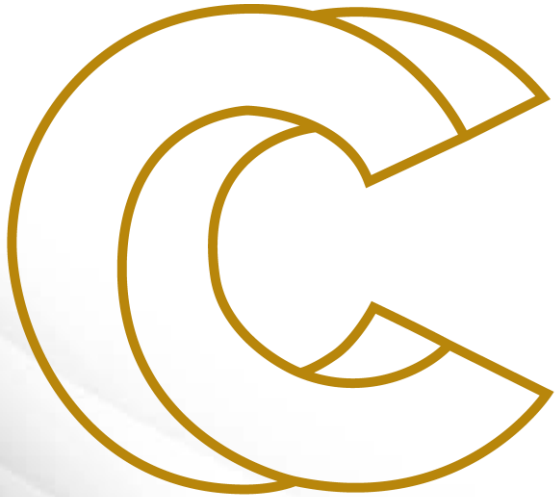


EURO^2

Introduction au calcul intensif

Par EuroCC Belgium

Plan

PARTIE 1

- Introduction
  - Exemple d'applications du calcul intensif
  - EuroHPC JU et le projet EuroCC
- État actuel des infrastructures de calcul intensif
  - Performances et la liste TOP500
  - Supercalculateurs en Europe et en Belgique

PARTIE 2

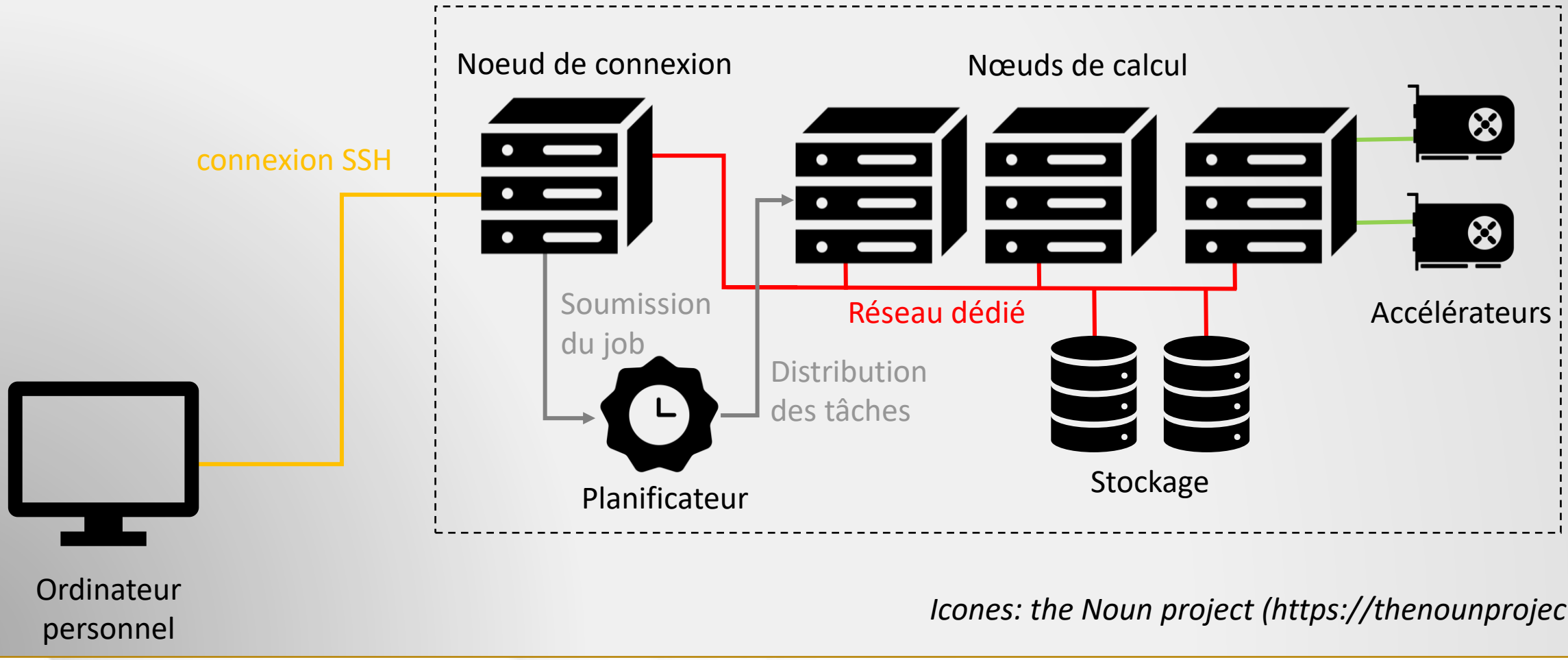
- Comment fonctionne un supercalculateur
  - Architecture & composants
  - Interagir avec un supercalculateur
- Comment utiliser efficacement les ressources
  - Parallélisation
  - Problèmes liés à la parallélisation
  - Quels outils utiliser?

PARTIE 2

Qu’est-ce qu’un superordinateur

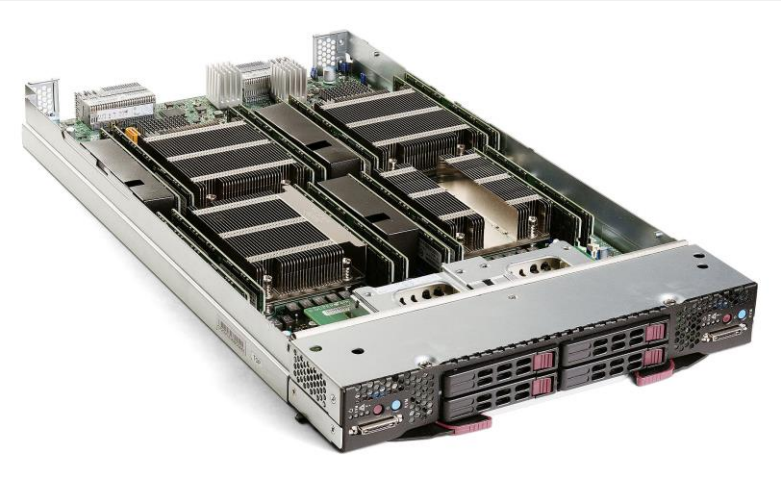
Comment ils fonctionnent

Anatomie d'un cluster



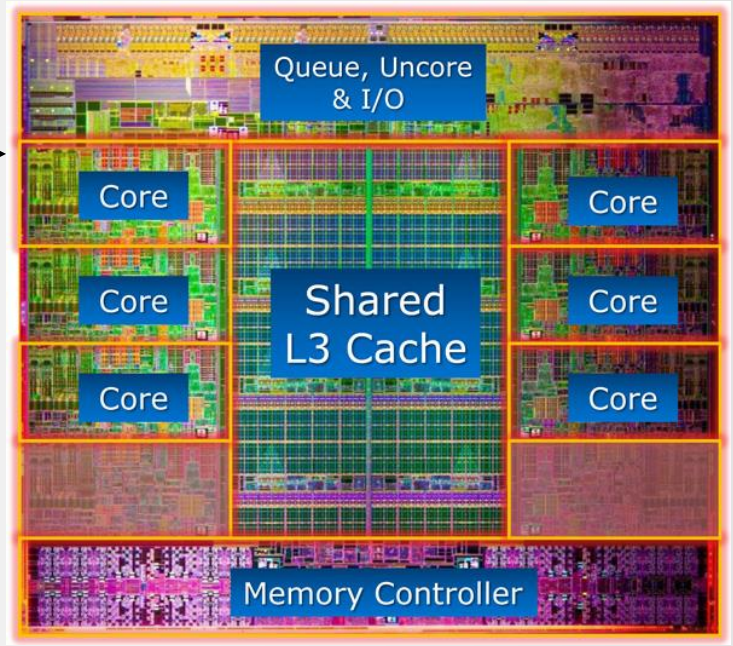
Icones: the Noun project (https://thenounproject.com/)

Composants



Noeud de calcul (Wikipedia)
Se comporte comme un ordinateur

- Socket (CPU)
- RAM
- Réseau
- Refroidissement
- Stockage local
- …

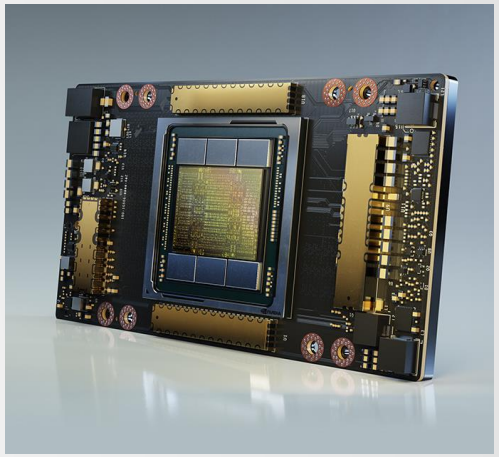


Intel sandybridge (https://www.anandtech.com/)

Composants: GPU / Accélérateurs

- GPGPU: general-purpose GPU
- À la place de quelques cœurs très performants (CPUs), de nombreux cœurs moins performants
- GPU commerciaux: bonnes performances en FLOPS pour la simple précision, pas pour la double précision
- GPU dédié pour le calcul intensif (e.g., NVIDIA Ampere or AMD Instinct)
- Futur du calcul intensif?

AMD Instinct (amd.com)

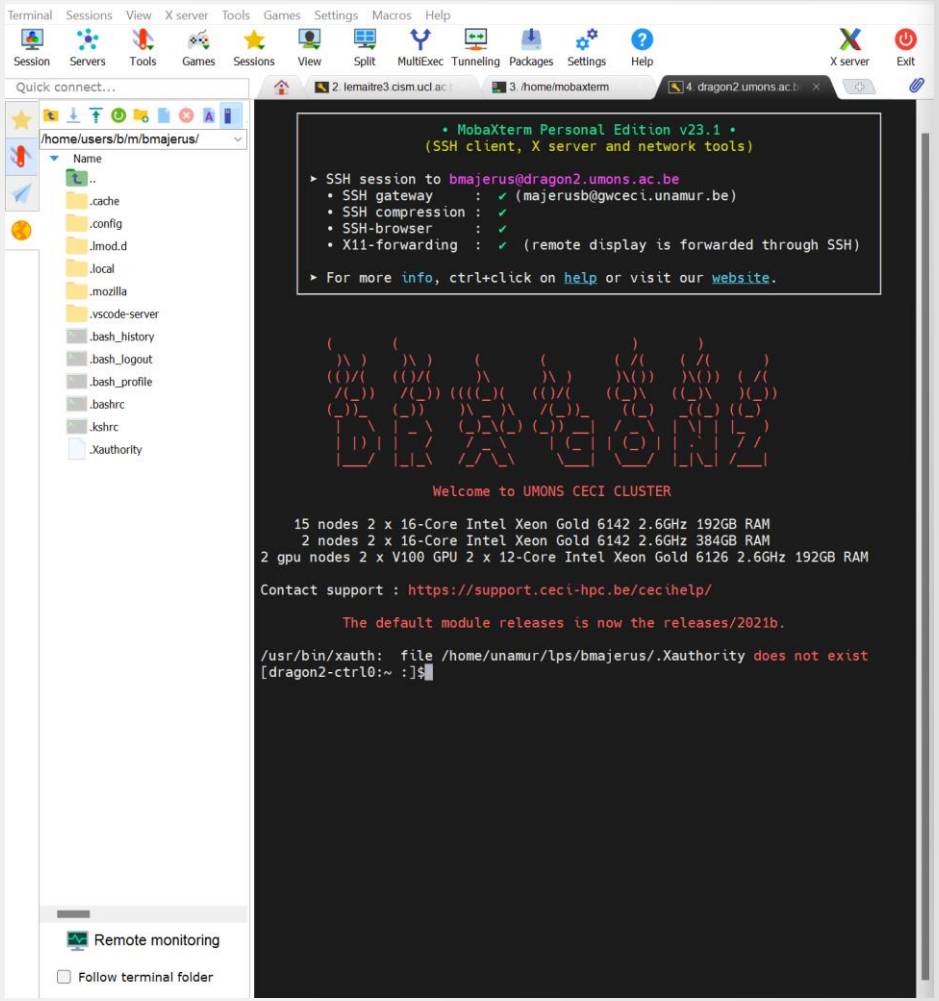


Ampere (nvidia.com)

Interagir avec le supercalculateur

Vous interagissez avec la ligne de commande (Linux, Mac) ou avec des logiciels spécifiques (Windows, Mac).

- Connexion avec un client SSH
- Un terminal pour envoyer ou recevoir du texte
- Une interface graphique est possible (mais pas encore disponible partout)



Le planificateur

- Pour chaque “job”, le planificateur (par exemple SLURM) doit connaitre le temps de calcul prévu, la mémoire RAM, le nombre de cœurs/nœuds demandés, et va distribuer les calculs sur les nœuds quand la place est disponible.
- Différents clusters ont différents buts, par exemple:
  - Grande quantité de mémoire RAM,
  - Petits jobs rapides,
  - Nœuds avec accélérateurs, ...

```
#!/bin/bash
#submission script for Lemaitre3
#SBATCH --job-name=CNT
#SBATCH --time=24:00:00 # hh:mm:ss
#SBATCH --ntasks=128
#SBATCH --mem-per-cpu=2000 # megabytes
#SBATCH --partition=batch


module load Python/3.6.3-foss-2017b
module load libxc

srun gpaw-python cnt_ph.py >log_ph
~
~
~
~
~
~
~
~
~
```

Comment utiliser ces ressources efficacement

Solutions... et difficultés

Particularités du calcul intensif

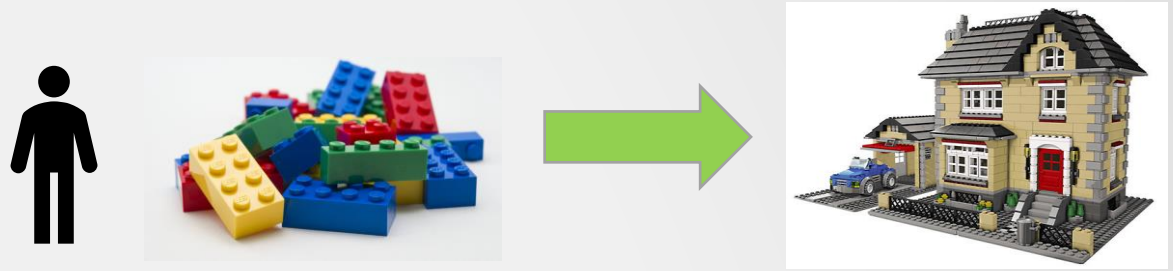
EURO²

- Avantages du calcul intensif
  - Plus grande taille: certains problèmes nécessitent de grande quantité de RAM
  - Plus rapide: certains problèmes ont des temps de résolution très long
- Solutions … et problèmes:
  - Plus de mémoire (mais difficulté avec la hiérarchie du stockage)
  - Parallélisme (avec ses difficultés intrinsèques)

Augmenter la vitesse: le calcul parallèle

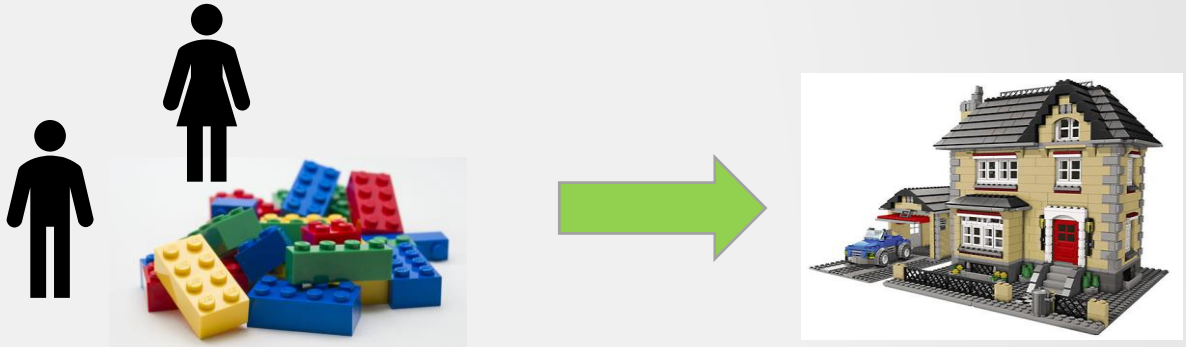
- Pour obtenir de telles performances, l'idée est d'utiliser **le calcul parallèle**: exécuter de nombreuses opérations en même temps.
- Mais le programme doit être adapté pour cela

Série: 1 travailleur (1 personne) construit



Images de LEGO (lego.com)

Parallèle : 2 travailleurs → environ 2 fois plus rapide



L'accélération est toujours décevante

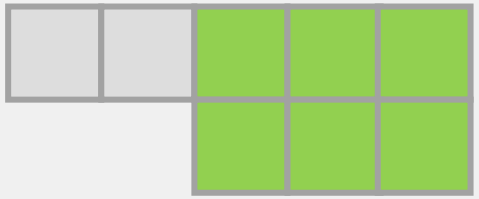
Dans la plupart des cas, un programme n'est jamais entièrement parallélisable (un tel cas est appelé « embarrassingly parallel »)

Non parallélisable (25%)

1 travailleur:

$T_{tot,1} = 8$

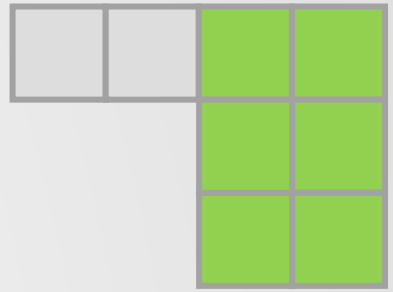
2 travailleurs:



$T_{tot,2} = 5$

Accél. = $T_{tot,1} / T_{tot,2} = 1.6$

3 travailleurs:



$T_{tot,3} = 4$

Accél. = $T_{tot,1} / T_{tot,3} = 2$

→ Peu importe la rapidité de la partie parallèle, c'est la partie en série qui limite la performance

L'acceleration est toujours limitée

Loi d'Amdahl:

$$S = \frac{1}{s + \frac{P}{N}}$$

S: accélération effective

s: proportion du code en série (%)

P: proportion du code en parallèle (%)

N: nombre de processeurs

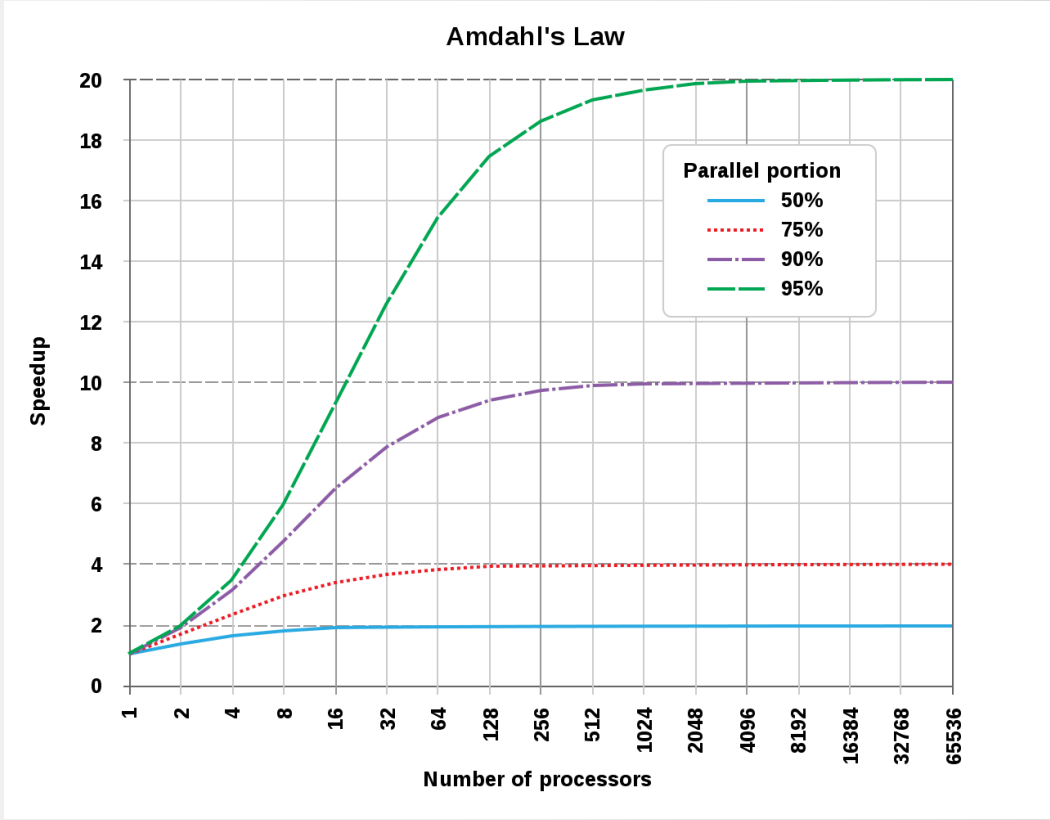
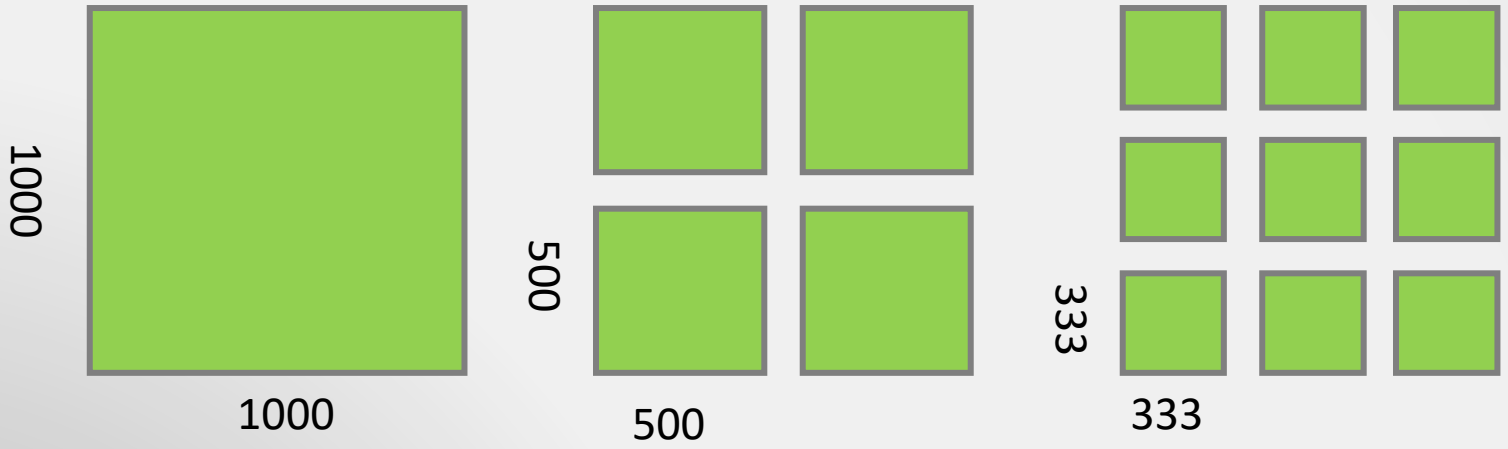


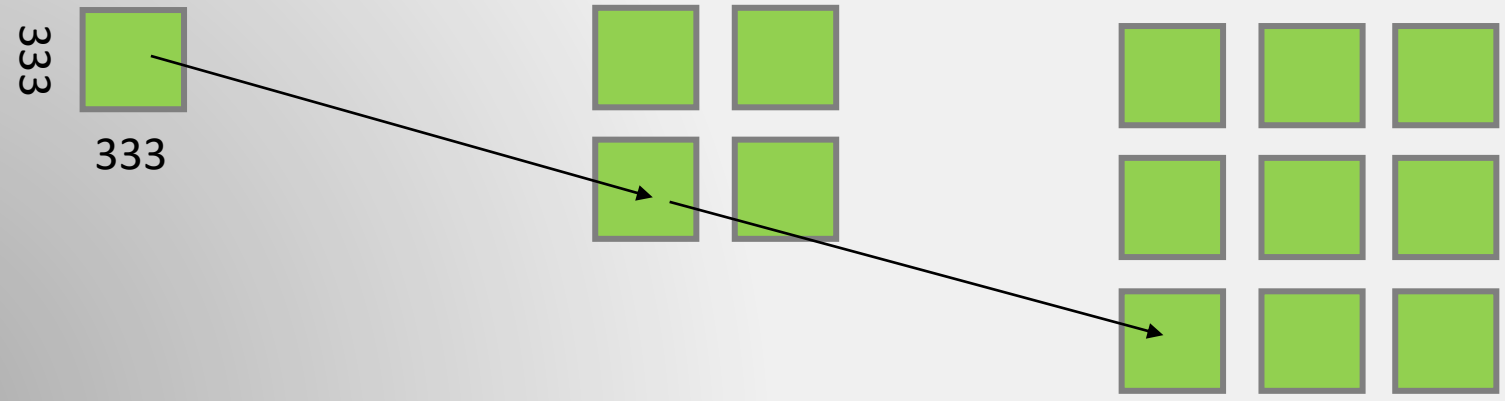
Image de Wikipédia (wikipedia.org)

Ca répond à la question : "Combien de processeurs je dois utiliser pour un problème donner?"

Strong vs weak scaling



Strong scaling: la taille du problème est constante et le problème est divisé sur plusieurs processeurs



Weak scaling: la taille du problème est le même sur chaque processeur

Le weak scaling c'est bien!

Si on augmente la taille du problème quand des processeurs sont ajoutés:

$$S = N - s\,(N - 1)$$

S: accélération effective

N: nombre de processeurs

s: proportion en série (%)

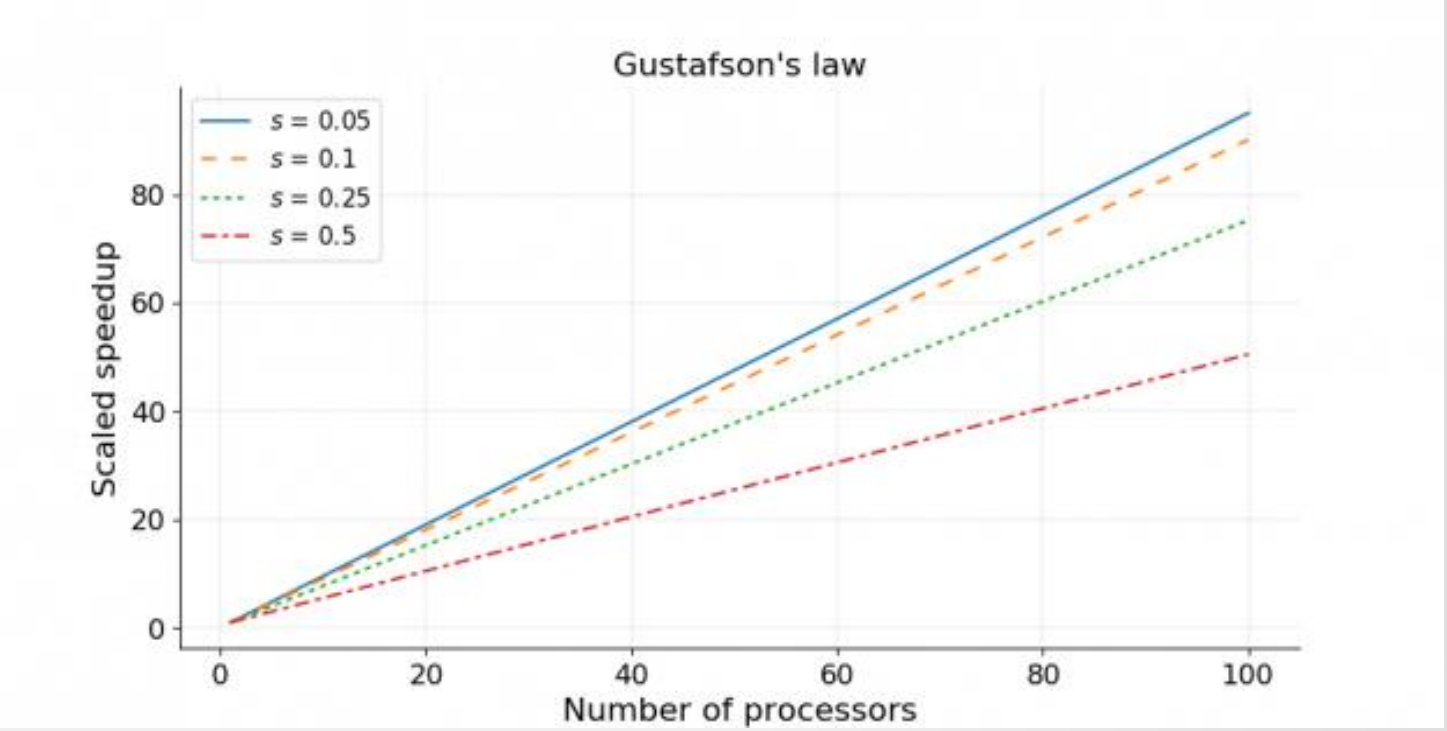
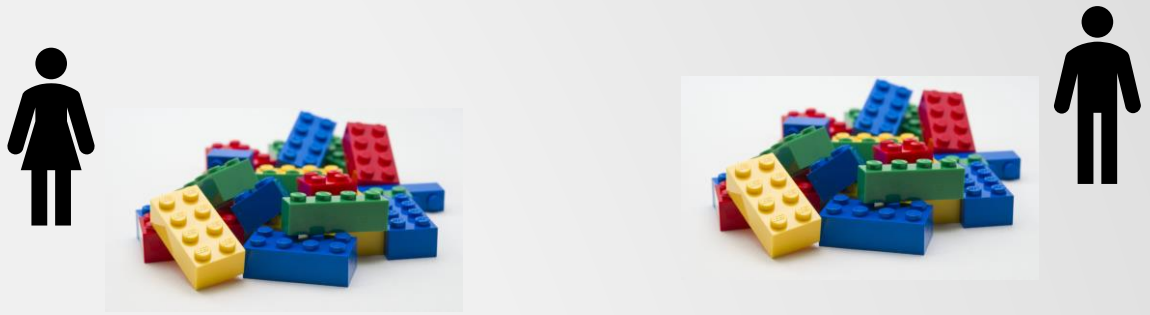


Image de Wikipédia (wikipedia.org)

Ca répond à la question: "De combien puis-je augmenter la taille de mon problème tel que le temps d'exécution est le même que si j'avais utilisé un seul processeur."

Mais il y a toujours des surcoûts en temps

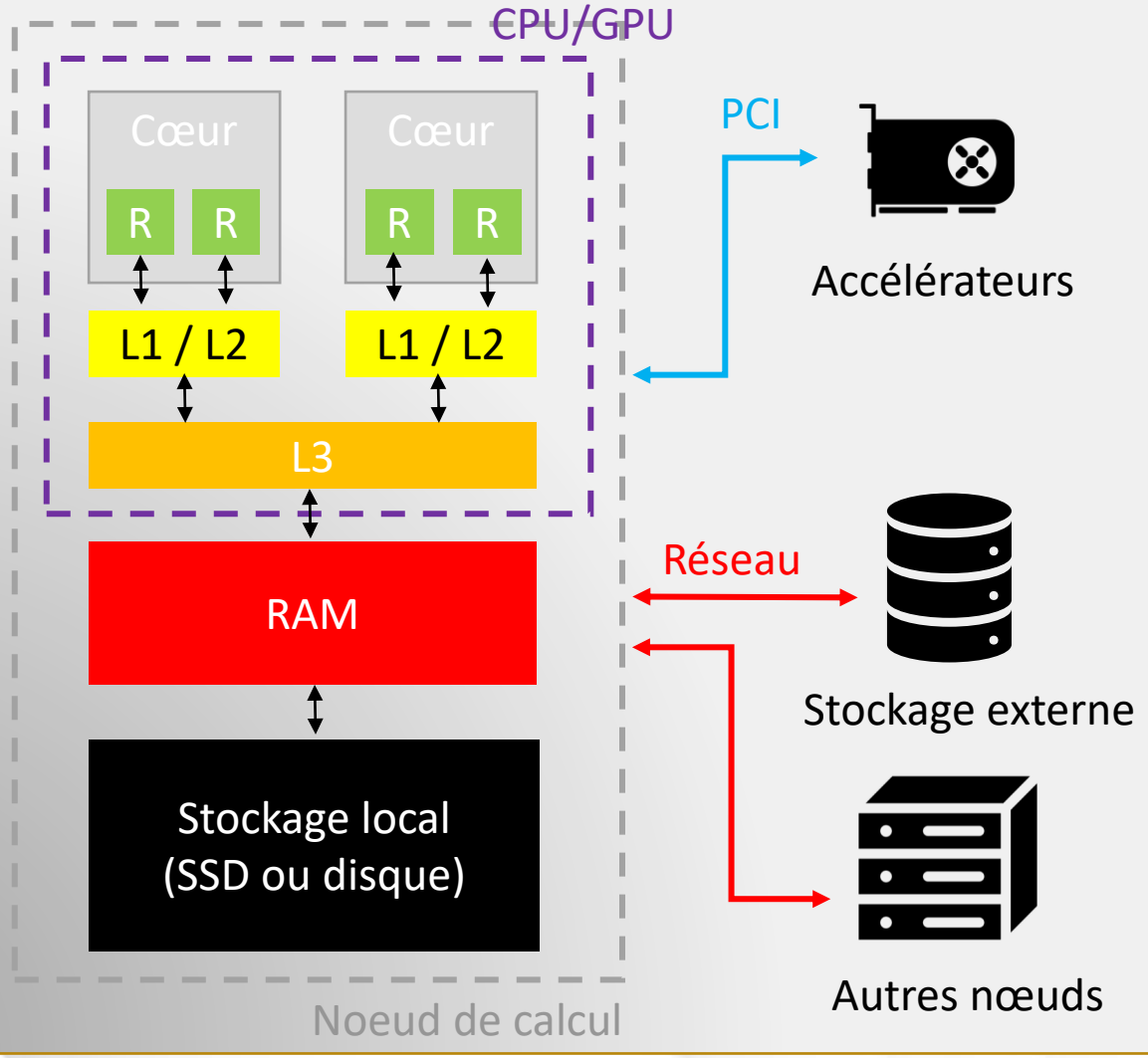
Modèle de mémoire partagée: tous les travailleurs (ici, les personnes) travaillent sur le même ensemble de données (ici, les lego). Faible surcoût dû à la synchronisation (toutes les personnes ne peuvent pas travailler sur la même partie en même temps).



Modèle de mémoire distribuée: chaque travailleur (ici, personne) travaille sur son propre ensemble de données (ici, pièces de lego). Généralement plus efficace (pas de collaboration pendant le travail), mais surcoûts dus à la communication (ici, distribution des pièces au début et assemblage du résultat à la fin).

En général, les deux sont mélangés. Cela nécessite également une bonne répartition de la charge (c'est-à-dire que chaque personne a la même quantité de travail à effectuer, ce qui n'est pas si facile à réaliser.

La hiérarchie de la mémoire



- Plus on s'éloigne du cœur, plus on est lent (mais, en général, plus la capacité est grande).
- I/O n'est pas nécessairement parallélisé
- Les communications sont limitantes lors de l'utilisation de plusieurs nœuds
- Mouvement des données vers et depuis un accélérateur

Icone: the Noun project (https://thenounproject.com/)

Quels outils utiliser?

EURO²

Programmation parallèle:

- Vectorisation (au niveau d'un cœur)
- Threading / OpenMP (au niveau d'un nœud)
- CUDA / HIP / OpenCL / OpenACC / OpenMP (au niveau de l'accélérateur)
- Socket / MPI / PGAS (au niveau d'un cluster)

Librairies optimisées:

- BLAS / LAPACK / MKL (algèbre linéaire)
- FFTW (fast-Fourier transform)
- HDF5 / netCDF (parallèle I/O)

Bonne nouvelle pour les novices en informatique : de nombreux programmes scientifiques ont déjà été conçus en version parallèle et sont disponibles sur les supercalculateurs → Pas de compétence en programmation nécessaire!

Conclusions

Conclusions

- La parallélisation est au cœur de l'efficacité des superordinateurs
- Il existe plusieurs paradigmes de parallélisation avec leurs propres avantages et inconvénients
- Il n'est pas nécessaire d'avoir de grandes compétences en programmation pour faire du calcul intensif

→ Plus d'informations sur:

- https://www.enccb.be/
- https://www.ceci-hpc.be/
- https://www.vscentrum.be/

Merci!

EURO²

Funded by
the European Union

Cenaero

fwo

VLAAMS SUPERCOMPUTER CENTRUM

Vlaanderen
is supercomputing

EuroHPC
Joint Undertaking

CÉCI

Avec le soutien de la
Wallonie

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia under grant agreement No 101101903.