Algorithmique du Parallélisme, Ordonnancement de tâches

Guillaume Pallez
Inria

M2 CISD, Enseirb-Matmeca,
Automne 2020

Quelques mots sur l'organisation du cours

Trois intervenants

- Guillaume Pallez, Inria
- Laercio Pilla, CNRS

Plan (tentatif) → 1 créneau 1h20, 6 créneaux de 2h40
`http://people.bordeaux.inria.fr/gaupy/ressources/teachings/2019/algo_hpc/` (`https://huit.re/algo_hpc`)

- Introduction à l'algorithmique du parallélisme [G. Pallez]
- L'origine du graphe de tâches [L. Pilla].
- Introduction à l'ordonnancement [G. Pallez].
- Ordonnancement avancé [G. Pallez].
- Activité pratique [L. Pilla].
- Équilibrage de charge, localité des données [L. Pilla].
- Applications : où trouver un problème d'ordonnancement [G. Pallez].

Ce à quoi on s'attend

- Le contenu de ce cours est très dense. Nous souhaitons vous donner un aperçu large du domaine.
- Le cours/les slides suffisent, pas besoin de lire de livres en plus.
- Pour bien comprendre tout, il faut réussir à prendre du recul et regarder le tout dans un ensemble.
- C'est un cours compliqué, il demande du travail personnel.
  Une heure de cours = Au moins une heure de travail en plus pour relire/comprendre bien les slides.
- Demandez vous, plus que le contenu direct, quel est le message qu'on essaie de faire passer.
- À chaque début de cours, vos questions sur les cours d'avant sont les bienvenues ! Vraiment !

Évaluation : Lecture d'article

Modalités :

- Examen final : étude bibliographique d'un article de recherche (monômes ou binômes).
- Rapport (4 à 8 pages), du pour le XXXX
- Présentation : 14 décembre 2020, 15 min (25 pour binômes) + 10 min de questions (durée à confirmer une fois les articles choisis).

Liste d'articles disponible sur :
`https://huit.re/algo_hpc`

Communication

- Un site web avec tous les slides, ainsi que les articles (pour l'évaluation)
  `https://huit.re/algo_hpc`

- Si vous avez des questions :
  mailto: guillaume.pallez@inria.fr, francieli.zanon-boito@inria.fr, pilla@lri.fr

- Connaissances demandées :
  - Techniques algorithmiques classiques (programmation dynamique, algorithmes gloutons)
  - Base : réseau, Système d'exploitations, complexité (P v NP v NP-c), algorithme d'approximation

Propagande

Nous travaillons dans des problématiques liées à ce cours. Si ça vous intéresse de continuer dans ce domaine, venez nous parler :

- Il y a peu de bourses de thèses, mais il y en a. Par contre il faut bosser.
- Si vous êtes intéressées par un stage de recherche (voir une thèse), on peut vous proposer des choses, ou vous diriger vers des gens intéressants

Selon vous, qu'est-ce que ça veut dire?

Selon vous, qu'est-ce que ça veut dire?

Trois notions critiques: application, machine, fonction objective.

Un logiciel est une solution pour connecter des "applications" et des "machines" dans l'espoir d'optimiser une "fonction objective".

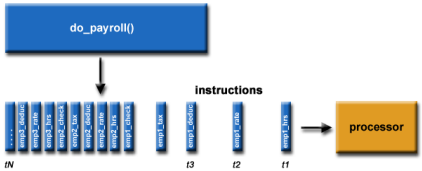
Plan du cours d'aujourd'hui

1. Le calcul parallèle
2. Les applications
   - Paradigmes applicatifs
   - Killer apps
3. Le Parallélisme de la machine
   - Grid/Volunteer computing
   - Cloud, Fog, Edge
   - Cluster, Calcul haute-performance
4. Parallélisme du nœud
   - Architecture de Von Neumann
   - Architectures mémoires
   - Réseaux de communications
5. Les fonctions objectives
   - Loi d'Amdhal et Speedup
   - Comment choisir son objectif
6. Les données
   - Big Data
   - In-Situ/In-Transit

Traditionnellement, un logiciel est écrit pour être calculé en **série** :

- On sépare le problème en une série d'instructions discrètes
- Elles sont exécutées les unes après les autres (séquentiellement)
- Exécutées sur une seule machine (processeur)
- Une exécution à chaque instant.

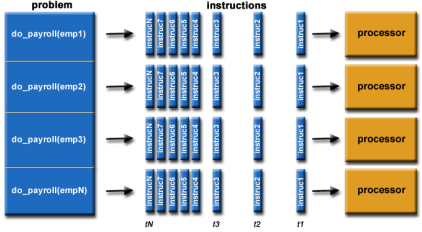
Exemple:



Au sens le plus simple, **le calcul parallèle** est l'utilisation simultanée de plusieurs ressources pour résoudre un problème :

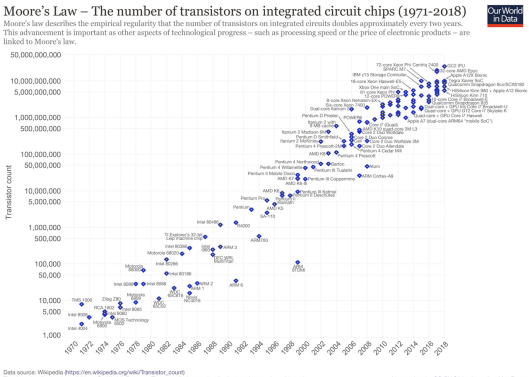
- On sépare le problème en une série d'instructions discrètes qui peuvent être résolues de manière concurrente
- Chaque partie est séparée en une série d'instructions
- Ces instructions sont exécutées sur plusieurs machines
- Système de coordination/contrôle.

Exemple:



Est-ce utile ?

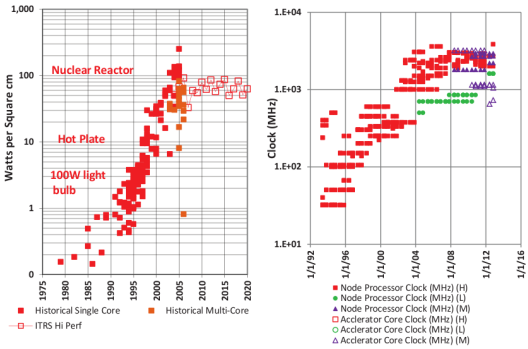
- Les limites aux capacités de calcul;



Loi de Moore : tous les deux ans on peut doubler la densité de transistor sur une carte à puce. Certes...

Est-ce utile ?

- Les limites aux capacités de calcul;

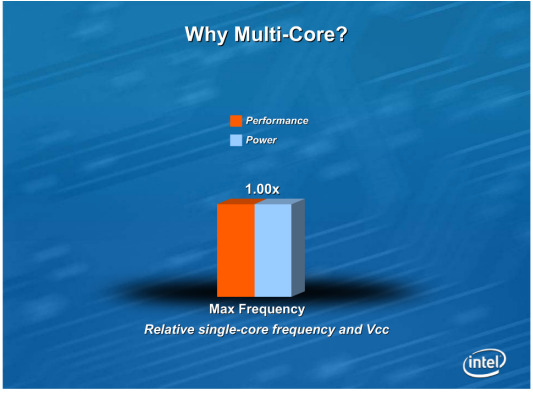


.. mais on ne peut plus les alimenter suffisamment.

Source: `Exascale computing trends`, Kogge and Shalf, 2013

Est-ce utile ?

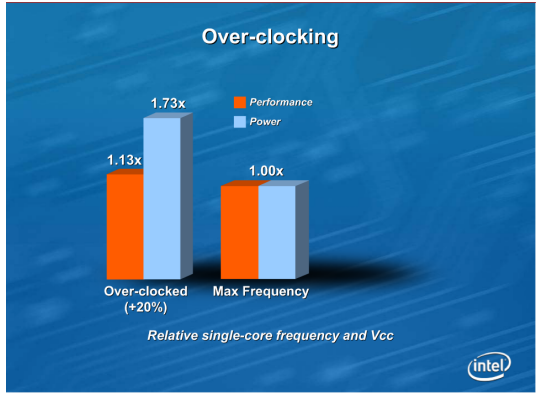
- Les limites aux capacités de calcul;
- Le parallélisme pour les économies d'énergie;



Source: Intel (source: Arnaud Legrand)

Est-ce utile ?

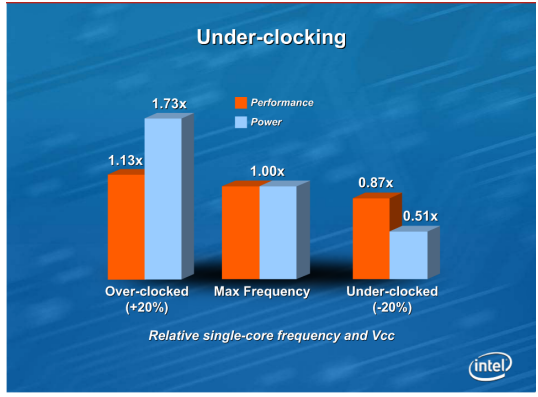
- Les limites aux capacités de calcul;
- Le parallélisme pour les économies d'énergie;



Source: Intel (source: Arnaud Legrand)

Est-ce utile ?

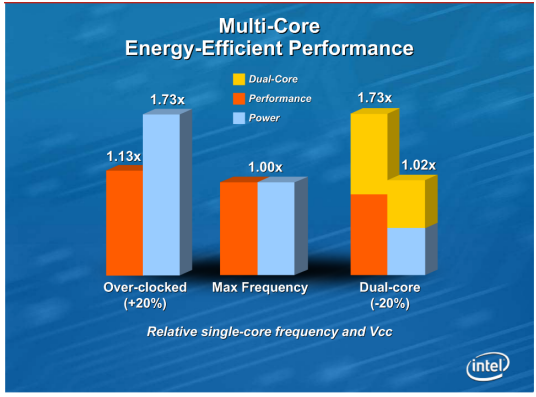
- Les limites aux capacités de calcul;
- Le parallélisme pour les économies d'énergie;



Source: Intel (source: Arnaud Legrand)

Est-ce utile ?

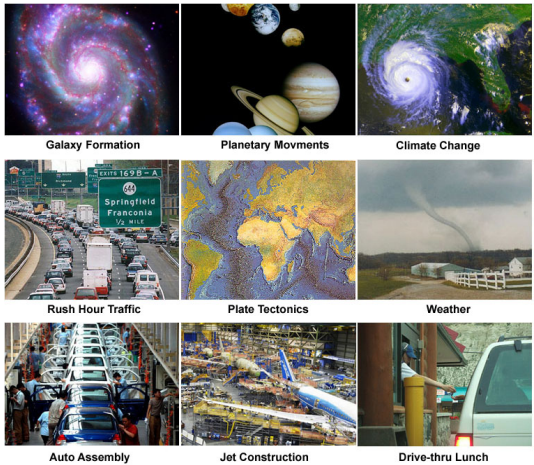
- Les limites aux capacités de calcul;
- Le parallélisme pour les économies d'énergie;



Source: Intel (source: Arnaud Legrand)

Est-ce utile ?

- Les limites aux capacités de calcul;
- Le parallélisme pour les économies d'énergie;
- Des applications intrinsèquement parallèles.



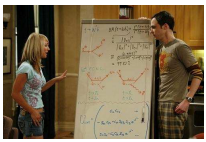
Source : Blaise Barney's tutorial

Plan

Computer Technology and other sciences

Pencil and paper alone cannot solve all our problems. Computer can be used be used as a scientific instrument.

Computer technology has brought us a two new scientific paradigms:


The Big Bang Theory

Big Data

- Dig huge amounts of data (sensors, transaction records, genome and protein databanks,...)
- Enables to discover phenomena or truths that would otherwise remain unseen

Computational Science

- Performing real experiment is very costly and even sometimes simply impossible
- Allows to explore and investigate designs or phenomena in a few hours instead of years

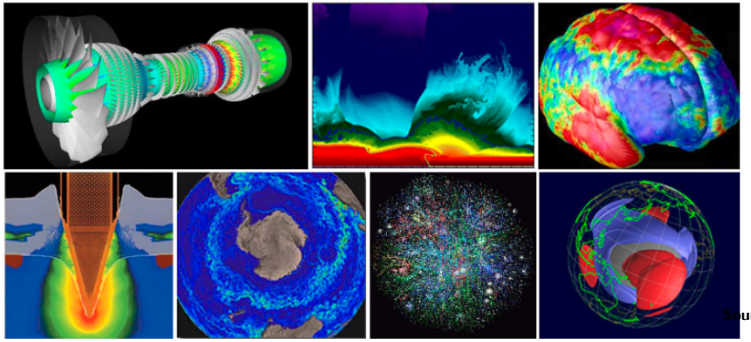
- Motivated the development of major computational infrastructures
- All fields of science (physics, genomics, astronomy, ecology,...) and industry (drug design, avionics, structural engineering, oil companies,...)

Plan

Quelles applications : recherche

Science and Engineering:

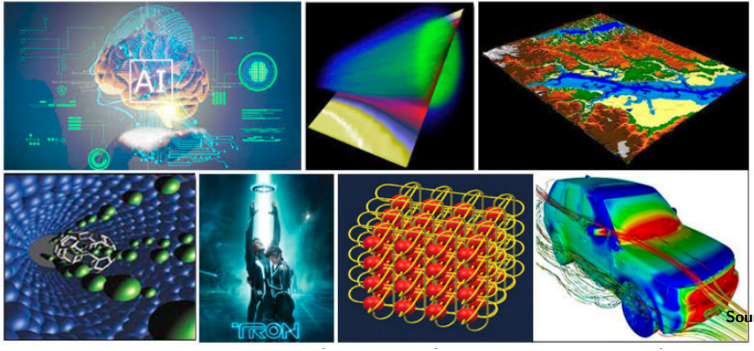
- Historically, parallel computing has been considered to be "the high end of computing", and has been used to model difficult problems in many areas of science and engineering:
  - Atmosphere, Earth, Environment
  - Physics - applied, nuclear, particle, condensed matter, high pressure, fusion, photonics
  - Bioscience, Biotechnology, Genetics
  - Chemistry, Molecular Sciences
  - Geology, Seismology
  - Mechanical Engineering - from prosthetics to spacecraft
  - Electrical Engineering, Circuit Design, Microelectronics
  - Computer Science, Mathematics
  - Defense, Weapons



Source : Blaise Barney

Industrial and Commercial:

- Today, commercial applications provide an equal or greater driving force in the development of faster computers. These applications require the processing of large amounts of data in sophisticated ways. For example:
  - "Big Data", databases, data mining
  - Artificial Intelligence (AI)
  - Web search engines, web based business services
  - Medical imaging and diagnosis
  - Pharmaceutical design
  - Financial and economic modeling
  - Management of national and multi-national corporations
  - Advanced graphics and virtual reality, particularly in the entertainment industry
  - Networked video and multi-media technologies
  - Oil exploration



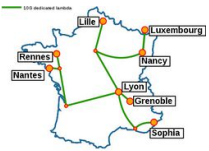
Source : Blaise Barney

Plan

Grid/Internet computing/Volunteer computing

- Essentiellement des ressources de calcul partagées et coordonnées
- Partagées entre de multiples utilisateurs
- Autonomes (auto-gouvernance des ressources)
- Souvent: hétérogènes, distribuées géographiquement.

Grid 5000, *un instrument scientifique support pour la recherche (en particulier, calcul distribué, HPC, Big Data et réseau).*

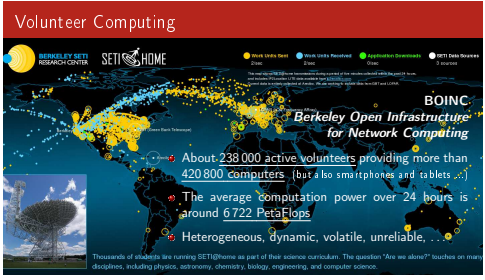


- 8 sites, 31 machines, 828 nœuds, 12328 cœurs.
- 10Gbps de réseau dédié entre les sites
- 550 utilisateurs

`https://www.grid5000.fr/`

Grid/Internet computing/Volunteer computing

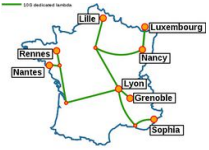
- Essentiellement des ressources de calcul partagées et coordonnées
- Partagées entre de multiples utilisateurs
- Autonomes (auto-gouvernance des ressources)
- Souvent: hétérogènes, distribuées géographiquement.



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

– Nobel committee (chemistry), 2013

Grid 5000, *un instrument scientifique support pour la recherche (en particulier, calcul distribué, HPC, Big Data et réseau).*



- 8 sites, 31 machines, 828 nœuds, 12328 cœurs.
- 10Gbps de réseau dédié entre les sites
- 550 utilisateurs

`https://www.grid5000.fr/`

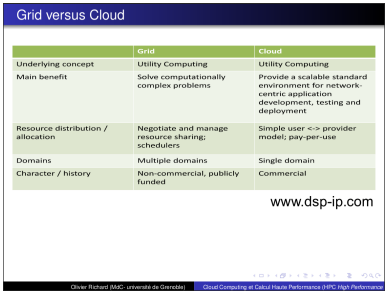
Source : Arnaud Legrand

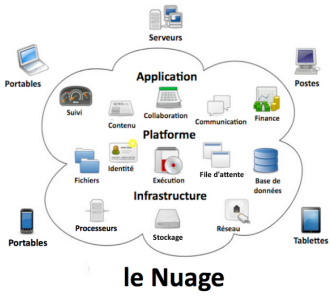
Plan

Cloud, Fog, Edge

Cloud : à l'origine Amazon propose l'utilisation de ses machines non utilisées (EC2).
Edge/fog : puis des *IOT* inutilisés.

- Simplicité d'utilisation, coût à l'usage
- XAAS ("X As A Service")





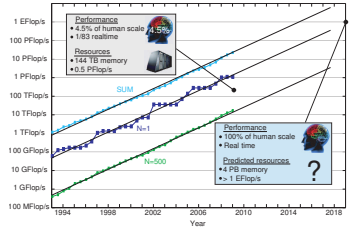
L'informatique en nuage, ou infonuagique.

Source :
`https://fr.wikipedia.org/wiki/Cloud_computing`

Plan

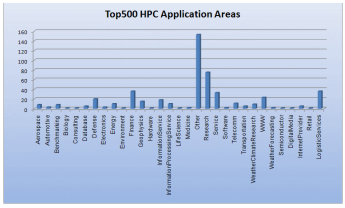
Cluster, Calcul haute-performance

- De *grosses* machines centralisées
- Utilisée pour des applications scientifiques majeures
  `https://en.wikipedia.org/wiki/Grand_Challenges`
- Capables de gérer beaucoup de calcul, beaucoup de données



Source : Data Top500.org, results Modha@IBM.

Source : Titan Supercomputer, Oak Ridge National Lab, US DOE.



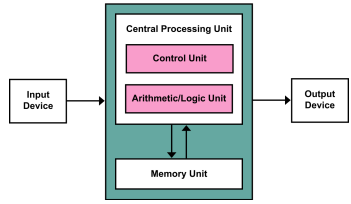
Source : Blaise Barney

Plan

Architecture de Von Neumann (1945)[1]

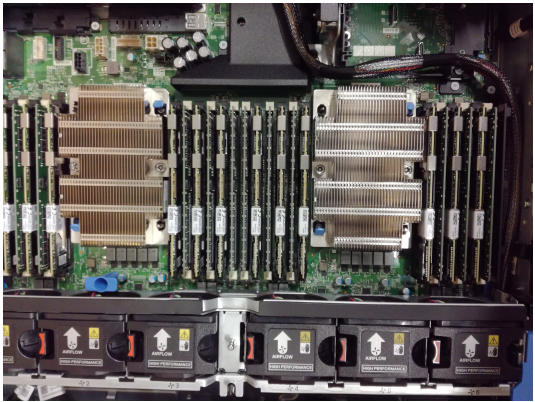
Quatre composants principaux:

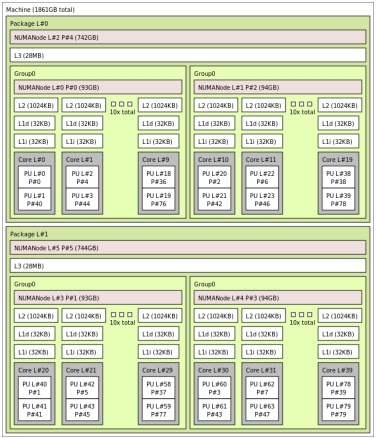
- Unité arithmétique et logique
  → fait les calculs de base
- Unité de contrôle
  → en charge de l'ordre des opérations
- Mémoire
  → contient les données, et le programme
- Entrées/Sorties
  → pour communiquer avec le monde extérieur



Aujourd'hui, les architectures parallèles reprennent essentiellement ce design de base, mais en multipliant le nombre d'unités.

[1] Voir https://en.wikipedia.org/wiki/Von_Neumann_architecture



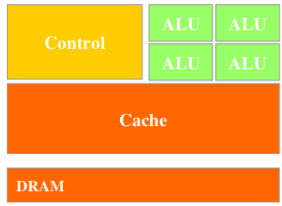


Représentation hwloc.

Processing Units (PU)

Rapide survol de différentes unités de calcul

- **CPU :** Modèle de base. Cache large, peut effectuer de nombreuses fonctions (Unité de contrôle sophistiquées), facile à programmer.

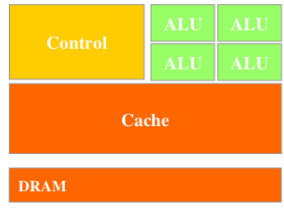


Processing Units (PU)

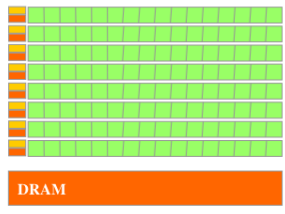
Rapide survol de différentes unités de calcul

- **CPU :** Modèle de base. Cache large, peut effectuer de nombreuses fonctions (Unité de contrôle sophistiquées), facile à programmer.

- **GPU, SIMD, Vector processors :** De nombreuses unités parallèles ; Modèle spécialisés à des opérations simples et répétées ; Idéal pour des applications telles que les calculs sur les images, vidéos, mais une flexibilité plus limitée.



CPU



GPU

Source: Nvidia → Slides JF Méhaut → Slides A Legrand.

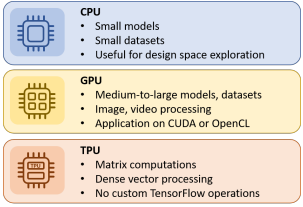
Processing Units (PU)

Rapide survol de différentes unités de calcul

- **CPU :** Modèle de base. Cache large, peut effectuer de nombreuses fonctions (Unité de contrôle sophistiquées), facile à programmer.

- **GPU, SIMD, Vector processors :** De nombreuses unités parallèles ; Modèle spécialisés à des opérations simples et répétées ; Idéal pour des applications telles que les calculs sur les images, vidéos, mais une flexibilité plus limitée.

- **TPU, VPU, NPU, QPU.. :** Nouveaux besoins applicatifs (machine learning, vision, quantum..), nouveaux moyens de calculs spécialisés ; Optimisés pour une fonction (calculs de tenseurs, ..).

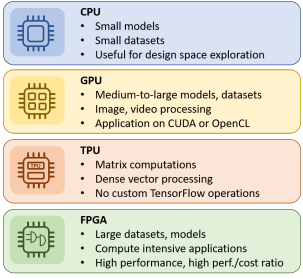


Source: `CPU, GPU, FPGA or TPU: Which one to choose for my Machine Learning training?`, inaccel.com

Processing Units (PU)

Rapide survol de différentes unités de calcul

- **CPU :** Modèle de base. Cache large, peut effectuer de nombreuses fonctions (Unité de contrôle sophistiquées), facile à programmer.
- **GPU, SIMD, Vector processors :** De nombreuses unités parallèles ; Modèle spécialisés à des opérations simples et répétées ; Idéal pour des applications telles que les calculs sur les images, vidéos, mais une flexibilité plus limitée.
- **TPU, VPU, NPU, QPU.. :** Nouveaux besoins applicatifs (machine learning, vision, quantum..), nouveaux moyens de calculs spécialisés ; Optimisés pour une fonction (calculs de tenseurs, ..).
- **FGPA :** Puce configurable : éviter de construire une nouvelle machine pour chacune des nouvelles applications ; Le logiciel commence par choisir les connections pour optimiser l'application.



Source: `CPU, GPU, FPGA or TPU: Which one to choose for my Machine Learning training?`, inaccel.com

Plan

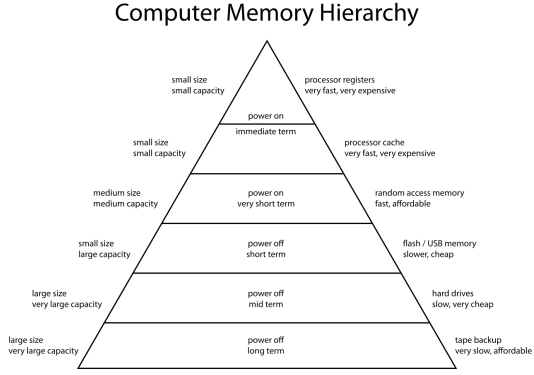
Hiérarchie Mémoire

Quatre principaux niveaux de stockage:

1. Interne: registres, caches..
2. Principal : RAM, ..
3. En ligne, stockage de masse : Stockages secondaires, Burst-Buffers..
4. Hors ligne, stockage de masse : Stockages tertiaires et hors-ligne etc.

Des grandeurs importantes :

- La latence (temps d'accès au stockage)
- La bande passante (vitesse de transfert de données)
- La capacité (espace disponible)
- Le coût



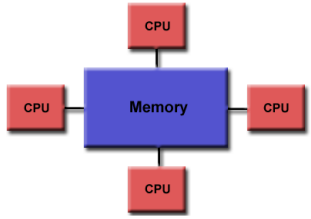
Source: `Wikipedia`, Danlash

Mémoires partagées

- Généralement tous les PU peuvent accéder (=lire/modifier) la ressource mémoire. Les changements faits par un PU sont visibles par les autres. Cohérence de cache matérielle (souvent) : si un PU écrit dans la mémoire, les autres doivent le savoir.

- **UMA** (Uniform memory access) :
  - Processeurs identiques (SMP), accès égaux (temps, localisation) à la mémoire.

- **NUMA** (Non-Uniform memory access) :
  - Souvent en connectant des SMP. Chaque SMP peut accéder à la mémoire des autres. Temps d'accès différents (non uniformes).

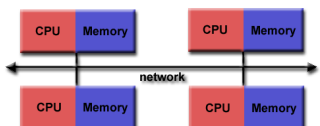


Source: Blaise Barney

L'un des désavantage des mémoires partagées est la limitation de passage à l'échelle : rajouter des CPU rajoute de la congestion sur les réseaux de communication. Encore pire en cas d'implémentation de cohérence de cache.

Mémoires distribuées

- Plusieurs PU avec chacun leur propre mémoire.
- Une caractéristique commune : un réseau de connection entre les mémoires de différents PU
- Un passage à l'échelle simple : rajouter un PU rajoute une mémoire ; pas de cohérence de cache.
- De manière générale, la gestion algorithmique de la ressource est plus compliquée (besoin de réfléchir à la distribution des données, coûts de communications non uniformes, ..).



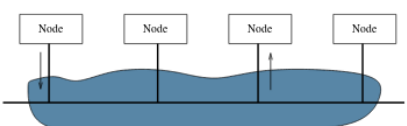
Source: Blaise Barney

Plan

Réseaux de communication

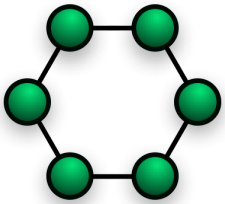
Au cœur du calcul parallèle : le réseau de communication.

- C'est par là que les résultats de calculs sont échangés.
- Ils doivent être efficaces pour profiter pleinement des unités de calculs.



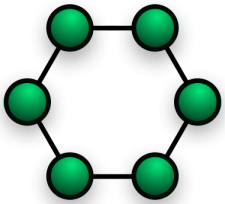
Quelques qualité nécessaires à un réseau :

- Topologie (comment les nœuds sont connectés = notion de distance)
- Latence/Bande passante (à quelle vitesse les messages circulent)
- Coût (combien de switchs, de tuyaux).
- Passage à l'échelle.



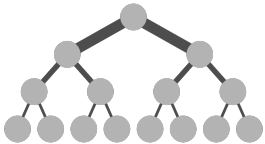
Anneaux

- Peu cher ☺
- Forte Latence ☹
- Mauvais passage à l'échelle ☹
- Non résilient ☹



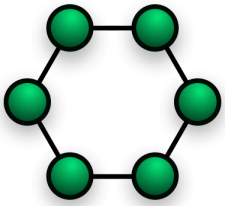
Anneaux

- Peu cher ☺
- Forte Latence ☹
- Mauvais passage à l'échelle ☹
- Non résilient ☹



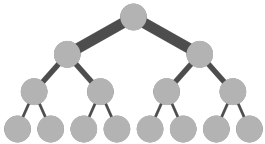
Arbres

- Peu cher ☺
- Faible Latence ☺
- La racine devient un goulot ☹(cf: *fat tree*)



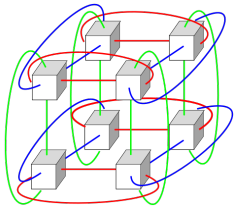
Anneaux

- Peu cher ☺
- Forte Latence ☹
- Mauvais passage à l'échelle ☹
- Non résilient ☹



Arbres

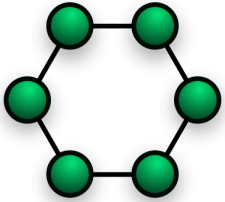
- Peu cher ☺
- Faible Latence ☺
- La racine devient un goulot ☹(cf: *fat tree*)



ND-Tores

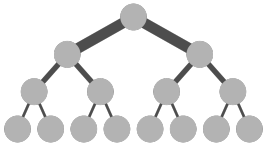
- Cher ☹
- Difficile à connecter ☹
- Latence : $O(n^{1/N})$ ☺
- Grande bande passante ☺

Topologies



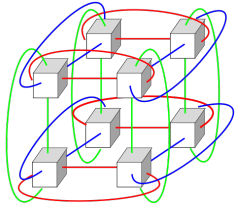
Anneaux

- Peu cher ☺
- Forte Latence ☹
- Mauvais passage à l'échelle ☹
- Non résilient ☹



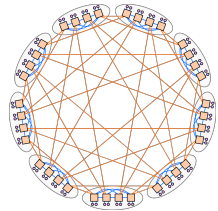
Arbres

- Peu cher ☺
- Faible Latence ☺
- La racine devient un goulot ☹(cf: *fat tree*)



ND-Tores

- Cher ☹
- Difficile à connecter ☹
- Latence : $O(n^{1/N})$ ☺
- Grande bande passante ☺



Dragonfly

- Peu cher ☺
- Latence : $O(1)$ ☺
- Bon passage à l'échelle ☺
- Saturations faciles de liens ☹

Plan

Loi d'Amdhal

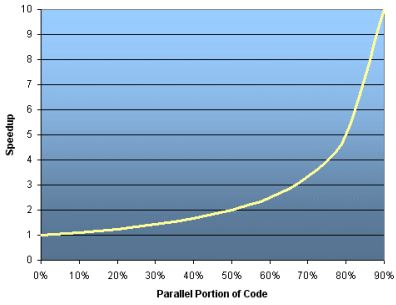
La loi d'Amdhal donne que le temps d'exécution d'un programme est une fonction de sa partie parallèle (proportion p) et de sa partie séquentielle (proportion $1 - p$) :

$$T = (1 - p)T + pT.$$

Son accélération (speedup) théorique est donné par:

$$S = \frac{1}{1 - p}$$

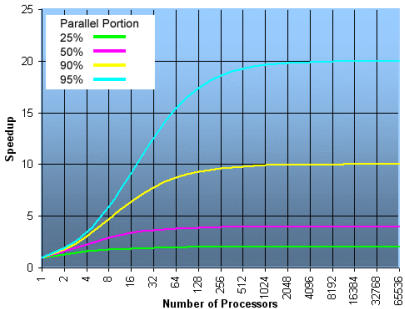
- Si le code ne peut pas être parallélisé ($p = 0$), on ne pourra jamais avoir un speedup plus grand que 1
- Si le code est entièrement parallèle ($p = 1$), le speedup (théorique) est infini.
- Si 50% du code peut être parallélisé, le maximum speedup est de 2: dans le meilleur des cas le code pourra aller deux fois plus vite.



Si on introduit le nombre de processeurs N, $T(N) = (1-p)T + \frac{p}{N}T$ et on peut calculer le speedup réel:

$$\text{Speedup} = \frac{1}{1 - p + p/N}$$

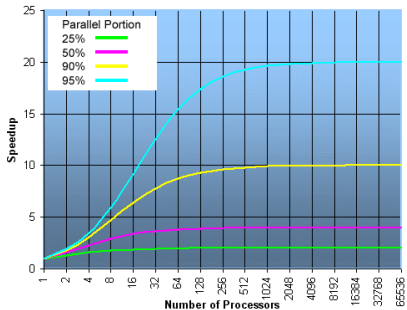
Les limitations du parallélisme :
Un code parallèle à 95% ne pourra jamais aller plus de 20 fois plus vite que la version séquentielle, quelque soit le nombre de processeurs utilisés !



Si on introduit le nombre de processeurs N, $T(N) = (1-p)T + \frac{p}{N}T$ et on peut calculer le speedup réel:

$$\text{Speedup} = \frac{1}{1 - p + p/N}$$

Les limitations du parallélisme :
Un code parallèle à 95% ne pourra jamais aller plus de 20 fois plus vite que la version séquentielle, quelque soit le nombre de processeurs utilisés !



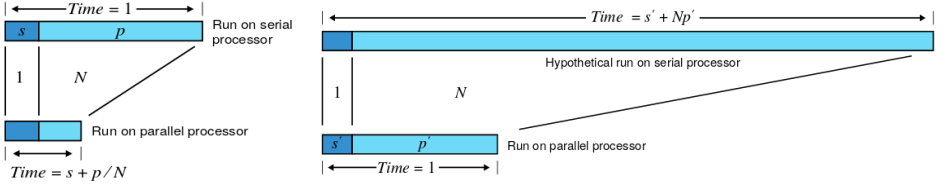
Le parallélisme ça ne sert pas à grand chose ?

Speedup (II)

One does not take a fixed-size problem and run it on various numbers of processors except when doing academic research, Gustafson, 1988.

En fait pour des algos parallèles on a souvent:

- La partie séquentielle : le temps pour l'initialisation, le chargement des données, l'IO.
- La partie parallèle : le "vrai" calcul, qui passe à l'échelle avec le nombre de processeurs (précision du calcul, taille des données etc).

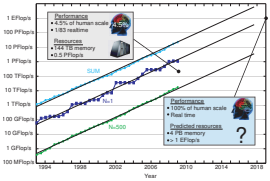


À gauche, le speedup théorique est limité par $1/s$. *À droite il peut tendre vers l'infini.*

Certains problèmes augmentent le pourcentage de la partie parallèle lorsque les données augmentent.

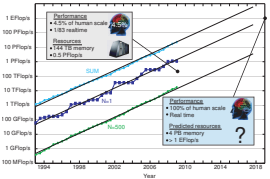
Plan

Quel objectif ?



Faire le plus de calcul possibles
(Rendement)

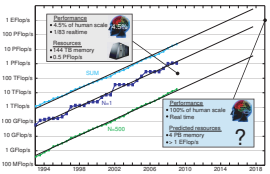
Quel objectif ?

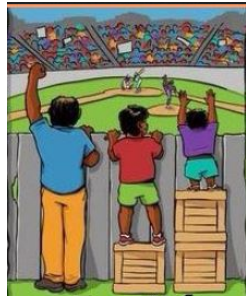


Faire le plus de calcul possibles (Rendement)

Aller le plus vite possible (Makespan)

Quel objectif ?



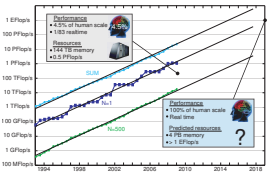


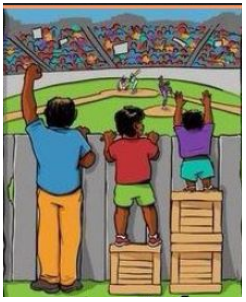
Faire le plus de calcul possibles (Rendement)

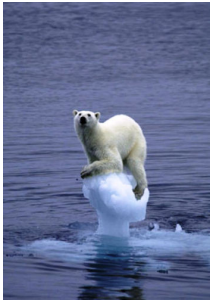
Aller le plus vite possible (Makespan)

Équité entre utilisateurs (Fairness)

Quel objectif ?







Faire le plus de calcul possibles (Rendement)

Aller le plus vite possible (Makespan)

Équité entre utilisateurs (Fairness)

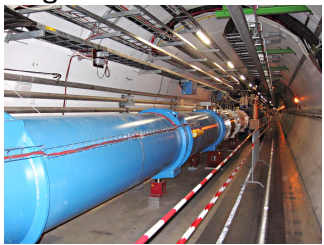
Consommation d'énergie (Energy)

Etc.

Plan

Le mot de la fin pour le truc sexy du moment, les données[2].

Large Hadron Collider :

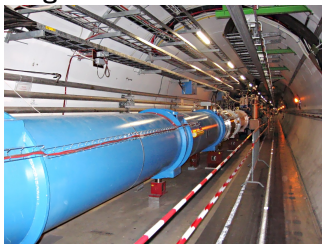


- 60 TB de données par jour (après tri, avant réplication)
- Equiv. Netflix : 7 ans de film généré chaque jour

[2] En équivalent Netflix: 16.25 MB/min pour une Full HD 1080.

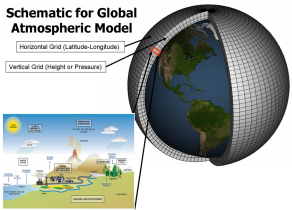
Le mot de la fin pour le truc sexy du moment, les données[2].

Large Hadron Collider :



- 60 TB de données par jour (après tri, avant réplication)
- Equiv. Netflix : 7 ans de film généré chaque jour

Modélisation climatique :

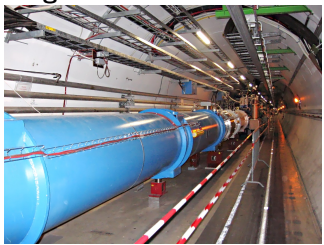


- $O(1)$ PB de données d'observations (en 2010) stockées par le NASA Center for Climate Simulation
- Equiv. Netflix : plusieurs centaines d'années de film.

[2] En équivalent Netflix: 16.25 MB/min pour une Full HD 1080.

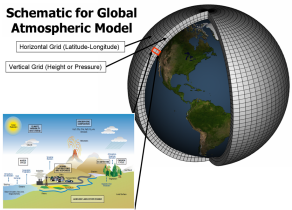
Le mot de la fin pour le truc sexy du moment, les données[2].

Large Hadron Collider :



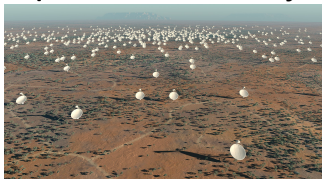
- 60 TB de données par jour (après tri, avant réplication)
- Equiv. Netflix : 7 ans de film généré chaque jour

Modélisation climatique :



- $O(1)$ PB de données d'observations (en 2010) stockées par le NASA Center for Climate Simulation
- Equiv. Netflix : plusieurs centaines d'années de film.

Square Kilometer Array :

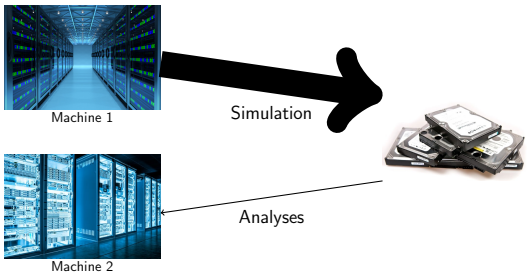


- 14 EB de données par jour (1PB/jour de stocké)
- Equiv. Netflix : 1 640 000 ans de film (117 années en stockage)

[2] En équivalent Netflix: 16.25 MB/min pour une Full HD 1080.

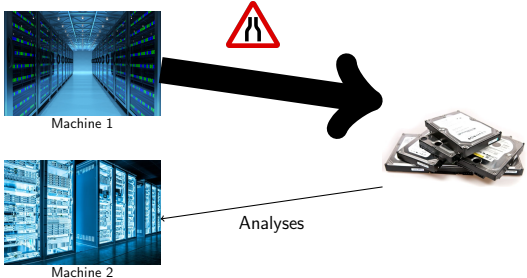
Plan

In Situ, In transit computing



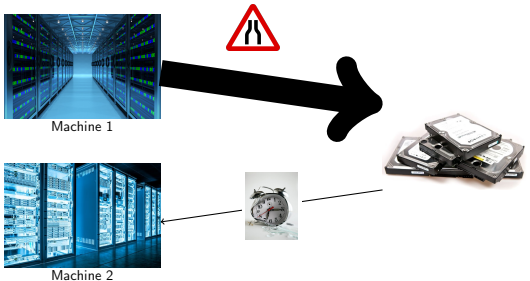
Source : Valentin Honoré

In Situ, In transit computing



Source : Valentin Honoré

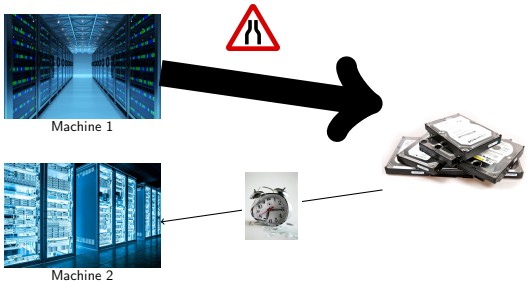
In Situ, In transit computing



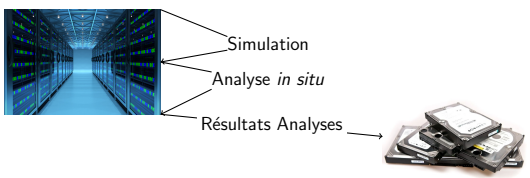
- Bonnes performances de calcul ☺
- Contention système IO ☹
- stockage insuffisant ☹

Source : Valentin Honoré

In Situ, In transit computing



- Bonnes performances de calcul ☺
- Contention système IO ☹
- stockage insuffisant ☹



- Limite la contention IO ☺
- Localité des données ☺
- Partage des ressources entre simu et analyse ☹

Source : `Valentin Honoré`

Références

Une grosse partie de ce cours est inspirés de divers cours piqués à droite et gauche, et en particulier de ceux-ci:

- Le tutoriel de Blaise Barney: *Introduction to Parallel Computing*. Tutorial of the Lawrence Livermore National Laboratory.
  `https://computing.llnl.gov/tutorials/parallel_comp/`.

- Le cours d'Arnaud Legrand et Vincent Danjean
  `http://polaris.imag.fr/arnaud.legrand/teaching/2013/M2R_PC.php`.