Programmation parallèle (PPAR)
Cours 1 : introduction au parallélisme

P. Fortin
pierre.fortin @ upmc.fr

D'après les cours de J.-L. Lamotte et C. Denis

Master 2 Informatique - UPMC

Motivations du parallélisme

Faire travailler simultanément plusieurs processeurs pour résoudre un même problème afin :

- de diminuer le temps de calcul,
- d'augmenter la taille du problème traité.

Dû aux besoins en calcul toujours croissants de la recherche et de l'industrie.

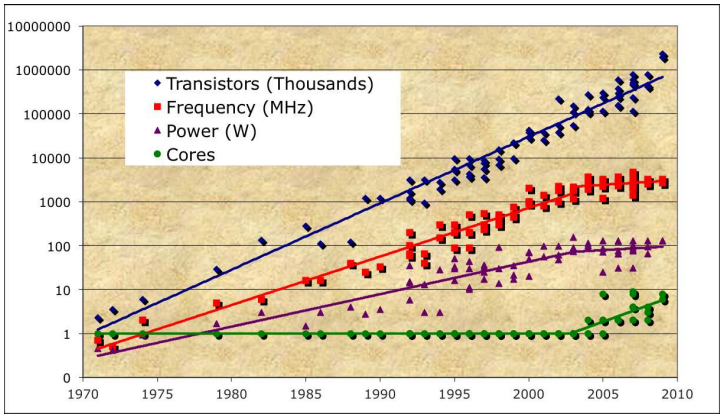
Différences entre parallélisme et

- calcul distribué : hétérogénéité, indépendance totale entre les différentes tâches, mode client-serveur, tolérance aux pannes... Exemple : le projet `SETI@home`
- calcul concurrent : compétition entre processus, problématiques de partage de ressources (système d'exploitation)

Motivations matérielles

Loi de Moore (empirique) :

≪ le nombre de transistors double tous les 2 ans ≫



CS267/E233, Applications of Parallel Computers, K. Yelick and J. Demmel, U.C. Berkeley

- Jusqu'en 2004-2005 : augmentation de la fréquence d'horloge des processeurs ⇒ augmentation des performances

Motivations matérielles (suite)

Depuis 2004-2005 :

- multiplication du nombre de cœurs (*cores*)

Même avant 2004 :

- course à la puissance de calcul :
  Puissance machine parallèle >> puissance machine séquentielle
- coût : une machine parallèle présente généralement un rapport coût/performance très avantageux

Remarque : les cœurs ≪ séquentiels ≫actuels sont *superscalaires* → utilisation de plusieurs pipelines d'instructions en parallèle

Course à la puissance de calcul justifiée : *1 calcul lancé il y a 10 ans sur une machine téraflopique (1 Tflops =* 10^{12} *opérations flottantes par seconde) et nécessitant 20 ans de calcul → aujourd'hui :*

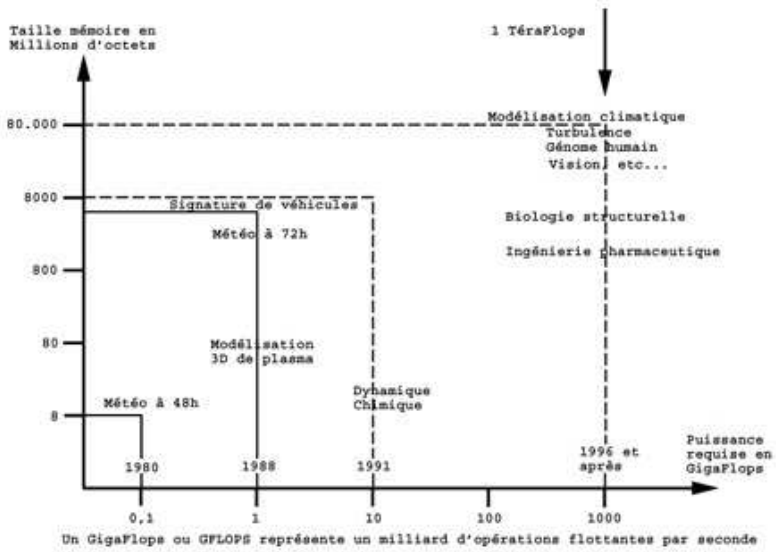
- *encore besoin de 10 ans sur la machine téraflopique ;*
- *temps de calcul total sur une machine pétaflopique (*10^{15} *opérations flottantes par seconde) :* < 8 *jours !*

Les grands domaines d'application du parallélisme

- Chimie, biologie : dynamique moléculaire
- Astrophysique : modélisation de la dynamique des galaxies
- Modélisation océan-atmosphère-climat
- Météorologie
- Mécanique des fluides et des structures
- Géophysique : analyse de la propagation d'ondes sismiques dans le sous-sol
- Bio-informatique : séquençage du génome
- Moteurs de recherche
- ...

Les ordres de grandeur

Giga (10^9), Tera (10^{12}), Peta (10^{15}), Exa (10^{18}) . . .
FLOPS : opérations en virgule flottante par seconde



Source : www.crihan.fr

Un exemple

- Soit le programme

```
Pour i variant de 1 à n faire
   a[i] = b[i] + c[i]
FinPour
```

- Les instructions $a[k] = b[k] + c[k]$ avec $1 \leq k \leq n$ **sont indépendantes**.
- Elles peuvent être exécutées dans un ordre quelconque donc simultanément.

Architectures parallèles

Pour fonctionner, une machine parallèle pose à son utilisateur un certain nombre de problèmes concernant :

Les éléments de calcul

- Combien de processeurs ?
- De quelle puissance ? Sont-ils homogènes ?
- La taille de la mémoire associée ?

Les communications

- Comment les processeurs sont-ils reliés les uns aux autres ?
- Leur protocole d'échange d'information ?
- Comment synchronisent-ils leurs efforts ?

Problèmes à résoudre

- Problèmes au fort potentiel de parallélisme ?
  - tâches indépendantes (applications trivialement parallèles)
  - tâches moyennent couplées (parallélisme à gros grain)
  - tâches fortement couplées (parallélisme à grain fin)
  - ou même algorithme intrinsèquement séquentiel . . .
- Degré de spécialisation des machines pour un problème donné ?
- Quels algorithmes ?
- Quels langages ?
- Quelle efficacité espérer ?
- Quelle expression pour le parallélisme : explicite ou implicite ?
  - explicite : MPI, PVM (Parallel Virtual Machine), threads POSIX. . .
  - (partiellement) implicite : OpenMP, HPF (High Performance Fortran). . .

Plan du cours

Les machines parallèles

Classification de Flynn (1966) :

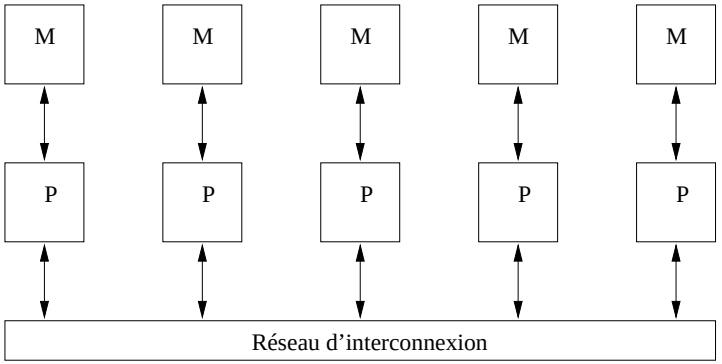
		Flot de données	
		unique	multiple
Flot d'instructions	unique	**SISD**	**SIMD**
	multiple	**MISD**	**MIMD**

- **machines SISD**
  Ce sont les machines séquentielles !
- **machines MISD**
  Chaque processeur reçoit des instructions distinctes opérant sur le même flot de données
  - la sortie d'une unité de traitement devient l'entrée d'une autre unité
  - analogie avec un pipeline

Classification de Flynn (suite)

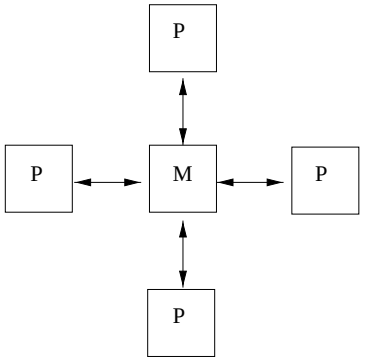
- **machines SIMD**
  Les unités de traitement exécutent simultanément la même opération sur leurs données propres.
  - fonctionnement synchrone
  - une seule unité de contrôle centralisée
  - exemples :
    - machines vectorielles (CRAY , NEC . . .), Connection Machine (CM1, CM2, CM200). . .
    - instructions SSE, AVX . . .(Intel, AMD . . .)
    - Graphics Processing Units (GPUs), processeur Cell . . .
- **machines MIMD**
  Les processeurs peuvent effectuer différentes opérations sur différentes données simultanément.
  - fonctionnement asynchrone
  - exemples :
    - IBM SP, IBM BlueGene, CM5, Cray T3D/E . . .
    - grappe (*cluster*) de PC

Classification selon l'organisation de la mémoire : **DM (Distributed Memory)**



- Chaque processeur dispose d'une mémoire locale qui lui est réservée (espace d'adressage local).
- Le réseau relie les processeurs entre eux.
- La communication entre processeurs se fait par échange de messages à travers le réseau d'interconnexion.

Classification selon l'organisation de la mémoire : **SM (Shared Memory)**



- Le (ou les) banc(s) mémoire est (sont) séparé(s) des processeurs. Le bus relie l'ensemble des processeurs à l'ensemble des bancs mémoire.
- Tout processeur peut accéder à l'intégralité de la mémoire (espace d'adressage global).
- La communication entre processeurs se fait par le biais de la mémoire.

- Attention aux conflits d'accès à un même emplacement mémoire !

Classification selon l'organisation de la mémoire (suite)

Race condition :

Il y a *race condition* lorsque

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

Exemple : `i = i + 1;`

DSM (Distributed Shared Memory) :

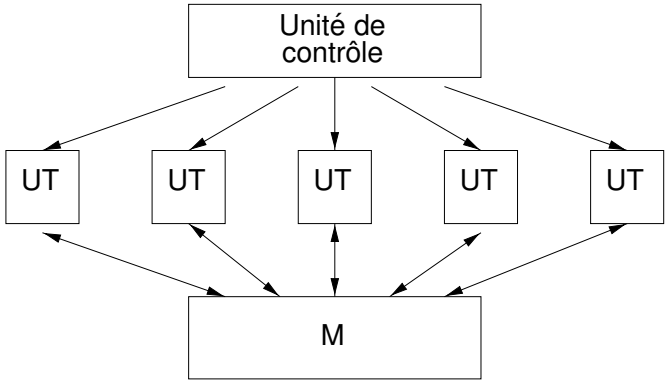
- Mémoire physiquement distribuée mais ≪ vue ≫ comme une mémoire unique (partagée).
- Attention : les temps d'accès varient fortement selon l'emplacement en mémoire !
- Sans une stratégie d'allocation mémoire adaptée, les performances s'écroulent. . .

Classification unifiée

- Le contrôle des instructions est centralisé (SI) ou distribué (MI).
- Les données sont toujours multiples (MD).

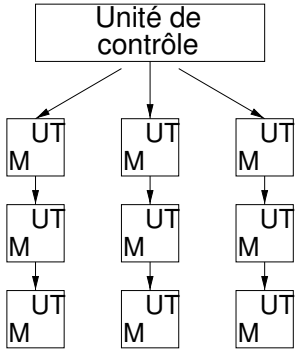
		Contrôle	
		centralisé	distribué
Mémoire	centralisée	**SIMD-SM**	**MIMD-SM**
	distribuée	**SIMD-DM**	**MIMD-DM**

Architecture SIMD-SM



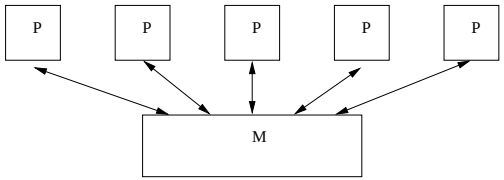
- UT : unité de traitement
- Machines vectorielles mono-processeur.
- Une unique instruction s'applique à des données multiples.

Architecture SIMD-DM



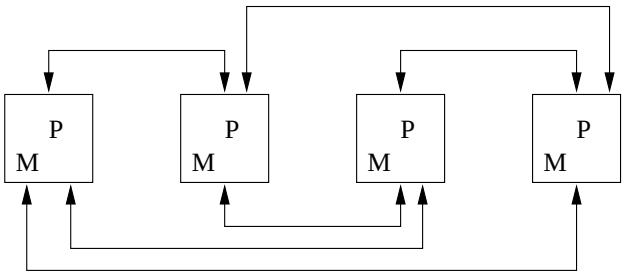
- Processeurs de faible puissance en grand nombre.
- Tous les processeurs exécutent la même instruction de manière synchrone.
- Ex : CM2 (jusqu'à 65536 processeurs élémentaires 1 bit)

Architecture MIMD-SM



- Machines UMA (*Uniform Memory Access*)
  - exemple : machines SMP (*Symmetric MultiProcessors*, avec une mémoire physique centralisée)
  - la complexité du bus d'interconnexion augmente rapidement avec le nombre de processeurs ⇒ limité à quelques dizaines de processeurs (32 par ex.)
- Machines NUMA (*Non-Uniform Memory Access*) basées sur une mémoire de type DSM
  - permet de regrouper plus de processeurs
  - attention aux temps d'accès à des zones mémoire ≪ lointaines ≫
  - exemple : SGI Origin 2000, machines avec plusieurs processeurs multicoeurs (effets NUMA entre les différents processeurs)
- Généralement avec cohérence de cache (*cache-coherent*) : cc-UMA et cc-NUMA

Architecture MIMD-DM



- Graphe complet ou partiel d'interconnexion.
  Topologies : anneau, grille, cube, hypercube, arbre, étoile (avec switch), tore 2D. . .
- Grand nombre de processeurs (éventuellement hétérogènes) de grande puissance.
- Les processeurs communiquent entre eux par envoi de messages.
- Les performances des applications sont liées aux performances en communication des machines (⇒ systèmes d'exploitation et environnements propres aux machines.)

Architectures MIMD hybrides

- Architectures MIMD combinant SM et DM
- Exemple : interconnecter plusieurs machines SMP
  $\rightarrow$ grappe (*cluster*) de nœuds SMP
  (par exemple : grappe de nœuds IBM p575)
- Permet de combiner les avantages des deux organisations mémoires (via une programmation hybride MPI-thread) :
  - mémoire partagée : confort de programmation grâce à l'adressage mémoire unique
  - mémoire distribuée : plus grand nombre de processeurs

Exemple de pipeline

- Analogie avec l'architecture MISD
- Exemple : la multiplication terme à terme de deux vecteurs de nombres en virgule flottante.
  Chaque multiplication $x(i) * y(i)$ est décomposée en plusieurs étapes (étages) indépendantes :
  - chargement de l'instruction
  - décodage de l'instruction
  - exécution de l'instruction :
    - addition des exposants,
    - normalisation des mantisses et ajustement des exposants,
    - multiplication des mantisses,
    - normalisation des mantisses et ajustement des exposants.
  - écriture du résultat dans un registre

  $\rightarrow$ En pratique : 4 cycles d'horloge pour la multiplication flottante sur CPU x86.

Le pipeline

Soit un opérateur $f_{n,1} = f_n \circ ... \circ f_2 \circ f_1$
n : profondeur du pipeline

		$\Rightarrow$	f_1	f_2	f_3	... f_n	$\Rightarrow$
d_3	d_2	d_1					
d_4	d_3	d_2	d_1				
d_5	d_4	d_3	d_2	$f_1(d_1)$			
d_6	d_5	d_4	d_3	$f_1(d_2)$	$f_2 \circ f_1(d_1)$		
				...			
d_{n+3}	d_{n+2}	d_{n+1}	d_n	$f_1(d_{n-1})$	...	$f_{n-1,1}(d_1)$	
d_{n+4}	d_{n+3}	d_{n+2}	d_{n+1}		...		$f_{n,1}(d_1)$

Temps d'exécution

- Soit T le temps nécessaire à l'exécution de l'opérateur $f_{n,1}$.
- Soit T_i le temps passé à l'étage i du pipeline : $\sum_i T_i = T$.
- On cadence tous les étages sur la base du plus lent des opérateurs, soit $T' = \max_i T_i$.
  $f_{n,1}(d_1)$ est disponible au temps $t = nT'$.

$\Rightarrow$ $(n-1)T'$ est le temps d'amorce du pipeline.

$\Rightarrow$ À partir du temps $t = nT'$, un résultat est disponible toutes les T' unités de temps.

$\Rightarrow$ Si l'on traite k données, le dernier résultat est disponible au temps $(k+n-1)T'$.

Facteur d'accélération du pipeline

Définition

On appelle facteur d'accélération la quantité :

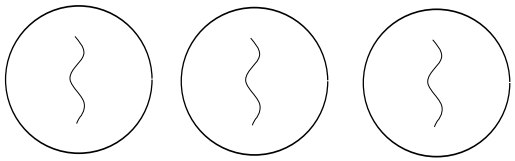
$$\mathcal{F} = \frac{\textit{temps sans pipeline}}{\textit{temps avec pipeline}}$$

- $\mathcal{F} = \frac{kT}{(k+n-1)T'}$
- Si $T' = \frac{T}{n}$, facteur d'accélération $= \frac{n}{1+\frac{(n-1)}{k}}$.
- Lorsque $k \to \infty$, le facteur d'accélération $\to n$, ce qui correspond au parallélisme maximum.

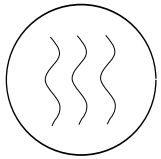
Notions de processus et de thread

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

Modèle de programmation

En mode MIMD :

- SPMD : *Single Process, Multiple Data*
  ou *Single Program, Multiple Data*
  Principe :
  - un unique code source
  - chaque processus/thread possède un identifiant (numéro, rang)
  - détermination du travail à effectuer en fonction de l'identifiant
- MPMD : *Multiple Program Multiple Data*
  Exemple : modèle client/serveur

Modèle de communication par :

- envoi de messages $\to$ MPI, PVM...
- mémoire partagée $\to$ OpenMP, threads POSIX...

Évaluation des performances

But : étudier le passage à l'échelle (ou extensibilité, ou *scalability* en anglais) d'un algorithme parallèle.

A savoir : l'algorithme (ou le programme) est-il (ou reste-t-il) efficace lorsque le nombre de processeurs augmente ?

- $T_1(n)$: temps nécessaire à l'exécution du **meilleur** algorithme séquentiel pour résoudre une instance de problème de taille n avec 1 processeur.
- $T_p(n)$: temps nécessaire à l'exécution de l'algorithme parallèle considéré pour résoudre une instance de problème de taille n avec p processeurs.

Définition (**accélération**, *speedup*)

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

Définition (**efficacité**, *efficiency*)

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

Strong scaling

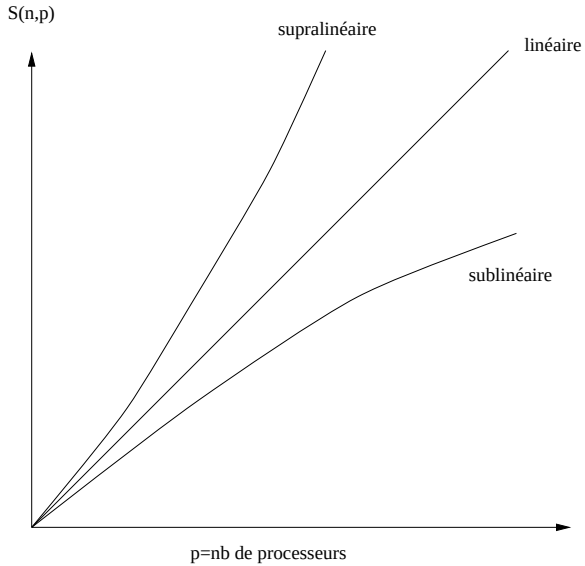
Etude des performances en fonction de p avec n fixé : ≪ *strong scaling* ≫

- **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.
- Pour un problème donné (à 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.

Étude graphique



Exemple

- Soit le programme séquentiel :

```
pour i= 0, 99
     a[i]=3*(i+2)
```

- Complexité : 500 instructions (incrément de l'indice de boucle, test, multiplication, addition et affectation)
- Code pour une machine parallèle SM à 10 processeurs :

```
mon_indice = indice_proc() /* coût appel = 1 */
nb = 100 / nb_procs()      /* coût appel = 1 */
/* 0 <= mon_indice < nb_procs() */
debut = mon_indice * nb
fin   = debut + nb - 1
pour i= debut, fin
     a[i]=3*(i+2)
```

- Chaque processeur : $2+3+2+3+((100/10)*5) = 60$ instructions
- Accélération : $8,33$ ($= \frac{500}{60}$) Efficacité : $0,83$
- Attention : pas de prise en compte du nombre de cycles par instruction $\Rightarrow$ en pratique : mesures de temps de calcul

Loi d'Amdahl

Soit f la fraction intrinsèquement séquentielle (non parallélisable) de l'algorithme à paralléliser. Alors l'accélération maximale sur p processeurs de l'algorithme parallèle correspondant est :

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

Exemple : 20% d'un algorithme n'est pas parallélisable, l'accélération est donc limitée à 5.

En pratique, le (sur)coût dû au parallélisme ≪ $1 - E(n,p)$ ≫ est dû aux parties séquentielles, mais aussi aux aspects suivants :

- démarrage et terminaison des tâches
- synchronisation
- communications
- surcoûts logiciels dûs aux compilateurs, bibliothèques, outils, systèmes d'exploitation...

weak scaling et loi de Gustafson-Barsis

En pratique, on ne travaille pas forcément avec un problème de taille fixée → cas du ≪ *weak scaling* ≫ :

- faire croître n et p ensemble
- objectif : traiter de plus gros problèmes tout en maintenant un temps d'exécution constant
- généralement, la fraction intrinsèquement séquentielle d'un algorithme séquentiel diminue lorsque n croît

Loi de Gustafson-Barsis :

Soit s la fraction intrinsèquement séquentielle d'un algorithme *parallèle* utilisant p processeurs. Sous l'hypothèse que la quantité totale de travail à effectuer en parallèle est proportionnelle à p (*weak-scaling*), l'accélération maximale de cet algorithme parallèle est :

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

Exemple : un calcul parallèle s'exécute sur 32 processeurs en 100 secondes, dont 5 secondes dans des parties séquentielles sur 1 seul processeur, alors $S(n,p) \leq 30,45$

Petit historique des machines parallèles

Les années 70

- Première machine parallèle : ILLIAC IV
  - composition : 64 processeurs avec leur propre mémoire, un réseau de communication en tore 2D, un ou plusieurs séquenceurs de contrôle
  - $>$ 10 ans de développement (1964 - 1976)
- Les grands calculateurs sont des machines vectorielles monoprocesseurs.
- Les premières machines multiprocesseurs apparaissent à la fin des années 70.

Les années 80

Les machines vectorielles

- Exemple : les machines CRAY vectorielles.
- Rapport coût/performance peu intéressant.

Les machines synchrones

- La Connection Machine 2 (CM2) :
  - entre 4096 et 65536 micro-processeurs de 1 à 4 bits,
  - petite mémoire locale et un séquenceur unique

Les machines parallèles à mémoires distribuées

- Micro-processeurs ordinaires indépendants
- Mémoire locale généralement plus importante
- Réseau de communication
- Exemple : Transputer, Paragon. . .

Les années 90

- Abandon progressif mais très lent des machines vectorielles.
- Âge d'or des machines MIMD-DM.
- Apparition des architectures basées sur des machines MIMD-SM (SMP) et des architectures de type grappe.

Les années 2000

- La machine parallèle du pauvre : réseaux PC connectés en Fast Ethernet (100 Mbits/s) ou Gigabit Ethernet (1 GBit/s).
- Grappes de nœuds SMP → exemple du White ASCI (2001-2006) :

- 8192 processeurs (512 nœuds de 16 processeurs)
- performance : 13 Tflops

Années 2000 et tendances actuelles (suite)

- Grappes de nœuds SMP (suite) : exemple de Jaguar (Cray)
  - noeuds SMP de 2 processeurs quad-cœur : 224162 cœurs au total
  - performance : 2,331 Pflops
  - #1 au TOP 500 de novembre 2009 (`www.top500.org`)
  - électricité : 1 million de dollars/an
- Architectures massivement parallèles : « plus de processeurs moins rapides »⟶ exemple de l'IBM BlueGene/L :
  - #1 au TOP 500 de novembre 2007
  - 106 496 nœuds de 2 processeurs PowerPC 440 (700 MHz, 2.8 GFlops)
  - 596 Tflops de performance crête

Années 2000 et tendances actuelles (suite)

- Les grilles de calcul : déploiement d'applications sur plusieurs centres de calculs dispersés géographiquement.
  Exemples : Globus, EGEE, Grid'5000 . . .
- A l'échelle d'internet (calcul distribué) : projets semblables à `SETI@home`
- Emergence des architectures multi-cœur :
  - hier : 2 ou 4 cœurs
  - actuellement : 6 ou 12 cœurs
  - bientôt : 16 cœurs . . .
  - Et aussi : *Simultaneous multithreading* (SMT, ou *Hyper-Threading* chez Intel) à 2 ou 4 voies : à chaque cycle, des instructions de threads différents peuvent être exécutées en même temps sur le processeur (superscalaire).
    Exemple : 1 thread → chargement mémoire, 1 thread → calcul
  - Attention aux effets "NUMA" avec plusieurs processeurs multicoeurs

Années 2000 et tendances actuelles (suite)

- Technologies accélératrices matérielles : GPU, processeur Cell, FPGA (*field-programmable gate array*), cartes spécialisées pour le calcul . . .

  Exemple de Titan : #1 au TOP 500 de novembre 2012
  - 18688 noeuds avec chacun : 1 processeur 16-coeur AMD et 1 GPU NVIDIA K20
  - 27,112 Pflops au total

  Exemple de Curie (CEA) :
  - partie "Curie noeuds hybrides" : 144 noeuds avec chacun 2 processeurs Intel et 2 GPU NVIDIA M2090, soit 192 Tflops au total
  - partie "Curie Thin nodes" : 5040 noeuds avec chacun 2 processeurs 8-coeurs Intel, soit au total 80640 coeurs et 1,667 Pflops (#11 TOP 500 novembre 2012)
- Green500 (`www.green500.org`) : basé sur les MFLOPS/W
  - 10 premiers de novembre 2012 : nœuds CPU + { GPU (NVIDIA, AMD), Intel Xeon Phi } ou IBM BlueGene/Q