

Programmation multicœurs et GPU

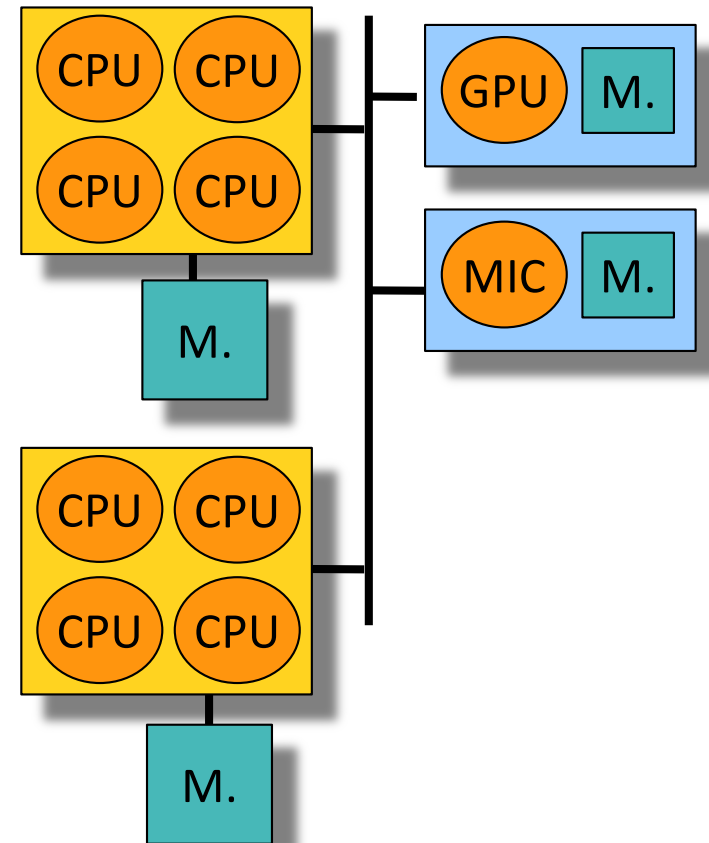
Raymond Namyst

Dept. of Computer Science
University of Bordeaux, France

<https://gforgeron.gitlab.io/it224/>

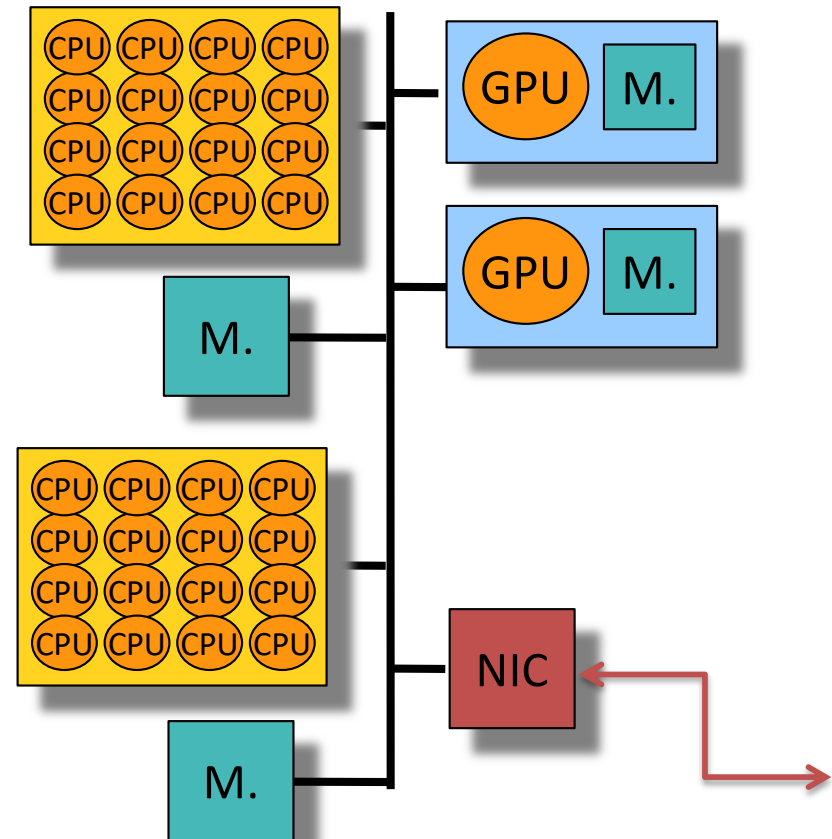
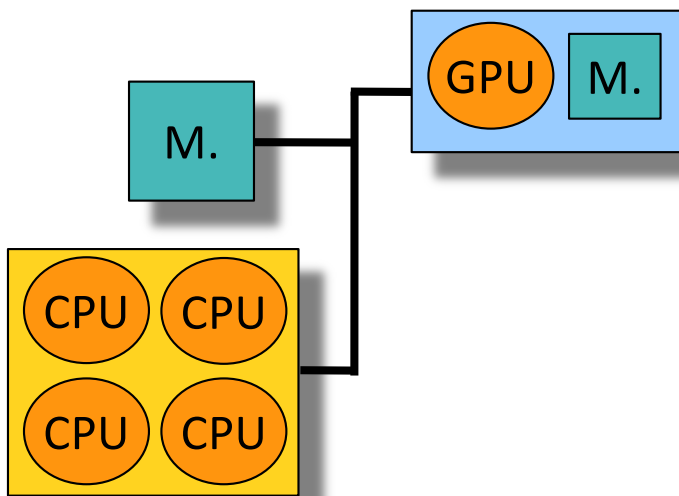
IT224 : Plan du cours

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



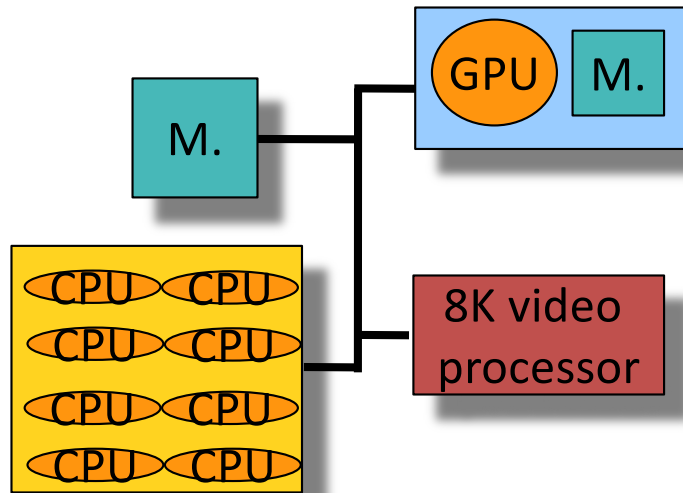
Est-ce important de savoir écrire des programmes parallèles ? De savoir programmer un GPU ?

Téléphone, PC portable,...



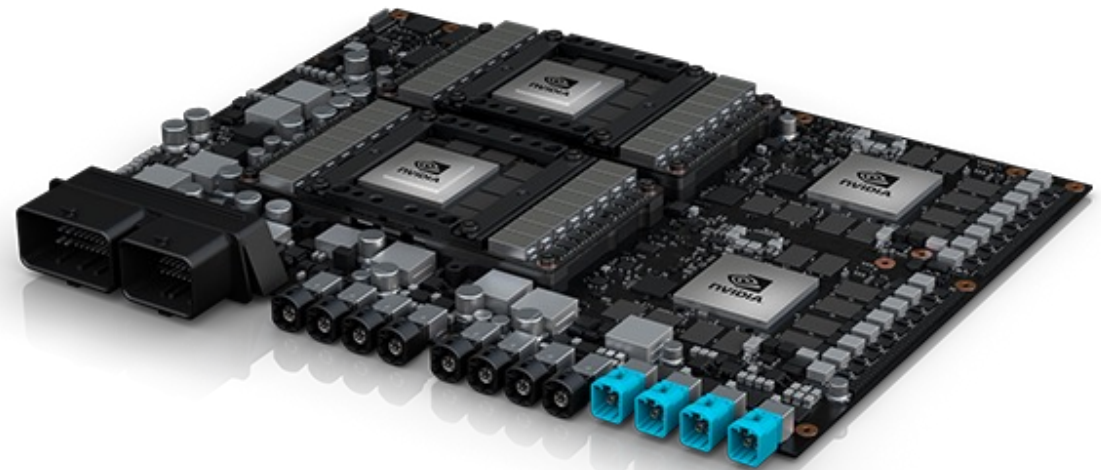
..., PC « gamer », serveur de calcul.

Le parallélisme au service de la conduite automatique



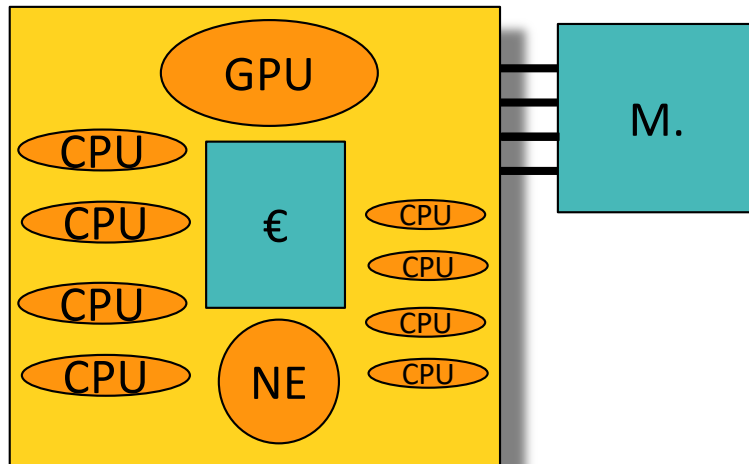
La puce Nvidia Xavier Intègre :

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



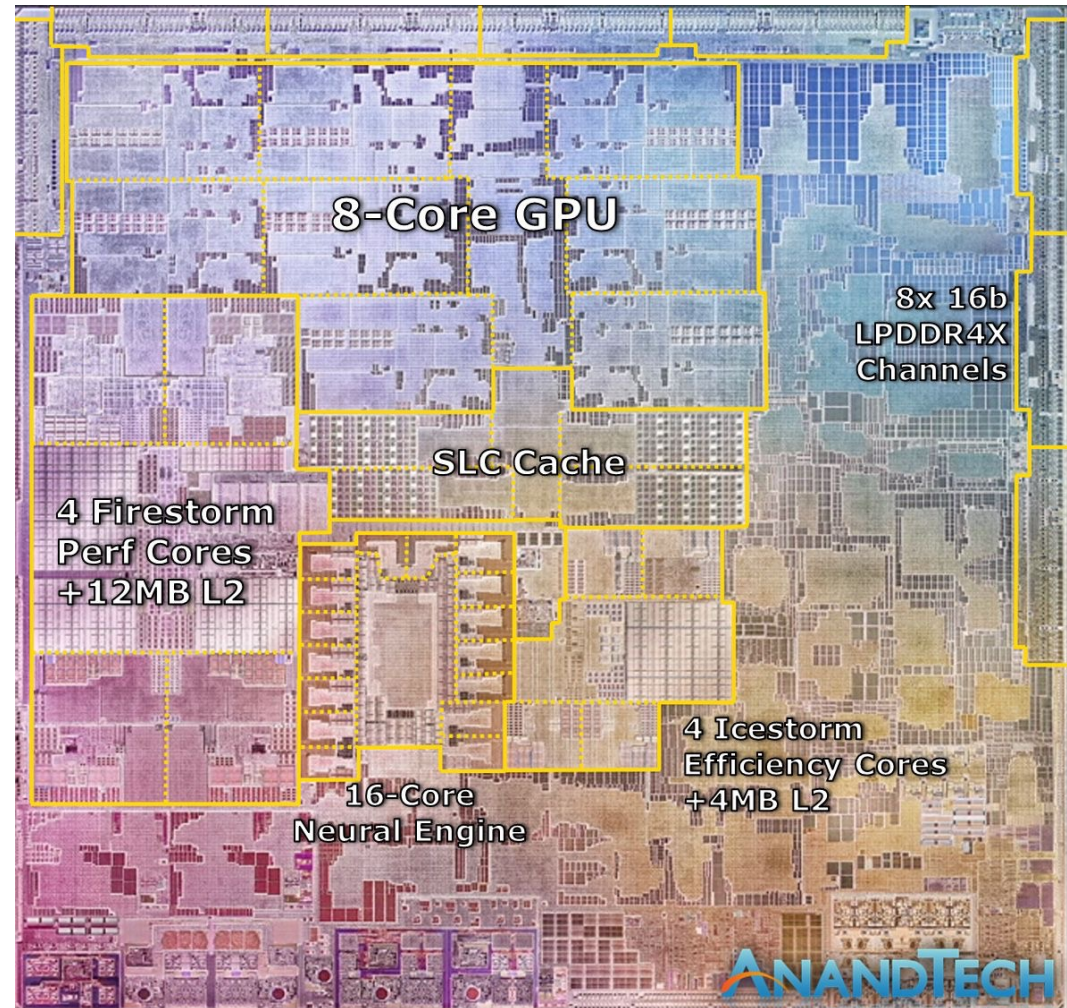
Un ordinateur embarqué composé de 2 Xavier

Des processeurs de plus en plus hétérogènes



La puce Apple M1 intègre :

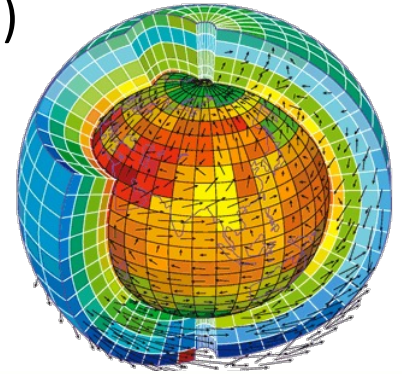
- 4 cœurs généralistes haute performance
- 4 cœurs généralistes basse consommation
- 1 GPU de 8 pipelines
- 1 *Neural Engine* de 16 cœurs



Circuit intégré de la puce M1

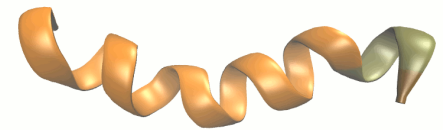
Pourquoi calculer en parallèle ?

- 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)

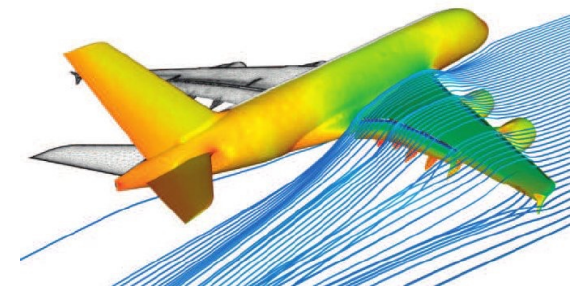


La simulation haute performance est devenue

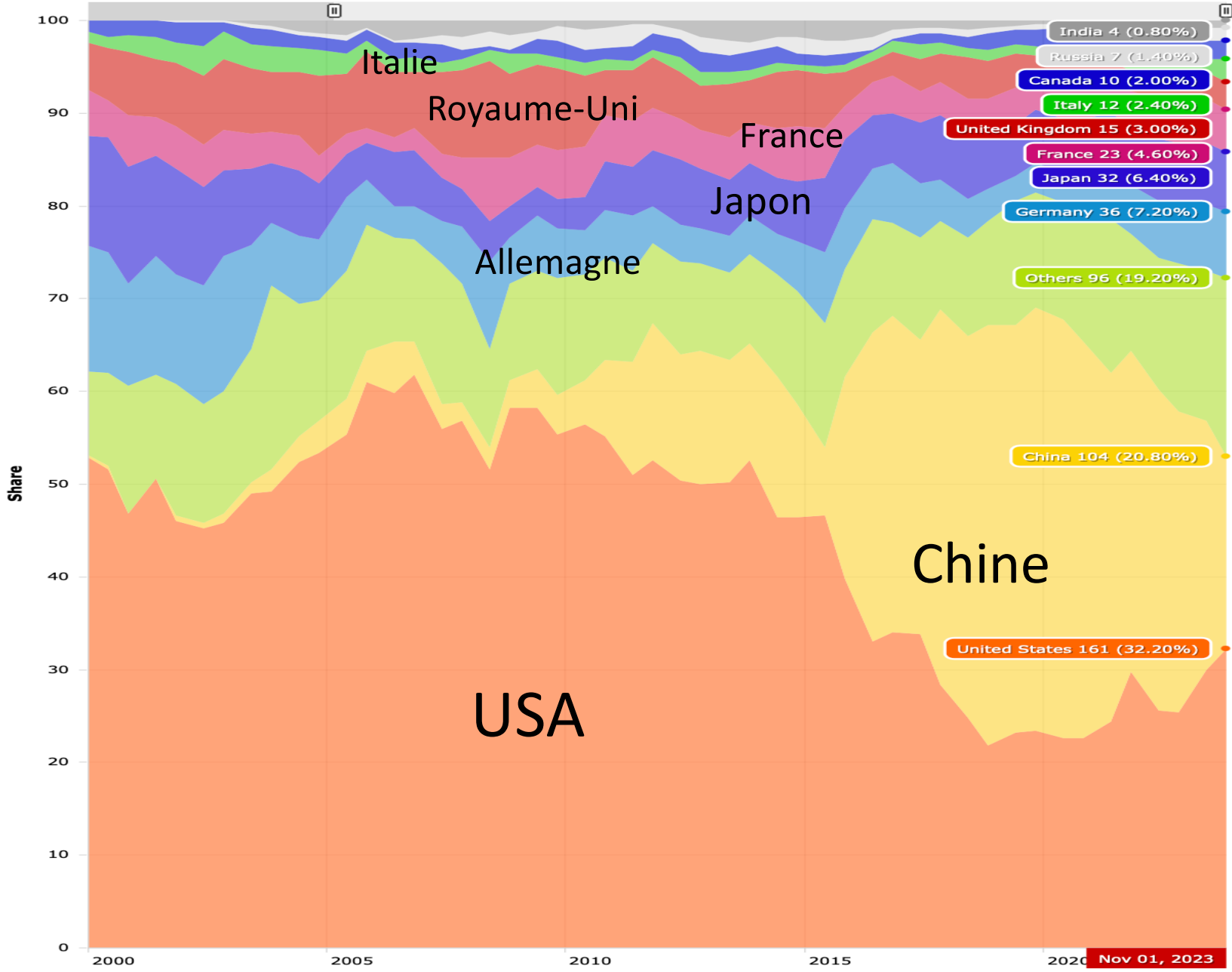
- Le *troisième pivot* de la science :
 - expérimentation / modélisation mathématique / simulation
- Incontournable dans les grands groupes industriels :
 - Simulation des phénomènes physiques / chimiques
 - Maquette numérique pour la conception et la fabrication



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



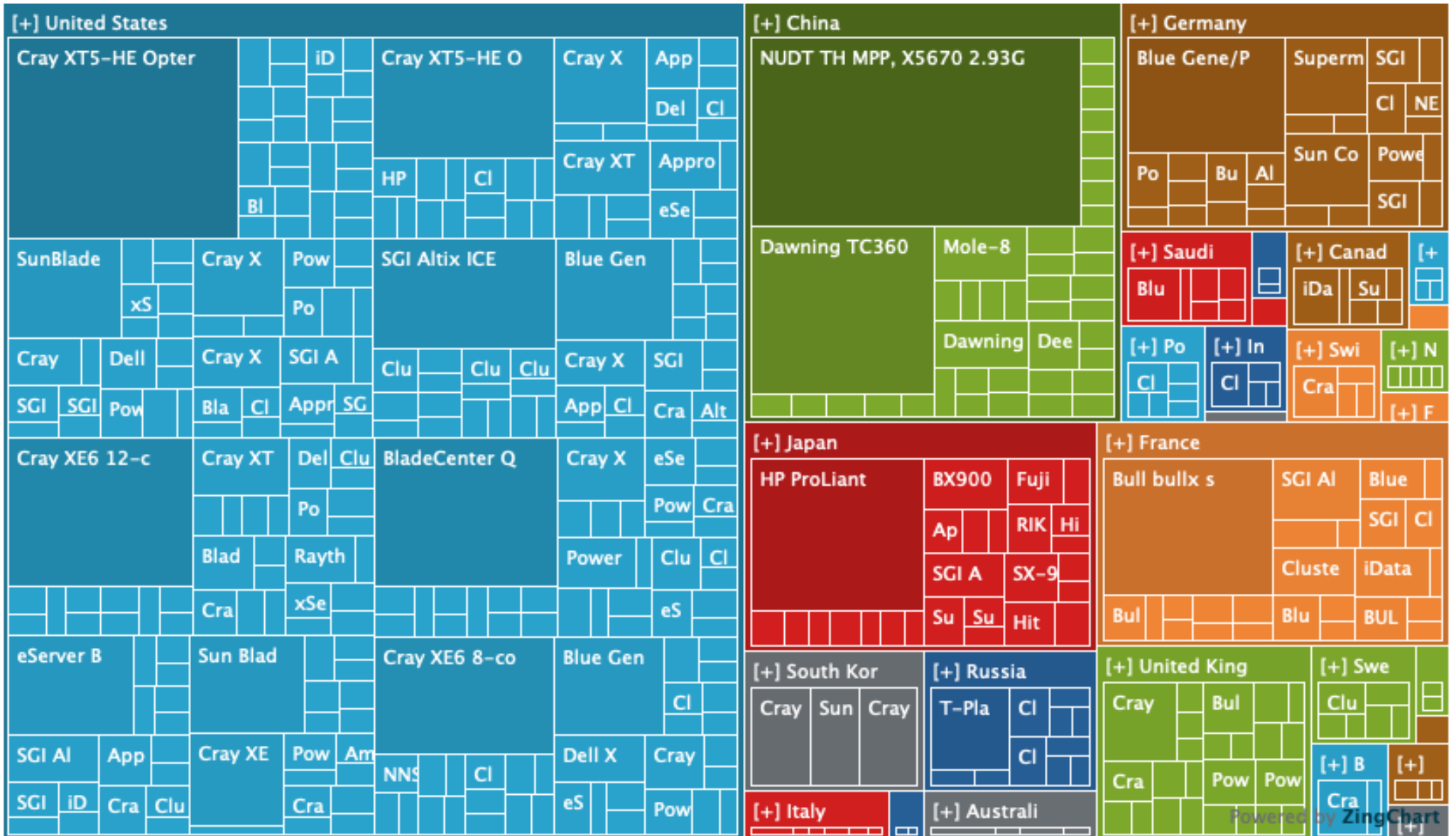
TOP 500



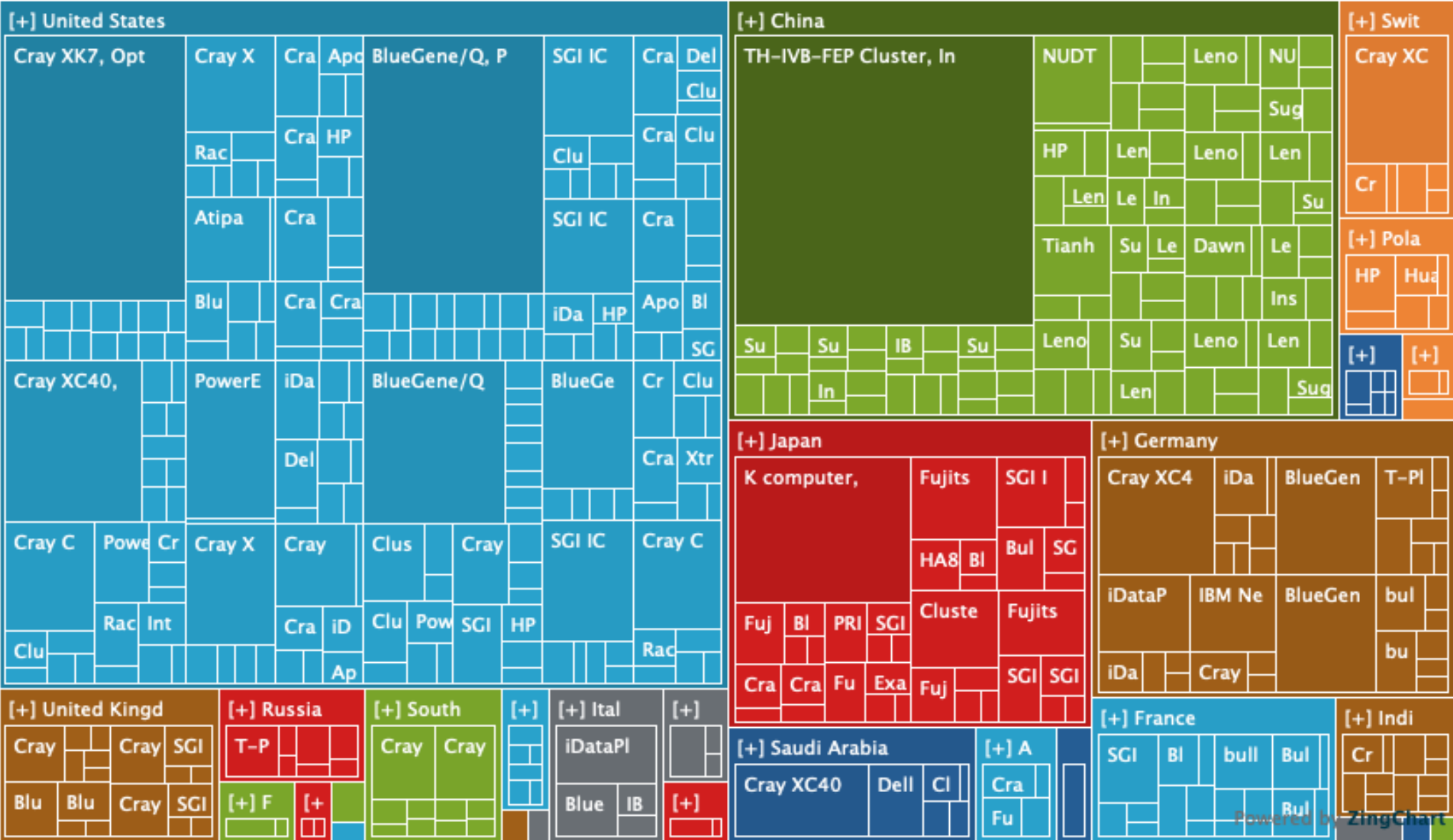
TOP 500 - 2005



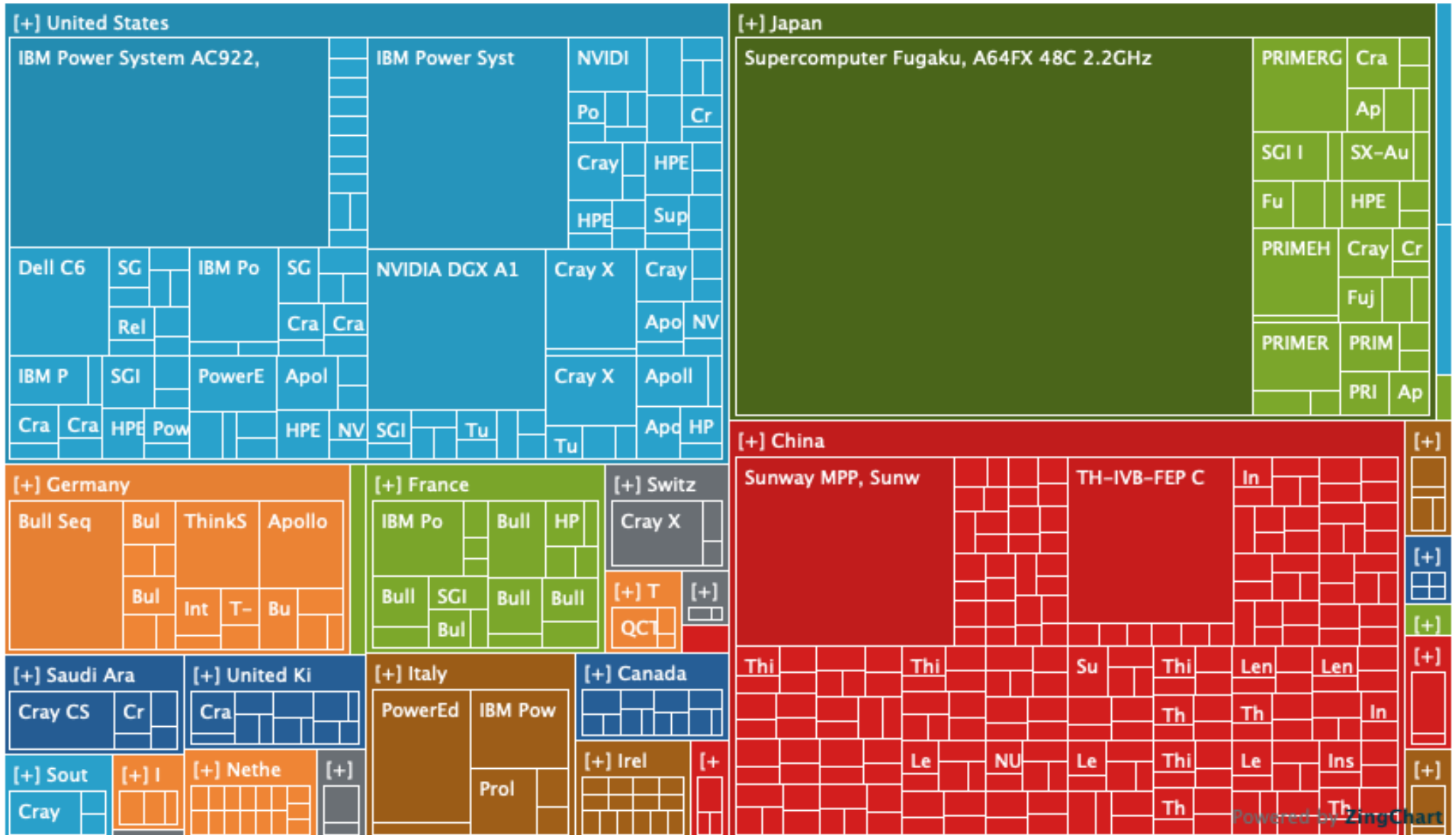
TOP 500 -2010



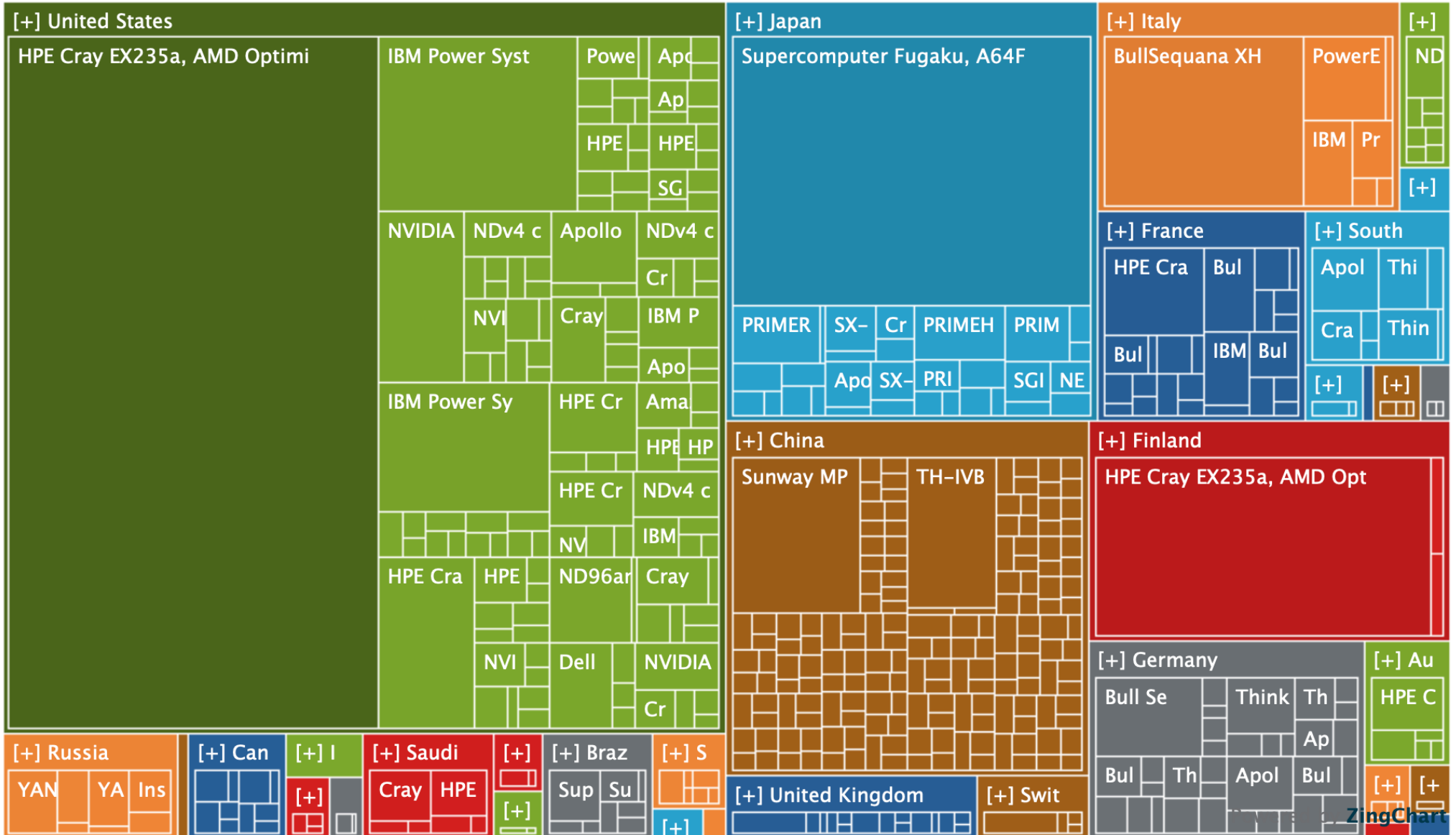
TOP 500 - 2015



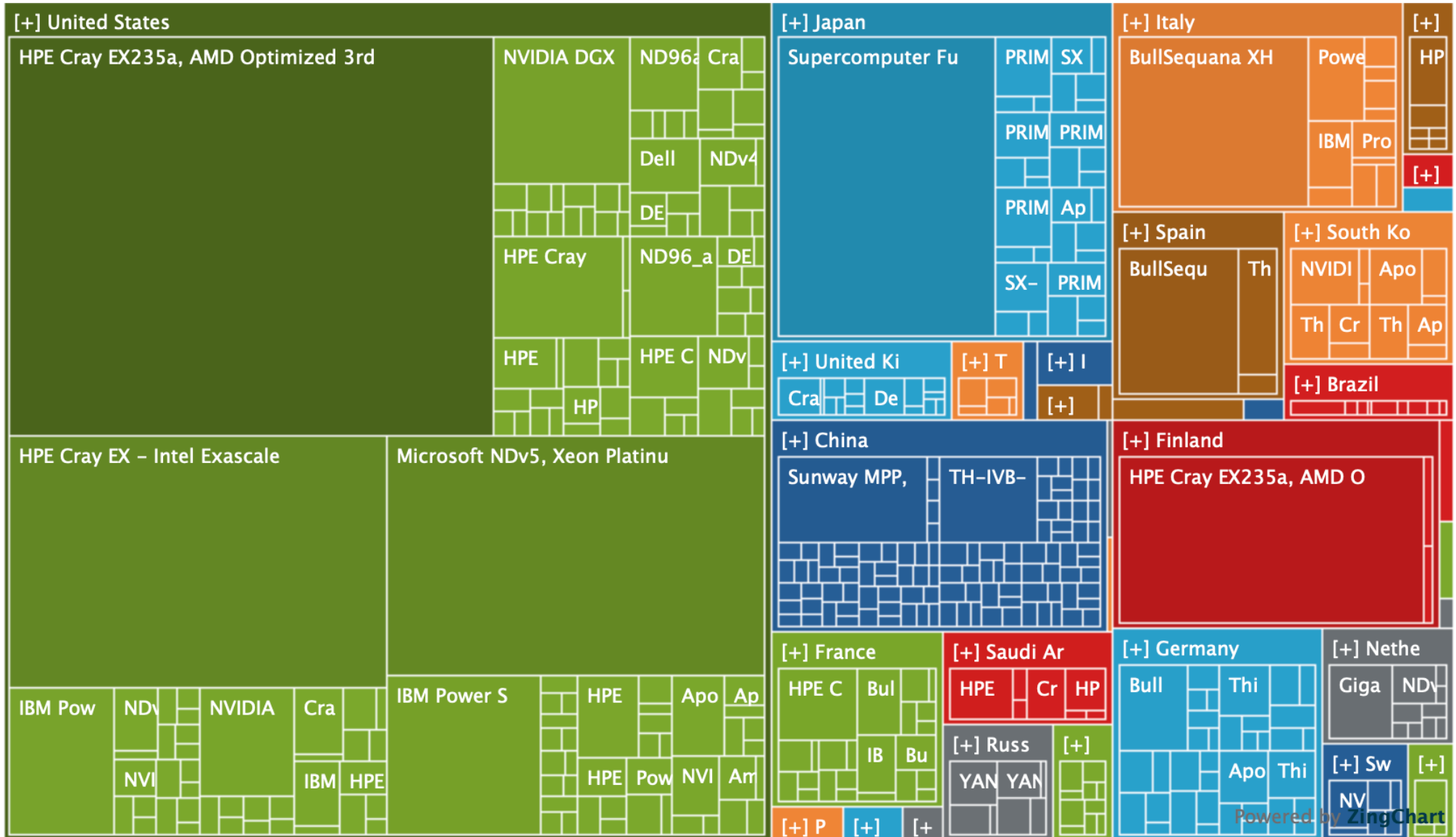
TOP 500 – 2020



TOP 500 - 2022



TOP 500 – 2023



Frontier (Oak Ridge, USA) #1



~40 000 GPU + 600 000= 8 730 112 cœurs AMD → 1,68 exaflops (1,68 10^{18} flops)
22,7 MW (= 2 TGV Duplex)

<https://www.olcf.ornl.gov/frontier/>

Aurora (Argonne DOE) #2



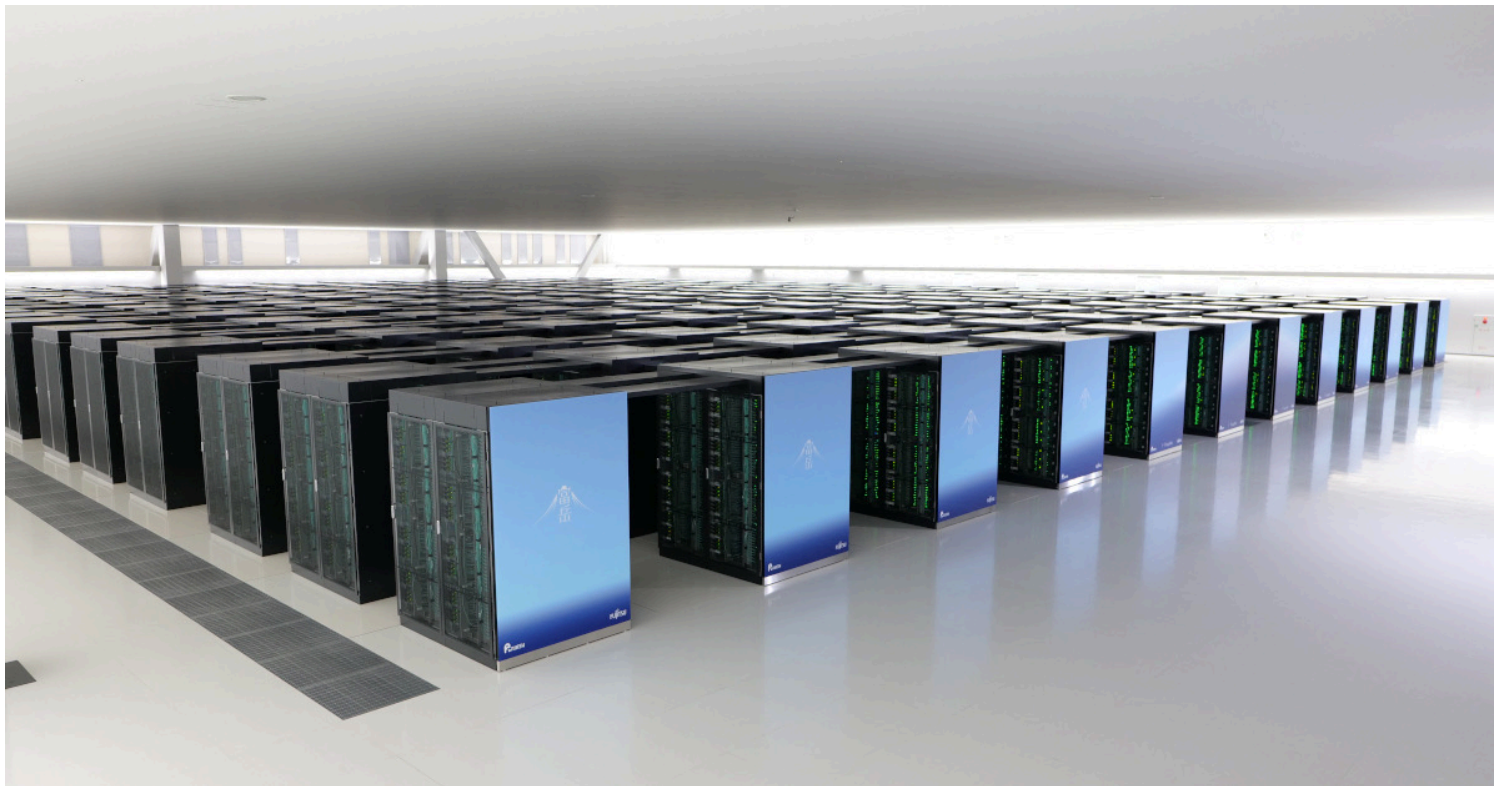
4,742,808 CPU → 1,06 exaflops - 25 MW (> 2 TGV Duplex)

EAGLE – MICROSOFT (#3)



1,123,200 coeurs → 850 petaflops

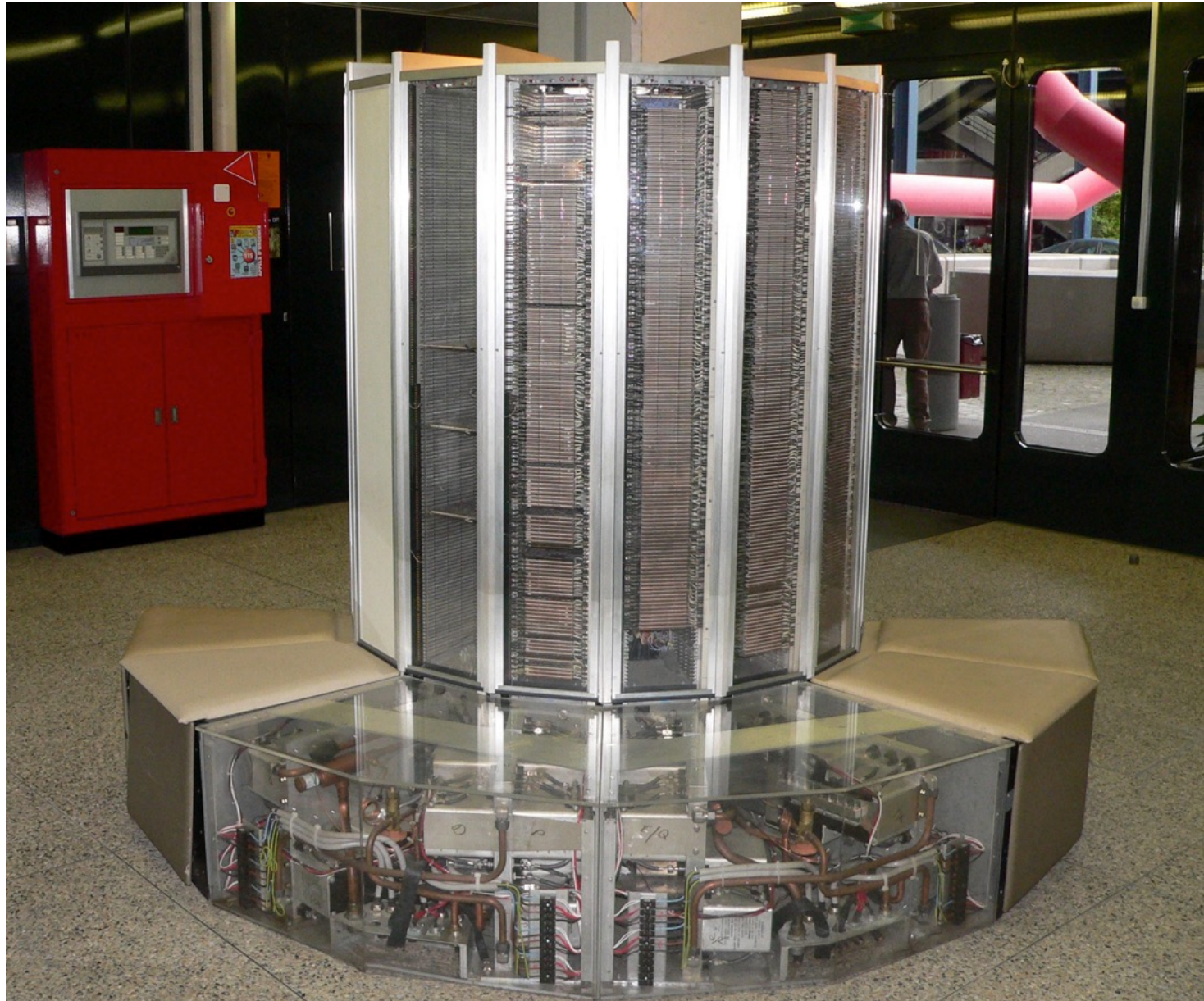
Fugaku (Riken, Japon) #4



7,630,848 CPU Fujitsu → 537 petaflops ($537 \cdot 10^{15}$ flops) - 30 MW (= 3 TGV Duplex)

<https://www.r-ccs.riken.jp/en/fugaku/covid-19/msg-en.html>

Cray one (136 MFLOPS, 1976)



Modélisation météorologique

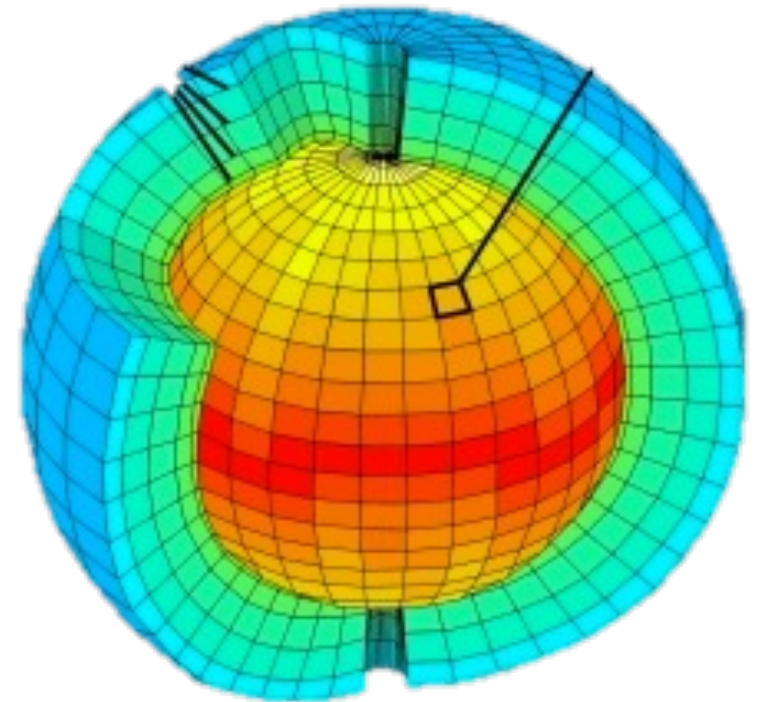
Un exemple de simulation numérique

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*

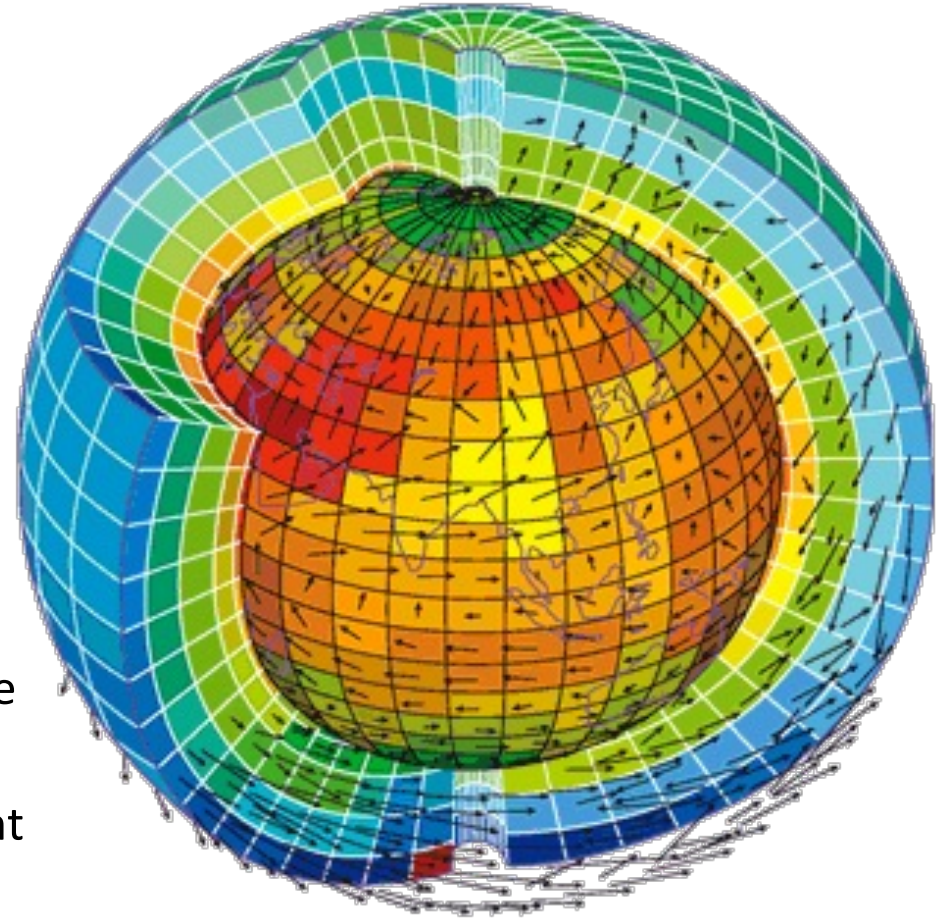
Simulation numérique déterministe :

- Discrétiser l'espace
 - Illustration : 1 point pour 2km^3 sur 20 km d'atmosphère $\Rightarrow 5 \cdot 10^9$ points
- Initialiser le modèle
 - Données satellites et autres capteurs
 - Nécessité d'interpoler les valeurs manquantes



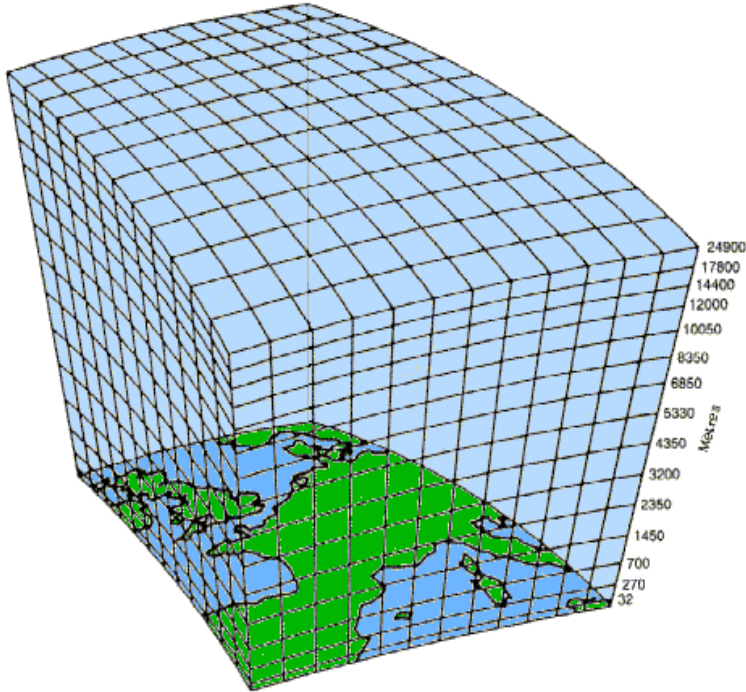
Modélisation météorologique

- 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



Discretisation du temps et de l'espace :

- pas de 60 secondes
- 1 point pour 2km^3 sur 20 km d'atmosphère => $5 \cdot 10^9$ points

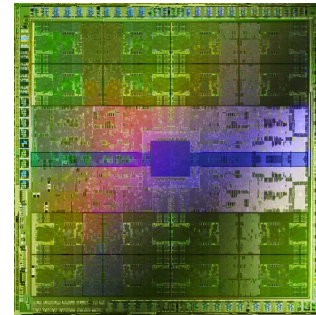
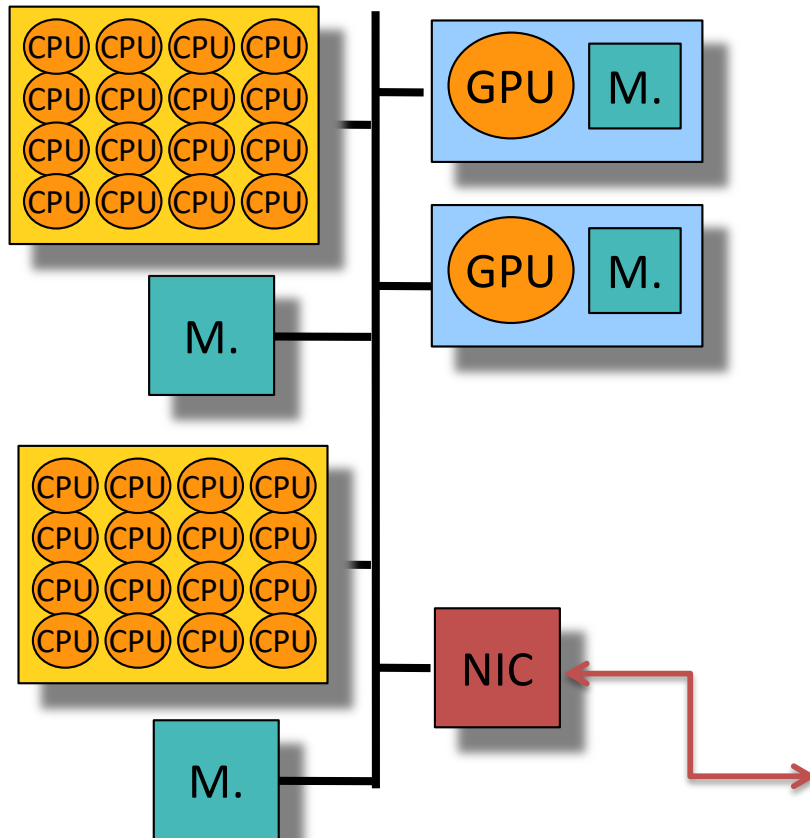
Nombre de calcul par point : 100 opérations (flop)

➔ **Simuler 1 minute (un pas) coûte $5 \cdot 10^{11}$ flop**

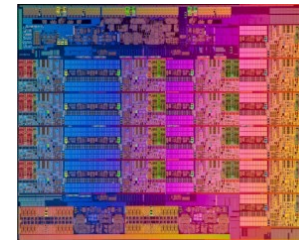
Puissance de calcul nécessaire à une simulation:

- Temps réel - **calculer 1 pas en 60s** : $5 \cdot 10^{11} / 60 = 8,33 \cdot 10^9 = 8,33 \text{ Giga flop/s}$
- Prévision - **calculer 7 jours en 1h** : $5 \cdot 10^{11} * 7\text{j} * 24\text{h} * 60\text{m} / 3600 = 1\,400 \text{ Gflop/s}$
- Climatologie - **50 ans en 1 jour** : $1\,060\,000 \text{ Gflop/s} = 1,06 \text{ péta flop / s}$

Composants d'un PC sur-vitaminé



Nvidia A100
128 SM
9,2 Tflops
(10k€)



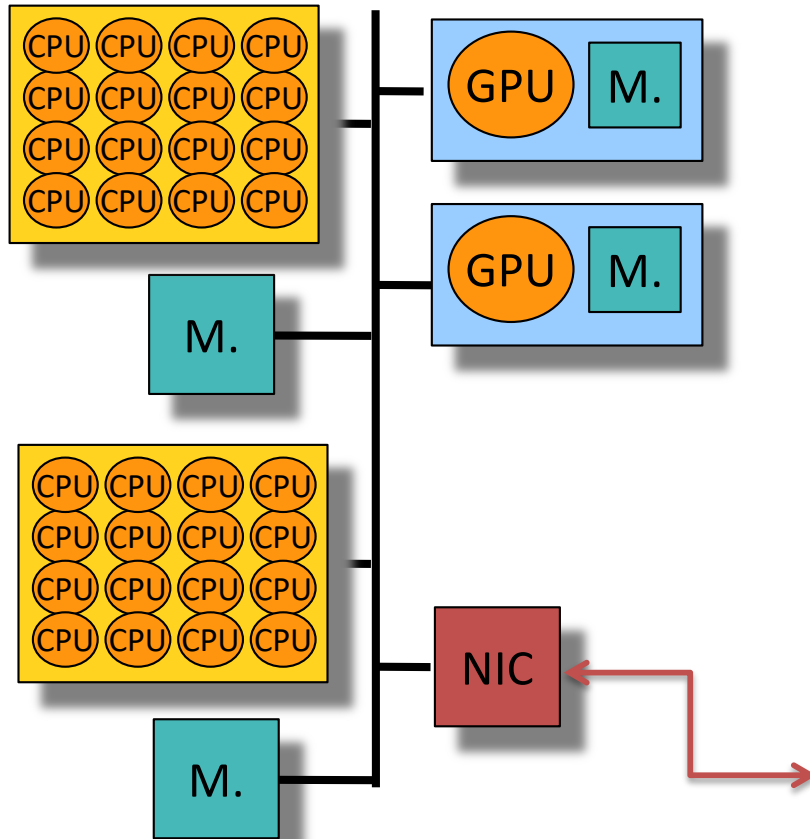
INTEL® XEON®
PLATINUM 9282
56 coeurs
2,4 TFlops



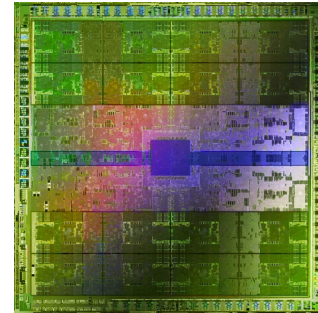
Carte réseau
Rapide 200 Gb/s

Les ordinateurs deviennent massivement parallèles.

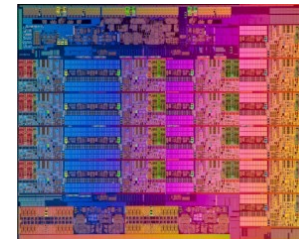
Performances d'un PC sur-vitaminé



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



Nvidia A100
128 SM
9,2 TFlops

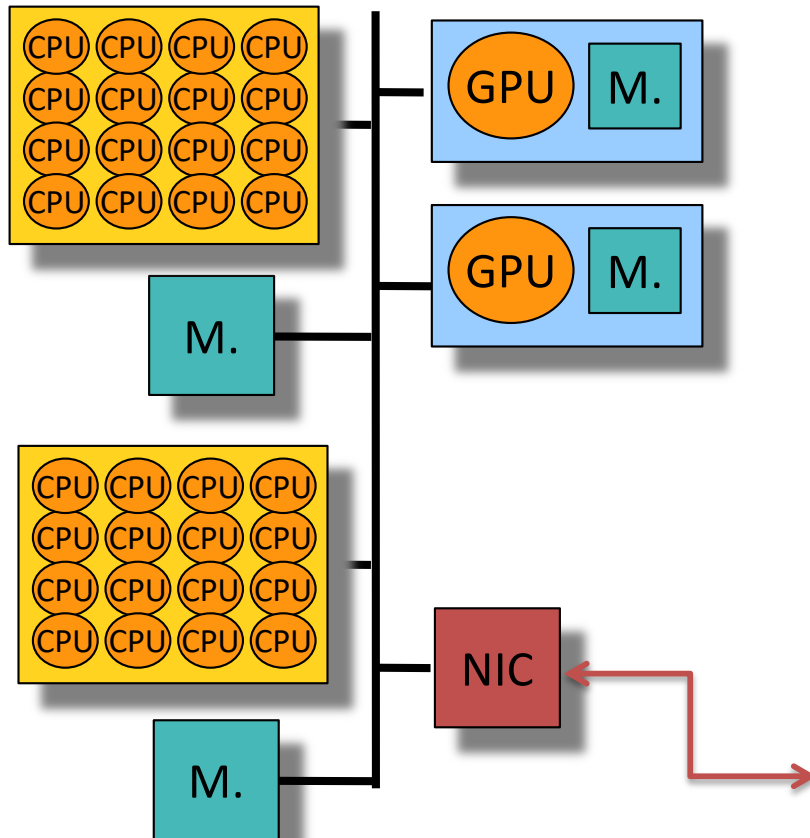


INTEL® XEON®
PLATINUM 9282
56 coeurs
2,4 TFlops

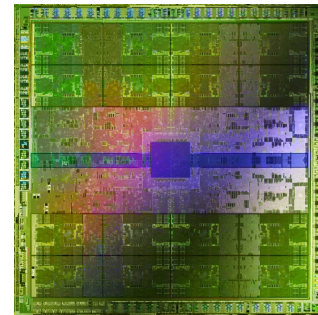


Carte réseau
Rapide 200 Gb/s

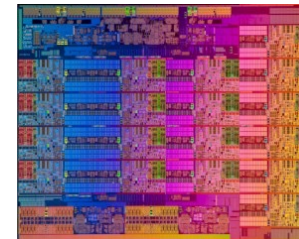
Temps pour calculer une prévision



1 PC sur-vitaminé = 2 GPU + 2 processeurs
= 1 prévision météo à 7 jours en 3 minutes 37



Nvidia A100
128 SM
9 minutes 07

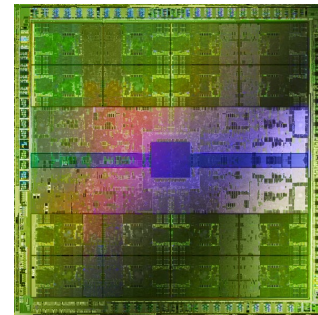
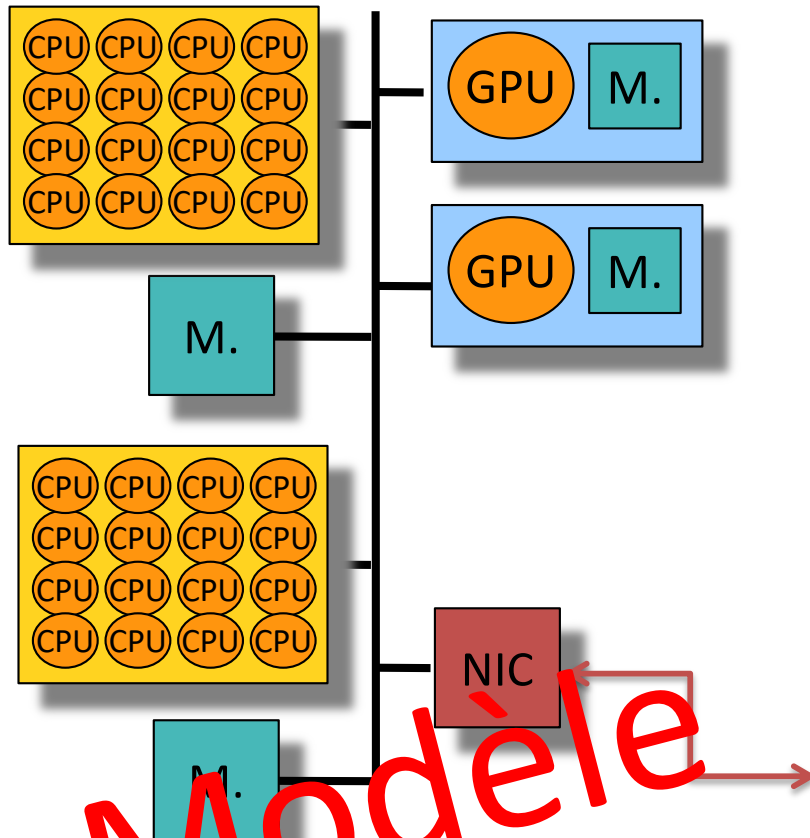


INTEL® XEON®
PLATINUM 9282
56 cœurs
34 minutes 40

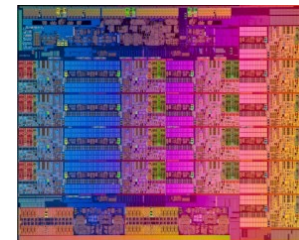


Carte réseau
Rapide 200 Gb/s

Temps pour calculer une prévision



Nvidia A100
128 SM
9 minutes 07



INTEL® XEON®
PLATINUM 9282
56 cœurs
34 minutes 40

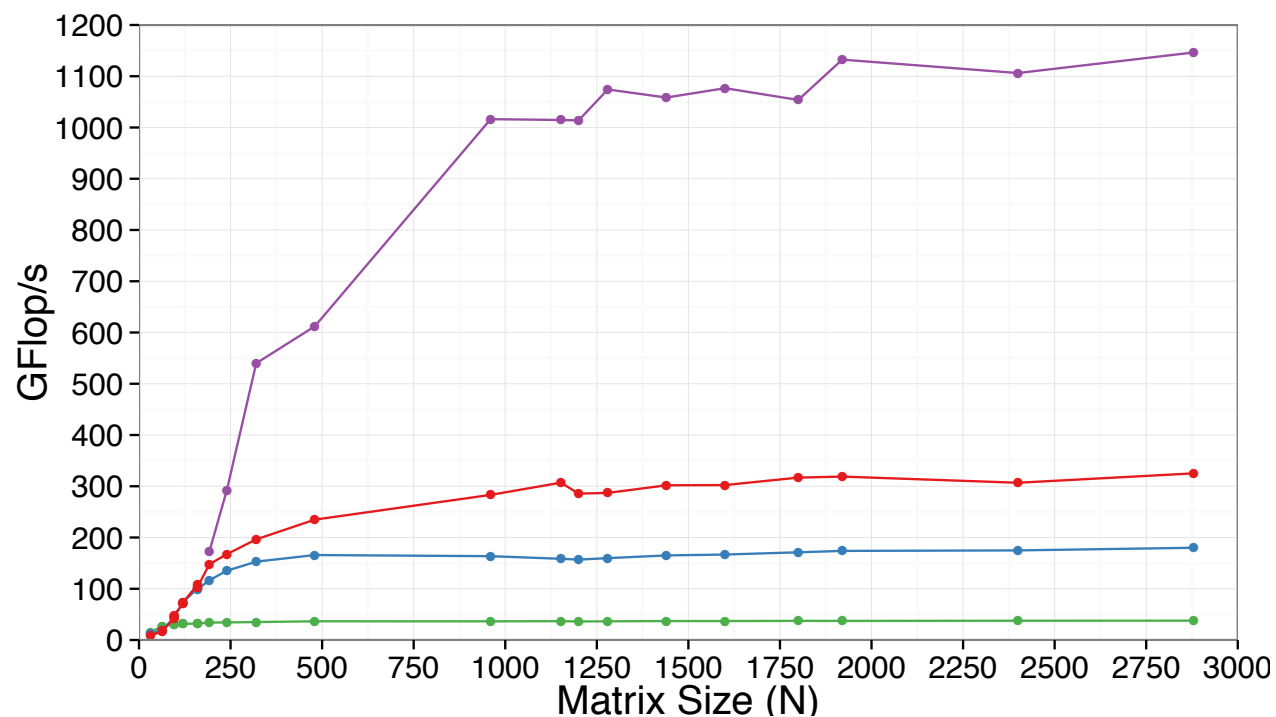
1 PC sur-vitaminé = 2 GPU + 2 processeurs
= 1 prévision météo à 7 jours en 3 minutes 37

**Modèle
simpliste**

- 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 -> 2013)



Efficacité

95% 1 cœur

93% 5 cœurs

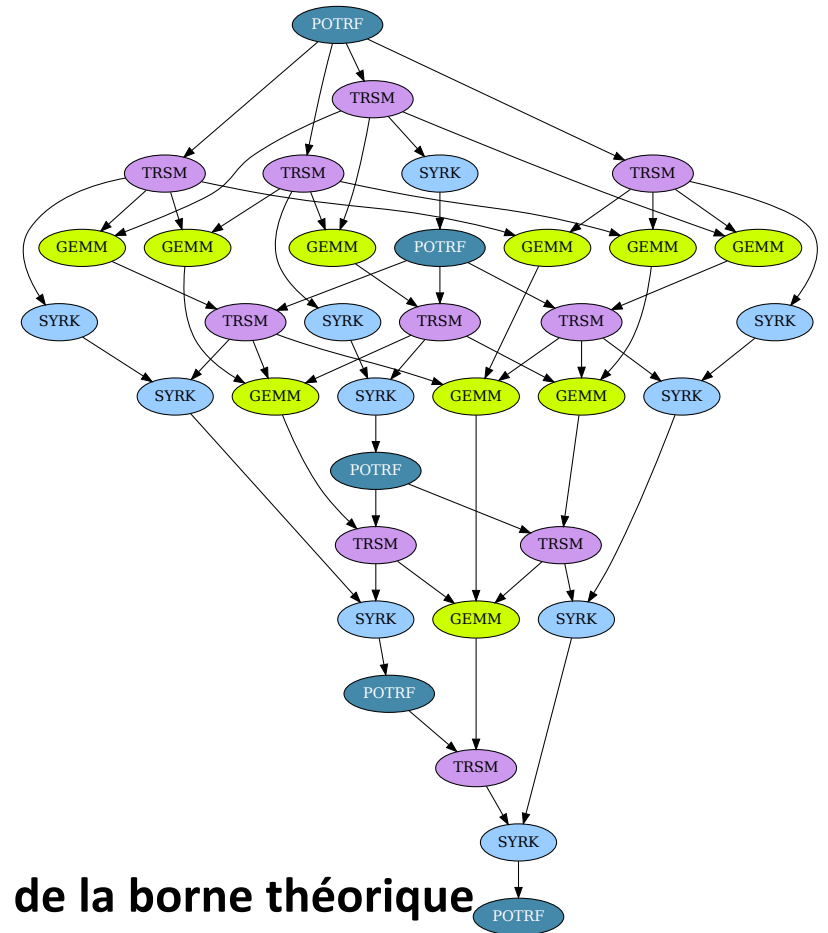
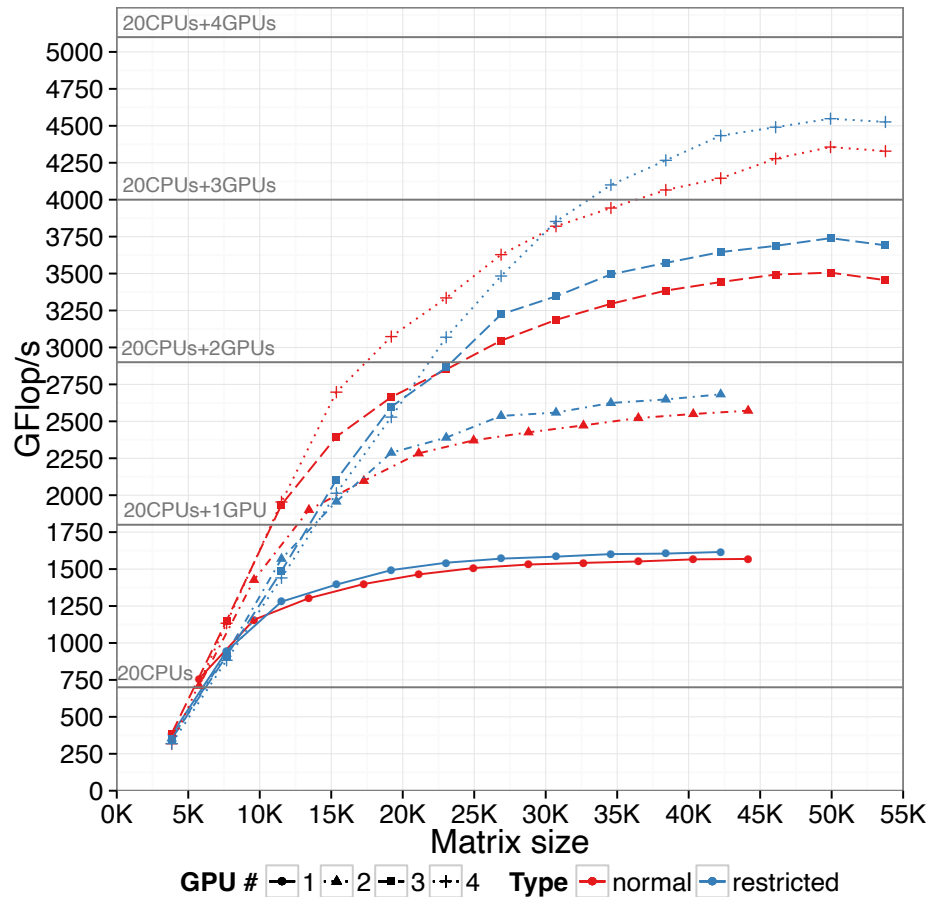
87% 10 cœurs

81% 1 GPU 2013

[9? % GPU 2020]

Est-il facile d'obtenir des performances ?

- Calcul matriciel plus complexe
(Factorisation de Cholesky)



~90% de la borne théorique
80% 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

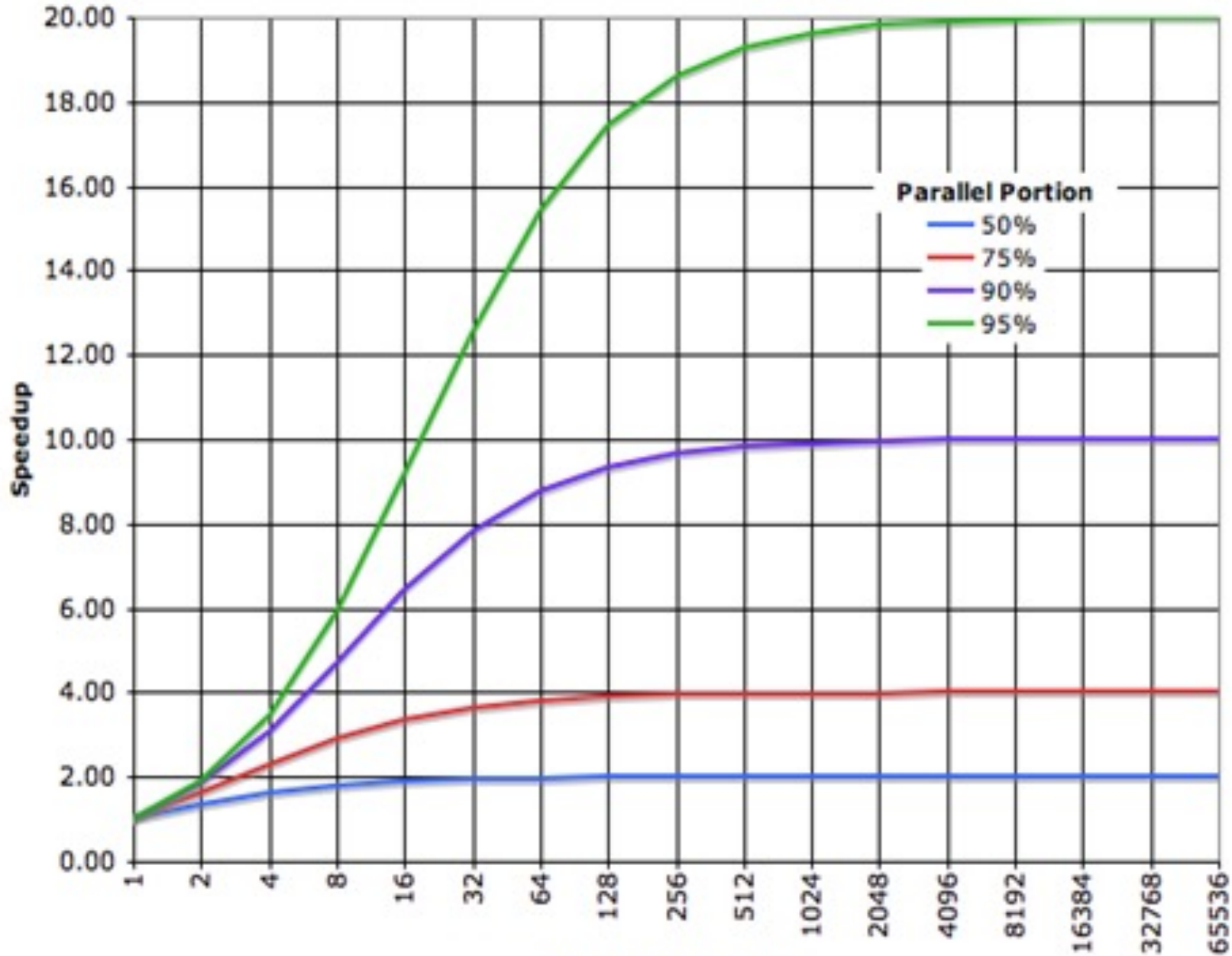
$$\text{durée parallèle} \geq \left(s + \frac{1-s}{p} \right) \times \text{durée séquentielle}$$

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

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

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

Loi d'Amdahl



Amdahl et chemin critique : tri fusion

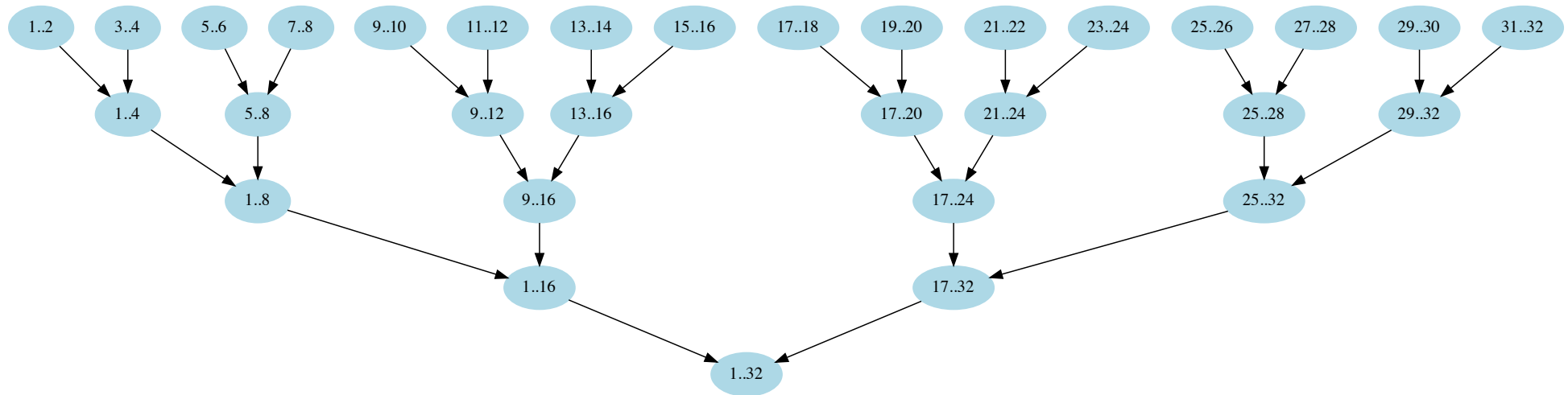


Tableau de 2^k éléments à trier à l'aide du tri fusion récursif simpliste :

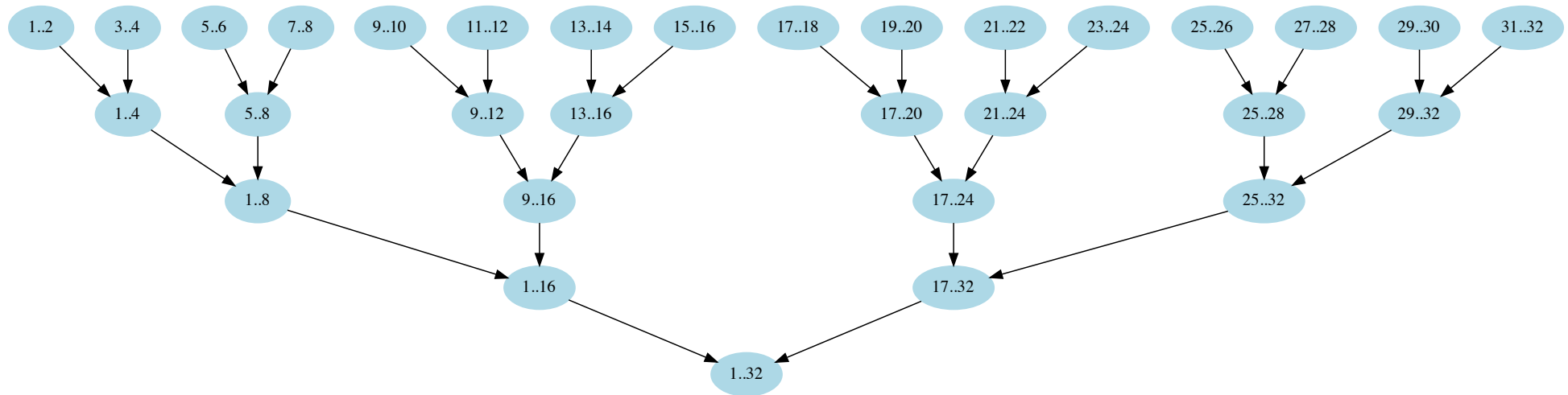
- on trie en parallèle chaque moitié du tableau
- on fusionne en séquentiel les moitiés

Part séquentielle = Chemin critique = toute branche d'une feuille à la racine

$$\text{durée parallèle} \geq \left(\text{branche} + \frac{\text{arbre} - \text{branche}}{p} \right)$$

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

Amdahl et chemin critique : 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

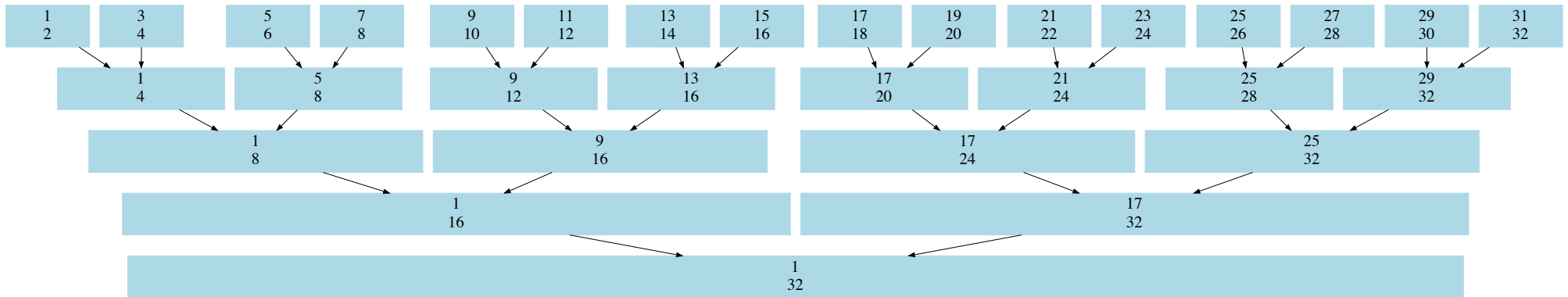
CoutPar(k) le coût d'une branche d'un tri de 2^k éléments

CoutSeq(k) = $k \cdot 2^k$ écritures

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

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

Amdahl et chemin critique : 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

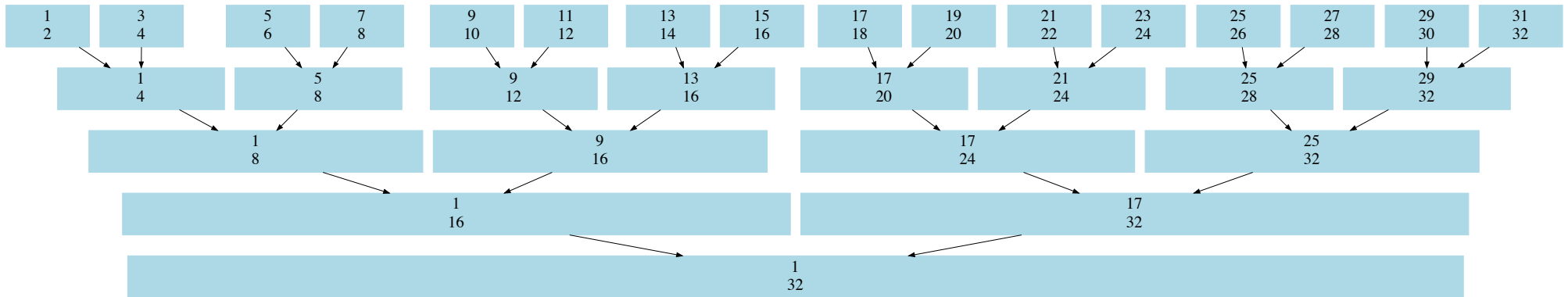
CoutPar(k) le coût d'une branche d'un tri de 2^k éléments

CoutSeq(k) = $k \cdot 2^k$ écritures

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

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

Exemple Amdahl : tri fusion



Tri fusion parallèle simpliste : on fusionne en séquentiel.

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

Nécessité de paralléliser la fusion pour améliorer l'accélération...

Nos objectifs

- Se plonger dans des environnements de programmation parallèle
 - Maitriser OpenMP
 - S'initier à OpenCL
- Adapter des algorithmes à une machine cible
 - Améliorer la localité mémoire
 - Paralléliser l'exécution
- Initier une démarche scientifique pour optimiser un programme
 - Observer, analyser, comprendre l'exécution d'un programme
 - En faisant varier des paramètres
 - Expliciter la démarche utilisée dans un rapport

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 *inutilement* limité par la bande passante de la mémoire

