Serveurs de calcul : contexte, enjeux et administration

Violaine Louvet, 13 septembre 2007

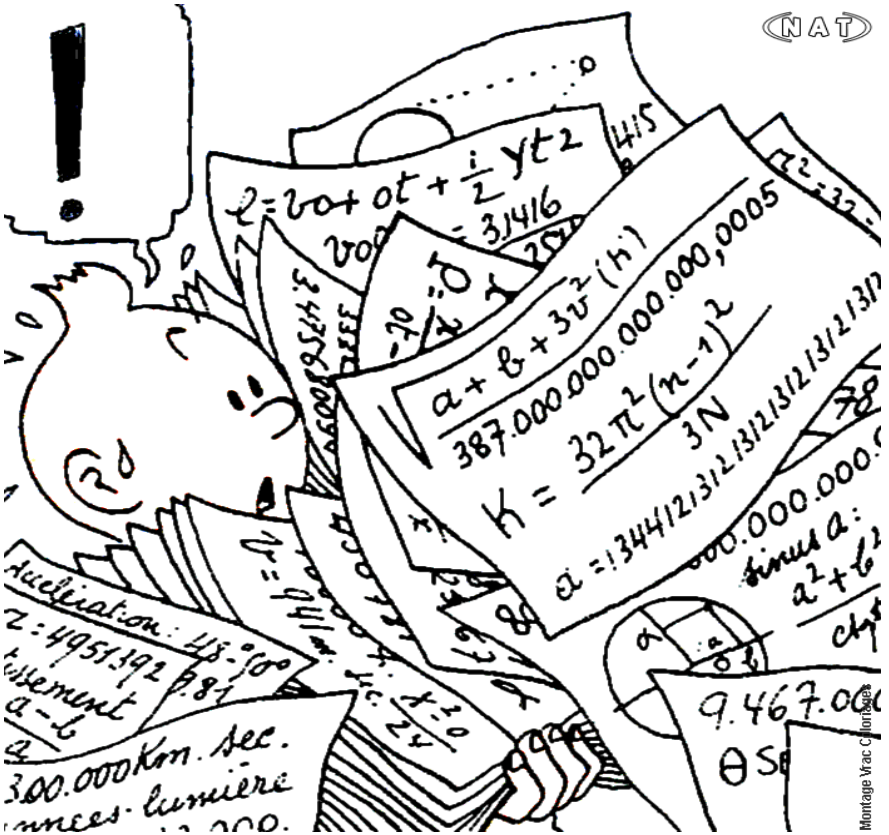
Calcul scientifique et serveurs de calcul

Journée proposée par :

- Le Groupe Calcul (http://calcul.math.cnrs.fr/)
- La Fédération des réseaux (métiers) d'Administrateurs Systèmes et Réseaux (ASR) ResInfo (http://www.urec.cnrs.fr/rubrique155.html)

Remerciements à :

- L'équipe vidéo du CC IN2P3,
- l'UREC
- l'IPNL



Deux mots sur le Groupe Calcul

- Un réseau métier de communication et d'échanges pour la communauté du Calcul en France.

- Une liste de diffusion regroupant à ce jour plus de 530 personnes.

- Un site web collaboratif http://calcul.math.cnrs.fr/ comprenant :
  - Des informations sur les activités du Groupe : formation, journées scientifiques, interventions dans des congrès ...
  - Un recensement (non exhaustif) des moyens de calcul en France : à compléter et à actualiser
  - Des actualités liées au Calcul ...

- L'organisation de journées de formation et d'information : licences et gestion de projets, compilateurs, C++ (STL et librairies), Python ...

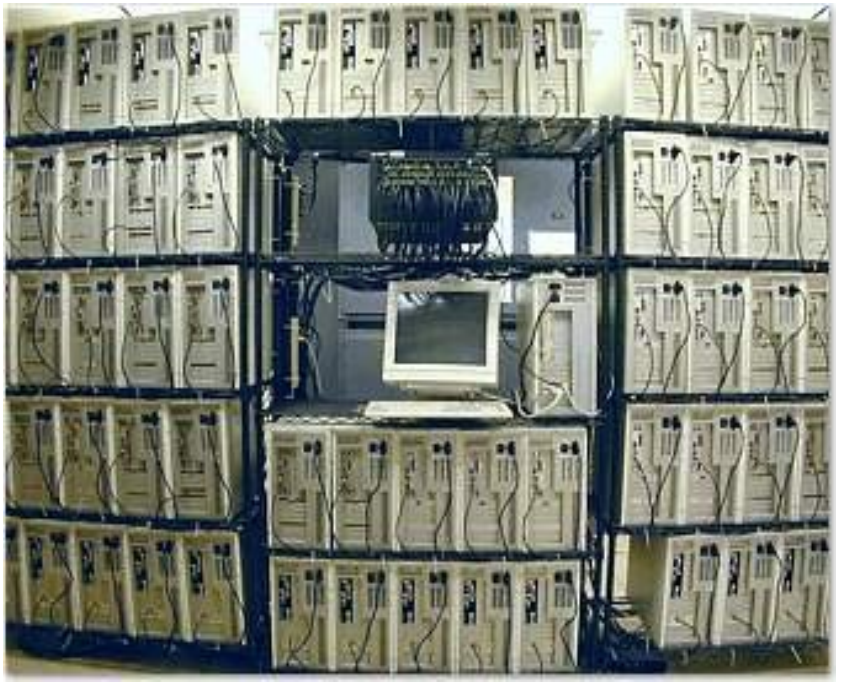
- La participation à des journées et congrès scientifiques : CANUM ...

- L'implication dans les logiciels libres par son partenariat avec le Projet Plume http://www.urec.cnrs.fr/rubrique251.html

- Des actions pour la valorisation de l'activités de développement de code par le projet CIEL (Code Informatique En Ligne) http://ciel.ccsd.cnrs.fr/

Programme

- Calcul paralléle : pourquoi, comment ?
- Quelles architectures et quel types de matériels ?
- Administration : outils systèmes et distributions

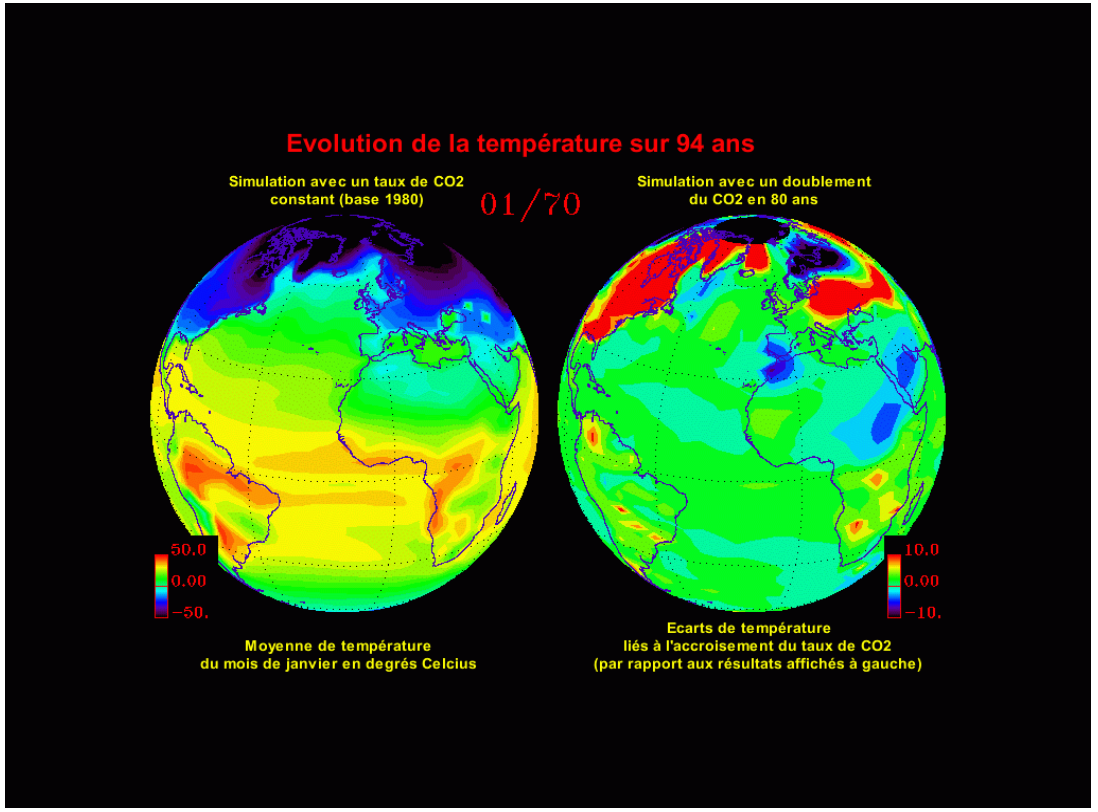


- Exploitation et Optimisation des ressources
- Grilles de calcul
- Retours d'expériences :
  - Achats
  - OpenMosix et OpenSSI
  - Torque/Maui
  - Moyens de calcul du CNES
  - Cluster de bi-opterons/réseau infiniband

Introduction

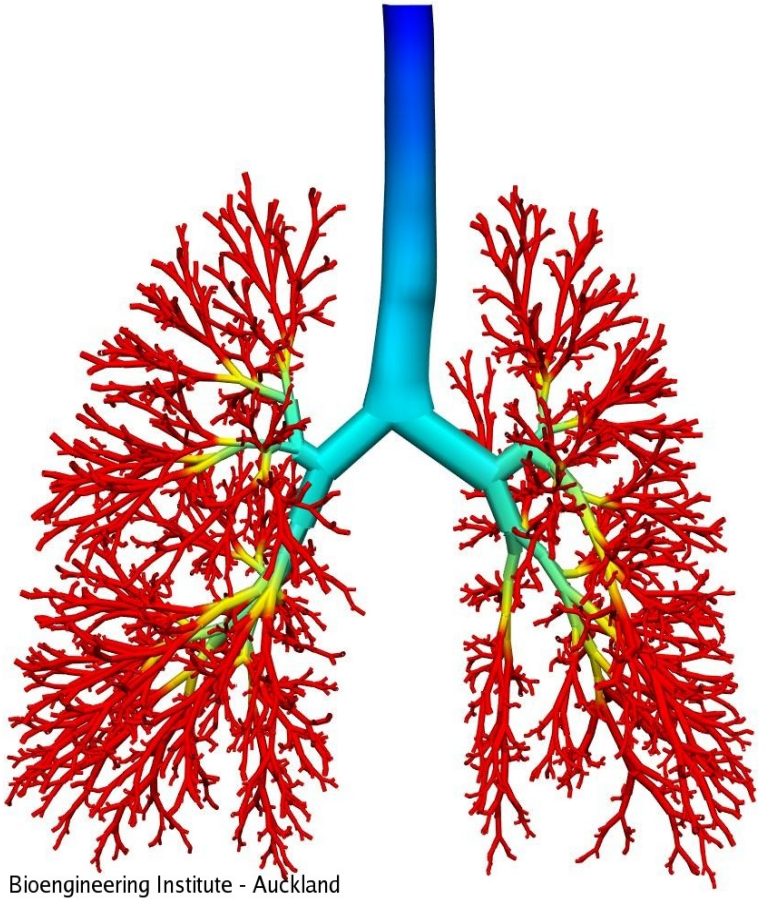
Grands challenges en calcul scientifique

- Simulations de mécanique des fluides : aéronautique, automobile, aérospatiale, météorologie, climat, prospection pétrolière ...



Source : Institut Simon Laplace

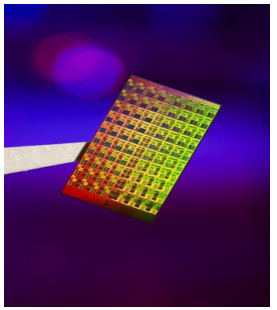
- Chimie, énergétique, biologie : catalyse chimique, combustion, plasmas, nouvelles thérapeutiques ...



Bioengineering Institute - Auckland

Besoins et solutions

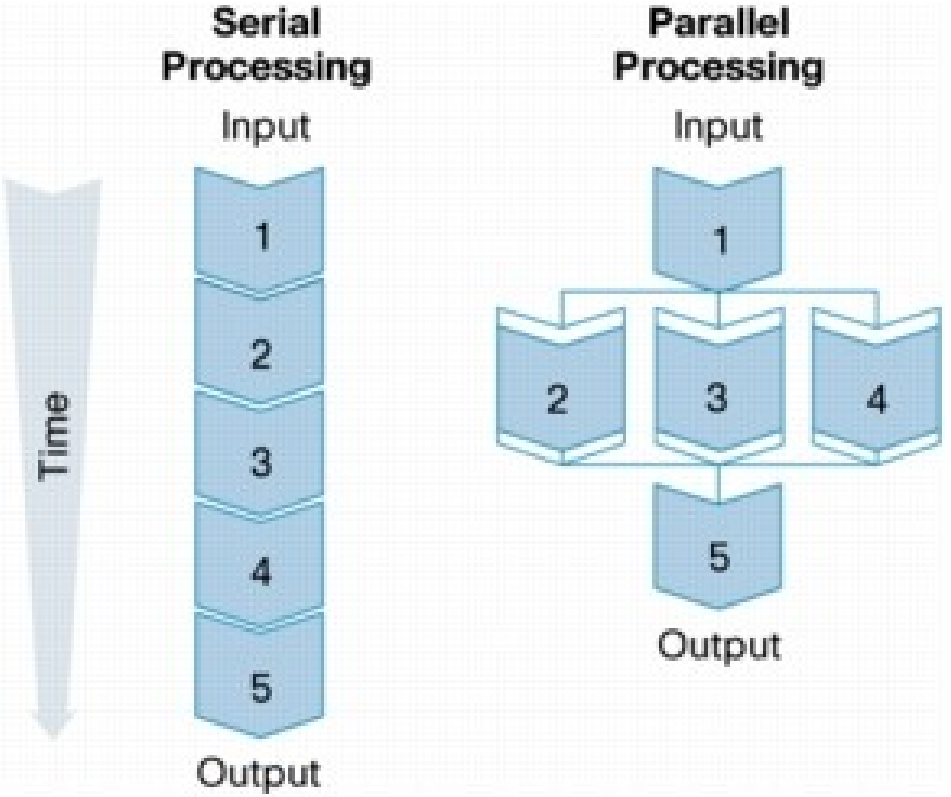
Quels besoins ?



- Besoins importants :
  - en puissance de calcul
  - en mémoire

- Ordres de grandeur : (G : giga, T : téra, P : péta)
  - 1 TeraFLOPs = $2^{40} \approx 1.000.000.000.000$ (un billion) opérations flottantes par seconde
  - 1 PetaByte : vidéo de 2300 ans, 1 milliard de livres...

Quels types de programmation ?



- **Programmation séquentielle**
  - Un seul flot d'exécution
  - Une instruction exécutée à la fois
  - Un processeur

- **Programmation parallèle**
  - Plusieurs flots d'exécution
  - Plusieurs instructions exécutées simultanément
  - Plusieurs processeurs

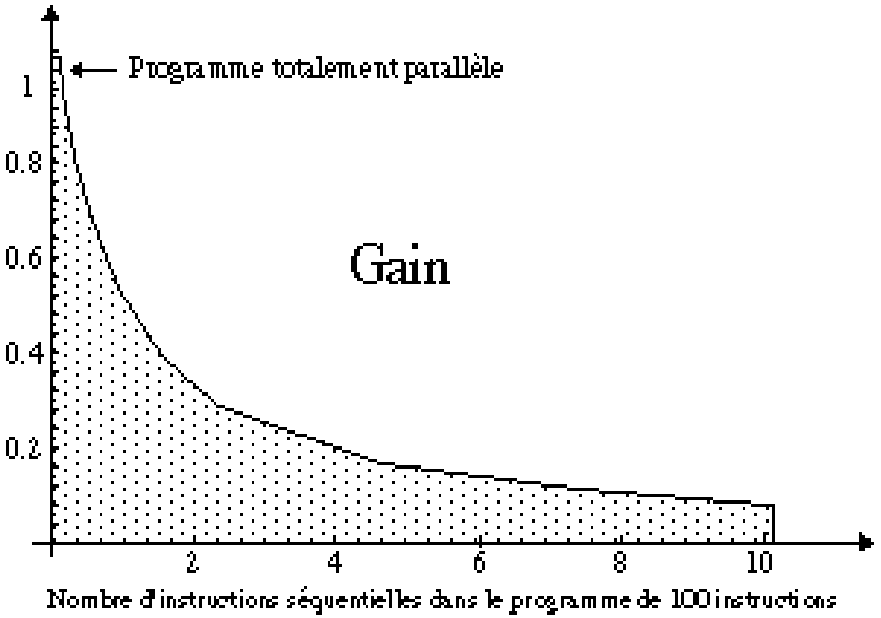
Pourquoi utiliser le parallélisme ?

Quels intérêts ?

- Rapidité : pour *N* processeurs, temps de calcul divisé par *N* en théorie
- Taille mémoire : pour *N* processeurs, on dispose de *N* fois plus de mémoire (en général, selon le type de machine)
- Résoudre des problèmes de plus grande taille ou plus complexes

Quels contraintes ?

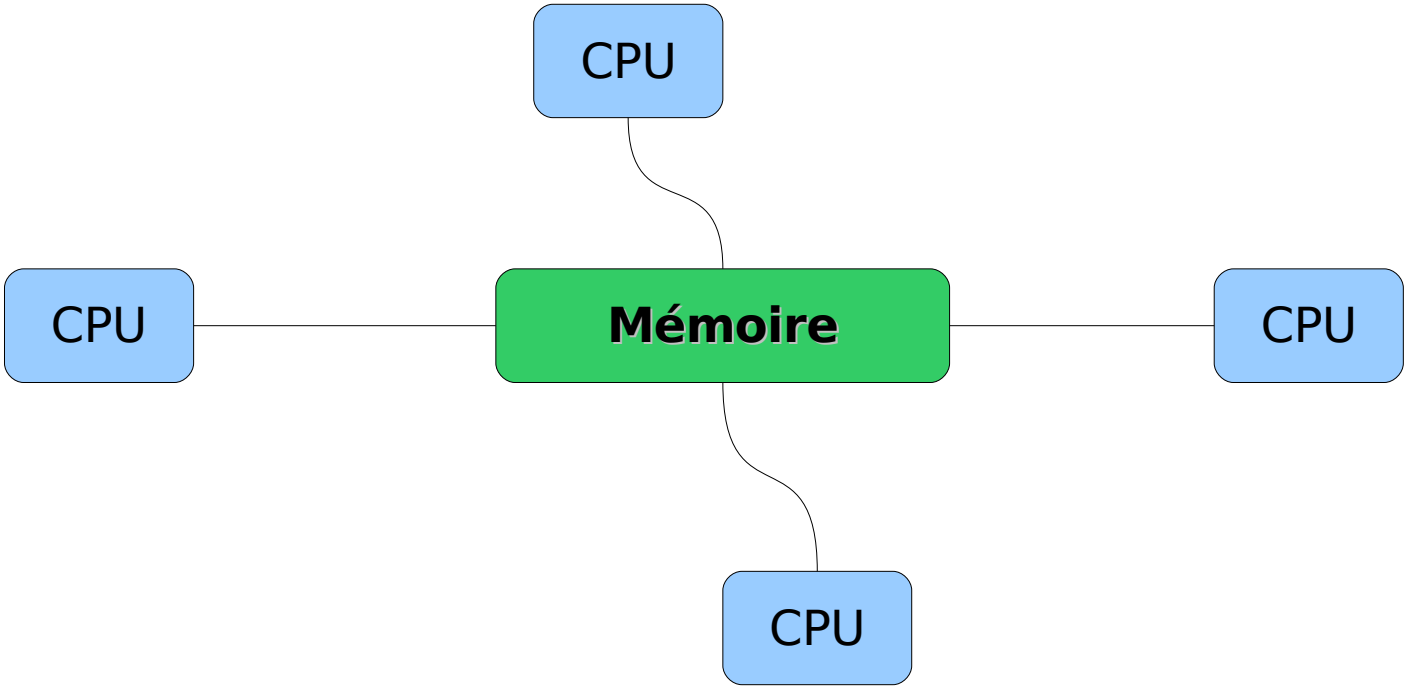
- Architecture : quel type de machine parallèle ?
- Outils : quelles librairies, quels compilateurs ?
- Programmation : quel granularité, quels algorithmes, quelles communications ?



Architecture mémoire : Systèmes à mémoire partagée et systèmes à mémoire distribuée

Système à Mémoire Partagée

- Tous les processeurs accèdent à toute la mémoire avec un même espace d'adressage global
- Chaque processeur travaille indépendamment des autres, mais les modifications effectuées par un processeur à un emplacement mémoire sont visibles par tous les autres



Mémoire Partagée : UMA/NUMA

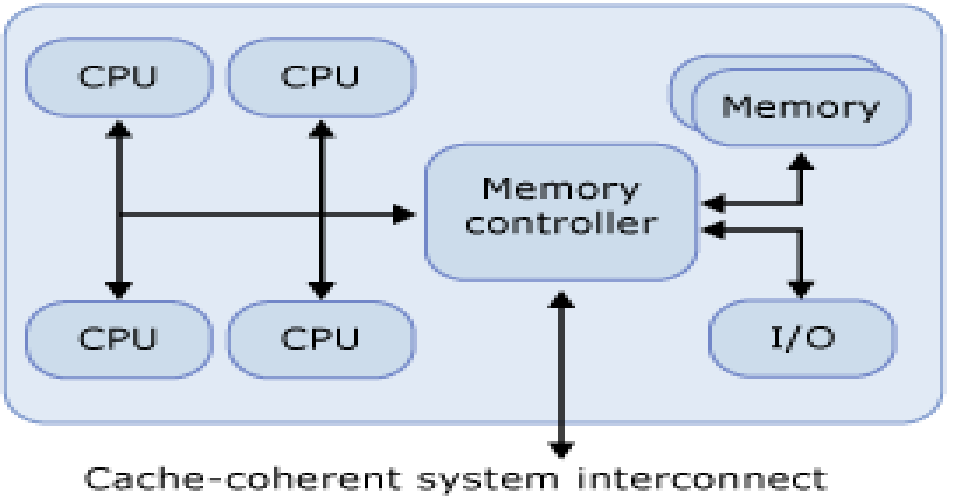
2 classes d'architecture à mémoire partagée :

- UMA (Uniform memory access) :
  Des processeurs identiques, ayant tous le même temps d'accès à la mémoire.
  Plus couramment appelées SMP (Symmetric MultiProcessor).
  Gestion de la cohérence des caches : si un processeur modifie un emplacement mémoire partagé, la modification est indiquée à tous les autres processeurs (on parle de CC-UMA).

- NUMA (None Uniform Memory Access) :
  En général fabriquées à base de plusieurs blocs SMP interconnectés.
  Les processeurs n'ont pas tous le même temps d'accès à la mémoire.
  Le temps d'accès via le lien d'interconnexion des blocs est plus lent.
  Gestion de la cohérence des caches, CC-NUMA : Cache Coherent NUMA

Exemples :

- CC-UMA : IBM Power 3, Alpha, Compaq
- CC-NUMA : SGI Altix, Bull Novascale
  Noeud SMP :
  IBM Power4/5,
  SUN V40Z



Architecture NUMA

Mémoire Partagée

Avantages :

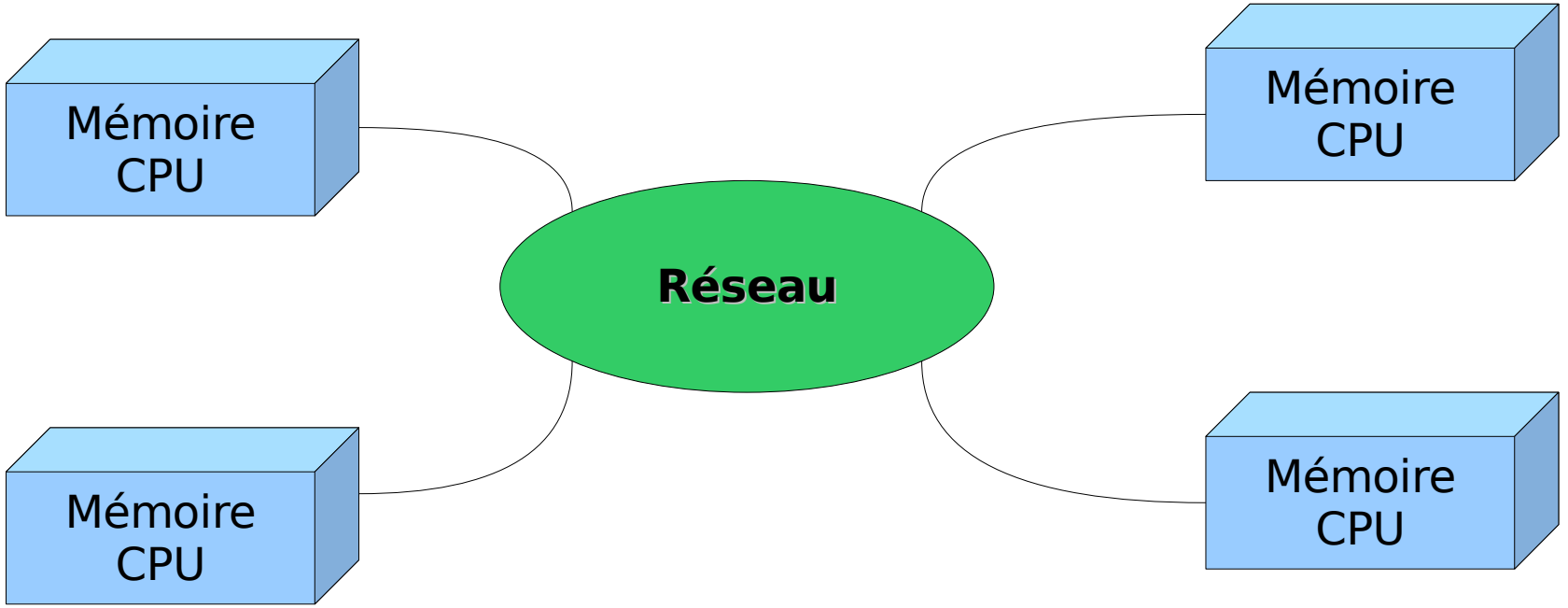
- Espace d'adresse globale : facilite le travail du programmeur
- Mémoire proche des CPUs : le partage des données entre tâches est rapide

Inconvénients :

- Manque de scalabilité : augmenter le nombre de CPUs accroît le trafic sur le chemin d'accès à la mémoire partagée
- Le programmeur doit gérer la synchronisation pour un accès correcte à la mémoire globale
- Très coûteux de construire des architectures à mémoire partagée avec un grand nombre de CPUs

Système à Mémoire Distribuée

- Un espace mémoire différent est associé à chaque processeur
- L'accès à la mémoire d'un autre processeur se fait via un réseau d'interconnexion.

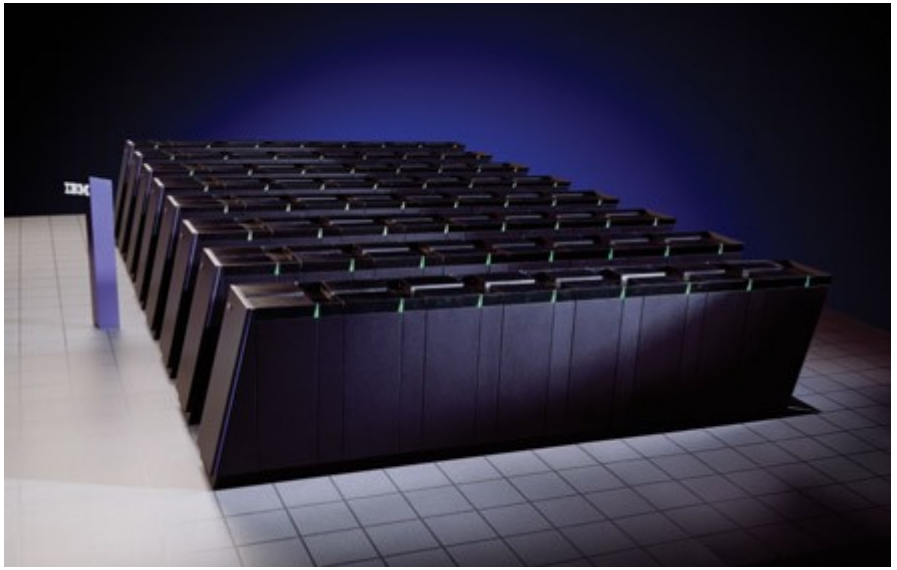


Mémoire Distribuée

- Chaque processeur a sa propre mémoire locale, il n'y a pas de notion d'espace d'adressage globale entre tous les processeurs.
- Les processeurs opèrent indépendamment les uns des autres, une modification en mémoire locale n'a pas d'effet sur la mémoire des autres processeurs.
- Si un processeur a besoin d'une donnée dans la mémoire d'un autre processeur, le programmeur doit définir explicitement la communication par envoi/réception de message.
- Les réseaux d'interconnexion sont divers, avec des niveaux de performance très variables.

Exemples :

- Cray XT3
- Cluster de PCS
- IBM Blue Gene



IBM Blue Gene

Mémoire Distribuée

Avantages :

- Bonne scalabilité : on peut augmenter le nombre de processeurs et la mémoire proportionnellement.
- Accès rapide à la mémoire sur chaque processeur sans interférence ni surcoût de gestion de cohérence de cache.
- Coût raisonnable (PCs en réseau)

Inconvénients :

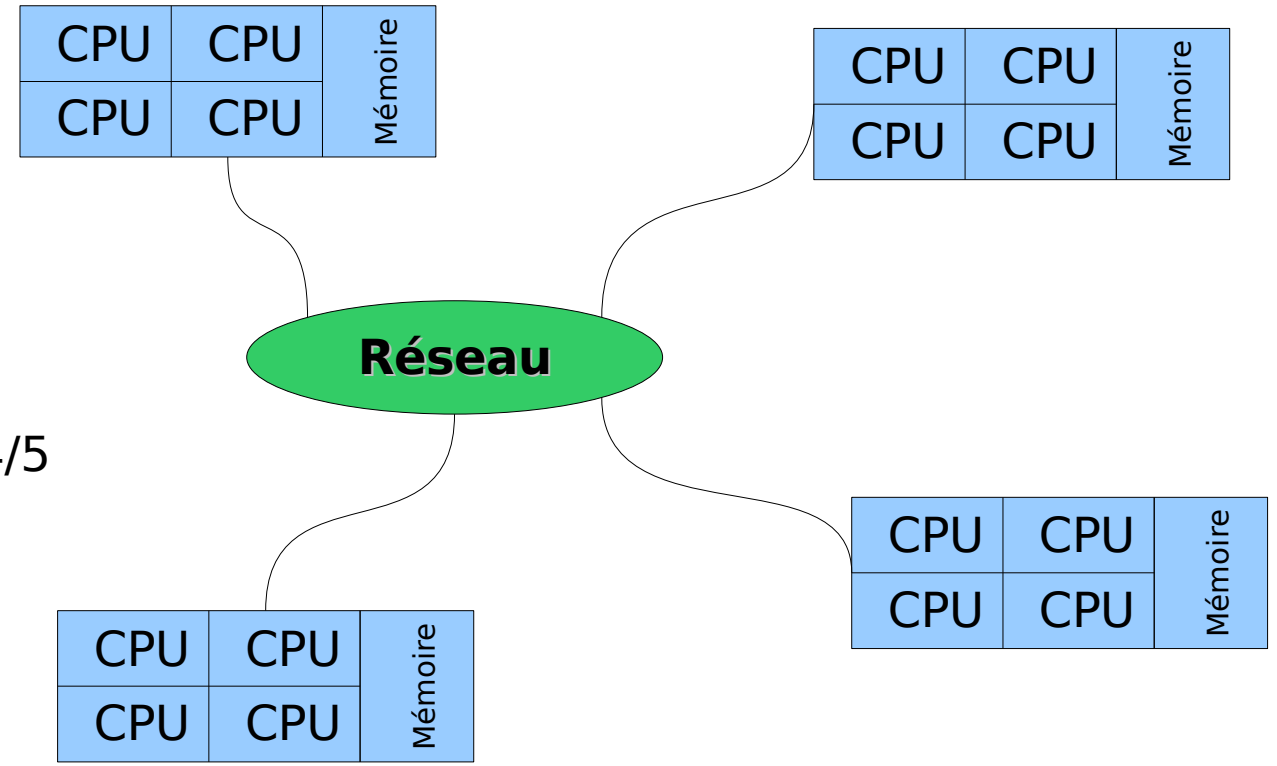
- Le programmeur doit gèrer lui-même les échanges de données entre processeurs. La mise en oeuvre est parfois difficile et nécessite l'emploi d'algorithme ad'hoc.
- Une distribution des données cohérente avec le travail des processeurs n'est pas toujours évidente. Dans certains cas, les temps de communication peuvent faire chuter considérablement les performances.

Système hybride

- Les ordinateurs les plus puissants au monde sont aujourd'hui un mixte de mémoire partagée et mémoire distribuée
- La brique de base (noeud) est un multiprocesseur à mémoire partagée
- Ces briques sont interconnectées par un réseau (type ethernet, myrinet, Infiniband, ...)

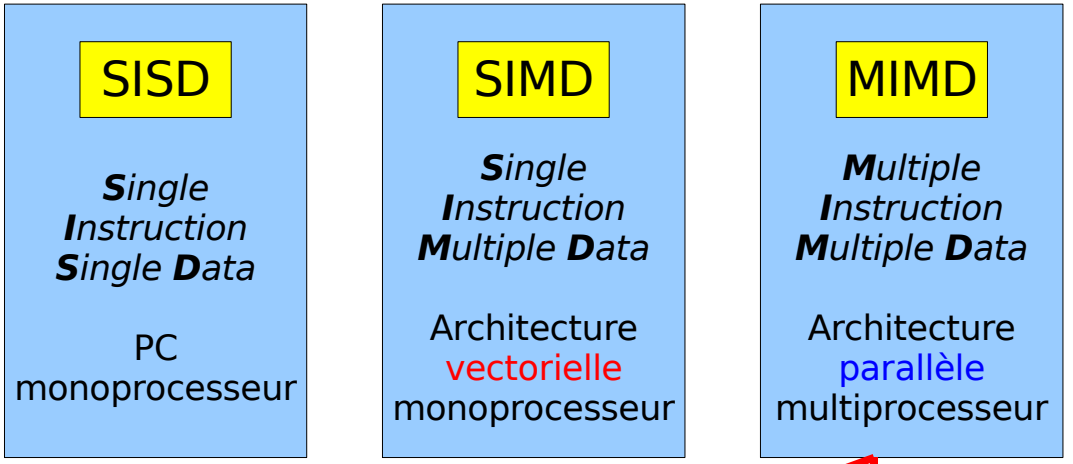
Exemples :

- Cluster de SMP :
  - SUN V40Z
  - IBM Power 4/5



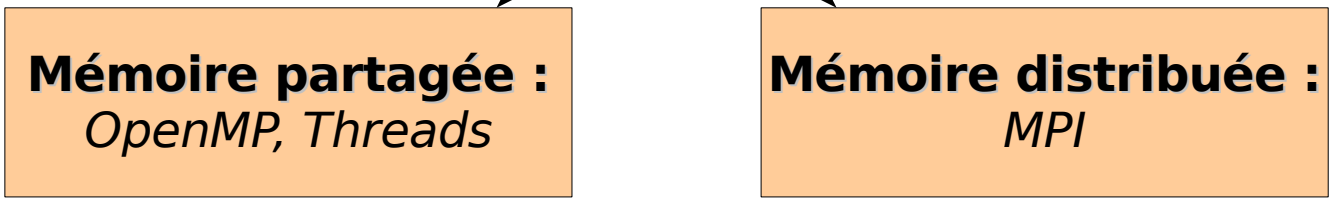
Modèles de programmation

- **Architecture matérielle:**



- **Modèle de programmation:**

- **Outils de parallélisation:**



Programmation en mémoire distribuée : échanges de messages

Pas d'espace d'adressage global :

- Il faut gérer la répartition des données sur les différents processeurs. Cela signifie que le découpage et le placement des données d'un ensemble (d'un tableau par exemple) doivent être gérés par le programmeur.

- L'échange d'informations entre les processeurs nécessite la communication d'un message entre un émetteur et un récepteur.
  C'est au programmeur d'identifier les émetteurs et les récepteurs et de placer des fonctions de communication dans leur programme.

- Le programmeur doit aussi gérer la coordination des processus ; c'est-à-dire contrôler l'exécution et gérer les sections critiques.

Environnement de programmation par passage de message :

- PVM (Parallel Virtual Machine), standard de fait, issu du milieu universitaire.

- MPI (Message Passing Interface) est une proposition de standard, présentée en 1993, émanant d'industriels et du milieu académique. Devenu de facto un standard de communication.

PVM

Permet à un ensemble de machines connectées sur un réseau local de se conduire comme une seule et même ressource appelée machine virtuelle.

Composition de PVM :

- Un démon,
- Une console,
- Des bibliothèques (C et Fortran) pour la programmation par échanges de messages.

Caractéristiques :

- Simple (cache les problème à l'utilisateur),
- Portable,
- Hétérogène : machines, architecture, OS, réseaux.

La machine virtuelle peut être composée de machines séquentielles et parallèles.

MPI

Environnement dans lequel une application MPI est un ensemble de processus autonomes exécutant chacun leur propre code et communiquant via des appels à des routines de la bibliothèques MPI.

Composition :

- Collection de fonctions C et C++ (et de sous-routines Fortran).

Caractéristiques :

- Portabilité,
- Architecture hétérogène,
- Transparence d'utilisation.

Implémentations MPI-2 (du domaine public, utilisent rsh et ssh) :

- MPICH,
- Open MPI,
- LAM ...

MPI vs PVM

PVM :

- Grand nombre d'architectures supportées,
- Beaucoup d'outils développé pour l'environnement de programmation (interface graphique, ...),
- Moins utilisé que MPI.

MPI :

- Mise en commun des avantages existants,
- Volonté de normaliser : sémantique précise, sans choix d'implantation,
- Objectifs : portabilité, puissance d'expression et bonnes performances.

Quelques bibliothèques scientifiques parallèles du domaine public utilisant MPI :

- *ScaLAPACK* : résolution de problèmes d'algèbre linéaire par des méthodes directes.
- *PETSc* : résolution de problèmes d'algèbre linéaire et non-linéaire par des méthodes itératives.
- *FFTW* : transformées de Fourier rapides.

Programmation en mémoire partagée : directives de compilation

OpenMP (Open Multi Processing)

- OpenMP est un **ensemble de directives de compilation** pour paralléliser un code sur une architecture SMP (interfaces Fortran, C et C++)
- Le **compilateur interprète les directives** OpenMP (si il en est capable!)
- Les standards d'OpenMP datent de 1997, ceux d'OpenMP-2 de 2000. Les développeurs de compilateurs les implémentent.

Un programme OpenMP est exécuté par un processus unique.
Ce processus active des processus légers (threads) à l'entrée d'une région parallèle.

Chaque processus léger exécute une tâche composée d'un ensemble d'instructions.
Un programme OpenMP est une alternance de régions séquentielles et de régions parallèles.

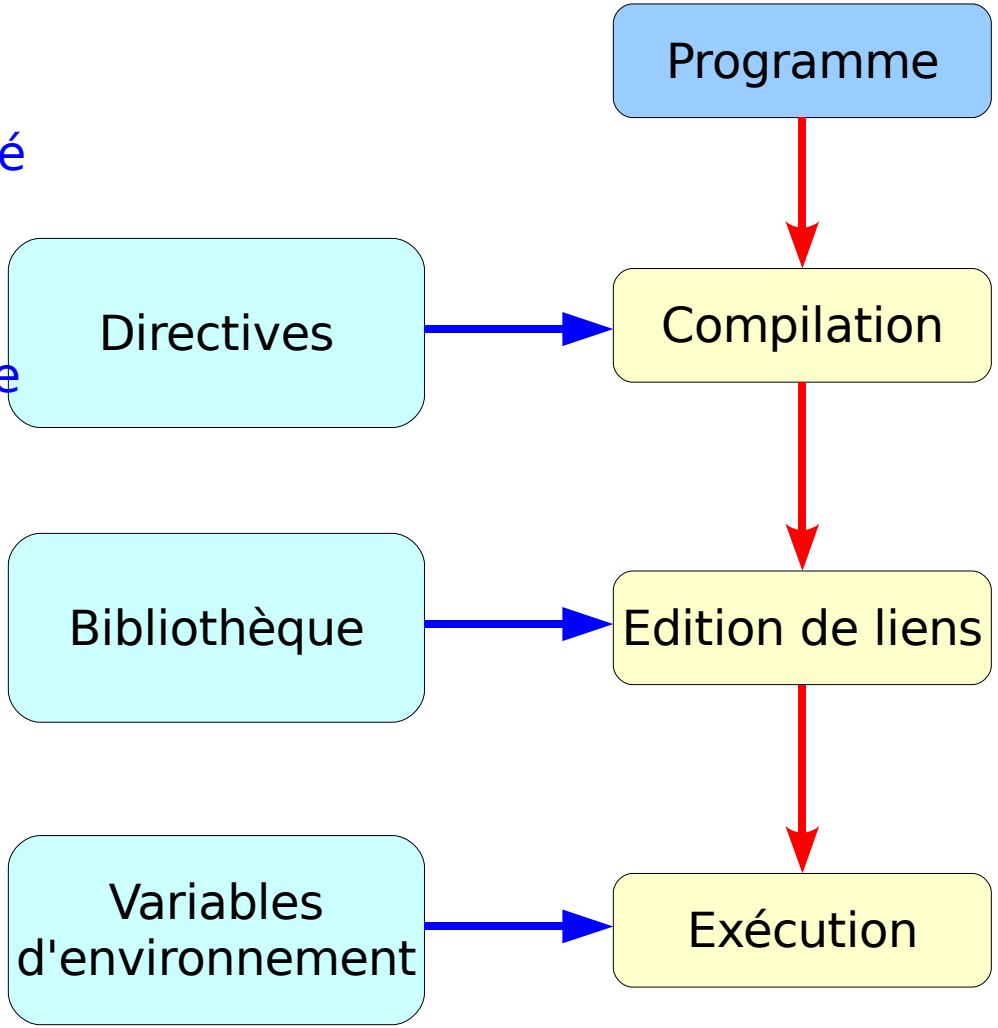
Une région séquentielle est toujours exécutée par la tâche maître, celle dont le rang vaut 0.

Programmation en mémoire partagée : openMP

- **Directives et clauses de compilation** :
  Elles servent à définir le partage du travail, la synchronisation et le statut privé ou partagé des données.
  Elles sont considérées par le compilateur comme des lignes de commentaires à moins de spécifier une option adéquate de compilation pour qu'elles soient interprétées.

- **Fonctions et sous-programmes** : ils font partie d'une bibliothèque chargée à l'édition de liens du programme.

- **Variables d'environnement** : une fois positionnées, leurs valeurs sont prises en compte à l'exécution.



Programmation en mémoire partagée : openMP vs bibliothèque de threads

OpenMP :

- Simple à apprendre et à utiliser
- Calcul automatique du nombre de threads
- Codes source séquentiel et parallèle identiques

Bibliothèque de threads :

- les bibliothèques de threads (pthreads ...) offrent un panel beaucoup plus large et plus précis de fonctions primitives permettant un contrôle très fin sur le multi-threading.

- Dans le cas où les threads doivent être gérés individuellement, ces bibliothèques seront un choix plus adaptée qu'OpenMP.
- L'utilisation d'OpenMP suppose un compilateur qui supporte ce standard.

Programmation mixte

Granularité :

- Grain fin :
  Peu de calcul entre les communications
  Facilite le load balancing
  Risque de surcoût dû aux communications et aux synchronisations
- Gros grain :
  Beaucoup de calculs entre 2 communications
  Plus simple d'améliorer les performances
  Plus difficile d'améliorer le load balancing

Une affaire de compromis :

- efficacité : minimiser les coûts de communication quand c'est possible.
  Parallélisation à grains fins en openMP et à grains plus grossiers en MPI.
- Associer la facilité de programmation d'openMP au contrôle fin du comportement du programme permis par MPI.
- Optimiser les performances des machines, souvent des clusters de multi-processeurs.

Outils de développement

Compilateurs, debuggers :

- Compilateurs compatibles OpenMP :
  Intel, gnu à partir de la version 4.2.x, Portland (dont HPF), IBM, ...
- Debuggers parallèles :
  TotalView (TotalView technologies), DDT (alinea)
- Profiling et traces MPI:
  Vtune (Profiling, Intel), VAMPIR (trace MPI, Intel), PE Benchmarker (IBM, traces et profiling), TAU (Tuning and Analysis Utilities, LANL, profiling, trace et visualisation), Paradyn (Université du Maryland , profiling), Jumpshot (ANL, visualisation de traces), ...
- Visualisation :
  pV3 (parallel Visual3, MIT, post-processing et visualisation de données distribuées)

Bibliothèques scientifiques parallèles :

- Algèbre linéaire :
  PBLAS, ScaLAPACK, PETSc, ...
- Divers :
  FFTW (Transformées de Fourier,), scilab (version parallèle avec PVM), otalView (TotalView technologies), DDT (alinea), ...

Utilisez les bibliothèques constructeurs optimisées !

Programmation parallèle : la panacée ?

Coût d'une application parallèle :

En général, l'application parallèle est plus complexe que l'appli. séquentielle correspondante.
Son coût doit tenir compte des différentes étapes du cycle de développement :

- Conception
- Codage
- Déboguage
- Tuning
- Maintenance (Attention à la qualité du développement si le code doit être retouché par d'autres personnes)

En définitive :

- ✗ Paralléliser une application est une opération compliquée
- ✗ Pas de recette magique (parallélisation automatique non performante)
- ✗ Il est nécessaire de **bien** réfléchir
  - Sur quelle architecture matérielle vais-je exécuter mon application ?
  - Quels sont mes critères d'évaluation ?
  - Ma solution logicielle est-elle réalisable ?

Programmation parallèle : la panacée ?

Performances : les mauvaises surprises

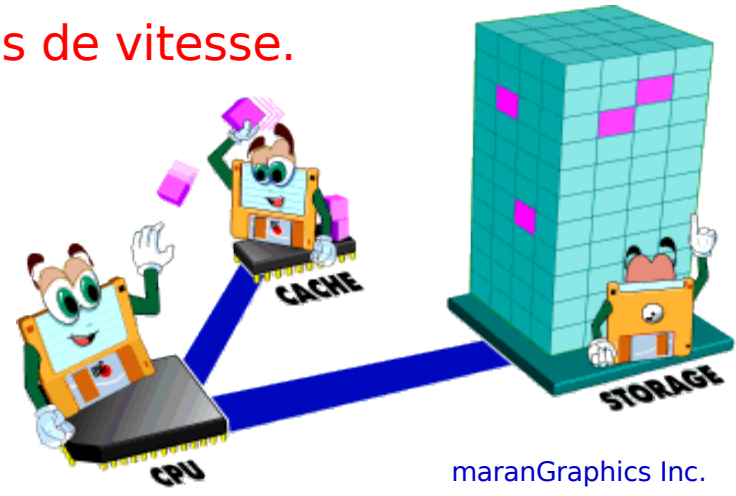
- Coût des communications : à minimiser
- Equilibrage de charges
- Algorithme dédié : il faut impérativement penser le parallélisme dès le départ

Critères d'évaluation :
- Temps d'exécution total
- Nombre de processeurs
- Coût

Scalabilité :
Généralement, des limites intrinsèques à l'algorithme.
Le hardware joue un rôle important (débit bus, mémoire-cpu sur SMP, débit réseau de comm., taille mémoire, vitesse d'horloge du proc., ...).
Plus de processeurs n'impliquent pas nécessairement plus de vitesse.

Défaut de cache :
Donnée non présente dans le cache
→ transfert depuis la mémoire
→ perte de temps !

Il faut connaître et tenir compte du hardware !!!



Références

- Cours de l'IDRIS : http://www.idris.fr/data/cours/cours-IDRIS.html
- Documentation du projet CIMENT : https://ciment.ujf-grenoble.fr/docs
- ORAP : http://www.irisa.fr/orap/
- Cours du LIFL (West Team) : http://www2.lifl.fr/west/courses/cshp/

Terminologie

- **Tâche** : une portion de travail à exécuter sur un ordinateur, du type un ensemble d'instructions d'un programme qui est exécuté sur un processeur.

- **Tâche parallèle** : une tâche qui peut s'exécuter sur plusieurs processeurs, sans risque sur la validité des résultats

- **Exécution séquentielle** : exécution d'un programme séquentiel, une étape à la fois

- **Exécution parallèle** : exécution d'un programme par plusieurs tâches, chaque tâche pouvant exécuter la même instruction ou une instruction différente, à un instant donné

- **Mémoire partagée** : d'un point de vue *physique*, réfère à une machine dont tous les proc ont un accès direct à une mémoire physique commune (par ex. via un bus).
  D'un point de vue *modèle de programmation* : toutes les tâches ont la même image mémoire et peuvent directement adresser et accéder au même emplacement mémoire logique, peu importe où il se trouve en mémoire physique.

- **Mémoire distribuée** : d'un point de vue *physique*, basée sur un accès mémoire réseau pour une mémoire physique non commune. D'un point de vue *modèle de programmation*, les tâches ne peuvent voir que la mémoire de la machine locale et doivent effectuer des communications pour accéder à la mémoire d'une machine distante, sur laquelle d'autres tâches s'exécutent.

Terminologie

- **Communications** : les tâches parallèles échangent des données, par différents moyens physiques : via un bus à mémoire partagée, via un réseau. Quelque soit la méthode employée, on parle de «communication ».

- **Synchronisation** : la coordination des tâches en temps réel est souvent associée aux communications, elle est souvent implémentée en introduisant un point de synchronisation au-delà duquel la tâche ne peut continuer tant que une ou plusieurs autres tâches ne l'ont pas atteint.

- **Granularité** : mesure qualitative du rapport calcul / communications
  - *Grain grossier (coarse)* : relativement bcp de calculs entre # communications
  - *Grain fin* : relativement peu de calcul entre # communications

- **Speedup** : mesure de l'amélioration des performances dues au parallélisme : (wall clock time of serial execution) / (wall clock time of //execution)

- **Scalabilité** : réfère à la capacité d'un système (hard ou soft) // à fournir une augmentation de l'accélération proportionnelle à l'augmentation du nombre de processeurs.