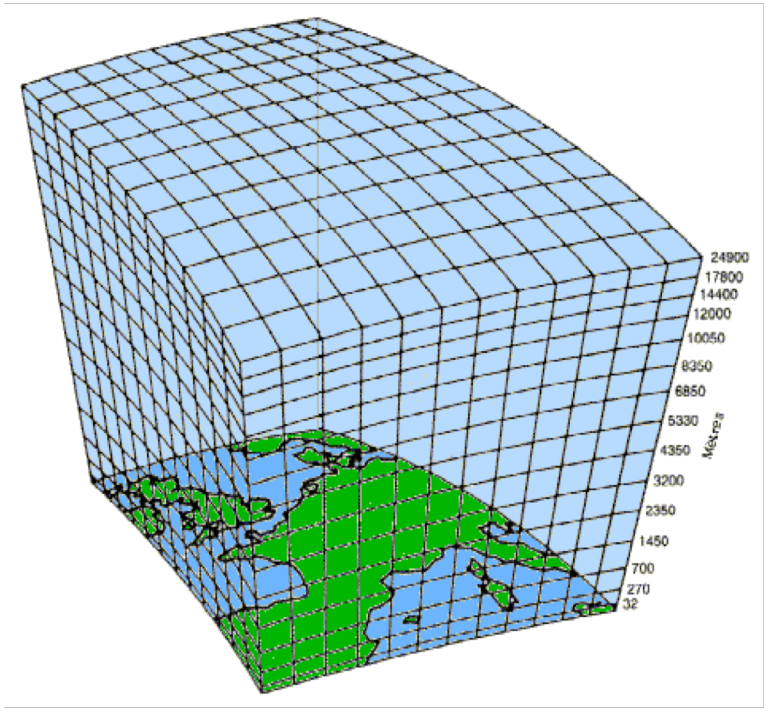
- Discrétiser l'espace
  - Illustration : 1 point pour $2km^3$
    sur 20 km d'atmosphère => $5.10^9$ points
- Initialiser le modèle
  - Données satellites et autres capteurs
  - Nécessité d'interpoler les valeurs manquantes

# Modélisation météorologique

- Faire évoluer le modèle
  - Discrétiser le temps
    - Ex: calculer par pas de 60 secondes
  - Résoudre pour chaque maille un système d'équations
    - calculer l'état suivant d'une maille en fonction de son voisinage, de l'apport solaire, de la rotation de la terre,…
    - Coût illustratif : 100 FLOP / point



# Modélisation météorologique
## illustration du coût d'une simulation




Discrétisation du temps et de l'espace :

- pas de 60 secondes
- 1 point pour 2km$^3$ sur 20 km d'atmosphère => 5.10$^9$ points

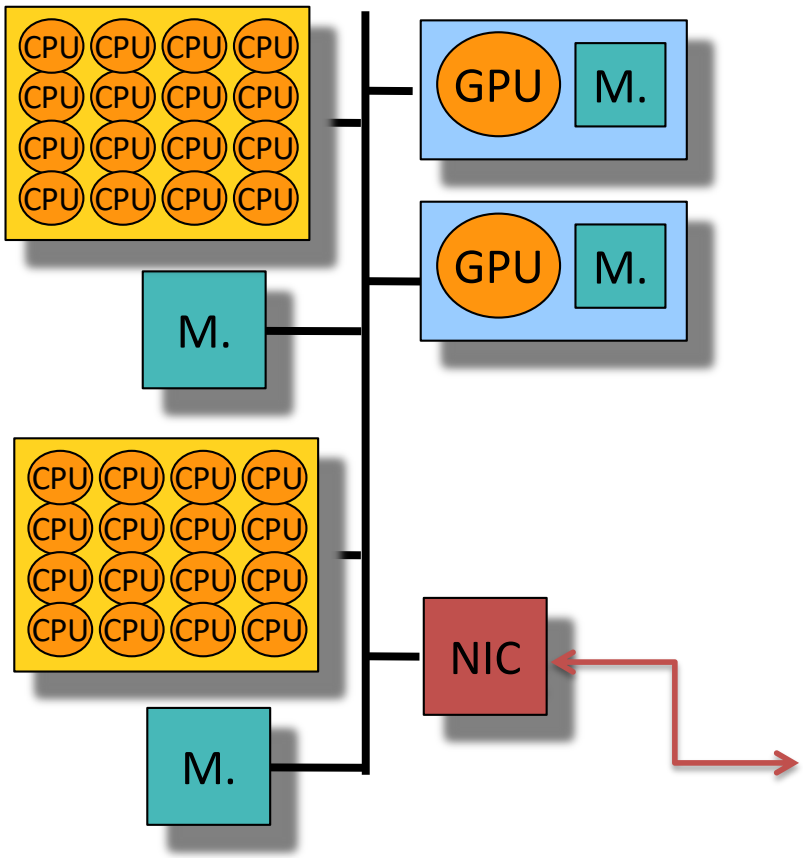
Nombre de calcul par point : 100 opérations (flop)

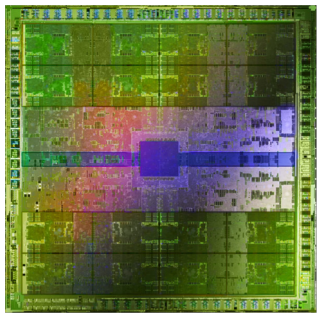
➔ **Simuler 1 minute (un pas) coûte *5.10$^{11}$ flop***

Puissance de calcul nécessaire à une simulation:

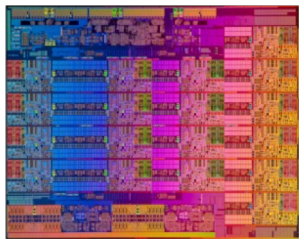
- Temps réel - **calculer 1 pas en 60s** : *5.10$^{11}$ / 60 = 8,33 . 10$^9$ =* ***8,33 Giga flop/s***
- Prévision - **calculer 7 jours en 1h** *: 5.10$^{11}$ ** 7j * 24h * 60m / 3600 = **1 400 Gflop/s**
- Climatologie - **50 ans en 1 jour** : 1 060 000 Gflop/s **= 1,06 péta flop / s**

# Composants d'un PC sur-vitaminé



GPU Nvidia V100
5120 cœurs cuda
(= 80 pipelines)

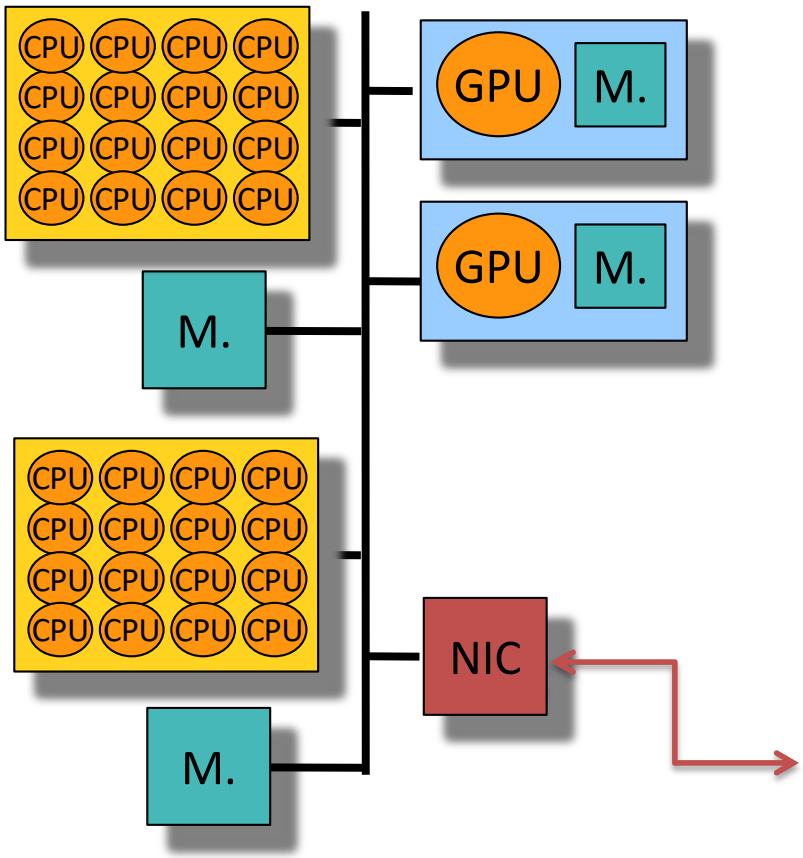


Intel E7-8890
48 cœurs virtuels
(= 24 pipelines)

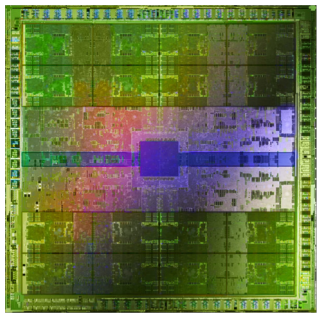
Carte réseau
Rapide 200 Gb/s

*Les ordinateurs deviennent massivement parallèles.*

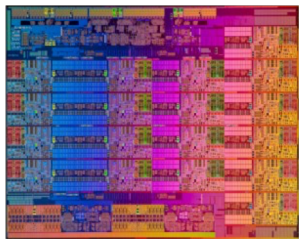
# Performances d'un PC sur-vitaminé




1 PC sur-vitaminé = 2 GPU + 2 processeurs
= 17,6 Téra flop/s



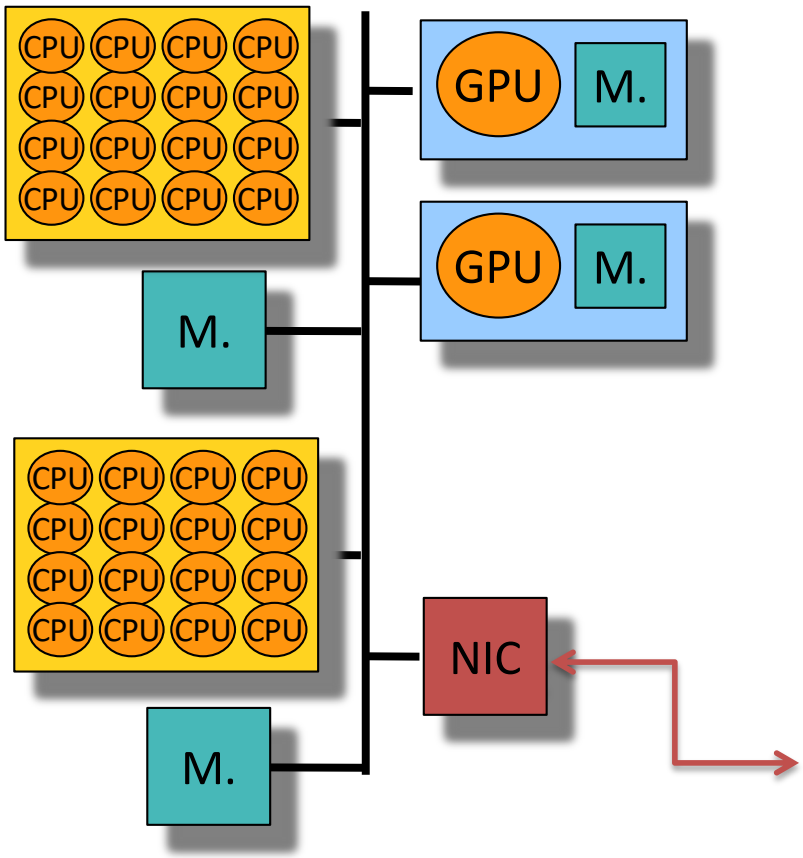
GPU Nvidia V100
5120 cœurs cuda
7.8 TFlops



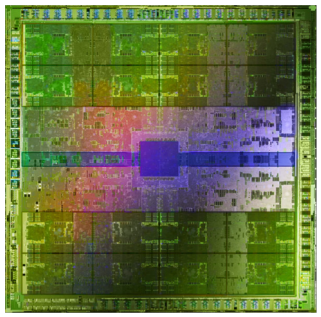
Intel E7-8890
48 cœurs virtuels
912 GFlops

Carte réseau
Rapide 200 Gb/s

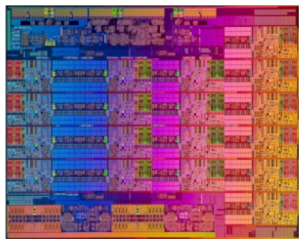
# Temps pour calculer une prévision




1 PC sur-vitaminé = 2 GPU + 2 processeurs
= 1 prévision météo à 7 jours en 4 minutes 45



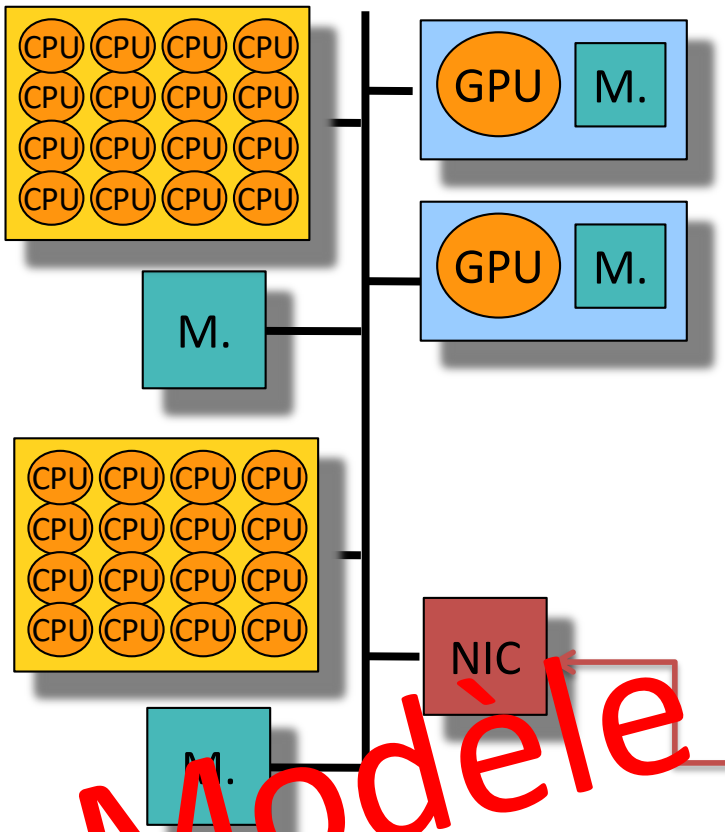
GPU Nvidia V100
5120 cœurs cuda
10 minutes 30



Intel E7-8890
48 cœurs virtuels
1 heure 30

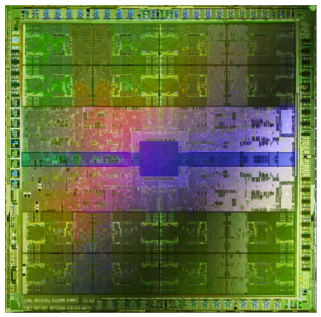
Carte réseau
Rapide 200 Gb/s

# Temps pour calculer une prévision

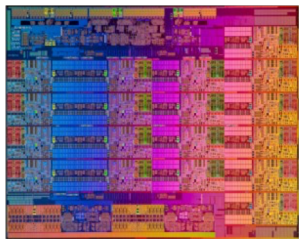



1 PC sur-vitaminé = 2 GPU + 2 processeurs
= 1 prévision météo à 7 jours en 4 minutes 45

Modèle simpliste



GPU Nvidia V100
5120 cœurs cuda
10 minutes 30

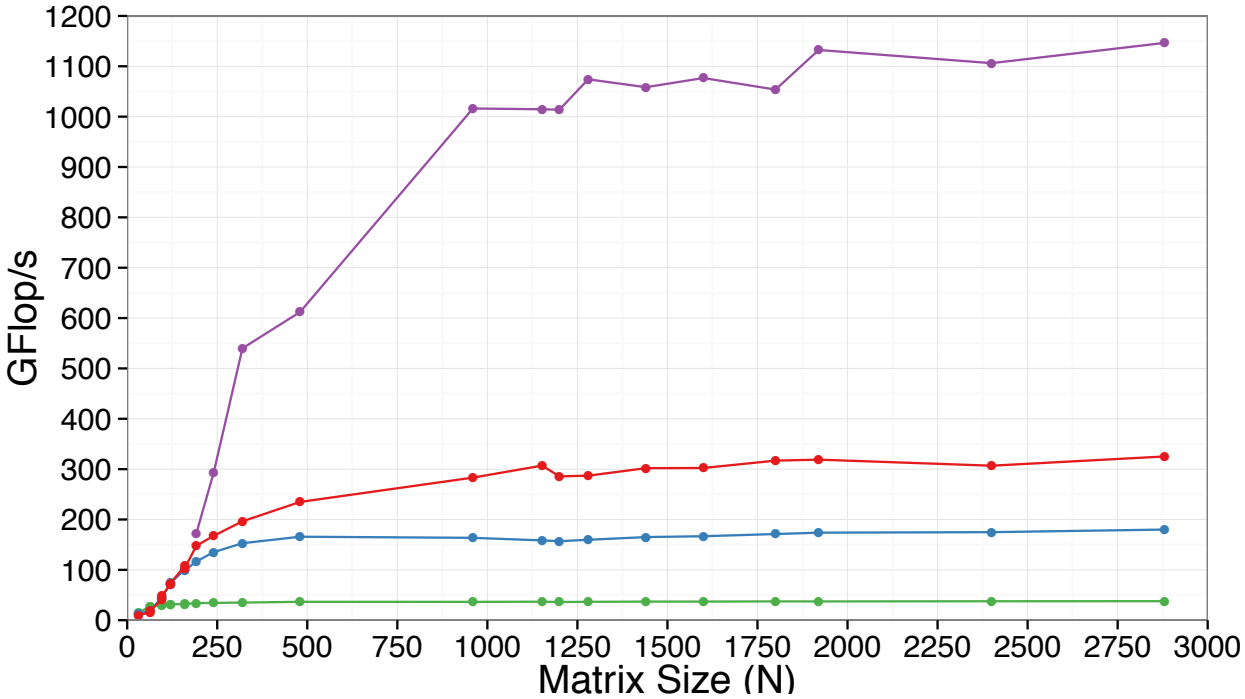


Intel E7-8890
48 cœurs virtuels
1 heure 30

- **Une prévision n'est pas une masse informe d'opérations flottantes**
- **Il faut aussi tenir compte du temps d'accès aux données**

# Est-il facile d'obtenir les performances crêtes ?

- C'est difficile, même dans les cas faciles
  - Produit de matrices à l'aide des bibliothèques Intel MKL et CudaBLAS
  - E5-2680 v3 - 2.5 MHZ
  - Kepler 40 (1430 GF/s)

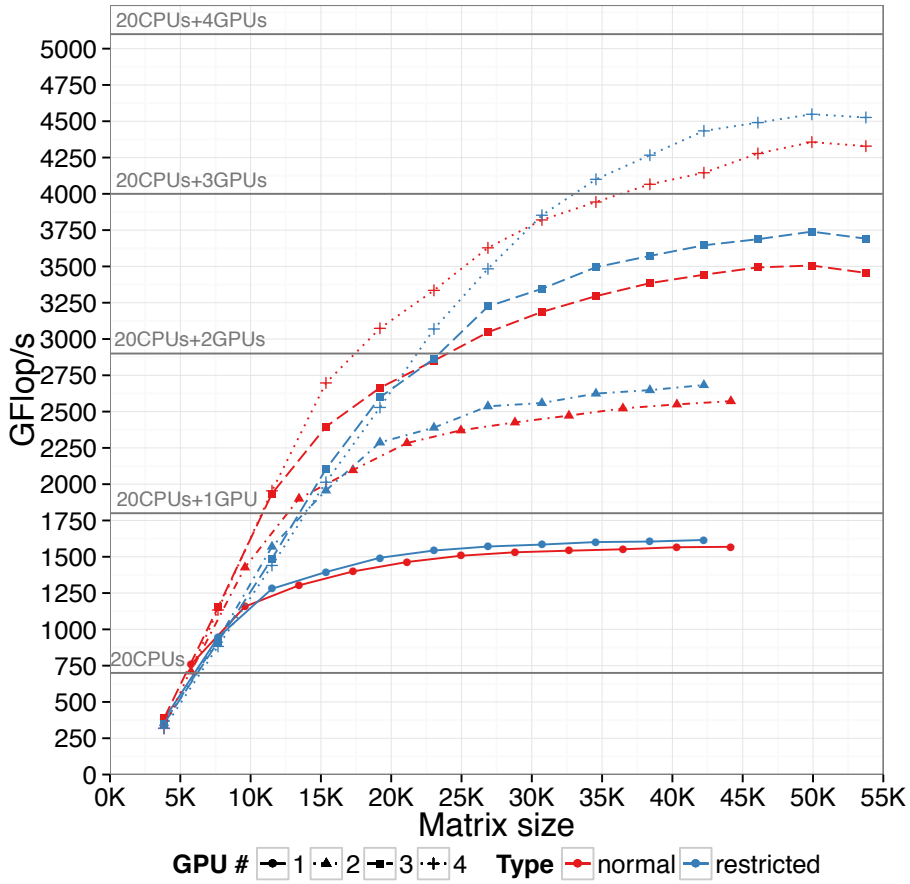


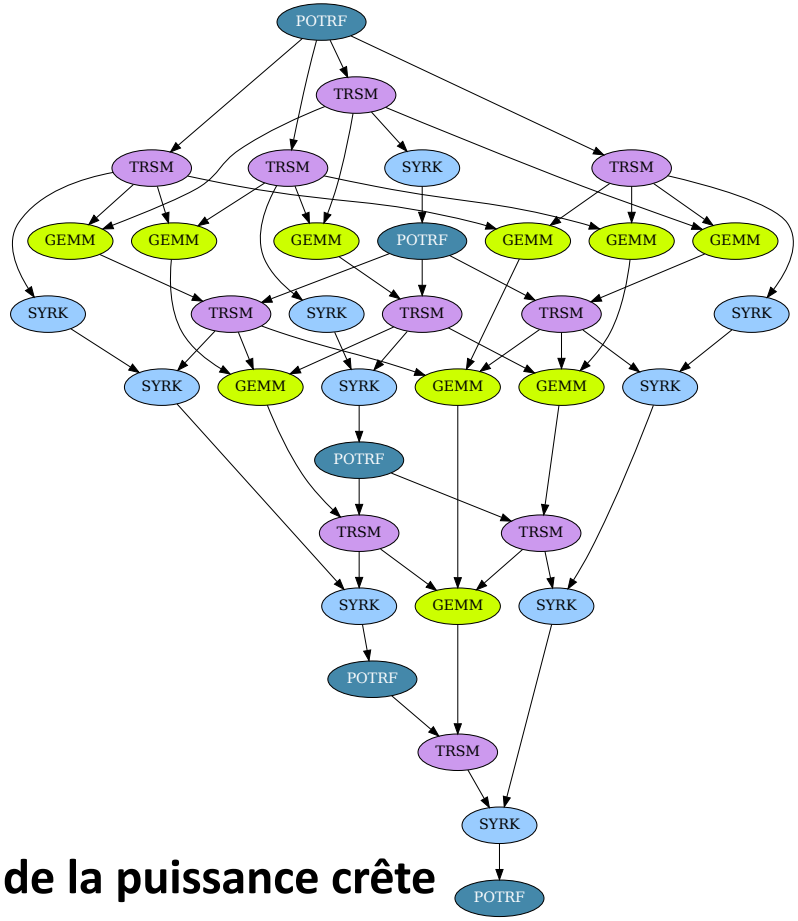

Efficacité
95% 1 cœur
93% 5 cœurs
87% 10 cœurs
81% 1 GPU

# Est-il facile d'obtenir des performances ?

- Calcul matriciel plus complexe

  (Factorisation de Cholesky)




**~60% de la puissance crête**

Intuitivement l'accélération est bornée par la durée d'exécution du chemin critique.

# Loi d'Amdahl

$$\text{accélération} = \text{speedup} = \frac{\text{temps du meilleur prog. séquentiel}}{\text{temps du prog. parallèle}}$$

*On note*

- *s* la part séquentielle d'un programme
- *1 - s* est la part parallélisable de l'exécution
- p le nombre de processeurs

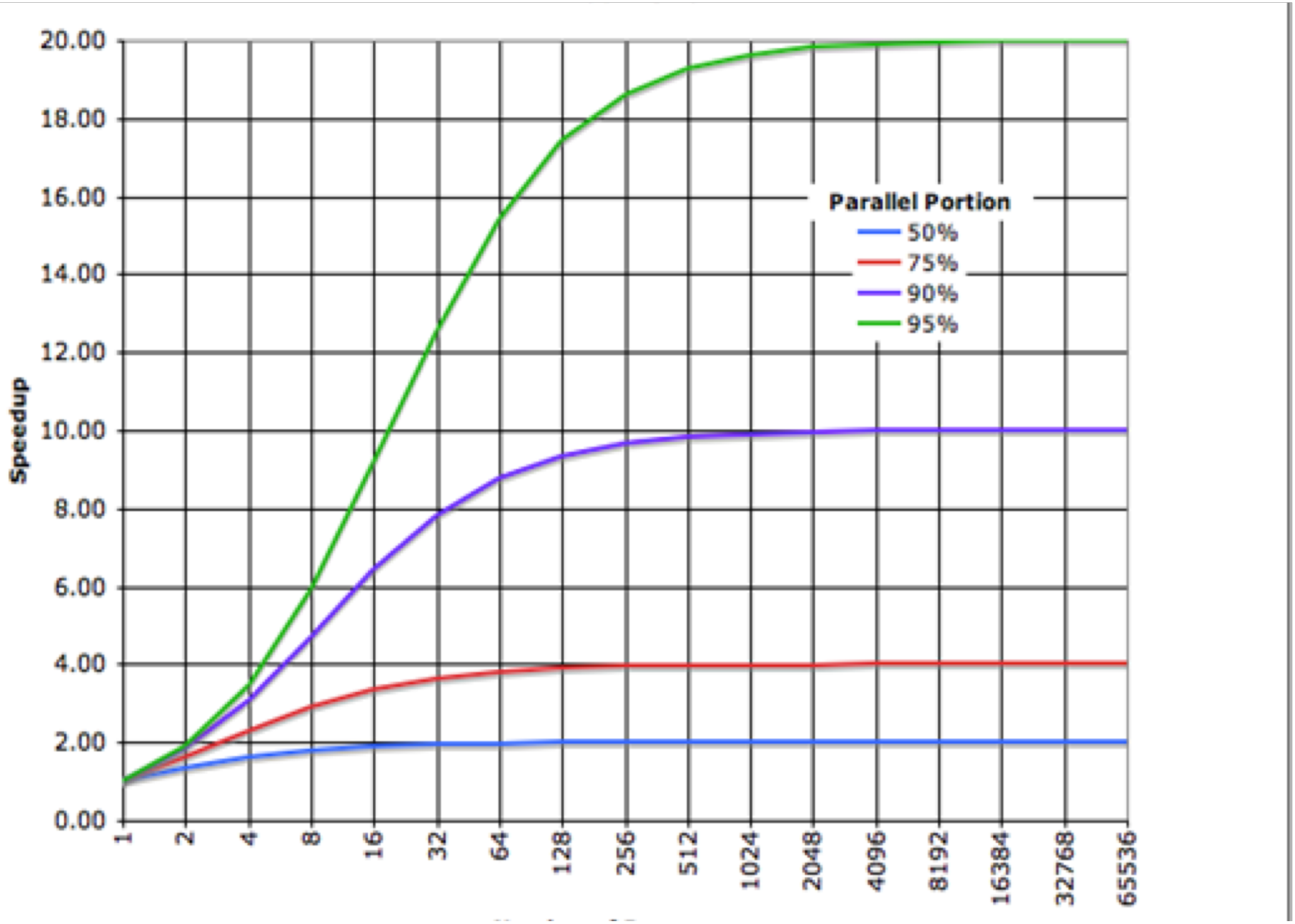
$$durée\ parallèle \geq \left(S + \frac{1-s}{p}\right) \times durée\ séquentielle$$

L'accélération de l'exécution sur une machine à p processeur est ainsi bornée par

$$accélération \leq \frac{1}{S + \frac{1-s}{p}} < \frac{1}{S}$$

*« si 1% de l'application est séquentielle on n'arrivera pas à aller 100 fois plus vite »*

# Loi d'Amdahl


Speedup
20.00
18.00
16.00
14.00
12.00
10.00
8.00
6.00
4.00
2.00
0.00
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
Parallel Portion
50%
75%
90%
95%

# Exemple Amdahl : tri fusion

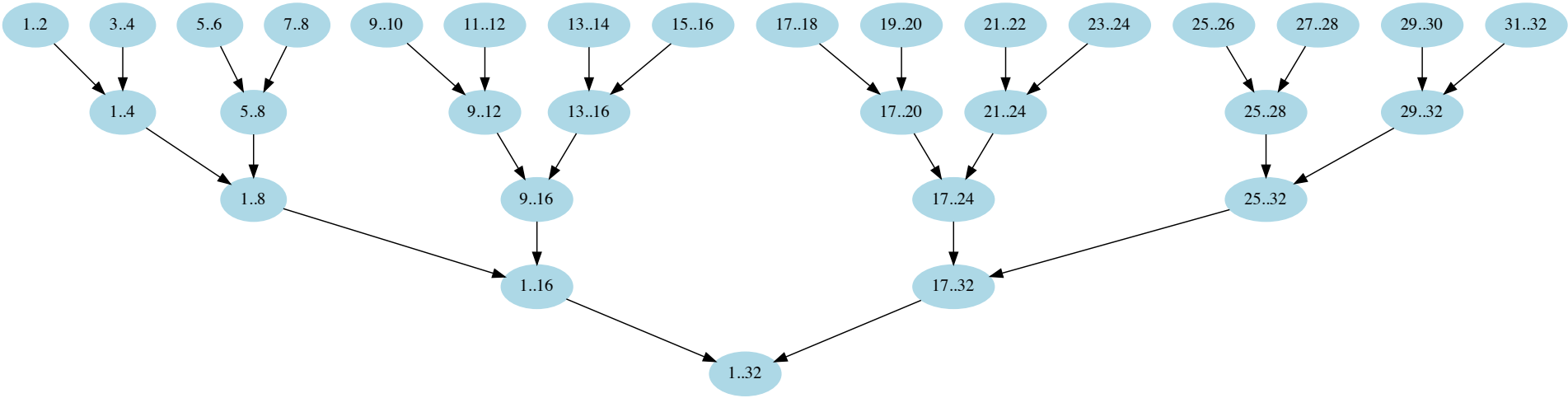


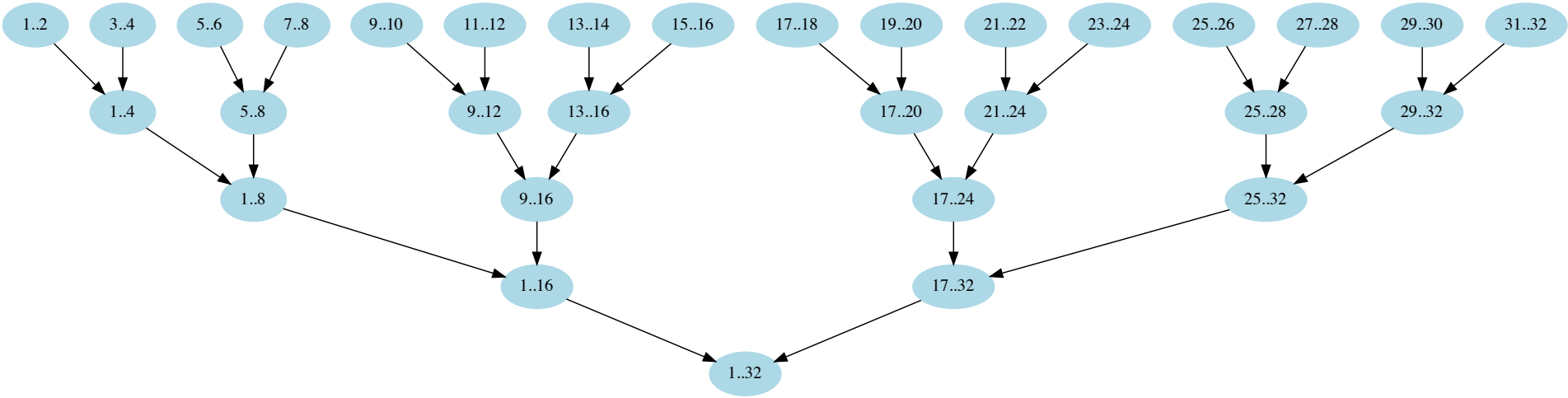

Tableau de $2^k$ éléments à trier à l'aide du tri fusion récursif simpliste :

- on trie en parallèle chaque moitié du tableau
- on fusionne en séquentiel les moitiés

Part séquentielle = Chemin critique = toute branche d'une anti-feuille à l'anti-racine

$$accélération \leq \frac{coût\ de\ l'arbre}{coût\ d'une\ branche}$$

# Exemple Amdahl : tri fusion



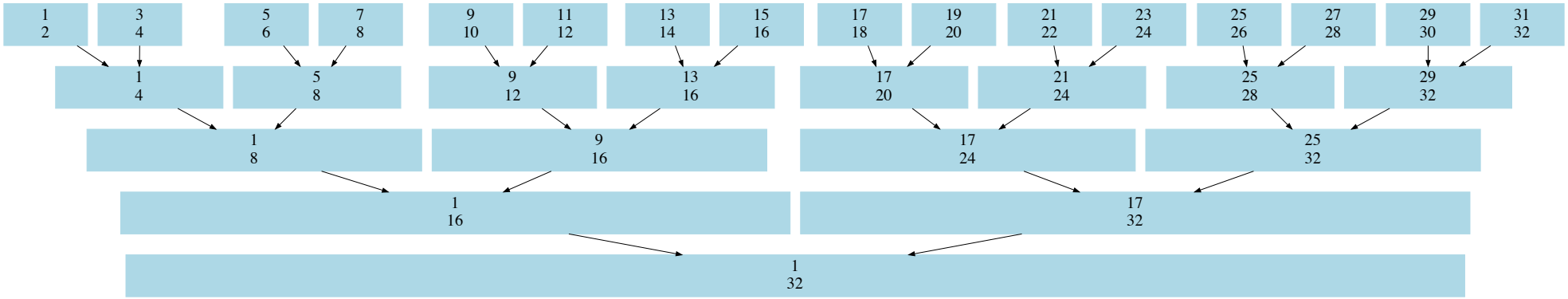

Pire cas - recopie de tous les éléments à chaque niveau de l'arbre : $2^k$ écritures par niveau

CoutSeq(k) le coût d'un tri de $2^k$ éléments
CoutPar(k) le coût d'une branche d'un tri de $2^k$ éléments

CoutSeq(k) = $k.2^k$ écritures
CoutPar(k) = $2 + 4 + \cdots + 2^k = 2^{k+1} - 2$ écritures

$$accélération(k) \leq \frac{coût\ de\ l'arbre}{coût\ d'une\ branche} = \frac{k.2^k}{2^{k+1}-2} \approx \frac{k}{2}$$ (dès que $2^k \gg 2$)

# Exemple Amdahl : tri fusion



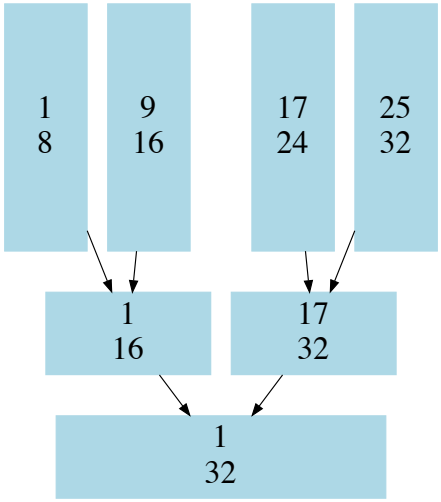

Pire cas - recopie de tous les éléments à chaque niveau de l'arbre : $2^k$ écritures par niveau

CoutSeq(k) le coût d'un tri de $2^k$ éléments
CoutPar(k) le coût d'une branche d'un tri de $2^k$ éléments

CoutSeq(k) = $k.2^k$ écritures
CoutPar(k) = $2 + 4 + \cdots + 2^k = 2^{k+1} - 2$ écritures

$$accélération(k) \leq \frac{coût\ de\ l'arbre}{coût\ d'une\ branche} = \frac{k.2^k}{2^{k+1}-2} \approx \frac{k}{2}$$ (dès que $2^k \gg 2$)

# Exemple Amdahl : tri fusion



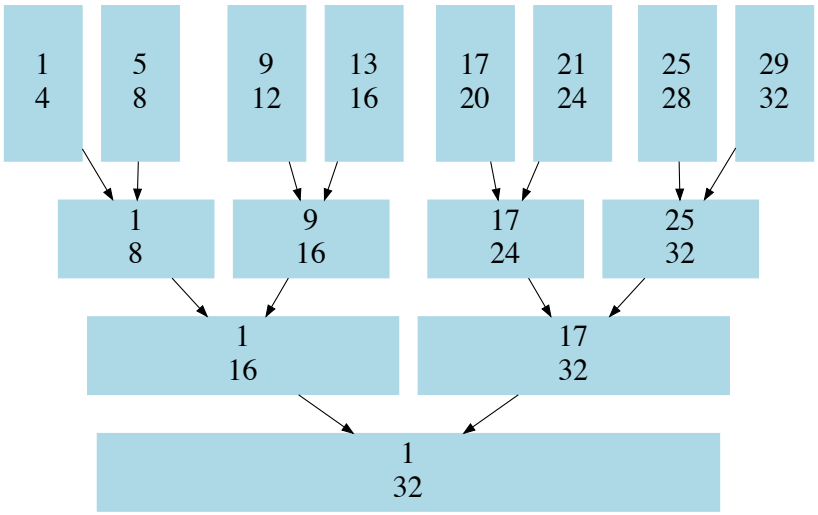

Quelle accélération espérer lorsqu'on coupe la parallélisation à partir de $2^n$ éléments ?

$$accélération(k,n) \leq \frac{coût\ de\ l'arbre}{coût\ d'une\ branche\ coupée}$$

CoutPar(k,n) = Coût d'une branche (coupée à partir de $2^n$ éléments)
CoutPar(k,n) = CoutPar(k) – CoutPar(n) + CoutSeq(n)

$$\text{CoutPar(k,n)} = 2^{k+1} - 2^{n+1} + n.\,2^n$$

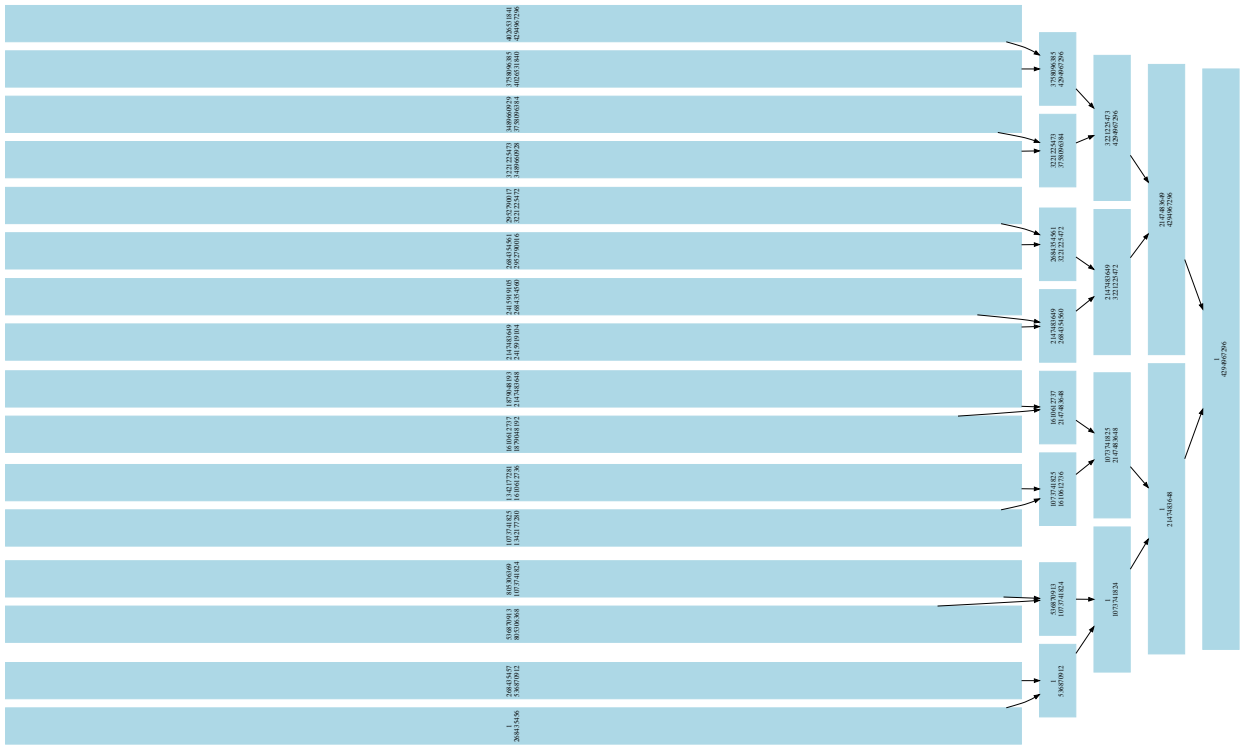
# Exemple Amdahl : tri fusion




$$accélération(k,n) \leq \frac{coût\ de\ l'arbre}{coût\ d'une\ branche\ coupée}$$

$$accélération(k,n) \leq \frac{k.2^k}{2^{k+1}-2^{n+1}+n.2^n} = \frac{k}{2+\frac{n-2}{2^{k-n}}}$$

# Exemple Amdahl : tri fusion



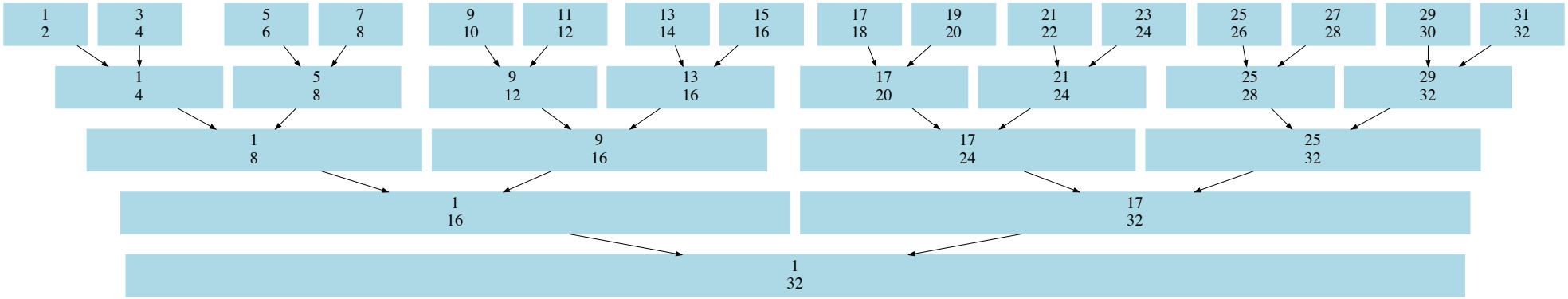
k=32, n=28

$$accélération(k,n) \leq \frac{k.2^k}{2^{k+1}-2^{n+1}+n.2^n} = \frac{k}{2+\frac{n-2}{2^{k-n}}}$$

| n | Accélération |
|---|---|
| 32 | 1 |
| 31 | 1,93939 |
| 30 | 3,55556 |
| 29 | 5,95349 |
| 28 | 8,82759 |
| 27 | 11,5056 |
| 26 | 13,4737 |
| 25 | 14,681 |
| 24 | 15,3408 |
| 23 | 15,6785 |
| 22 | 15,8453 |
| 21 | 15,9261 |
| 20 | 15,9649 |
| 19 | 15,9834 |
| 18 | 15,9922 |
| 17 | 15,9963 |
| 16 | 15,9983 |
| 15 | 15,9992 |

Tableau de $2^{32}$ éléments

# Exemple Amdahl : tri fusion




Tri fusion parallèle simpliste : on fusionne en séquentiel.

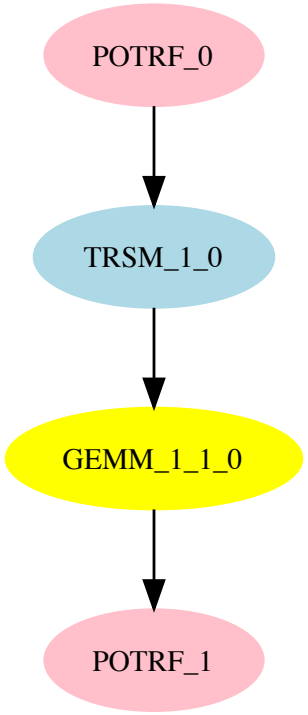
Accélération limitée ici à $\frac{k}{2} \Longrightarrow$ inutile de paralléliser très profondément.

Nécessité de paralléliser la fusion pour améliorer l'accélération

- On considère l'élément médian du premier tableau
- Recherche dichotomique du 1er élément supérieur au médian dans le 2nd
  - Complexité parallèle $O\left(\frac{n \log n}{p}\right)$ en moyenne et $O\left(\frac{n^2}{p}\right)$ dans le pire cas
  - Plus de calcul que pour la version séquentielle

# Factorisation de Cholesky

## *speed-up maximal en fonction du nombre de tuiles*

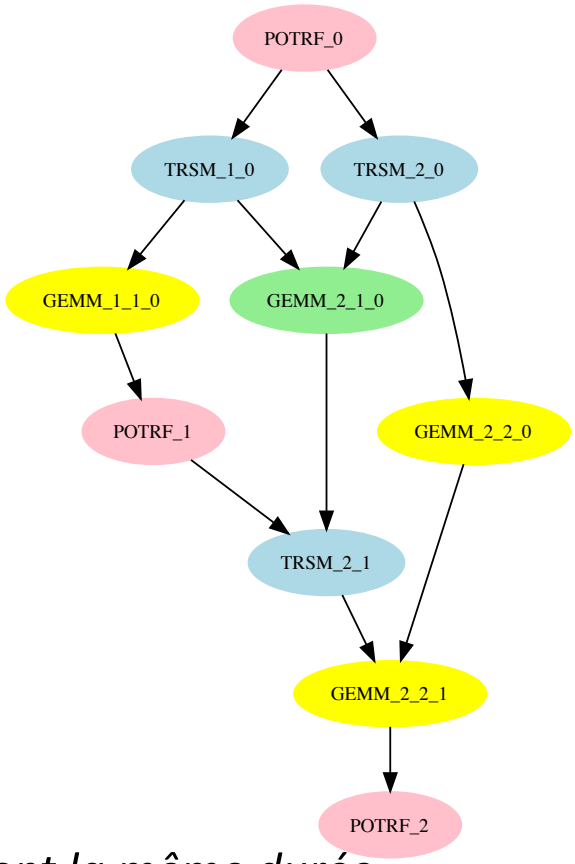



*On suppose que toutes les tâches ont la même durée.*
*Le rapport entre le nombre de tâches et la longueur du chemin critique borne l'accélération que l'on peut espérer en parallélisant l'exécution des tâches.*

# Tâches : 4
Longueur chemin critique : 4
Rapport : 1

# Factorisation de Cholesky

## *speed-up maximal en fonction du nombre de tuiles*

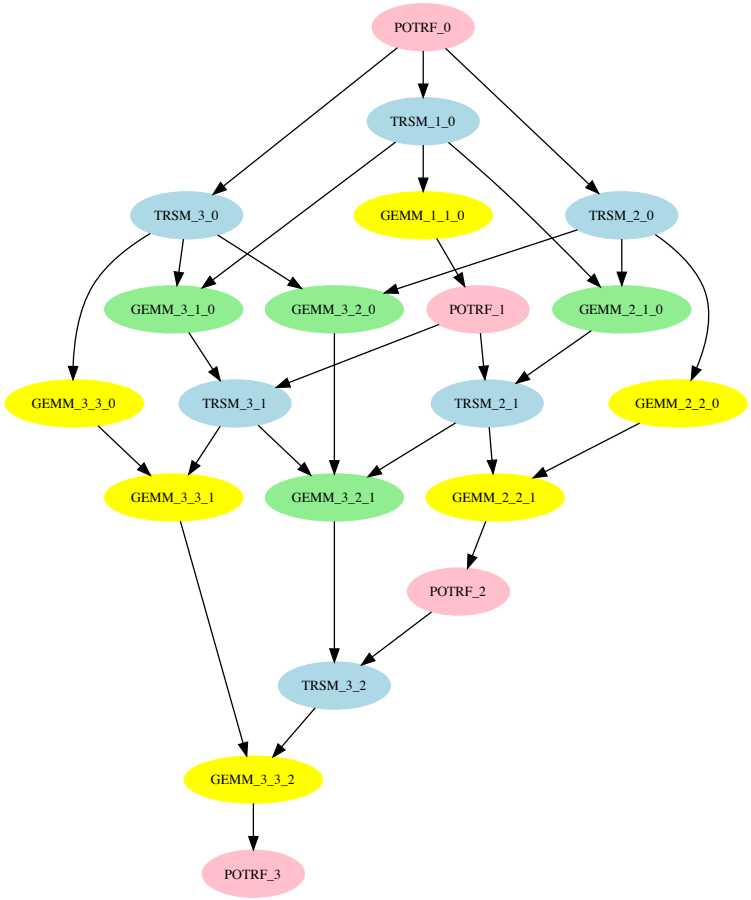



*On suppose que toutes les tâches ont la même durée.*
*Le rapport entre le nombre de tâches et la longueur du chemin critique borne l'accélération que l'on peut espérer en parallélisant l'exécution des tâches.*

# Tâches : 10
Longueur chemin critique : 7
Rapport : 1,42

# Factorisation de Cholesky

*speed-up maximal en fonction du nombre de tuiles*



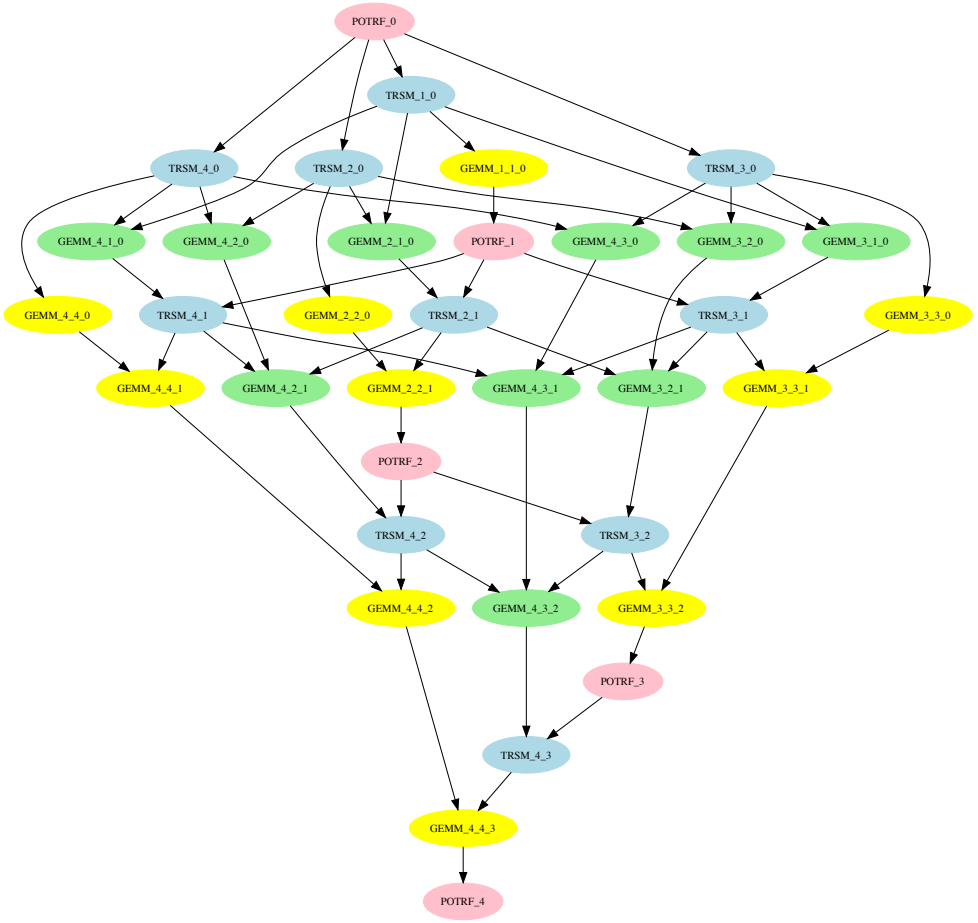

# Tâches : 20

Longueur chemin critique : 10

Rapport : 2

# Factorisation de Cholesky

*speed-up maximal en fonction du nombre de tuiles*



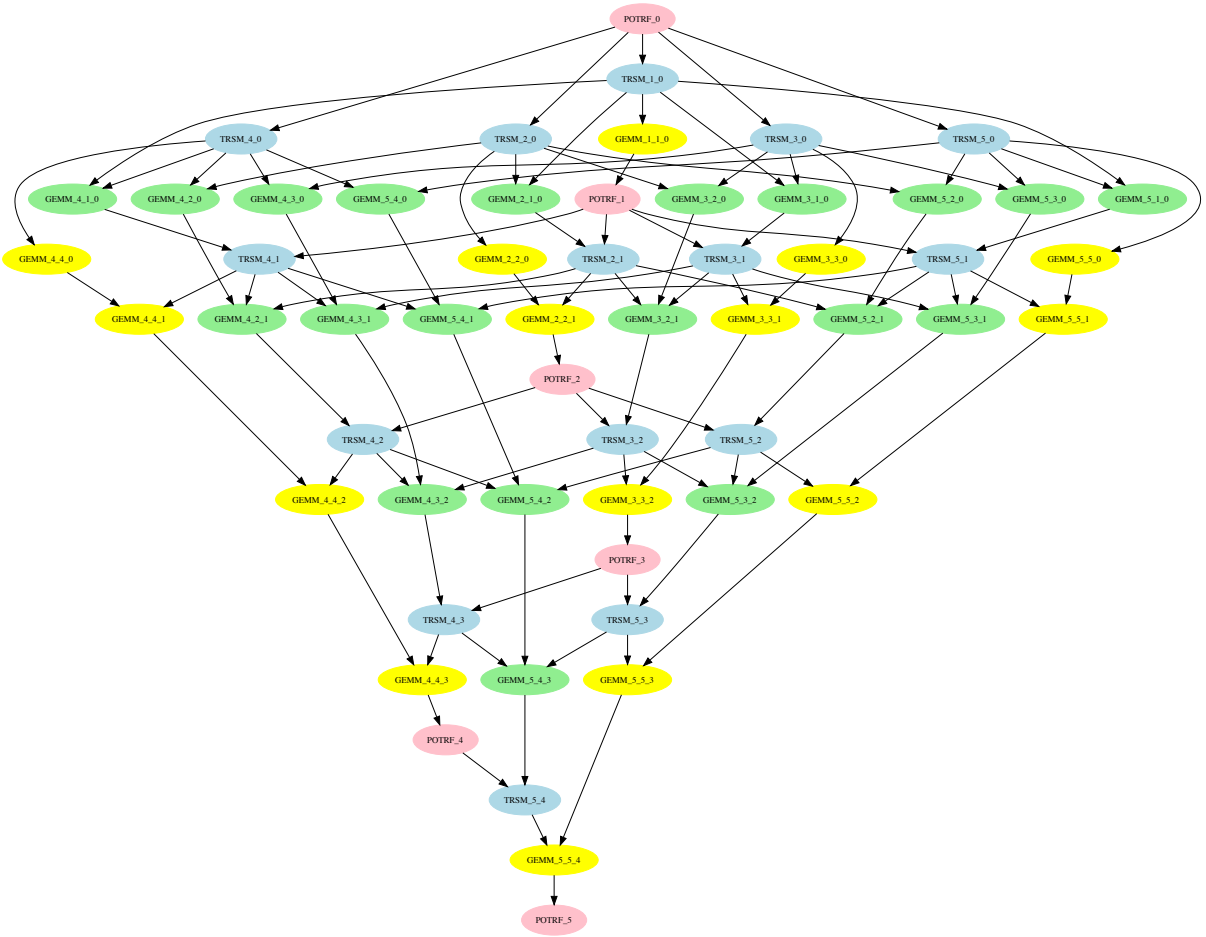

# Tâches : 35

Longueur chemin critique : 13

Rapport : 2,69

# Factorisation de Cholesky

*speed-up maximal en fonction du nombre de tuiles*



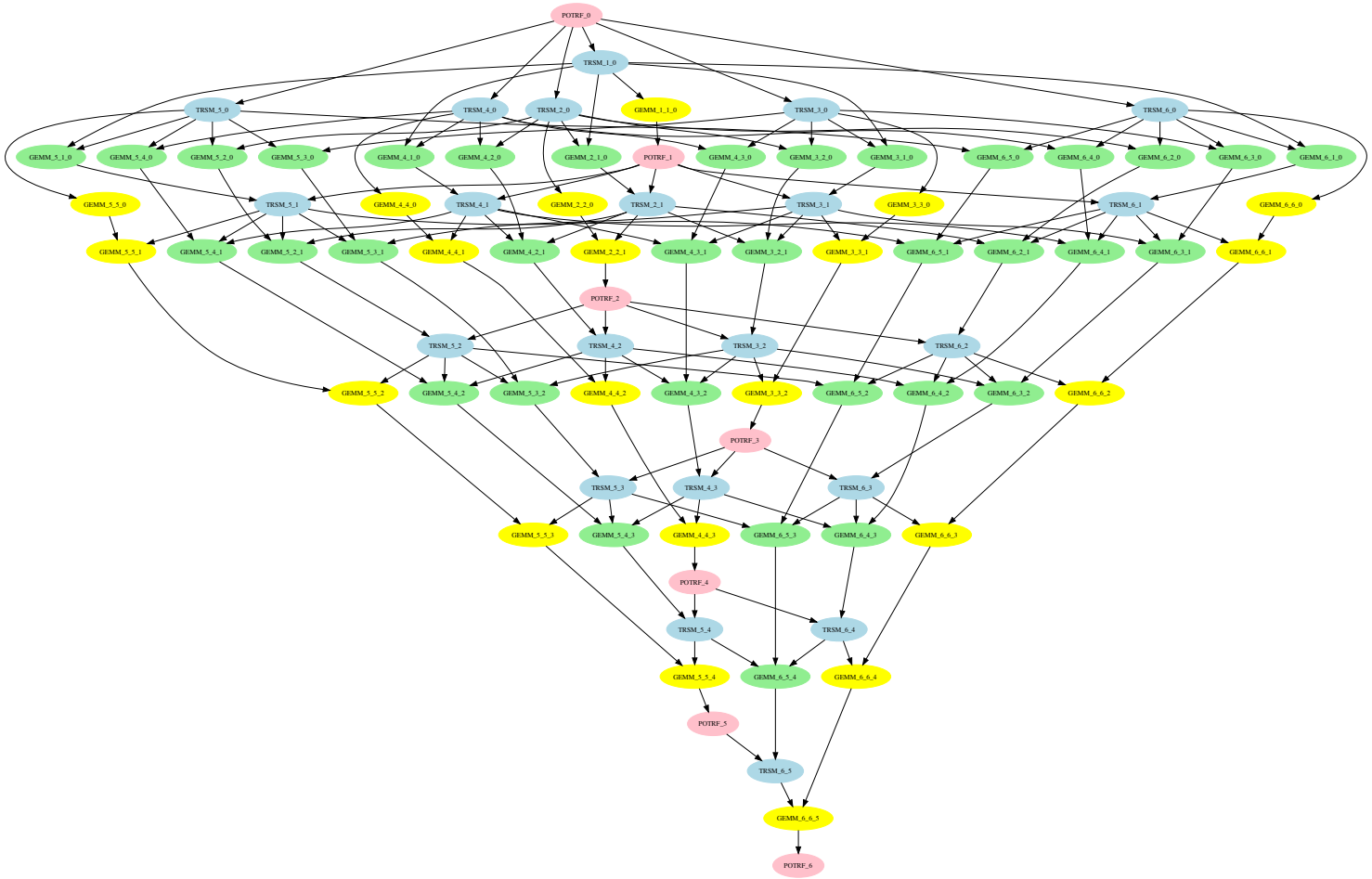

# Tâches : 56

Longueur chemin critique : 16

Rapport : 4

# Factorisation de Cholesky

*speed-up maximal en fonction du nombre de tuiles*




# Tâches : 84

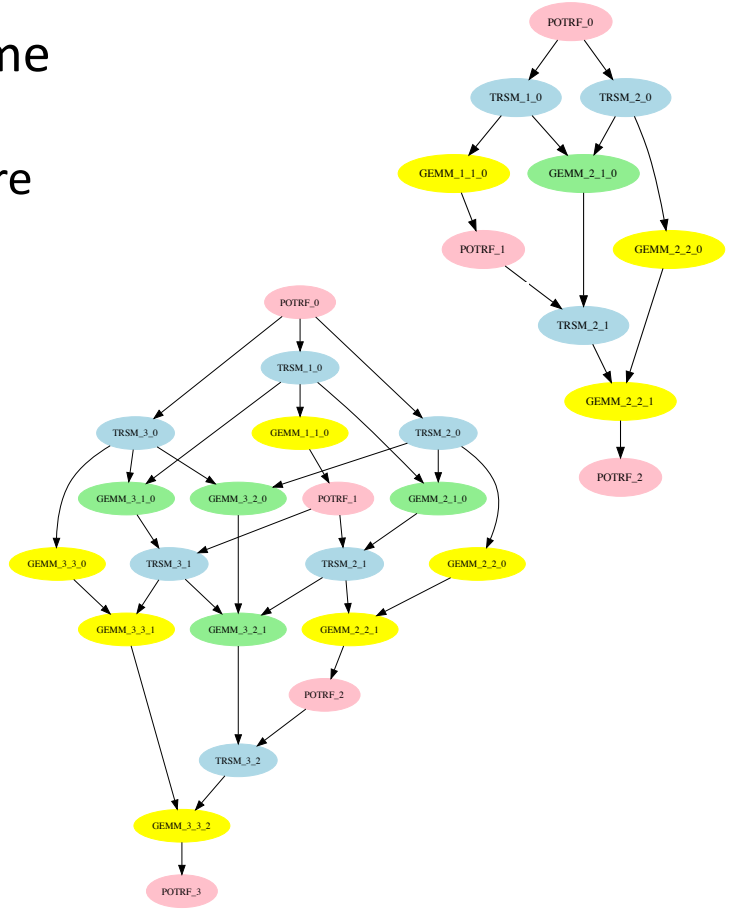
Longueur chemin critique : 19

Rapport : 4,42

# Comment obtenir des performances optimales ?

- Il faut extraire suffisamment de parallélisme
  - Pour occuper *toutes* les unités de calcul
  - Nécessite souvent du calcul supplémentaire

- Quelle finesse de grain de parallélisation ?
  - Gros grain
    - ✓ Efficacité des unités de calcul
    - ? Parallélisme
  - Grain fin
    - ? Efficacité  des unités de calcul
    - ? Surcoût
    - ✓ Parallélisme




- Il faut utiliser correctement la mémoire
  - ne pas être *bêtement* limité par la bande passante de la mémoire

# Nos objectifs

- Se plonger dans des environnements de programmation parallèle
  - Maitriser le cœur d'OpenMP
  - S'initier à OpenCL
- Modifier des algorithmes séquentiels pour les adapter à une machine cible
  - Améliorer la localité mémoire
  - Paralléliser un calcul
- Améliorer le temps d'exécution d'un programme
  - Observer l'exécution d'un programme
  - Explorer des pistes d'amélioration
  - Tirer des conclusions : comment coder des programmes efficaces ?