Credits

- Ce cours est basé sur celui de Pierre Fortin (SU $\rightsquigarrow$ ULille)
- Celui de P. Fortin était basé sur celui de J.-L. Lamotte (SU)

Bibliographie

- *Parallel Programming in C with MPI and OpenMP* (M.J. Quinn)
- *MPI : A Message-Passing Interface Standard*
- *OpenMP Application Programming Interface*
- ...

Plan du cours

1. Introduction au parallélisme (et machines parallèles)
2. Standard MPI
3. Algorithmes parallèles
4. Programmation multi-thread (OpenMP de base)
5. Programmation multi-thread avancée (task OpenMP)
6. Programmation SIMD / vectorisation
7. Introduction au calcul haute performance

Hors-programme : GPU et autres accélérateurs (RDV en M2)

Cours 1 : Introduction

Charles Bouillaguet
(charles.bouillaguet@lip6.fr)

2020-01-27

Qu'est-ce que le HPC ?

***High-Performance Computing* (HPC)**

Mélange de :

1. Matériel spécialisé (parallèle)
2. Algorithmes adaptés (parallèles)
3. Techniques de programmation *ad hoc*

+ Programmeurs spécialement formés !

L'importance de la simulation numérique

Méthode scientifique traditionnelle

1. Elaboration d'une théorie à partir des observations
2. Expérimentation physique pour valider la théorie
3. Confrontation des résultats de l'expérimentation aux observations

On itère le processus si les résultats de l'expérimentation ne correspondent pas aux observations.

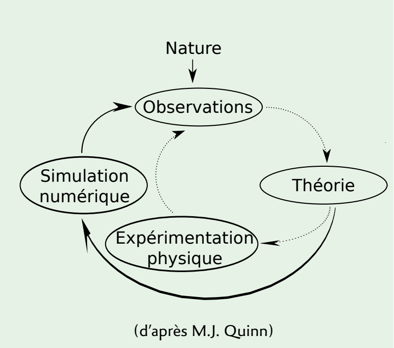
L'expérimentation peut être trop...

- Difficile (*Wind Tunnel* contenant un Airbus A380 ?)
- Chère (*crash test*, ...)
- Lente (évolution du climat, dynamique des galaxies, ...)
- Dangereuse (essais cliniques, ☣, ☢, ✺—)

Simulation numérique

Utilisation d'ordinateurs pour simuler et analyser le phénomène

- à partir de lois physiques
- grâce à des méthodes numériques
- avec des ordinateurs toujours plus puissants



(d'après M.J. Quinn)

⇝ 3ème pilier de la science ?

Les grands domaines d'application du HPC

"Grands challenges" scientifiques

- Chimie, biologie : dynamique moléculaire
- Bio-informatique : séquençage du génome
- Astrophysique : dynamique des galaxies, de l'Univers
- Climatologie : échanges océan $\leftrightarrow$ atmosphère
- Méca. des fluides : écoulements turbulents, combustion
- ...

Et aussi :

- Géophysique : propagation d'ondes sismiques
- Industrie automobile : simulation d'accidents
- « *Deep Learning* »
- ...

Puissance de calcul

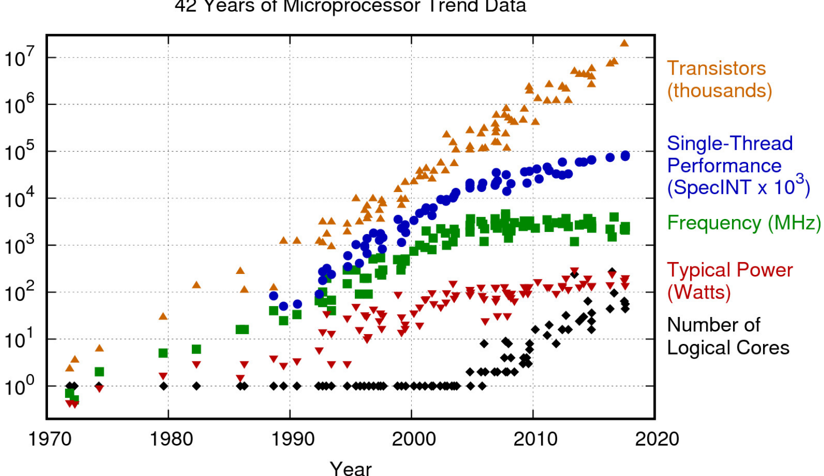
Mesure de performance

Unité : FLOP (*Floating Point OPeration*), FLOP/s (ou FLOPS)

- Giga ($10^9 \approx 2^{30}$)
- Tera ($10^{12} \approx 2^{40}$)
- Peta ($10^{15} \approx 2^{50}$)
- Exa ($10^{18} \approx 2^{60}$) ...

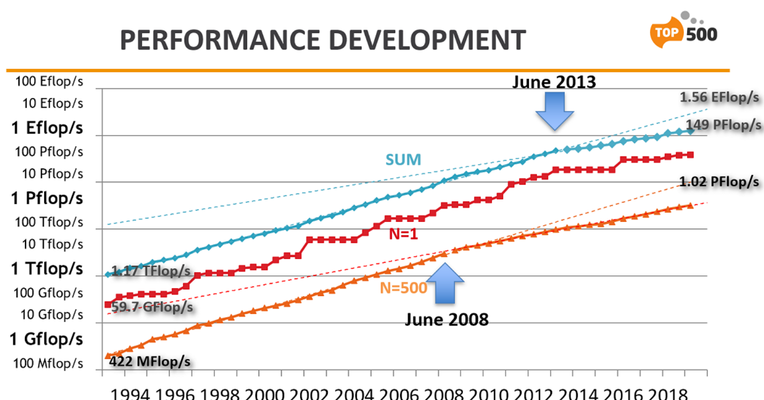
- Machines pour gros calculs = **machines parallèles**
- Le parallélisme est une **tendance de fond**

La “loi de Moore” et la fin du “*Dennard Scaling*”



La “loi de Moore” et la fin du “*Dennard Scaling*”

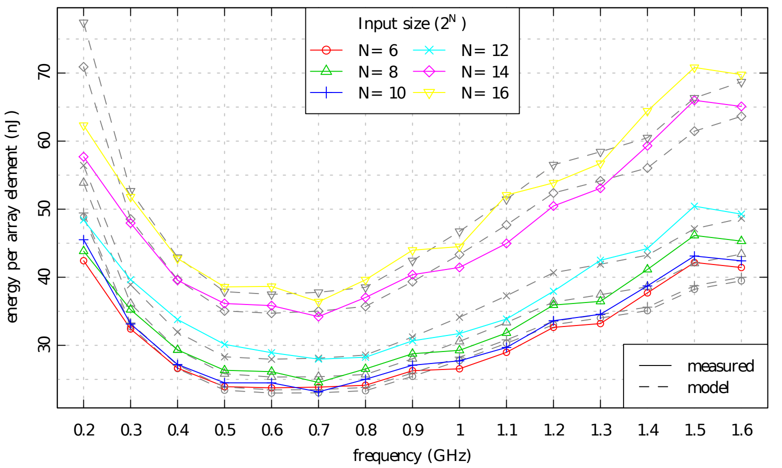
Repercussion sur le TOP500



Amélioration de l'efficacité énergétique (FLOP/Watt)

⇝ réduire fréquence, augmenter # processeurs

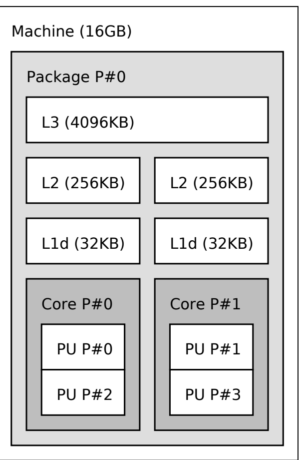
CPU = Cortex A9 (ARM, smartphone). Calcul = (morceau de la) FFT.



Glossaire

machine	ensemble des composants
cluster	des serveurs de calcul réliés à un réseau (~~COVID~~)
noeud	un « ordinateur » indépendant
baie	armoire contenants noeuds, switch, etc.
SMP	noeud qui contient plusieurs processeurs
processeur	objet qui contient au moins un *coeur* + cache, etc.
coeur	circuit qui exécute du code de façon autonome
multicoeur	processeur qui contient plusieurs coeurs
thread matériel	contexte d'exécution autonome dans un coeur
SMT	coeur qui héberge plusieurs threads matériels

Glossaire



- thread matériel $\neq$ thread logiciel (OS)
- “coeur physique” $\approx$ coeur
- “coeur logique” $\approx$ thread matériel

Notions de processus et de thread dans les OS

Processus	« flot d'exécution » + « espace mémoire »
Thread	« flot d'exécution »

Notions de processus et de thread dans les OS

Processus « flot d'exécution » + « espace mémoire »

Thread « flot d'exécution »

Eléments propres à chaque processus	Eléments propres à chaque thread
Espace d'adressage	Compteur ordinal
Variables globales	Registres
Fichiers ouverts	Pile (dont variables locales)
Processus enfant, signaux...	Etat
Mode multi-processus	Mode multi-thread

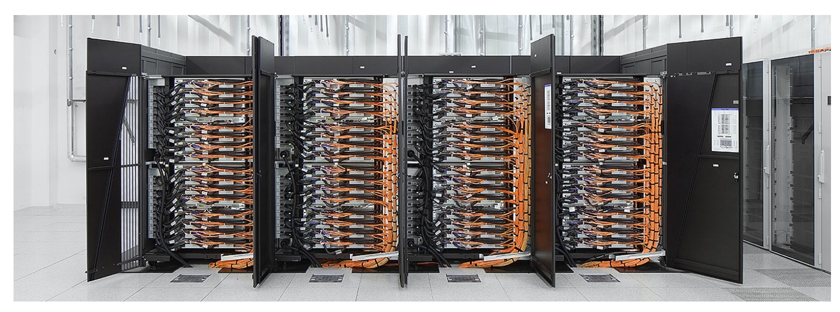
Une machine de HPC : `Turing`

IBM BlueGene/Q (2012–2019) @ IDRIS (CNRS)

Une machine de HPC : Turing

IBM BlueGene/Q (2012–2019) @ IDRIS (CNRS)

6 baies, 98 304 coeurs ("petit") ⇝ Sequoia (USA) en a 1 572 864



Une machine de HPC : Turing

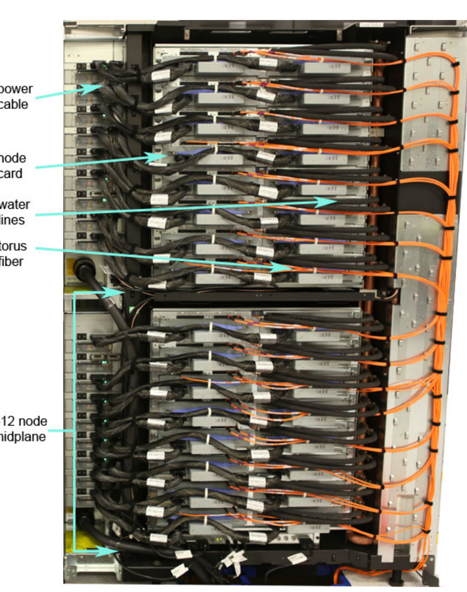
IBM BlueGene/Q (2012–2019) @ IDRIS (CNRS)

1 baie =

- 16 *node cards*
- + alimentation
- + *water cooling*
- + noeuds I/O

Total =

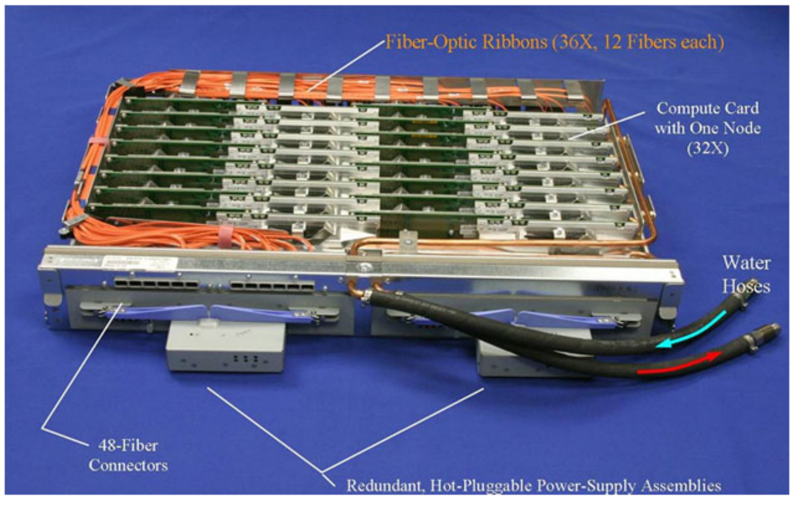
- 1024 noeuds
- 16 384 coeurs
- 16 To de RAM



Une machine de HPC : `Turing`

IBM BlueGene/Q (2012–2019) @ IDRIS (CNRS)

1 *node card* = 32 *compute cards*



Une machine de HPC : Turing

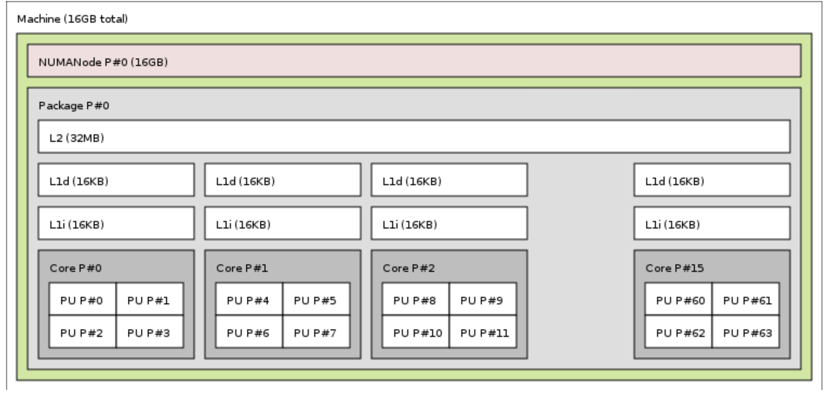
IBM BlueGene/Q (2012–2019) @ IDRIS (CNRS)

1 *compute card* =

- 1 processeur (PowerPC A2, 1.6Ghz, 16 coeurs)
- 16Go de RAM ECC (36 chips de 512Mo)

Une machine de HPC : `Turing`

IBM BlueGene/Q (2012–2019) @ IDRIS (CNRS)



Quelques exemples (au CNRS)

1. `Turing` (IDRIS / CNRS, 2012–2019), 1.2 PetaFLOPS
   - IBM BlueGene/Q ($\approx$ 20 millions €)
   - 6144 noeuds, 98 304 coeurs, 98To de RAM
     - 16Go + 1 $\times$ IBM PowerPC A2, 16 coeurs @ 1.6 Ghz
     - 1 CPU = ??? € = 200 GigaFLOPS
   - Réseau = “maison”, tore 5D, 10 $\times$ 2Go/s par noeud.
   - 600kW (cold water cooling)

Quelques exemples (au CNRS)

1. `Turing` (IDRIS / CNRS, 2012–2019), 1.2 PetaFLOPS
   - IBM BlueGene/Q ($\approx$ 20 millions €)
   - 6144 noeuds, 98 304 coeurs, 98To de RAM
     - 16Go + 1 $\times$ IBM PowerPC A2, 16 coeurs @ 1.6 Ghz
     - 1 CPU = ??? € = 200 GigaFLOPS
   - Réseau = "maison", tore 5D, 10 $\times$ 2Go/s par noeud.
   - 600kW (cold water cooling)
2. `Jean-Zay` (IDRIS / CNRS, 2019–), 28 PetaFLOPS
   - HPE SGI 8600 (25 millions €)
   - 1528 noeuds "CPU" $\rightarrow$ 61 120 coeurs, 4.9 PetaFLOPS
     - 192Go + 2 $\times$ Intel Xeon Gold 6248, 20 coeurs, 2.5Ghz
     - 1CPU = 3000€ = 1.6 TeraFLOPS
   - réseau = Intel OmniPath 100Gbit/s
   - $\approx$ 1MW (warm water cooling)

Quelques exemples (au CNRS)

1. `Turing` (IDRIS / CNRS, 2012–2019), 1.2 PetaFLOPS
   - IBM BlueGene/Q (≈ 20 millions €)
   - 6144 noeuds, 98 304 coeurs, 98To de RAM
     - 16Go + 1 × IBM PowerPC A2, 16 coeurs @ 1.6 Ghz
     - 1 CPU = ??? € = 200 GigaFLOPS
   - Réseau = “maison”, tore 5D, 10 × 2Go/s par noeud.
   - 600kW (cold water cooling)

2. `Jean-Zay` (IDRIS / CNRS, 2019–), 28 PetaFLOPS
   - HPE SGI 8600 (25 millions €)
   - 1528 noeuds “CPU” → 61 120 coeurs, 4.9 PetaFLOPS
     - 192Go + 2 × Intel Xeon Gold 6248, 20 coeurs, 2.5Ghz
     - 1CPU = 3000€ = 1.6 TeraFLOPS
   - 662 noeuds “GPU” → 199 040 coeurs, 21.5 PetaFLOPS
     - idem + 4 × NVIDIA V100 SMX2 16/32Go, 80 coeurs
     - 1 GPU = 8500€ = 8 TeraFLOPS
   - réseau = Intel OmniPath 100Gbit/s
   - ≈ 1MW (warm water cooling)

Quelques exemples (TOP 500)

1. #2 = `Summit` (DoE, USA, 2018–), 200 PetaFLOPS
   - IBM ($\approx$ $ 500 millions)
   - 4 608 noeuds
     - $2\times$ IBM Power9, 22 coeurs, 3Ghz
     - $6\times$ NVIDIA V100 16Go
     - 512Go DDR4 + 96Go HBM + 1.6To non-volatile memory
     - 1 noeud $\approx$ 120 000€ = 42 TeraFLOPS

   $\rightarrow$ 2.8Po de RAM, 202 752 coeurs CPU, 2 211 840 coeurs GPU
   - Réseau = Infiniband 100Gb/s
   - 13MW

Quelques exemples (TOP 500)

1. #2 = `Summit` (DoE, USA, 2018–), 200 PetaFLOPS
   - IBM (≈ $ 500 millions)
   - 4 608 noeuds
     - $2\times$ IBM Power9, 22 coeurs, 3Ghz
     - $6\times$ NVIDIA V100 16Go
     - 512Go DDR4 + 96Go HBM + 1.6To non-volatile memory
     - 1 noeud ≈ 120 000€ = 42 TeraFLOPS

   → 2.8Po de RAM, 202 752 coeurs CPU, 2 211 840 coeurs GPU
   - Réseau = Infiniband 100Gb/s
   - 13MW

2. #6 = `Frontera` (Univ. Texas, 2019–), 38.7 PetaFLOPS
   - Dell (≈ $60 millions)
   - 8008 noeuds
     - 192Go + 2 $\times$ Intel Xeon Platinum 8280, 28 coeurs, 2.7Ghz
     - 1 CPU = 10 000€ = 2.4 TeraFLOPS

   → 1.5Po RAM, 448 448 coeurs
   - réseau = Infiniband 100Gbit/s

Et bien sûr...

- #1 = `Fugaku` (RIKEN, Japon, 2021–), 400 PetaFLOPS
  - Fujitsu ($\approx$ 900 millions €)
  - $\geq$ 150 000 noeuds
    - 32Go HBM2 + $1\times$ Fujitsu A64FX (arm), 48 coeurs, 2Ghz
    - 1 CPU = 2.7 TeraFLOPS

  $\rightarrow$ 4.8Po de RAM, 7 200 000 coeurs
  - Réseau = Tore 6D (tofu)

Beintôt des machines (pre-)exascale en Europe

Votre futur (proche)...

- #9999 = Une salle de TP, ≈ 0.8 TeraFLOPS
  - 16 noeuds
    - 4Go RAM + $1\times$ Core i5 (4 coeurs)
    - 1 CPU = 50 GigaFLOPS

  $\rightarrow$ 64Go de RAM, 64 coeurs
  - Réseau = (gigabit ?) ethernet

Enfin, ça c'était dans le monde d'avant !

Grande variété architecturale

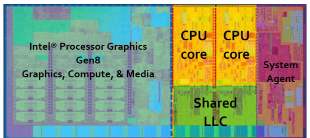
Comment obtenir de la puissance de calcul ?

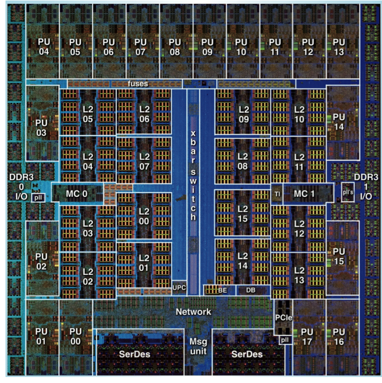
- CPUs les plus puissants possible (`frontera`)
- Plein de CPUs peu puissants (IBM BlueGene)
- Accélérateurs matériels (GPU) (`summit`, `jean-zay`)
- Convergence CPU/GPU (`fugaku`)

Difficulté de portage

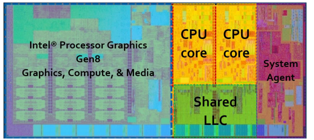
À l'extrême, des programmes sont parfois optimisés pour UNE machine donnée.

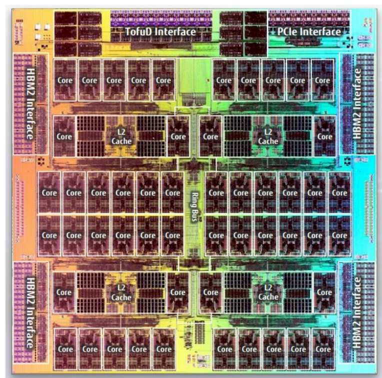
Pas des ordinateurs « normaux »





Pas des ordinateurs « normaux »

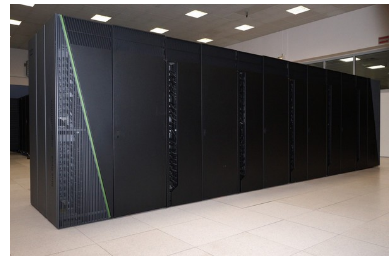




$\Rightarrow$ Programmeurs spécialement formés

⇒ Programmeurs spécialement formés

⇒ Programmeurs spécialement formés



⇒ Programmeurs spécialement formés

⇒ Programmeurs spécialement formés

Produit matrice-matrice (`C += A*B`)

```
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        for (int k = 0; k < n; k++)
            C[i][j] += A[i][k] * B[k][j];
```

Sur un serveur de calcul moderne :

- $\approx 0.05\%$ des performances max. théoriques
- BLAS multi-thread optimisé : $\approx 60\%$.

Les machines parallèles

Classification de Flynn (1966) :

		Flot de données	
		unique	multiple
Flot d'instructions	unique	**SISD**	**SIMD**
	multiple	**MISD**	**MIMD**

Les machines parallèles

Classification de Flynn (1966) :

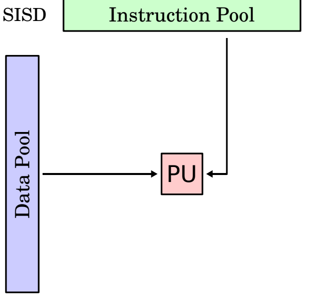
		Flot de données	
		unique	multiple
Flot d'instructions	unique	**SISD**	**SIMD**
	multiple	**MISD**	**MIMD**

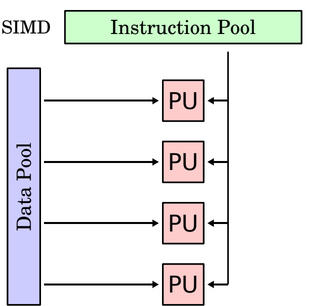
- **machines SISD** Ce sont les machines séquentielles !
- **machines MISD** Ça n'existe pas vraiment.
  (Chaque processeur recevrait des instructions distinctes opérant sur le même flot de données ???)

Classification de Flynn (suite)

- **machines SIMD**
  Les unités de traitement exécutent simultanément la même opération sur leurs données propres.
  - fonctionnement synchrone
  - une seule unité de contrôle centralisée
  - exemples :
    - machines vectorielles des années 1970–90 (CRAY , NEC ...)
    - instructions vectorielles (SSE, AVX2, AVX-512, NEON, ...)
    - Graphics Processing Units (GPUs)

SIMD





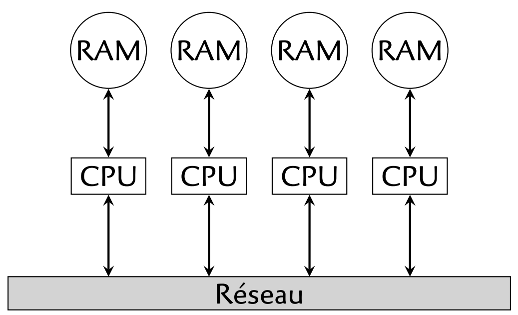
(image : Wikipédia)

Classification de Flynn (suite)

- **machines SIMD**
  Les unités de traitement exécutent simultanément la même opération sur leurs données propres.
  - fonctionnement synchrone
  - une seule unité de contrôle centralisée
  - exemples :
    - machines vectorielles des années 1970–90 (CRAY , NEC ...)
    - instructions vectorielles (SSE, AVX2, AVX-512, NEON, ...)
    - Graphics Processing Units (GPUs)
- **machines MIMD**
  Les processeurs peuvent effectuer différentes opérations sur différentes données simultanément.
  - fonctionnement asynchrone
  - exemples :
    - grappe (*cluster*) de PC
    - Toutes les grosses machines de HPC

Mémoire distribuée

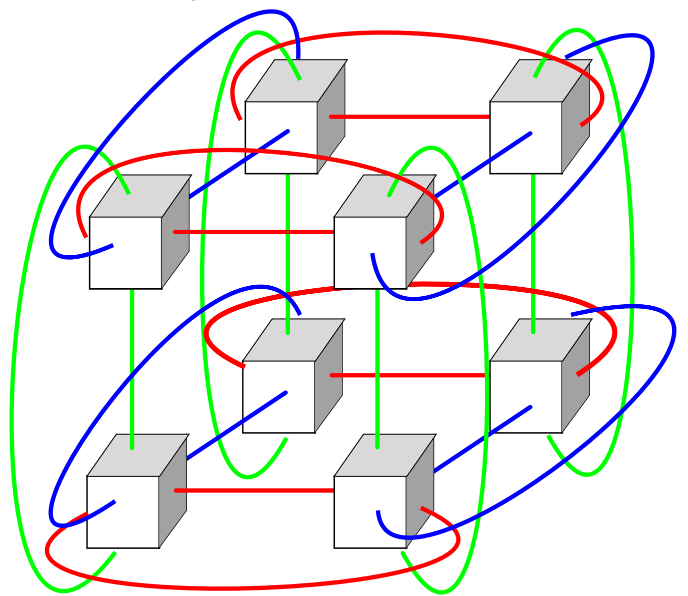
Classification selon l'organisation de la mémoire



- Chaque processeur possède sa propre mémoire.
- Communication = échange de messages sur le réseau.

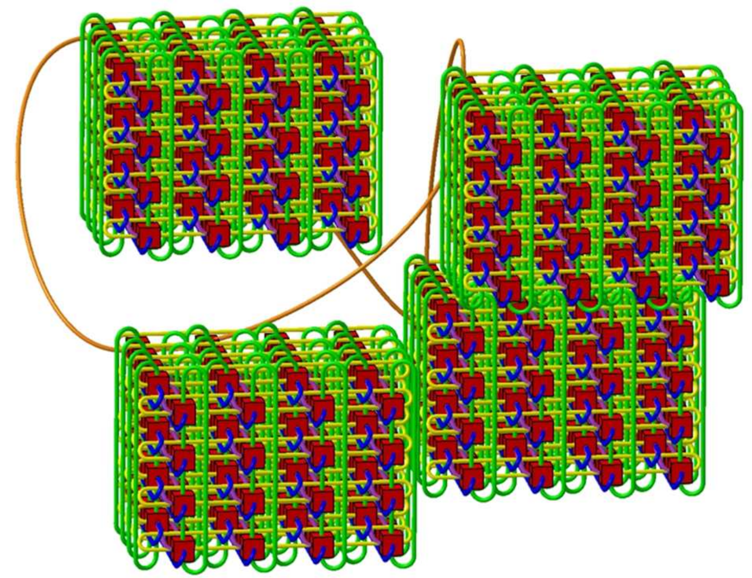
Topologie Réseau

Tore 3D (IBM BlueGene/P, Cray XT3, ...)



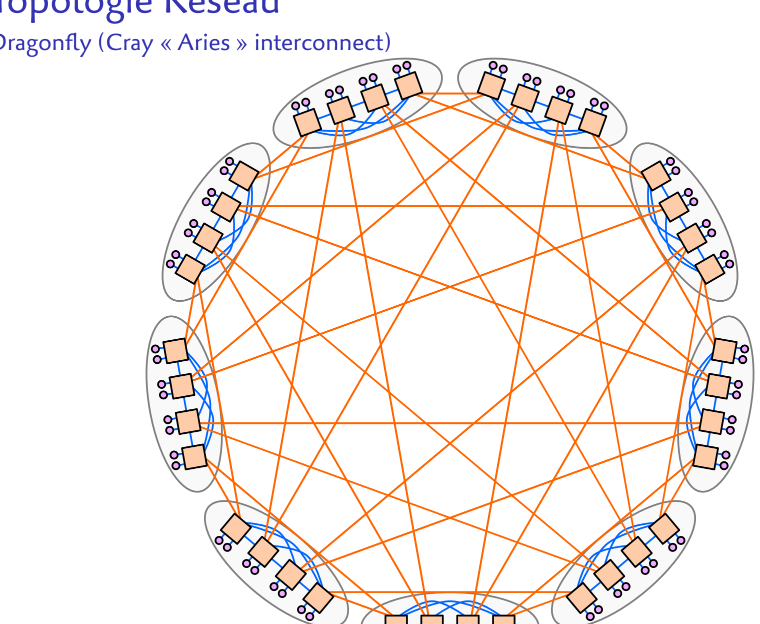
Topologie Réseau

Tore 5D (IBM BlueGene/Q)



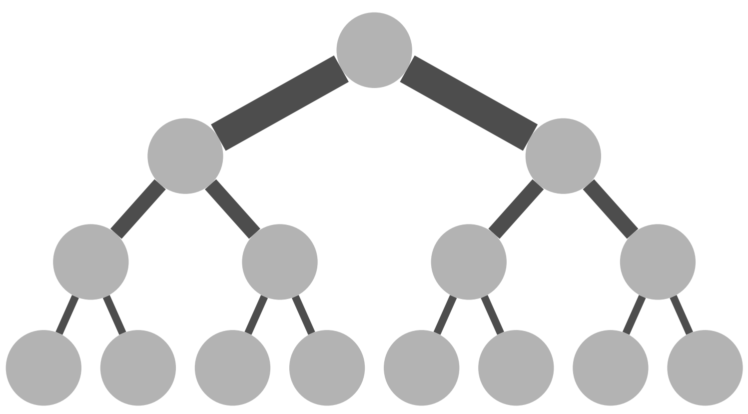
Topologie Réseau

Dragonfly (Cray « Aries » interconnect)



Topologie Réseau

Fat Tree (Infiniband)

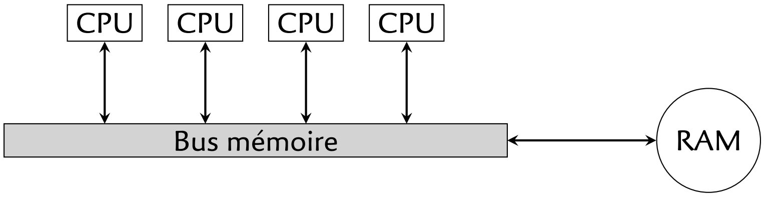


Topologie Réseau

Au final, ça ressemble à ça

Mémoire partagée

Classification selon l'organisation de la mémoire



- CPUs reliés à l'ensemble de la mémoire par un bus.
- Chacun peut accéder à l'intégralité de la mémoire.
- Communications $\leftrightarrow$ lecture/écriture en mémoire.

- Attention aux conflits d'accès à une même adresse !

Mémoire partagée

Classification selon l'organisation de la mémoire

Race condition :

Il y a *race condition* lorsque

- au moins 2 processeurs accèdent à la même variable
- au moins un processeur y accède en écriture
- ces accès sont potentiellement concurrents (ils peuvent être effectués « au même moment »)

```
i = i + 1;
```

Solution

mécanismes de synchronisation (section critique, opération atomique, barrière de synchronisation)

Non-Uniform Memory Access

Classification selon l'organisation de la mémoire

Mémoire partagée : problème de passage à l'échelle

Contention sur le bus mémoire...

Solution

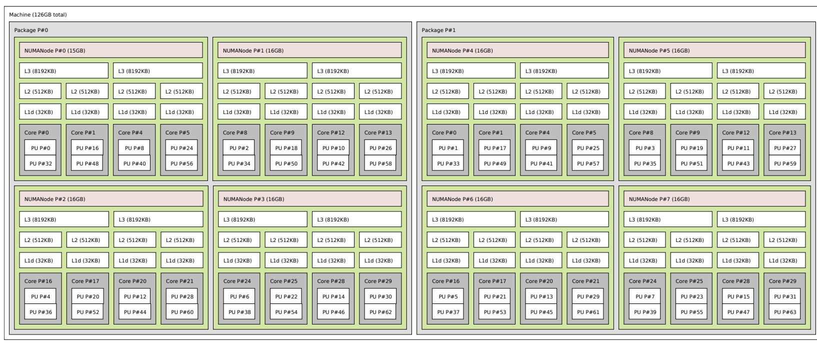
- Simuler mémoire partagée avec mémoire distribuée.
- Chaque CPU « contrôle » une partie de la mémoire.
  - Il y accède *directement*
- Pour accéder au reste de la mémoire, il faut passer par les autres CPUs.

$\Rightarrow$ Réseau *ad hoc*, très rapide.

Non-Uniform Memory Access

Un vrai exemple

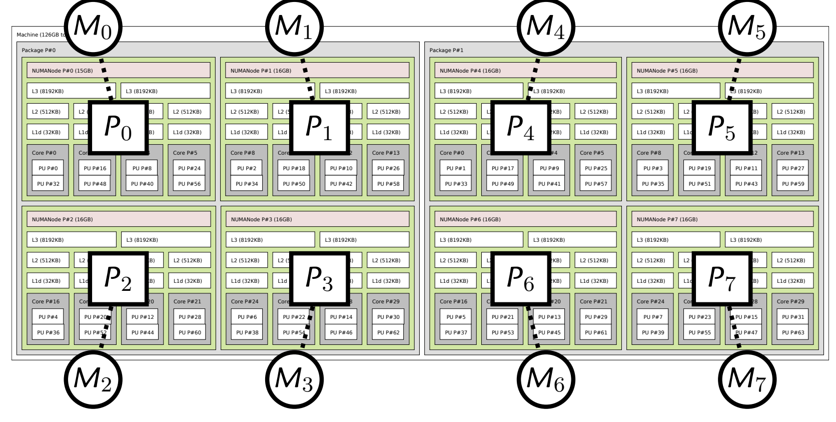
- un noeud SMP avec $2\times$ CPU AMD EYPC « Naples »
- 4 $\times$ contrôleur RAM indépendants par CPU



Non-Uniform Memory Access

Un vrai exemple

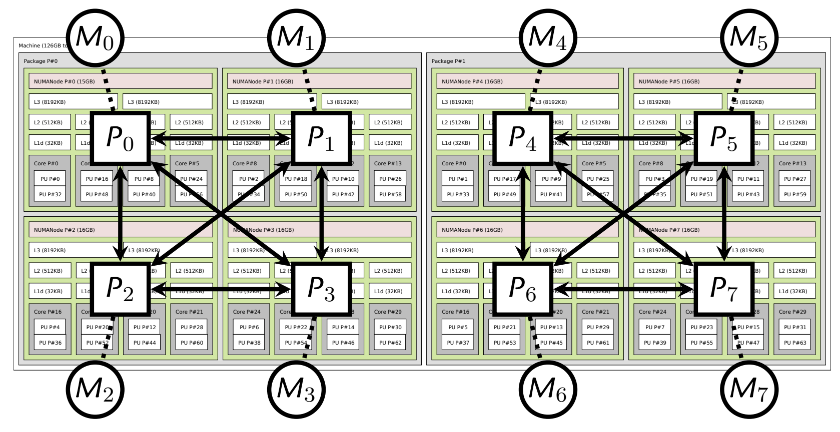
- un noeud SMP avec $2\times$ CPU AMD EYPC « Naples »
- $4\times$ contrôleur RAM indépendants par CPU



Non-Uniform Memory Access

Un vrai exemple

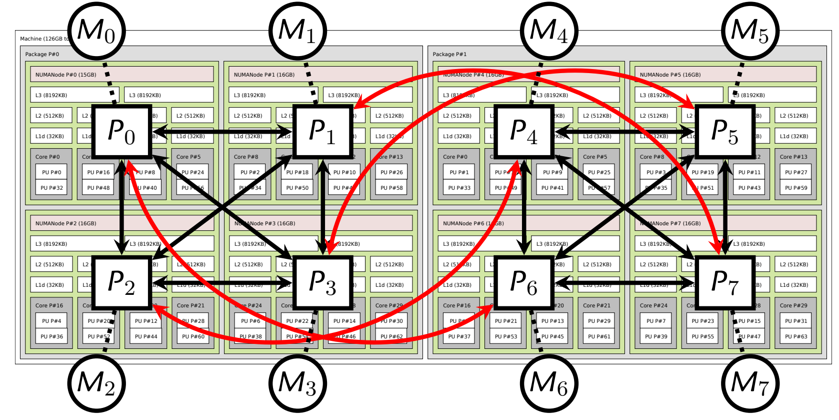
- un noeud SMP avec $2\times$ CPU AMD EYPC « Naples »
- 4 $\times$ contrôleur RAM indépendants par CPU
- Graphe complet à l'intérieur d'un CPU



Non-Uniform Memory Access

Un vrai exemple

- un noeud SMP avec $2\times$ CPU AMD EYPC « Naples »
- 4 $\times$ contrôleur RAM indépendants par CPU
- Graphe complet à l'intérieur d'un CPU
- Interconnection partielle entre les 2 sockets
- Lien rouge = $\approx 2\times$ plus de latence



En pratique...

- machine = noeuds reliés par un réseau
  - MIMD / mémoire *distribuée*
- Un noeud = plusieurs processeurs
  - Sûrement NUMA
- Dans un noeud = tous les coeurs ont accès à la RAM
  - Mémoire *partagée*
- Dans un coeur = instructions vectorielles
  - SIMD

Bref, on a droit à tout à la fois.

Instruction-Level Parallelism (ILP)

```
slwi 10,9,3
add 8,11,10
lwzx 10,11,10
lwz 7,4(8)
or. 10,10,7
bne 0,.L146
addi 5,5,8
stw 3,0(8)
stw 4,4(8)
cmplw 7,6,5
bne 7,.L24
lwz 9,144(19)
li 10,1
stw 10,20704(31)
addi 9,9,1
```

code = liste *ordonnée* d'instructions

Parallélisme au sein d'un coeur

- *Pipeline(s)*
- *Superscalarité*
- *Out-of-order execution*
- *Simultaneous Multi-Threading*

$\Rightarrow$ Mécanismes parallèles

$\Rightarrow$ Sémantique séquentielle

Parallélisme dans les applications

```
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        for (int k = 0; k < n; k++)
            C[i][j] += A[i][k] * B[k][j];
```

- Parallélisme de données
- parallélisme de contrôle
- Granularité
- Portions séquentielles

Évaluation des performances

Étudier le passage à l'échelle (*scalability*) d'un algorithme parallèle

Juge de paix : horloge murale.

- $T_1(n)$: temps nécessaire à l'exécution du **meilleur** algorithme séquentiel pour résoudre une instance de problème de taille n
- $T_p(n)$: temps nécessaire à l'exécution de l'algorithme parallèle considéré pour résoudre une instance de problème de taille n avec p processeurs.

Accélération, *speedup*

$$S(n,p) = \frac{T_1(n)}{T_p(n)}$$

Efficacité, *efficiency*

$$E(n,p) = \frac{S(n,p)}{p}$$

Strong scaling (« extensibilité forte »)

Objectif de la parallélisation

- Etude des performances en fonction de p avec n fixé.
- Résoudre un problème de taille fixe le plus vite possible.

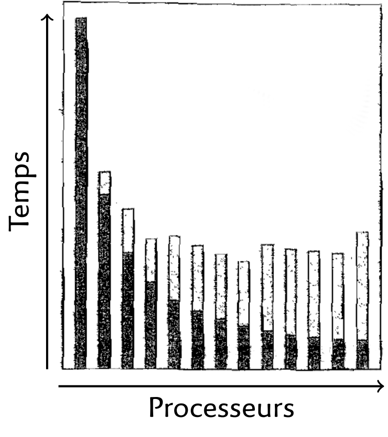
- **Accélération linéaire ($\rightarrow$ l'idéal) :** les processeurs sont occupés à 100%

$$S(n,p) = p \qquad E(n,p) = 1$$

- **Accélération sublinéaire :** les processeurs sont occupés à moins de 100%
- **Accélération supralinéaire :** difficile à envisager. Toutefois cela peut arriver si la mémoire est mieux utilisée (utilisation des caches), ou si l'on économise des calculs.

Strong scaling

Exemple d'une application non triviale

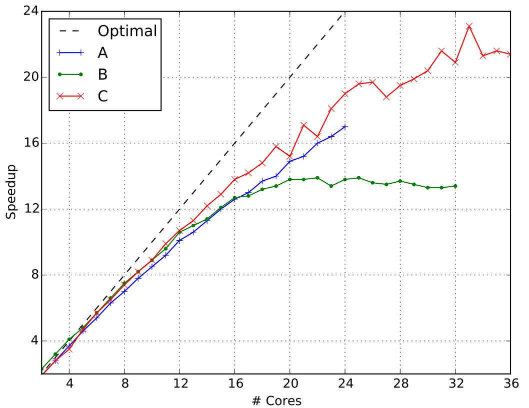


(d'après M.J. Quinn)

- en grisé : temps de calcul
  - décroît (ou stagne) lorsque $p \nearrow$
- en blanc : temps de communication
  - augmente lorsque $p \nearrow$
- Pour un problème donné (n fixé), il existe un nombre maximal de processeurs utilisables efficacement.
  - Au-delà, l'ajout de processeurs supplémentaires n'apporte plus de gain de performance.

Exemple avec un (vrai) programme multi-thread

Calcul de rang de matrices creuses



Name	CPU Type	# CPU	cores/CPU	L3 Cache/CPU
A	Xeon E5-2670 v3	$2\times$	12	30MB
B	Xeon E5-4620	$4\times$	8	16MB
C	Xeon E5-2695 v4	$2\times$	18	45MB

Loi d'Amdahl

Un algorithme dont une fraction f est non parallélisable. Alors :

$$S(n,p) \leq \frac{1}{f + (1-f)/p}$$

$\Rightarrow$ 20% d'un algo est séquentiel, l'accélération est limitée à 5.

(Sur)coût dû au parallélisme « $1 - E(n,p)$ » dû aux parties séquentielles et :

- démarrage et terminaison des tâches
- communications et/ou synchronisations
- qualité de l'équilibrage de charge (voir cours suivants)
- surcoûts logiciels dûs aux compilateurs, bibliothèques, outils, systèmes d'exploitation...

Weak scaling (« extensibilité faible »)

Objectif de la parallélisation

- Etude des performances quand n augmente avec p.
- Résoudre de plus gros problèmes en temps fixe.

Effet d'Amdahl (empirique)

- La fraction intrinsèquement séquentielle d'un algo diminue généralement quand $n \nearrow$
- Coûts de communication $\nearrow$ (car $p \nearrow$), mais *moins vite* que la quantité de calculs (car $n \nearrow$).

Loi de Gustafson-Barsis

On exécute un programme parallèle sur p processeurs.

On mesure que la fraction du temps d'exécution passé dans des portions intrinsèquement séquentielles est s.

Alors :

$$S(n,p) \leq p - (p-1)s$$

Exemple

Un calcul parallèle s'exécute sur 32 processeurs en 100 secondes, dont 5 secondes dans des parties séquentielles sur 1 seul processeur, alors $S(n,p) \leq 30.45$

Comment écrire des programmes parallèles ?

- Parallélisation automatique

Comment écrire des programmes parallèles ?

- Parallélisation automatique
- « Annotations » de parallélisme dans des programmes C

OpenMP

```
#pragma omp parallel for
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        for (int k = 0; k < n; k++)
            C[i][j] += A[i][k] * B[k][j];
```

Comment écrire des programmes parallèles ?

- Parallélisation automatique
- « Annotations » de parallélisme dans des programmes C
- plain C + bibliothèques (`pthread`, MPI)

Librairies

```
 for(int t=0; t<NUM_THREADS; t++)
      pthread_create(&threads[t], NULL, ThreadFunction, (void *) t);
...
pthread_exit();
```

Comment écrire des programmes parallèles ?

- Parallélisation automatique
- « Annotations » de parallélisme dans des programmes C
- plain C + bibliothèques (`pthread`, MPI)
- Nouveaux langages ?

Nouveaux langages (Go)

```
func main() {
    s := []int{7, 2, 8, -9, 4, 0}
    c := make(chan int)
    go sum(s[:len(s)/2], c)
    go sum(s[len(s)/2:], c)
    x, y := <-c, <-c
    fmt.Println(x, y, x+y)
}
```

Comment écrire des programmes parallèles ?

- Parallélisation automatique
- « Annotations » de parallélisme dans des programmes C
- plain C + bibliothèques (`pthread`, MPI)
- Nouveaux langages ?
- Paradigme répandu (et pratique) : SPMD

Single Program Mutiple Data : Data Parallelism

```
int i = <<rank>>;
for (int j = 0; j < n; j++)
    for (int k = 0; k < n; k++)
        C[i][j] += A[i][k] * B[k][j];
```

Comment écrire des programmes parallèles ?

- Parallélisation automatique
- « Annotations » de parallélisme dans des programmes C
- plain C + bibliothèques (`pthread`, MPI)
- Nouveaux langages ?
- Paradigme répandu (et pratique) : SPMD

Single Program Mutiple Data : Control Parallelism

```
if (<<rank>> == 0) {
  <<Portion purement séquentielle>>
} else {
  <<Attend>>
}
```