A rattrapper !!!!!

On trouve une puissance de calcul de l'ordre de $10^3 5$ op/sec par kg de masse.

Aussi, avec un calcul sur l'entropie, on aurait comme limite de mémoire $2 \times 10^{31} \frac{\text{bit}}{\text{kg}}$

- DNA storage: stocker des bits sur des brins d'ADN

  Capacité: 500 exabytes (10^{18}) par gramme d'ADN

- Cerveau humain: quelle sont ses performances ?

$\approx 10^{11}$ neurones, 10^4 synapse par neurone à 10 Hz: -> Peta operations par seconde pour une puissance consommée: 100 Watts

- Remarques: Faut-il plutôt beaucoup de Processing Elements (PE) à faible puissance ou 1 seul à très grande puissance ?

Si μ est la fréquence du processeur, et qu'il y en a n, on a une puissance de calcul qui est $R = n\mu$

La puissance consommée $P = n\mu^2$

Donc le rapport $\frac{R}{P} = \frac{n\mu}{n\mu^2} = \frac{1}{\mu}$ que l'on veut maximiser

Donc $\mu \to 0$ et $n \to \inf$

2. On peut aussi trouver des performances avec des améliorations architecturales

- Comment agencer les différents composants pour maximiser les performances du tout ?
  - Mémoire cache: avoir de la mémoire proche du CPU (Même chip) pour diminuer le goulet d'étranglement de Von Neumann
  - Processeurs vectoriels → registres vectoriels (Dans le CPU, on a un registre (registre vectoriel) qui contient beaucoup de données (512 mots). On a des données en mémoire. On prend d'un coup plusieurs données de la mémoire)
  - Exécution en pipeline
  - Parallélisme: avoir plusieurs PE ainsi qu'à tous les niveaux -> plusieurs instructions en même temps dans un coeur (ILP: Instruction Level Parallelism)
  - Comment faire mieux ? Memory Wall ! L'accès aux données est l'élément limitatif tant du point de vue des performances que de l'énergie consommée. -> Il faudrait fonc calculer directement dans la mémoire -> par exemple les Automates Cellulaires

3. Algorithmes

Solution de systèmes d'équations linéaire

- Élimination de Gauss
- Gauss-Seider

- SUR
- ...
- Adaptivité , multigrid

Exemple: (1970: utilisation des méthodes de Gauss; années 2000: utilisation Adaptivite -> gain de 10^4) (Puissance des ordinateurs: technologie + architecture -> facteur 10^4 entre les années 1970 et les années 2000)

Autre exemple: cryptographie: factoriser de très grands nombres pour casser un code style RSA (En 1970, on estimait à $*15 * 10^9$ ans le temps de calcul pour casser RSA. Mais en 1990, il a fallu 8 mois sur 600 stations de travail. Ce succès a été obtenu grâce à un algo très sophistiqué)

Évolution des machines HPC (High Performance Computers) Il y a toujours eu des vesoins pour obtenir plus de performances que les ordinateurs séquentiels du moment le permettaient.

- ILLIAC IV (1970) 64 processeurs pour faire des prévisions météo. (ils ont fait 4 fois moins de processeurs et cela a coûté 4 fois plus cher que prévu)
- 1975 - 1990 -> machines vectorielles. C'était la réfécence du superordinateur durant ces années.
- 1990 - 2000 -> parallélisme s'imposer (machines SIMD, MIND, MPP, SMP (Simetric Multi Processors), ...) Pas de langages de programmation communs
- 1995 Cluster de PC (Beowolf) avec des processeurs du marché de plus en plus puissants. MPI est né à ce moment pour coupler ces processeurs de façon standard
- GRID, Cloud (2005)
- 2010 - 20.. -> Multicoeurs, GPU, ...

⇒ **Parallélisme a toujours été présent dans les esprits, mais il a toujours été suplantés par des techniques plus simples jusqu'à maintenant.**

Systèmes parallèles et répartis (distribués) Plusieurs PE ⇒ concurrence

- calcul parallèle
- calcul réparti

Parallélisme: Plusieurs processeurs coopérant à la solution d'un même problème

Réparti: ensemble de processeurs qui résolvant plusieurs problèmes couplés

Tableau différentiateur

	couplage	mise en commun délibérée	granularité	hypothèses sur le système	connaissance mutuelle	sécurité
parallélisme	fort	oui	fine	oui	oui	non

	couplage	mise en commun délibérée	granularité	hypothèses sur le système	connaissance mutuelle	sécurité
réparti	faible	non	grossière	non	non	oui

Exploiter le parallélisme *Comment faire coopérer plusieurs processeurs pour efficacement résoudre un problème ?*

Pour illustrer des stratégies de coopération, on va utiliser un exemple de la vie de tous les jours:

La famille Dupont, Mme, M, Marie et Pierre doivent préparer un goûter d'anniversaire avec un grand nombre de tartines.

Les tâches pour chaque tartine sont les suivantes: 1. Couper une tranche de pain 2. Beurrer la tranche 3. Mettre la confiture 4. Ranger sur un plat 1 ère stratégie (séquentielle) - Mme Dupont fait toutes les tartines car le reste de la famille est incapable et ralentirait le processus.

Parfois, un seul processeur, puissant et bien programmé, fait bien le travail.

```
tartine nb
      ^
      |            t1 t2 t3 t4
      |t1 t2 t3 t4
      |_______________
                 temps
```

Pour n tartines, il faut $T_{seq} = N4\tau$ où τ est le temps de chacune des 4 tâches

2 ème stratégie: Travail à la chaîne

```
           Mme    M   Pierre   Marie
tartine -> t1 -> t2 -> t3   ->  t4
```

Dès que Mme a coupé sa tranche, M peut la beurrer et Mme, pendant ce temps, coupe la suivante

```
tartine nb
      ^
      |   t1 t2 t3 t4      overlap des tâches et on peut avoir jusqu'à 4 tartines en prépara
      |t1 t2 t3 t4
      |_______________
                 temps
```

$T_{pipeline} = 4\tau + (n-1)\tau$ avec 4τ le temps de la première tartine et $(n-1)$ car à chaque temps τ, une nouvelle tartine est finie

Gain: $\frac{T_{seq}}{T_{pipeline}} = \frac{4\tau n}{4\tau+(n-1)\tau}$ $\frac{4\tau n}{n\tau} = 4$ On a 4 travailleurs -> 4 fois plus vite.

Mais il y a des limites au travail à la chaîne:

- Toutes les tâches doivent être de même durée, et les PE de même puissance
- Si le pipeline s'interrompt, il faut le 'recharger' pour avoir des performances
- Il n'y a du travail que pour 4 personnes car il n'y a que 4 tâches... Si le voisin veut aider, il n'aura rien à faire: Gain limité à 4.

3 ème stratégie: SIMD (Single Instruction flow, Multiple Data flow) (C'est un peu une organisation militaire... Il y a une liste de choses à faire... Chacun fait la même chose en même temps). Avantages:

- Autant de travailleurs jusqu'à N le nombre de tartines.
- Tâches peuvent être de durée différente -> on reste synchronisé.

tartine nb

```
                t1 t2 t3 t4
  ^             t1 t2 t3 t4
p |t1 t2 t3 t4
  |t1 t2 t3 t4
  |_______________
              temps
```

Si on a p travailleurs $T_{SIMD} = \frac{N}{p} 4\tau$

Gain: $\frac{T_{seq}}{T_{SIMD}} = \frac{N4\tau}{\frac{N}{p}4\tau} = p$ avec $p = 1 \dots n$