

Introduction à la Programmation Parallèle



Babacar Diop

Dpt. d'Informatique

UFR des Sciences Appliquées et de Technologies

Université Gaston Berger de Saint-Louis

Année scolaire: 2017/2018

Notions d'algorithmes

Notion d'algorithmes

- ◎ En mathématiques et en informatique, un algorithme est une séquence autonome d'actions à réaliser pour résoudre un problème donné
- ◎ Les algorithmes peuvent effectuer des tâches de calcul, de traitement des données et de raisonnement automatisé.

◎ **Exemples:**

- ◎ Chercher le maximum d'une liste de nombre
- ◎ Calculer le PGDC/PPMC entre plusieurs nombres

Notion d'algorithmes

- ◉ Le concept d'algorithme existe depuis des siècles;
- ◉ **ÉTYMOLOGIE DU NOM**: **Al-Khwarizmi (nom arabe)**, mathématicien perse du 9ème siècle
- ◉ Cependant, l'idée formelle a été posée par **David Hilbert** en 1928.
- ◉ Des formalisations ultérieures ont été formulées comme des tentatives de définition :
 - ◉ les fonctions récursives de Gödel-Herbrand-Kleene de 1930, 1934 et 1935,
 - ◉ le calcul lambda d'Alonzo Church de 1936,
 - ◉ La « Formulation 1 » d'Emil Post de 1936
 - ◉ et les machines Turing d'Alan Turing de 1936 à 1939.
 - ◉ Donner une définition formelle des algorithmes, correspondant à la notion intuitive, demeure un problème difficile.

Classification des algorithmes

Les algorithmes peuvent être classifiés par :

- ◎ **Implémentation**
- ◎ Paradigmes
- ◎ Types de problèmes
- ◎ Domaine d'études et d'applications
- ◎ Complexité

Classification des algorithmes

◎ Récursif vs itératif

- ◎ Un algorithme récursif s'invoque lui-même à plusieurs reprises jusqu'à ce qu'une certaine condition (également appelée condition d'arrêt) soit satisfaite.
- ◎ Les algorithmes itératifs utilisent des constructions répétitives comme des boucles et parfois des structures de données supplémentaires comme des piles pour résoudre les problèmes donnés.

◎ Logique

- ◎ Un algorithme peut être considéré comme une déduction logique contrôlée.
- ◎ **Utilisation:** Intelligence artificielle

◎ Séquentiel vs parallèle

- ◎ Un algo séquentiel exécute une seule instruction à la fois (un seul processeur)
- ◎ Un algo parallèle exécute plusieurs instructions à la fois (>1 processeurs)

Classification des algorithmes

◎ Déterministe ou non déterministe

- ◎ Un algorithme déterministe résout un problème avec une décision exacte à chaque étape de l'algorithme
- ◎ Un algorithme non déterministe résout un problème en avançant de manière inexacte

◎ Exact ou approximatif

- ◎ Bien que de nombreux algorithmes atteignent une solution exacte, un algorithme d'approximation recherche une approximation plus proche de la vraie solution.
- ◎ **Utilisation:** pour résoudre des problèmes difficiles.

Caractéristiques d'un algorithme

- ◎ Un algorithme doit pouvoir être exprimé dans un langage formel
- ◎ Un algorithme se déroule à partir d'un état initial, procède à un nombre fini d'états successifs, et se termine à un état final
- ◎ Un algorithme prend des données en entrées et produit des données de sorties

Notions de parallélisme

Pourquoi le parallélisme ??



Galaxy Formation



Planetary Movements



Climate Change



Rush Hour Traffic



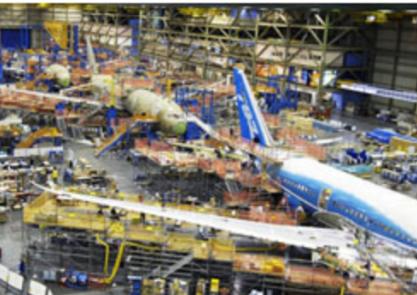
Plate Tectonics



Weather



Auto Assembly



Jet Construction



Drive-thru Lunch

Dans le monde naturel, **de nombreux événements complexes et interdépendants se produisent en même temps**, mais dans une séquence temporelle.

Comparée à la programmation séquentielle, **la programmation parallèle est beaucoup mieux adaptée à la modélisation, la simulation et la compréhension de phénomènes complexes et réels.**

Pourquoi le parallélisme ??

Le parallélisme au quotidien
de l'homme pour augmenter ses
performances ...



Algorithme parallèle

Définitions:

- Un algorithme peut être exécuté simultanément sur plusieurs processeurs différents, puis combiné pour obtenir le résultat final.
- Incontournable pour traiter d'énormes volumes de données dans un temps rapide.
- Ce cours fournit une introduction à la conception et à l'analyse d'algorithmes parallèles. En plus, il explique les modèles suivis dans des algorithmes parallèles, leurs structures et leur mise en œuvre.

Qu'est-ce que le parallélisme?

- Processus de traitement **simultané** de plusieurs instructions.
- Réduit le temps de calcul total.
- Peut être implémenté en utilisant des ordinateurs parallèles, c'est-à-dire un ordinateur avec de nombreux processeurs.
- **Remarques:** Les ordinateurs parallèles nécessitent un algorithme parallèle, des langages de programmation, des compilateurs et un système d'exploitation prenant en charge le multitâche.

Algorithme parallèle

Selon l'architecture des ordinateurs, nous avons deux types d'algorithmes:

- **Algorithme séquentiel** - Un algorithme dans lequel les étapes consécutives d'instructions sont exécutées dans un ordre chronologique pour résoudre un problème.
- **Algorithme parallèle** - Le problème est divisé en sous-problèmes et sont exécutés en parallèle pour obtenir des résultats individuels . Ensuite, ces résultats individuels sont combinés pour obtenir le résultat final souhaitée.

Problèmes derrière les algorithmes parallèles

- Difficulté de diviser un gros problème en sous-problèmes, à cause de la dépendance de données entre sous-problèmes.
- Par conséquent, les processeurs doivent communiquer entre eux pour résoudre le problème.
- **Il a été constaté que le temps nécessaire aux processeurs pour communiquer entre eux est supérieur au temps réel de traitement.**
- Ainsi, lors de la conception d'un algorithme parallèle, une utilisation correcte de l'UC devrait être considérée comme un algorithme efficace.

- Pour concevoir un algorithme correctement, il faut avoir une idée claire du modèle basique de calcul dans un ordinateur parallèle.

Catégories d'ordinateurs

Catégories d'ordinateurs

Selon le **flux d'instructions** et le **flux de données**, les ordinateurs peuvent être classés en quatre catégories:

- ✓ Ordinateurs à flux **d'instruction unique**, et à **flux de données unique** (**SISD**)
- ✓ Ordinateurs à flux **d'instruction unique**, et à **flux de données multiples** (**SIMD**)
- ✓ Ordinateurs à flux **d'instructions multiples**, et à **flux de données unique** (**MISD**)
- ✓ Ordinateurs à flux **d'instructions multiples**, et à **flux de données multiples** (**MIMD**)

S I S D Single Instruction stream Single Data stream	S I M D Single Instruction stream Multiple Data stream
M I S D Multiple Instruction stream Single Data stream	M I M D Multiple Instruction stream Multiple Data stream

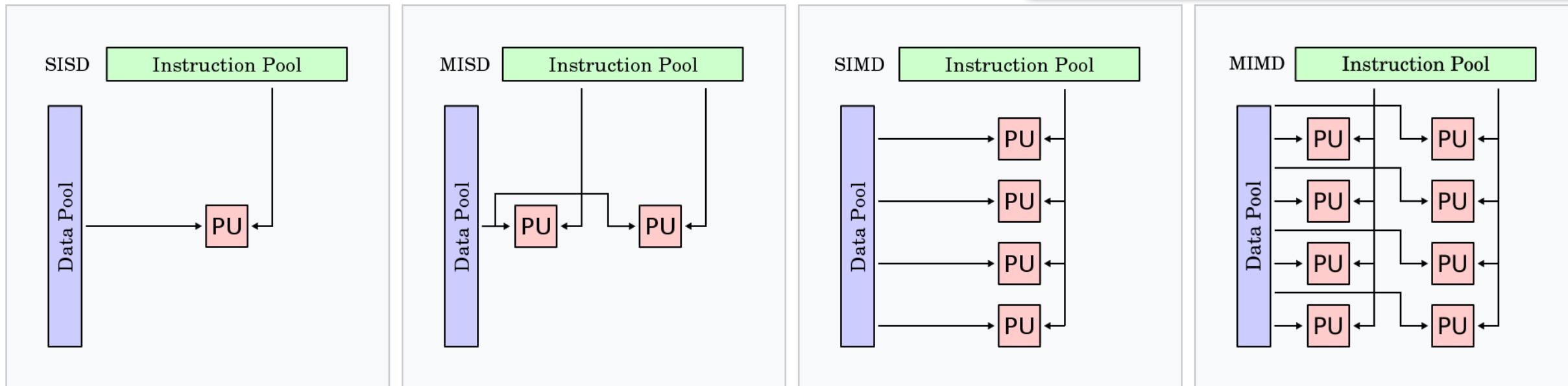
Taxinomie de Flynn

Flynn, M., *Some Computer Organizations and Their Effectiveness*, IEEE Trans. Comput., Vol. C-21, p. 948, 1972.

Taxinomie de Flynn

Wikipedia

https://fr.wikipedia.org/wiki/Taxonomie_de_Flynn



SISD

MISD

SIMD

MIMD

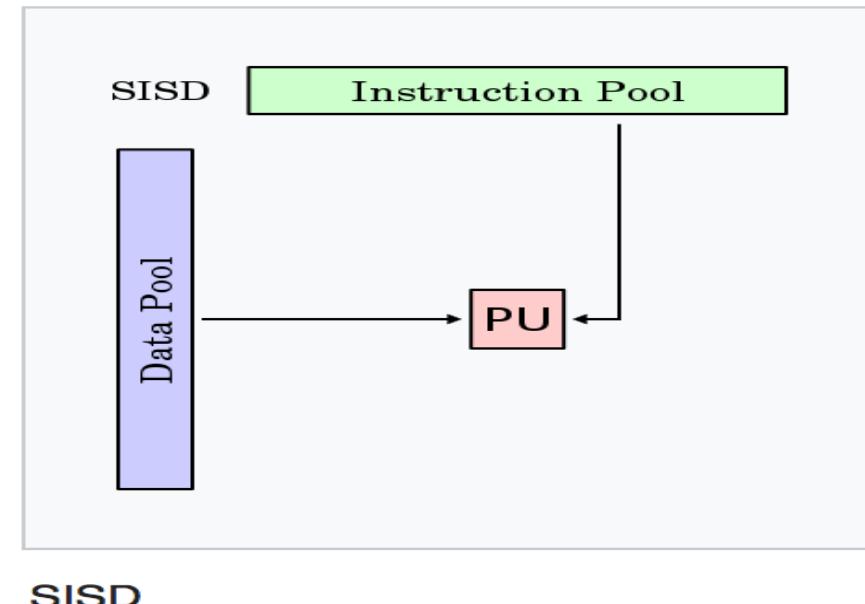
SISD

Contient :

- une unité de contrôle,
 - une unité de traitement
 - et une unité de mémoire.
-
- Le processeur reçoit **un seul flux d'instructions** de l'unité de commande (UC) et opère sur **un seul flux de données** de la mémoire.
 - Lors d'une opération, à chaque étape, le processeur reçoit **une instruction** de l'unité de commande et fonctionne sur **une seule donnée reçue** de l'unité de mémoire.

SISD

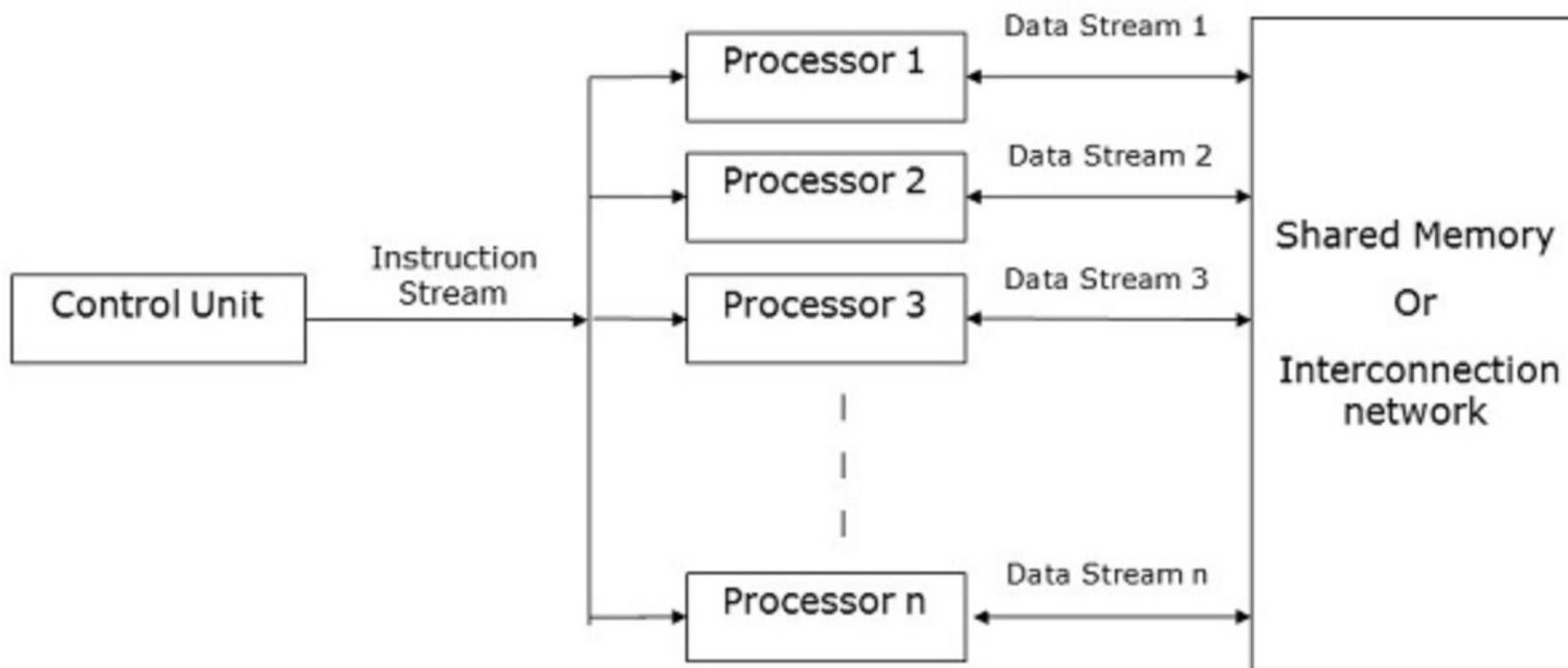
- Le processeur reçoit **un seul flux d'instructions** de l'unité de commande (UC) et opère sur **un seul flux de données** de la mémoire.
- Lors d'une opération, à chaque étape, le processeur reçoit **une instruction** de l'unité de commande et fonctionne sur **une seule donnée reçue** de l'unité de mémoire.



SIMD

Contient :

- une unité de contrôle,
- plusieurs unités de traitement
- et une mémoire partagée ou un réseau d'interconnexion.



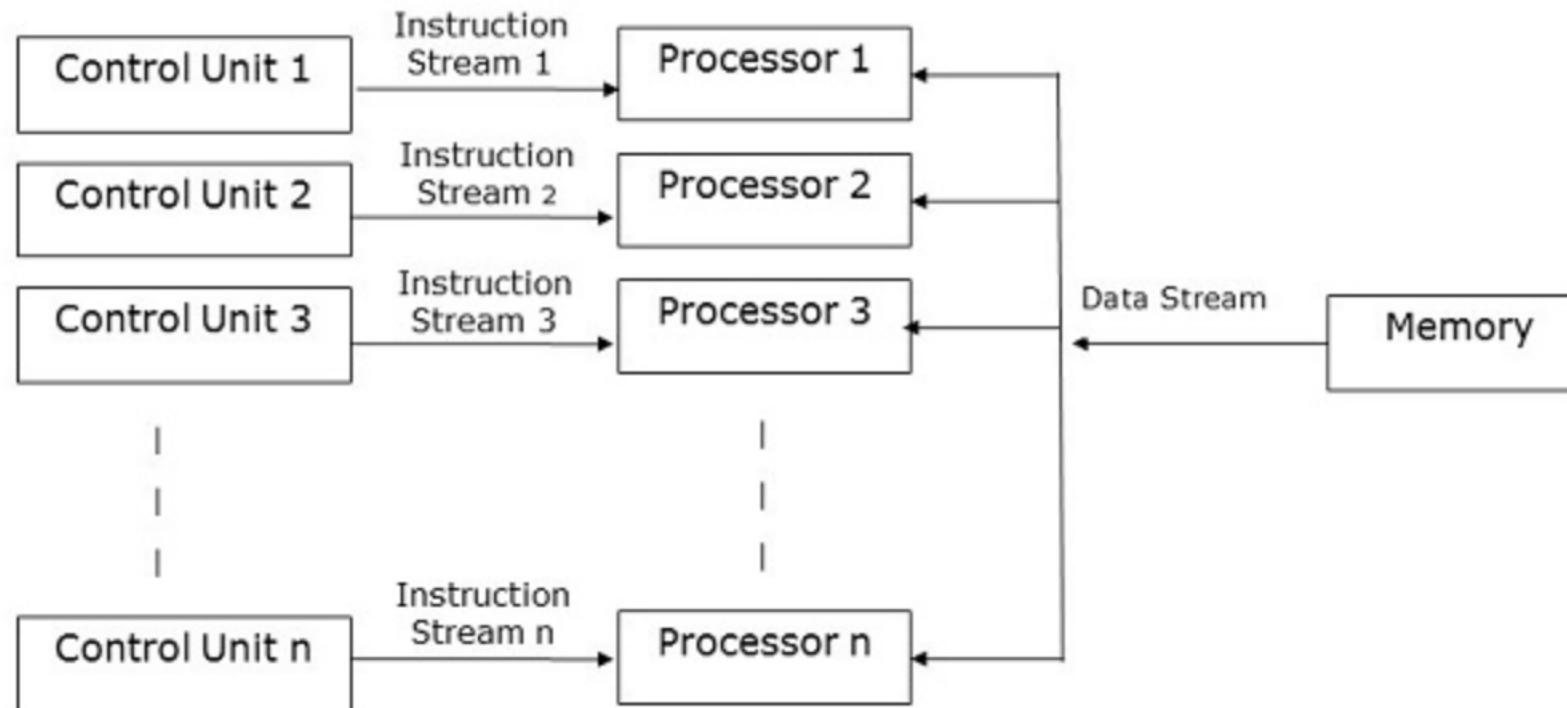
SIMD

- Une seule UC envoie des instructions à toutes les unités de traitement.
- Pendant le calcul, à chaque étape, chaque processeur reçoit **un seul ensemble d'instructions** de l'UC et fonctionne sur **differents ensembles de données** de la mémoire.
- Chacune des unités de traitement possède sa propre mémoire locale pour stocker les données et les instructions. **Les processeurs doivent communiquer entre eux.** Cela se fait par **mémoire partagée** ou par réseau d'interconnexion.
- Pendant qu'une partie des processeurs exécute un ensemble d'instructions, les autres attendent leur prochaine série d'instructions. Les instructions de l'UC décident quel processeur sera actif pour exécuter les instructions ou inactif pour attendre la prochaine instruction.

MISD

Contient :

- plusieurs unités de contrôle,
- plusieurs unités de traitement
- et une unité de mémoire commune.



Chaque processeur possède sa propre unité de contrôle et partage une unité de mémoire commune.

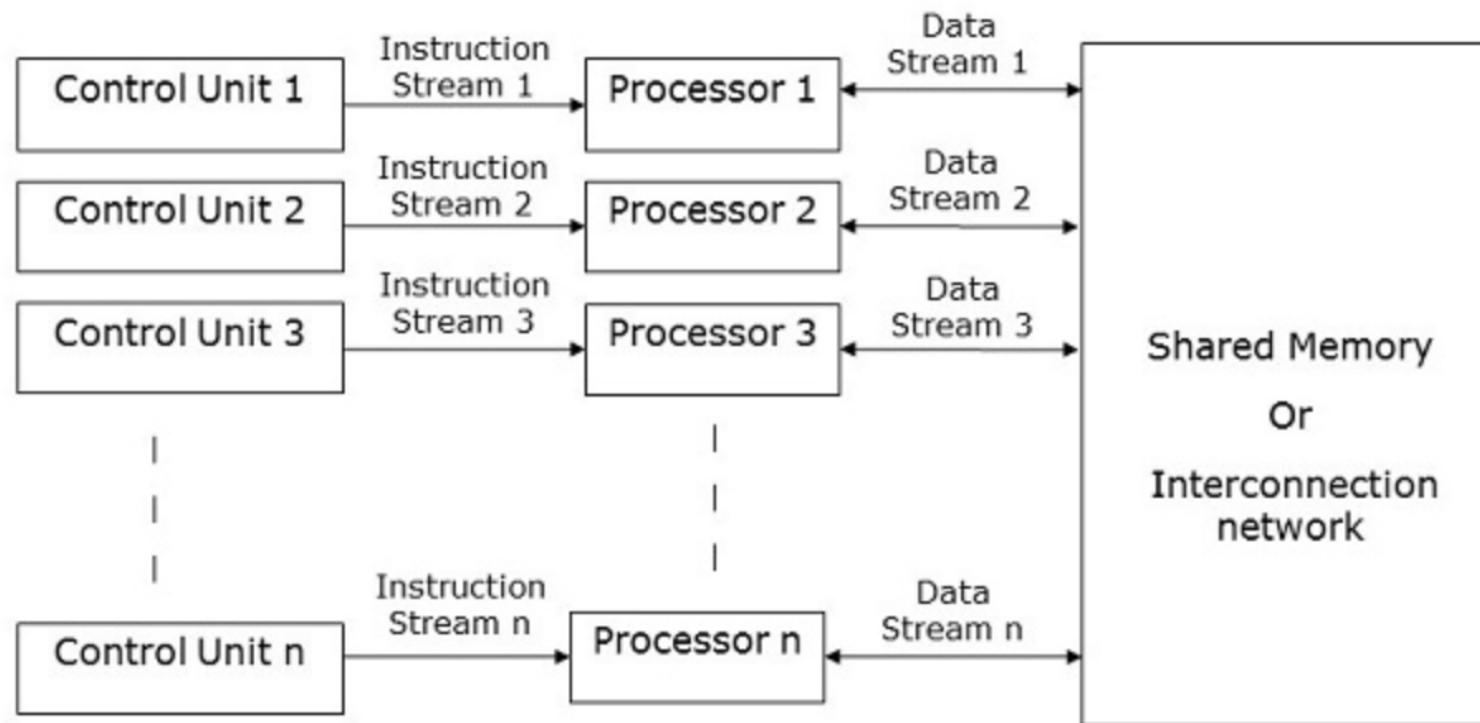
Tous les processeurs obtiennent des instructions individuellement à partir de leur propre unité de contrôle et **ils fonctionnent sur un seul flux de données** selon les instructions qu'ils ont reçues de leurs unités de contrôle respectives.

Les processeurs fonctionnent simultanément.

MIMD

Possède:

- plusieurs unités de contrôle,
- plusieurs unités de traitement
- et une mémoire partagée ou un réseau d'interconnexion.



MIMD

Chaque processeur possède sa propre unité de contrôle, son unité de mémoire locale et son unité arithmétique et logique.

Ils reçoivent différents ensembles d'instructions de leurs unités de contrôle respectives et fonctionnent sur différents ensembles de données.

Algorithmes parallèles

Analyse d'un algorithme parallèle (1)

Les **algorithmes parallèles** sont conçus pour améliorer la vitesse de calcul d'un ordinateur.

Pour analyser un algorithme parallèle, on considère généralement les paramètres suivants:

- La complexité temporelle (temps d'exécution),
- Nombre total de processeurs utilisés,
- Et coût total.

Analyse d'un algorithme parallèle (2)

Analyse temporelle

- ◎ L'objectif principal d'un algorithme parallèle c'est de réduire le temps de calcul d'un algorithme séquentiel.
- ◎ Evaluer le temps d'exécution d'un algorithme est extrêmement important pour analyser son efficacité.
- ◎ Temps d'exécution total = à partir du moment où l'algorithme commence à s'exécuter jusqu'au moment où il s'arrête.
- ◎ Temps d'exécution total = le moment où le premier processeur a commencé son exécution jusqu'au moment où le dernier processeur arrête son exécution.

Analyse d'un algorithme parallèle (3)

La **complexité temporelle** d'un algorithme peut être classée en trois catégories:

- ◎ **Complexité au pire cas** - lorsque le temps nécessaire à un algorithme pour une entrée donnée est maximal.
- ◎ **Complexité au moyen des cas** - Lorsque la quantité de temps requise par un algorithme pour une entrée donnée est moyenne.
- ◎ **Complexité au meilleur des cas** - Lorsque le temps requis par un algorithme pour une entrée donnée est minimum.

Analyse d'un algorithme parallèle (4)

Accélération d'un algorithme

La performance d'un algorithme parallèle est déterminée en calculant son accélération.

L'accélération est définie comme le rapport du temps d'exécution au pire des cas de l'algorithme séquentiel le plus rapide connu pour un problème particulier au temps d'exécution au pire des cas de l'algorithme parallèle.

$$\text{Accélération} = \frac{\text{Complexité temporelle de l'algorithme séquentiel}}{\text{Complexité temporelle de l'algorithme parallèle}}$$

Analyse d'un algorithme parallèle (5)

Nombre de processeurs utilisés

- ◎ **Le nombre de processeurs** utilisés est un facteur important dans l'analyse de l'efficacité d'un algorithme parallèle.
- ◎ Le coût d'achat, de maintenance et d'exécution des ordinateurs est calculé.
- ◎ Plus grand est le nombre de processeurs utilisés par un algorithme pour résoudre un problème, plus coûteux devient le résultat obtenu.

Analyse d'un algorithme parallèle (6)

Coût total

Le coût total d'un algorithme parallèle est le produit de la complexité du temps et du nombre de processeurs utilisés dans cet algorithme particulier.

$$\text{Coût total} = \text{Complexité temporelle} \times \text{Nombre de processeurs utilisés}$$

Par conséquent, l'efficacité d'un algorithme parallèle est égale à :

$$Eff = \frac{\text{Complexité temporelle de l'algorithme séquentiel (au pire des cas)}}{\text{Complexité temporelle de l'algorithme parallèle (au pire des cas)}}$$

Algorithme parallèle: modèles

Le modèle d'un algorithme parallèle est développé en considérant **une stratégie pour diviser les données** et la méthode de traitement et en appliquant une **stratégie appropriée pour réduire les interactions**.

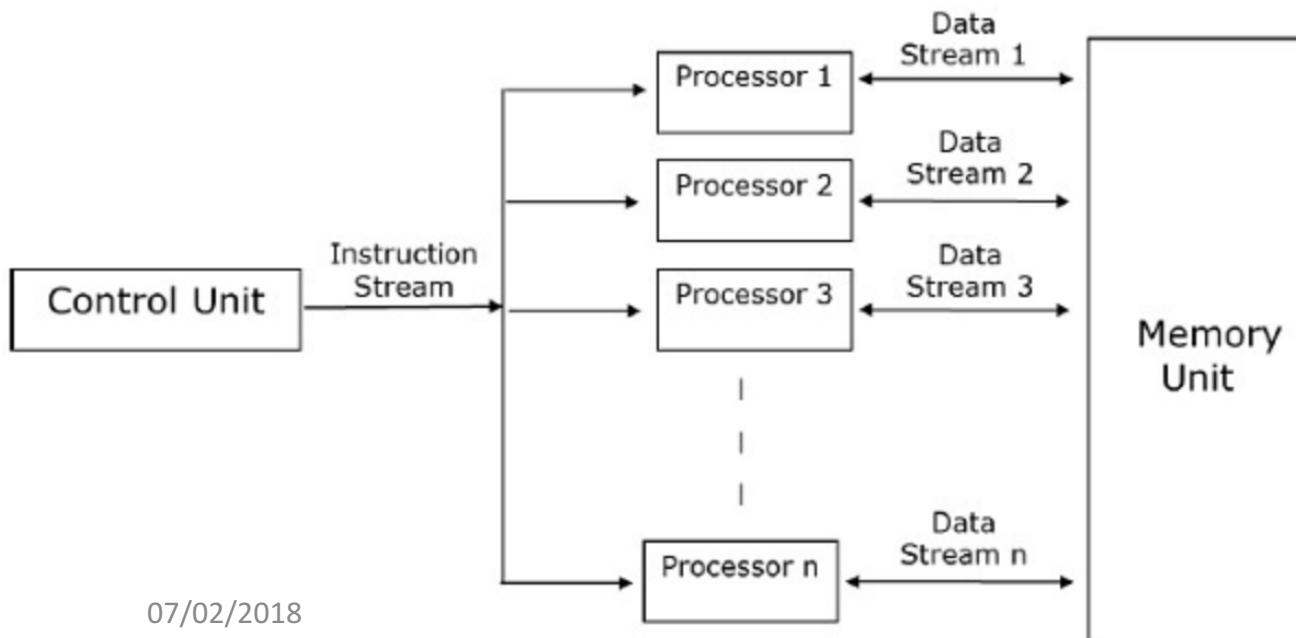
Nous discuterons des modèles d'algorithme parallèle suivants:

- ✓ Modèle parallèle de données
- ✓ Modèle de graphe des tâches
- ✓ Modèle de pool de travail
- ✓ Maître esclave
- ✓ Producteur-Consommateur ou modèle de pipeline
- ✓ Modèle hybride

Données parallèles

Dans ce modèle, les tâches sont attribuées aux processus et chaque tâche effectue des opérations similaires sur différentes données.

Le parallélisme des données est une conséquence d'opérations individuelles qui sont appliquées sur plusieurs éléments de données.



Principale caractéristique :
le parallélisme des données augmente avec la taille du problème, ce qui implique d'utiliser **plus de processus** pour résoudre des problèmes **plus grands en termes de données**.

Modèle de graphe des tâches

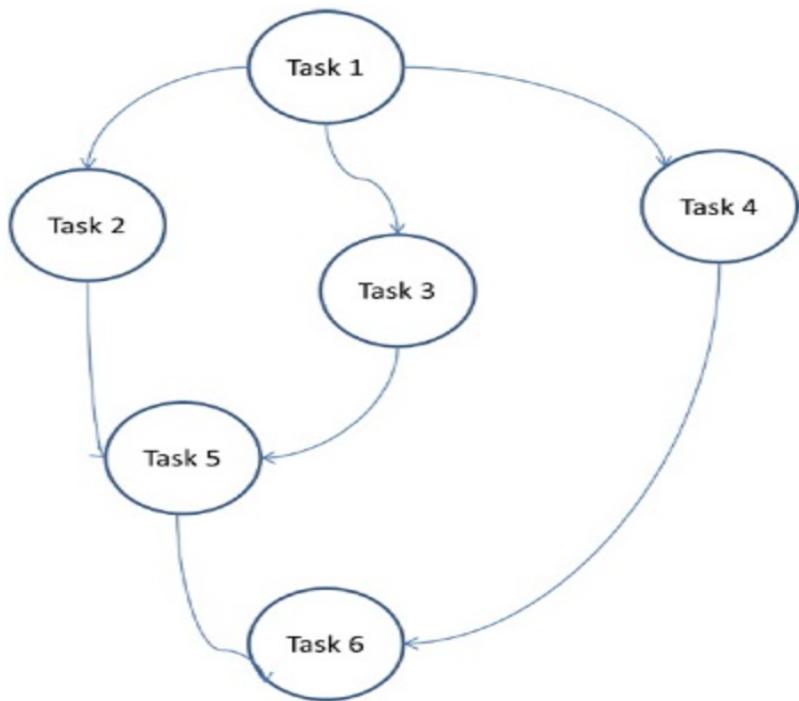
Dans ce modèle de graphique de tâche, le parallélisme est exprimé par un graphe de tâches.

La corrélation entre les tâches est utilisée pour promouvoir la localisation ou minimiser les coûts d'interaction.

Ce modèle est appliqué pour résoudre des **problèmes où la quantité de données associée aux tâches est énorme par rapport au nombre de calcul** qui leur est associé.

Les tâches sont assignées pour aider à améliorer le coût du mouvement des données parmi les tâches.

Modèle de graphe des tâches



- Les problèmes sont divisés en tâches atomiques et exécutées en graphe.
- Chaque tâche est une unité de travail indépendante qui a des dépendances sur une ou plusieurs tâches antérieures.
- Une fois la tâche terminée, la sortie d'une tâche antécédente passe à la tâche dépendante.
- Une tâche avec une tâche antécédente ne commence l'exécution que lorsque sa tâche antécédente est terminée.
- Le résultat final du graphe est reçu lorsque la dernière tâche dépendante est terminée (Tâche 6 de la figure ci-dessus).

Modèle de pool de travail (1)

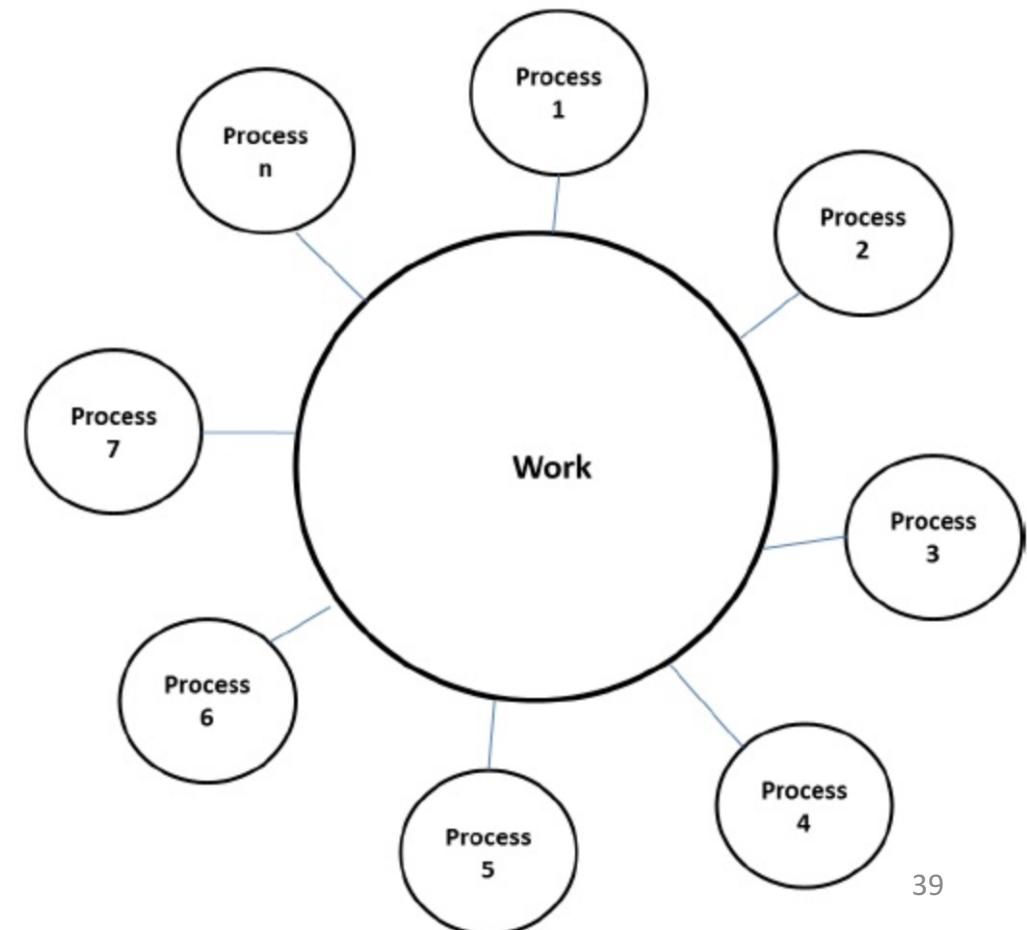
- Dans ce modèle, les tâches sont dynamiquement affectées aux processus d'équilibrage de la charge.
- Tout processus peut éventuellement exécuter une tâche quelconque.
- Modèle utilisé lorsque la quantité de données associée aux tâches est comparativement inférieure au calcul associé aux tâches.
- Il n'y a pas de pré-assignation des tâches sur les processus.
- L'attribution des tâches est centralisée ou décentralisée.

Modèle de pool de travail (2)

Les pointeurs sur les tâches sont enregistrés sur :

- ✓ une liste partagée physiquement ou,
- ✓ dans une file d'attente de priorité ou,
- ✓ dans une table ou
- ✓ dans un arbre de hashage,
- ✓ ou dans une structure de données

... répartie physiquement.

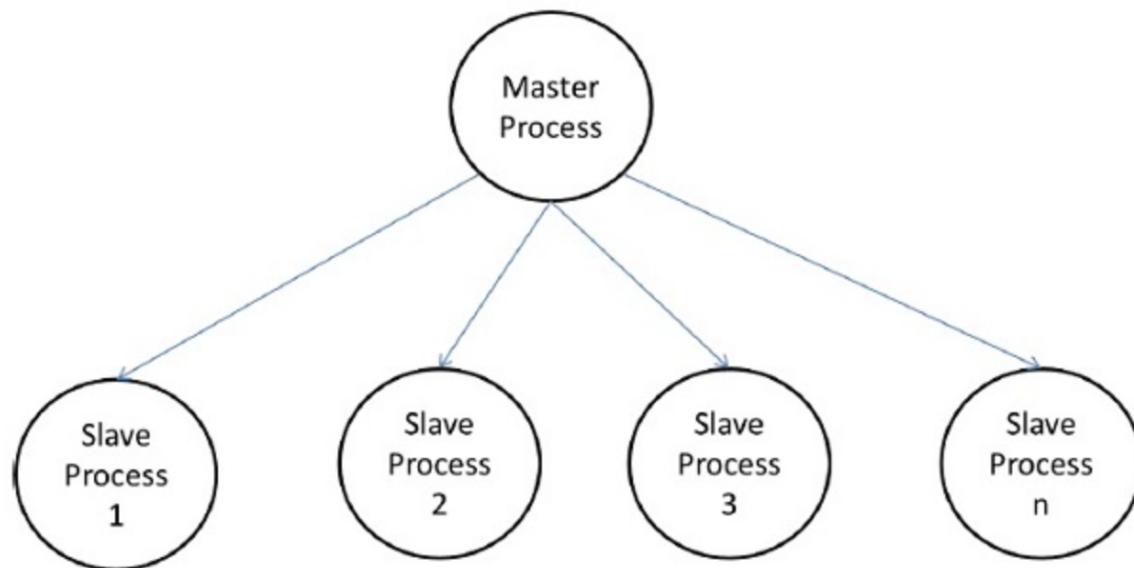


Modèle de pool de travail (3)

- La tâche peut être disponible au début, ou peut être générée dynamiquement.
- Si la tâche est générée dynamiquement et qu'une assignation décentralisée de la tâche est effectuée, un algorithme de détection de terminaison est requis pour que tous les processus puissent détecter l'achèvement de l'ensemble du programme et arrêter de chercher plus de tâches.

Modèle maître-esclave (1)

Dans ce modèle, un ou plusieurs processus maîtres génèrent une tâche et l'attribuent aux processus esclaves.



Modèle maître-esclave (2)

- Dans certains cas, une tâche doit être complétée par étapes et la tâche dans chaque phase doit être terminée avant que la tâche des phases suivantes ne puisse être générée.
- Le modèle maître-esclave peut être généralisé à un modèle maître-esclave hiérarchique ou multi-niveau dans lequel le maître de niveau supérieur alimente la grande partie des tâches au maître de second niveau, qui subdivise les tâches entre ses propres esclaves et peut effectuer une partie des tâches lui-même.

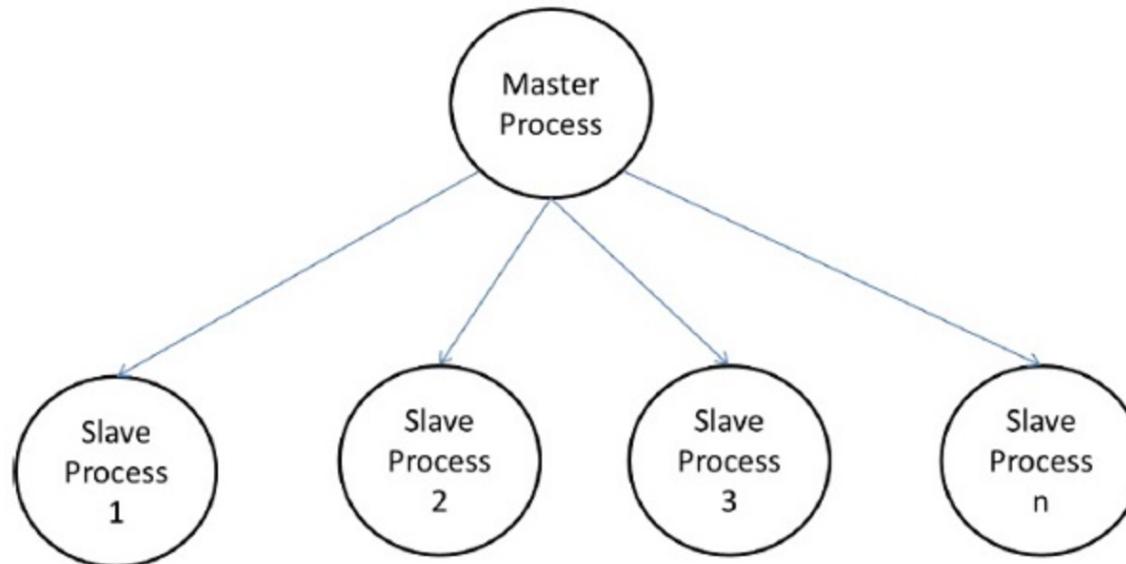
Modèle maître-esclave (3)

Les tâches peuvent être attribuées au préalable si :

- ✓ Le maître peut estimer le volume des tâches ou,
- ✓ Une assignation aléatoire peut faire un travail satisfaisant d'équilibrage de la charge ou,
- ✓ Les esclaves reçoivent des tâches plus petites à différents moments.

Ce modèle est
d'espace d'adr
l'interaction est

gmes
message, car



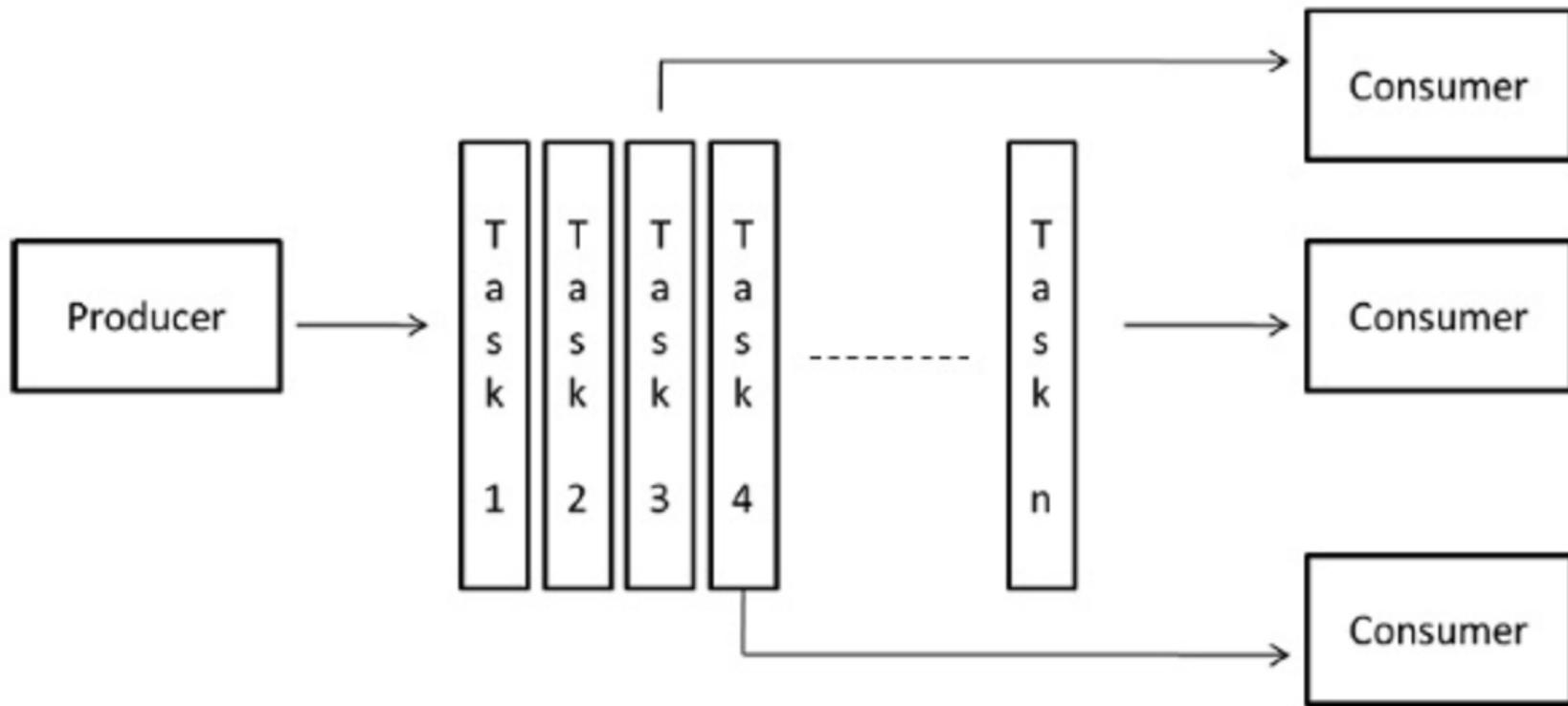
Modèle Pipeline (1)

- Un ensemble de données est transmis à travers une série de processus, dont chacun réalise une tâche.
- L'arrivée de nouvelles données génère l'exécution d'une nouvelle tâche par un processus en file d'attente.
- Les processus pourraient constituer une file d'attente sous la forme de tableaux linéaires ou multidimensionnels, d'arbres ou de graphiques généraux avec ou sans cycles.

Modèle Pipeline (2)

- Chaîne de producteurs et de consommateurs.
- Chaque processus en file d'attente peut être considéré comme un consommateur de données pour le processus qui l'a précédé dans la file d'attente et en tant que producteur de données pour le processus qui le suit dans la file d'attente.
- La file d'attente n'a pas besoin d'être une chaîne linéaire; Il peut s'agir d'un graphe dirigé.
- La technique de minimisation d'interaction la plus courante applicable à ce modèle est l'interaction qui se chevauche avec le calcul.

Modèle Pipeline (3)

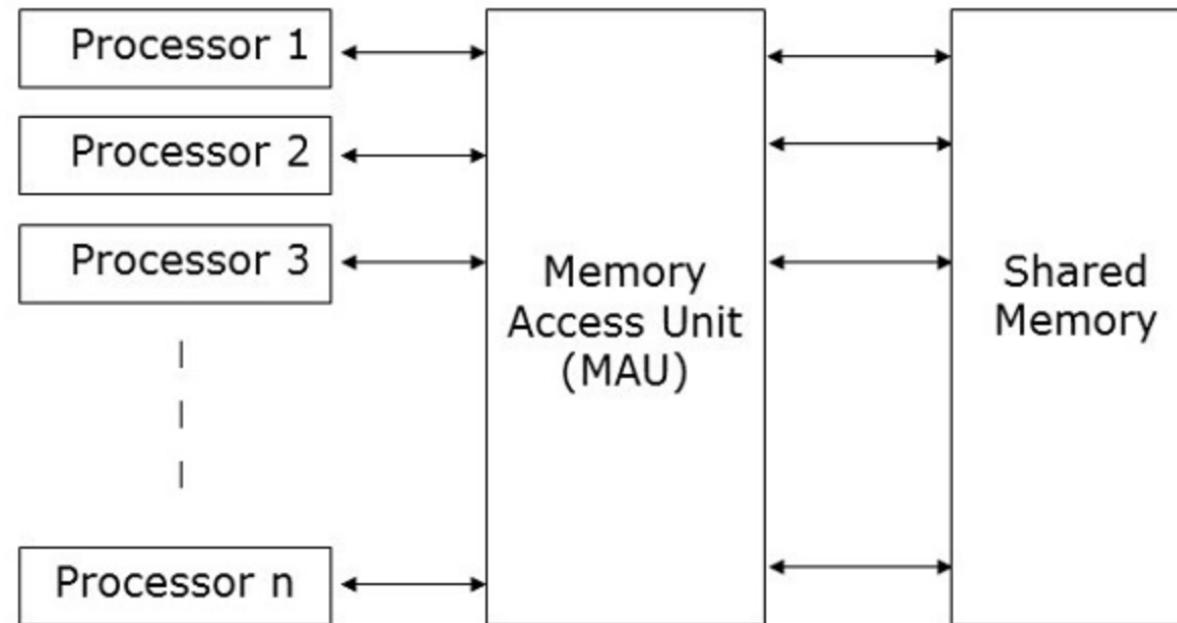


Modèles hybrides

- Un modèle d'algorithme hybride est nécessaire lorsque plus d'un modèle peut être nécessaire pour résoudre un problème.
- Un modèle hybride peut être composé soit de modèles multiples appliqués soit hiérarchiquement ou séquentiellement à différentes phases d'un algorithme parallèle.

Parallel Random Access Machines (PRAM)

- Parallel Random Access Machines (PRAM) est un modèle, où plusieurs processeurs similaires sont attachés à un seul bloc de mémoire.
- Les processeurs peuvent communiquer entre eux par la seule mémoire partagée.



Parallel Random Access Machines (PRAM)

- Ici, un certain nombre de processeurs peuvent souhaiter effectuer en même temps des opérations indépendantes sur un certain nombre de données.
- Cela peut entraîner un accès simultané du même emplacement de mémoire par différents processeurs.

Comment garantir l'accès concurrent aux données de la mémoire partagée de manière cohérente et sans blocage ??

Parallel Random Access Machines (PRAM)

Pour résoudre ce problème, les contraintes suivantes ont été appliquées sur le modèle

- ✓ **Lecture exclusive** et **Écriture exclusive** (EREW) - Aucun processeur n'est autorisé à lire ou à écrire dans le même emplacement de mémoire en même temps.
- ✓ **Lecture exclusive** et **Écriture simultanée** (ERCW) - Aucun processeur n'est autorisé à lire simultanément le même emplacement de mémoire, mais il est autorisé à écrire sur le même emplacement de mémoire en même temps.
- ✓ **Lecture simultanée** et **Écriture exclusive** (CREW) - Tous les processeurs sont autorisés à lire à partir du même emplacement de mémoire en même temps, mais ils ne sont pas autorisés à écrire sur le même emplacement de mémoire en même temps.
- ✓ **Lecture simultanée** et **Écriture simultanée** (CRCW) - Tous les processeurs sont autorisés à lire ou à écrire dans le même emplacement de mémoire en même temps.

Implémentation du modèle PRAM

Il existe de nombreuses méthodes pour implémenter le modèle PRAM, mais les plus connues sont:

- Modèle de mémoire partagée
- Modèle de passage de message
- Modèle parallèle de données

Mémoire partagée (1)

Met l'accent sur le parallélisme de contrôle et non sur le parallélisme des données.

Plusieurs processus s'exécutent sur différents processeurs indépendamment, mais ils partagent un espace mémoire commun.

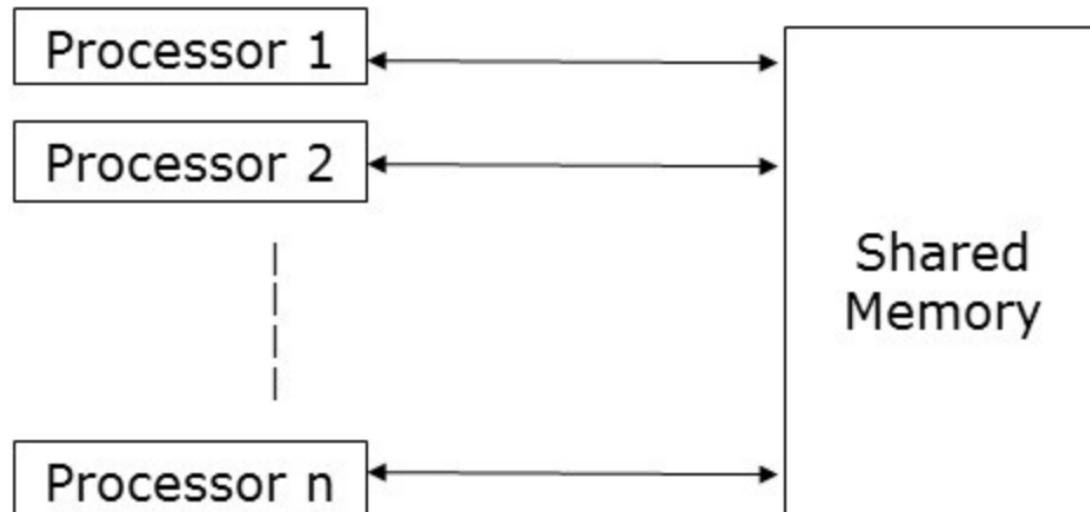
En raison de toute activité du processeur, s'il y a un changement dans n'importe quel emplacement de la mémoire, il est visible pour le reste des processeurs.

➤ Problème: **Accès concurrent à la mémoire**

➤ Exemple: plusieurs processeurs écrivent et y lisent en même temps

Mémoire partagée (2)

- Problème: **Accès concurrent à la mémoire**
 - Exemple: plusieurs processeurs écrivent et y lisent en même temps
- Solution: Mécanismes de contrôle
 - Verrouillage
 - Sémaphore
- Exemples:
 - SolarisTM threads [Solaris]
 - POSIX threads [Linux]
 - Win32 threads [Windows]
 - JavaTM threads



Mémoire partagée (3)

➤ Avantages

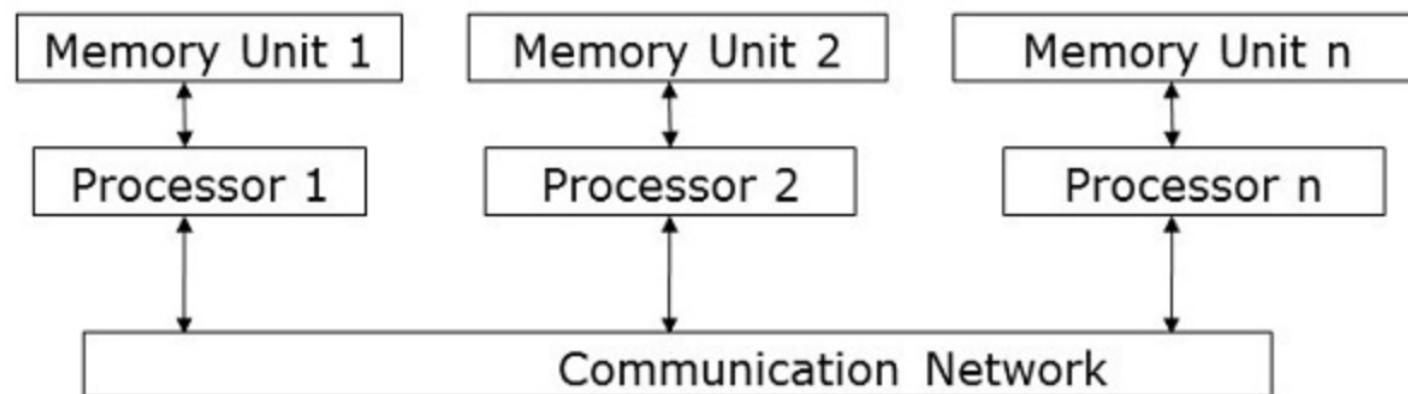
- Adressage globale
- Accès rapide aux données partagées
- Pas besoin de spécifier la communication
- Facile à implémenter

➤ Inconvénients

- Absence de portabilité
- Gestion locale des données difficile

Modèles de messages (1)

- Modèles le plus utilisé en programmation parallèle
- Chaque processeur a une mémoire locale
- Les processeurs échangent des données via un réseau de communication
- Les processeurs utilisent des bibliothèques pour la communication entre eux



Modèles de messages (2)

- Trois grandes méthodes de communication par message :
 - Communication point à point
 - Communication collective
 - Interface de passage de message

Modèles de messages (3)

Communication point à point

- Forme la plus simple de passage de message.
- Un message peut être envoyé du processeur émetteur à celui récepteur via deux modes :
 - **Mode synchrone** - Le message suivant est envoyé uniquement après avoir reçu une confirmation que son message précédent a été envoyé, pour maintenir la séquence du message.
 - **Mode asynchrone** - Pour envoyer le message suivant, il n'est pas nécessaire de recevoir la confirmation de la réception du message précédent.

Modèles de messages (4)

Modèles de communication collective

- Plus de deux processeurs sont concernés pour le passage des messages.
 - **Mode barrière** - Possible si tous les processeurs inclus dans la communication exécutent un bock particulier (connu sous le nom de bloc de barrière) pour le passage du message.
 - **Broadcasting** – Le broadcasting est de deux types
 - **One-to-All** - Ici, un processeur avec une seule opération envoie le même message à tous les autres processeurs.
 - **All-to-All** - Ici, tous les processeurs envoient un message à tous les autres processeurs.

Modèles de messages (5)

➤ **Avantages**

- Fournit un faible contrôle du parallélisme;
- Portabilité;
- Moins susceptible aux erreurs;
- Moins de coût dans la synchronisation parallèle et la distribution de données.

➤ **Inconvénients**

- Nécessite généralement un coût logiciel plus élevé

Interface de passage de message

Il existe de nombreuses bibliothèques pour la transmission de messages.

- Ici, nous discuterons des deux des bibliothèques de messagerie les plus utilisées
 - **Interface de passage de message (MPI)**
 - **Machine virtuelle parallèle (PVM)**

Interface de passage de messages (1)

- Norme universelle pour fournir une communication entre tous les processus simultanés dans un système de mémoire distribuée.
- La plupart des plates-formes informatiques parallèles couramment utilisées fournissent au moins une implémentation d'interface de passage de message.
- Implémenté comme une collection de fonctions prédéfinies appelées bibliothèque et peut être appelé à partir de langages telles que C, C ++, Fortran, etc.

Interface de passage de messages (2)

➤ Avantages

- Fonctionne uniquement sur des architectures de mémoire partagée ou des architectures de mémoire distribuée;
- Chaque processeur possède ses propres variables locales;
- Par rapport aux gros ordinateurs de mémoire partagée, les ordinateurs de mémoire distribués sont moins chers.

➤ Inconvénients

- D'autres modifications de programmation sont nécessaires pour l'algorithme parallèle;
- Parfois difficile à débarrasser;
- Ne fonctionne pas bien dans le réseau de communication entre les nœuds.

Machine virtuelle parallèle (PVM)

- PVM est un système de transmission de message, conçu pour connecter **plusieurs machines hôtes hétérogènes** distinctes pour former **une seule machine virtuelle**.
- Gère les tâches suivantes:
 - tout le routage des messages,
 - la conversion des données,
 - la planification des tâches dans le réseau d'architectures informatiques incompatibles.

Machine virtuelle parallèle (PVM)

- Domaines d'Applications :
- Grands problèmes de calcul tels que
 - Études de supraconductivité,
 - Simulations de dynamique moléculaire
 - Algorithmes matriciels
- Exemples :
 - Grilles de calcul
 - Data centers

Caractéristiques d'un PVM

- Très facile à installer et à configurer;
- Plusieurs utilisateurs peuvent utiliser PVM en même temps;
- Un utilisateur peut exécuter plusieurs applications;
- Supporte C, C ++, Fortran;
- Pour une série donnée d'un programme PVM, les utilisateurs peuvent sélectionner le groupe de machines;
- C'est un modèle de passage de message, Calcul par processus;
- Supporte une architecture hétérogène.

Modèles parallèle de données

- Consiste à effectuer des opérations sur un ensemble de données simultanément.
- L'ensemble de données est organisé en une structure comme un tableau, un hypercube, etc.
- Les processeurs effectuent des opérations collectivement sur la même structure de données.
- Chaque tâche est effectuée sur une partition différente de la même structure de données.
- Applicables qu'à certains algorithmes

Algo parallèles et structures de données

Structures de données

- Pour appliquer correctement un algorithme, il est très important que vous choisissez une structure de données appropriée.
- **Une même opération effectuée sur une structure de données peut prendre beaucoup plus de temps si elle est effectuée sur une autre structure de données.**
- Pour sélectionner une structure de données, il faut tenir compte de l'architecture et du type d'opérations à effectuer.
- Structures de données couramment utilisées :
 - Liste liée
 - Tableau
 - Réseau Hypercube

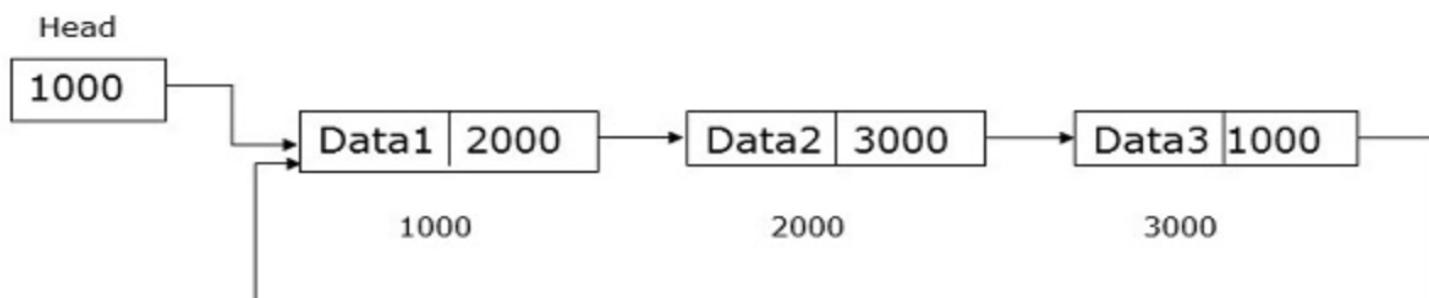
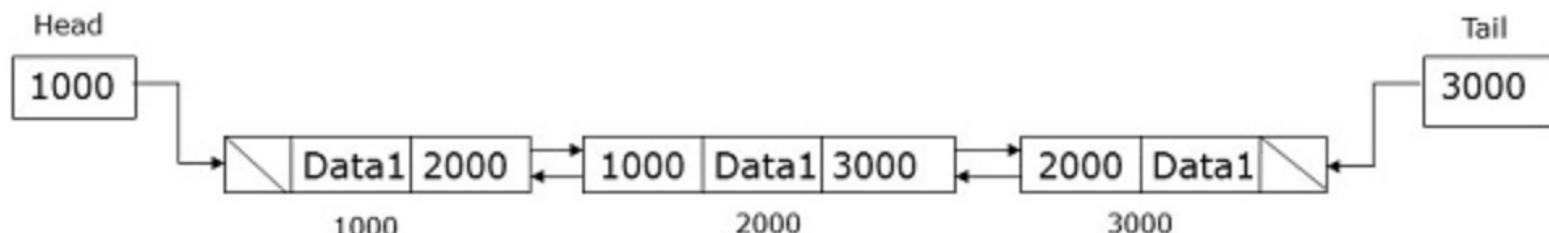
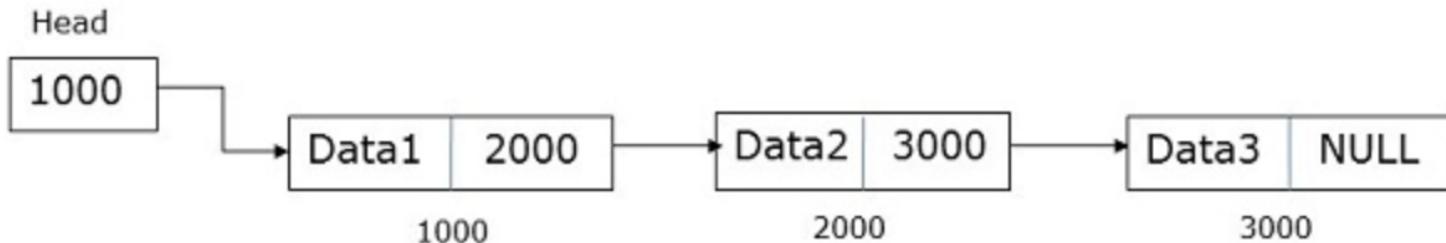
Les Listes chainées

- Une liste chainée est une structure de données ayant zéro ou plus de nœuds connectés par des pointeurs.
- Les noeuds peuvent ou non occuper des emplacements de mémoire consécutifs.
- Chaque nœud comporte deux ou trois parties: une partie de données qui stocke les données et les deux autres sont des champs de liens qui stockent l'adresse du noeud précédent ou suivant.
- L'adresse du premier nœud est stockée dans un pointeur externe appelé tête.
- Le dernier nœud, connu sous le nom de queue, ne contient généralement aucune adresse.

Les Listes chainées

➤ Il existe trois types de listes liées:

- Liste individuelle liée
- Liste doublement liée
- Liste liée circulaire



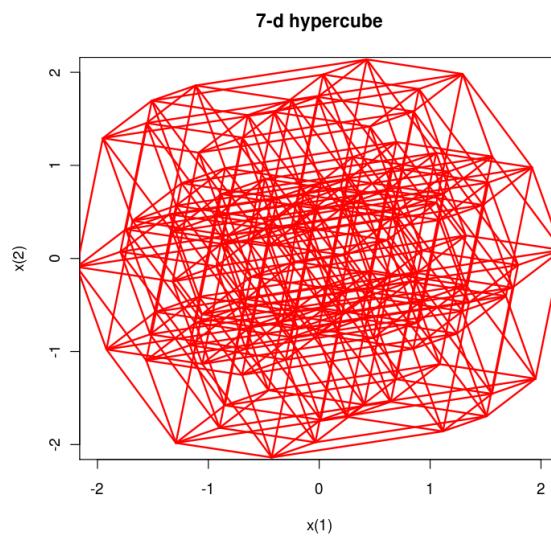
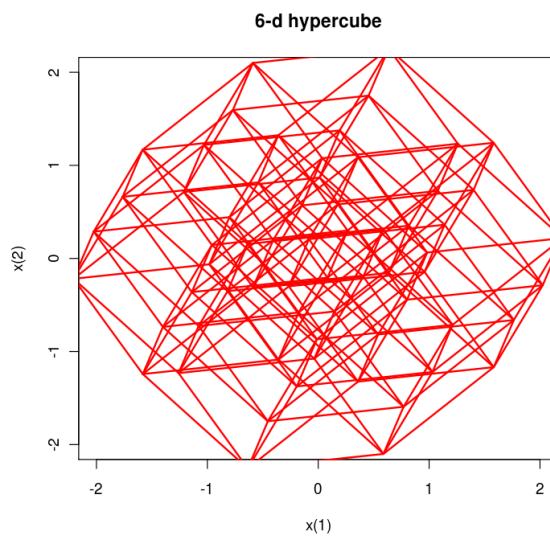
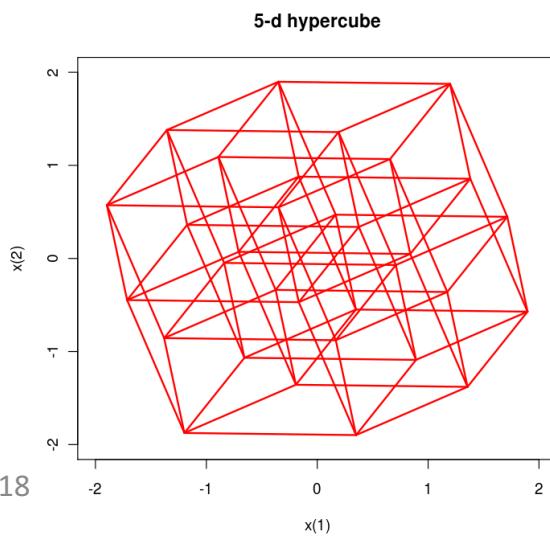
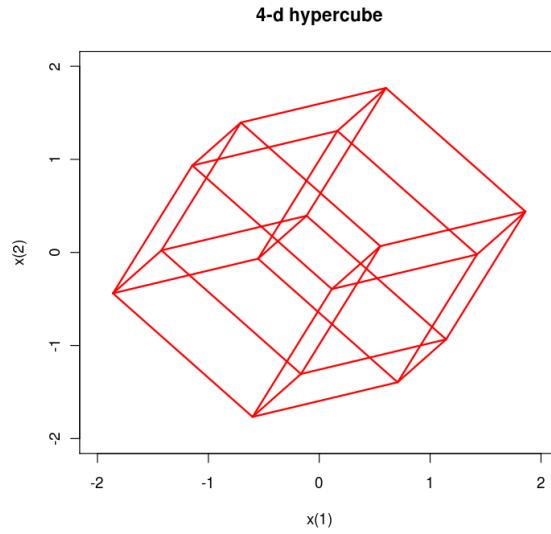
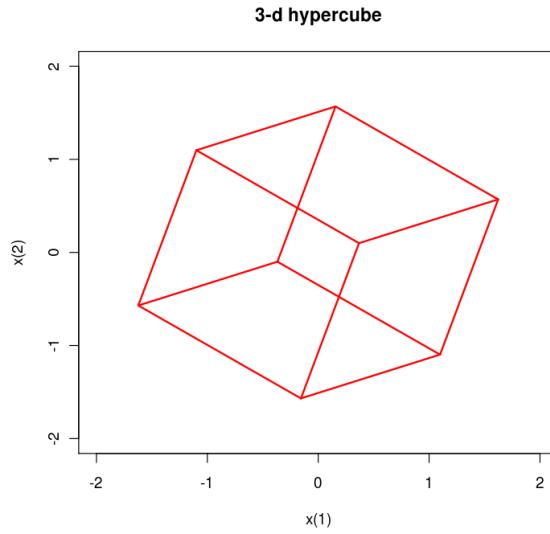
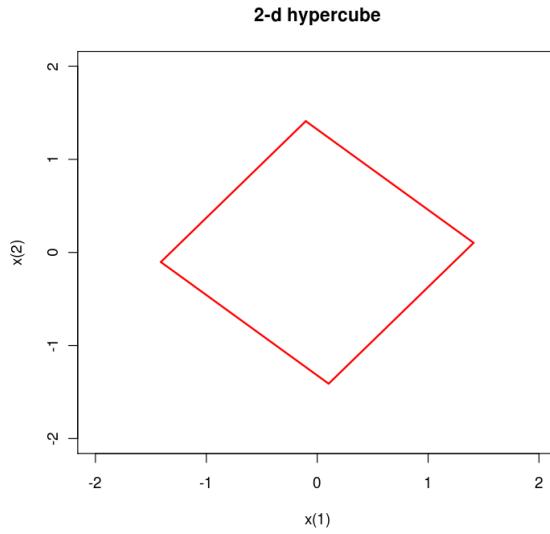
Les tableaux

- Un tableau est une structure de données où on peut stocker des types de données similaires.
- Il peut être unidimensionnel ou multidimensionnel.
- Les tableaux peuvent être créés statiquement ou dynamiquement.
 - Dans les tableaux déclarés statiquement, la dimension et la taille des tableaux sont connues au moment de la compilation.
 - Dans les tableaux dynamiquement déclarés, la dimension et la taille du tableau sont connues au moment de l'exécution.
- Pour la programmation de la mémoire partagée, les tableaux peuvent être utilisés comme mémoire commune et pour la programmation en parallèle de données, ils peuvent être utilisés en divisant en sous-tableaux.

Les Hypercubes

- L'architecture Hypercube est utile pour les algorithmes parallèles dans lesquels chaque tâche doit communiquer avec d'autres tâches.
- La topologie Hypercube peut intégrer facilement d'autres topologies telles que l'anneau et la maille.
- Il est également connu sous le nom de n-cubes, où n est le nombre de dimensions.
- Un hypercube peut être construit récursivement.

Les Hypercubes



Programmation Parallèle

- Langage de programmation
 - Occam
 - CUDA
 - OpenCL
 - ...

Occam

- Occam est un langage de programmation concurrent qui s'appuie sur la logique de **processus séquentiels communicants** (CSP)[1].
- Nommé d'après William Ockham du célèbre Rasoir d'Occam.
- Occam est un langage de procédure impérative (comme Pascal).
- Il a été développé par David May et d'autres à l'INMOS, conseillé par **Tony Hoare**, comme langage de programmation native pour leurs microprocesseurs de transputer, mais des mises en œuvre pour d'autres plates-formes sont disponibles.
- La version la plus connue est occam 2

Occam

- La communication entre les processus fonctionne à travers les canaux appelés « channels ».
- Un processus produit les données vers un canal via "!" Tandis qu'un autre saisit des données avec "?".
- L'entrée et la sortie ne peuvent se poursuivre que lorsque l'autre extrémité est prête à accepter ou à offrir des données. (Dans le cas «pas en cours», on dit souvent que le processus «bloque» sur le canal.

Occam

➤ Exemples : Soit c est une variable

```
keyboard ? c
```

```
screen ! c
```

```
SEQ
```

```
  x := x + 1  
  y := x * x
```

```
PAR
```

```
  p()  
  q()
```

Occam (Transputers)



Inmos T212, PREQUAL



Inmos T222, PREQUAL



STMicroelectronics
IMST225 (Inmos T225).



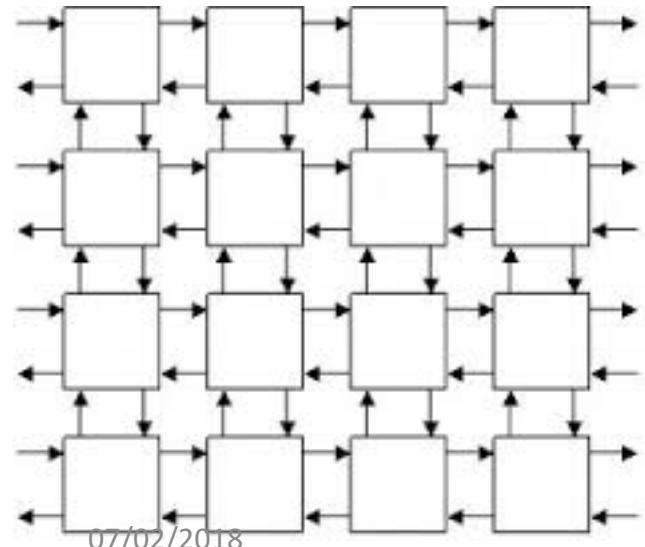
Inmos T400



Inmos T414



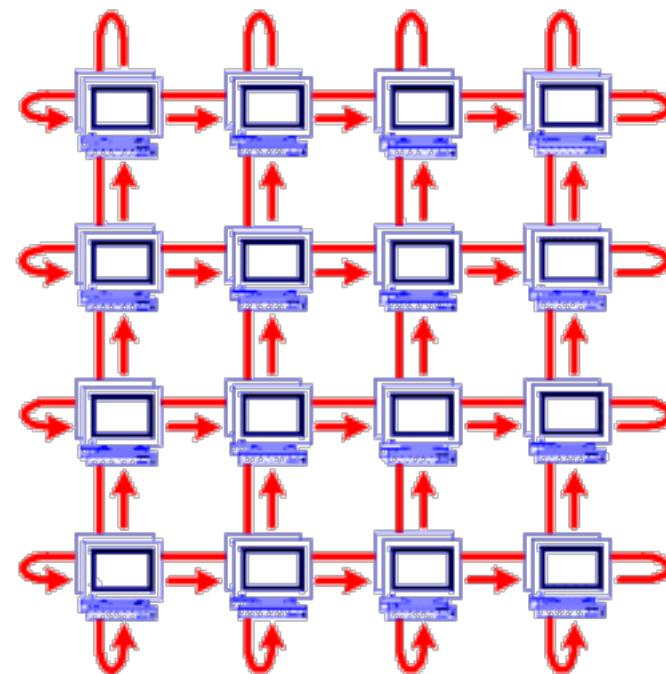
Inmos T425



Les **Transputers** étaient une série de microprocesseurs pionniers des années 1980, avec des mémoire et des liens de communication intégrées, destinées à l'informatique parallèle (Occam).

Conçu et produit par **Inmos**, une société de semi-conducteurs basée à Bristol, au Royaume-Uni.

Occam : exemples



CUDA : Qu'est-ce que CUDA?



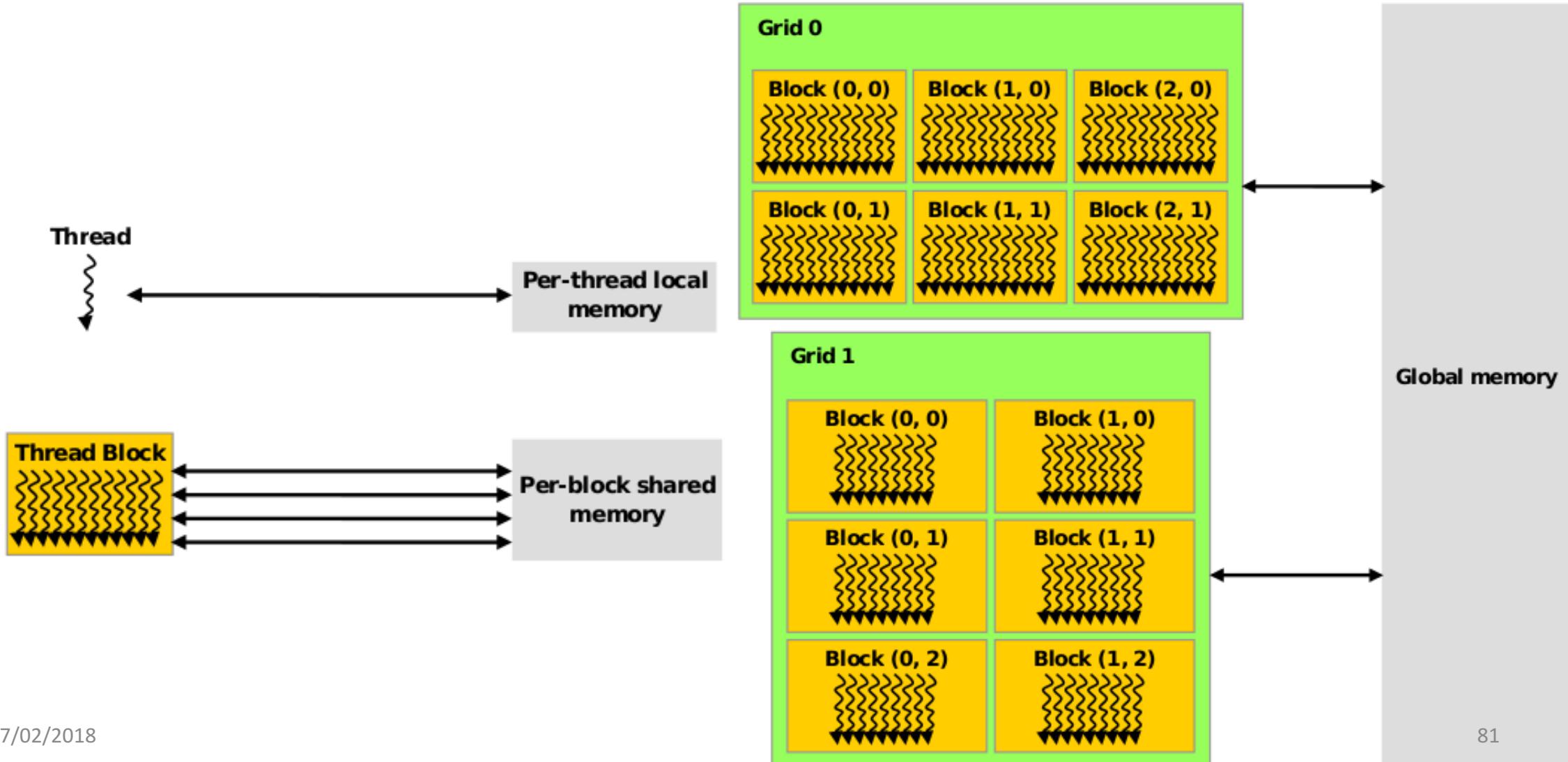
- **Une plate-forme informatique parallèle et un modèle d'interface de programmation d'applications (API) créé par Nvidia.**
- Permet aux développeurs et aux ingénieurs logiciels d'utiliser une unité de traitement graphique (GPU) pour augmenter les performances d'un programme.
- Utilise le processeur (CPU) et la carte graphique (GPU) pour des programmes parallèle

CUDA



- **GPU ?** L'unité de traitement graphique (GPU), en tant que processeur spécialisé, répond aux exigences des tâches en 3D à haute résolution en calcul haute intensité de calcul.
- **Contexte?** Depuis 2012, les GPU sont devenus des systèmes multi-core très parallèles permettant une manipulation très efficace de gros blocs de données en mémoire.
- **GPU vs CPU ?** Plus efficace que l'unité de traitement central (CPUs) à usage général pour les algorithmes dans les situations où le traitement de gros blocs de données se fait en parallèle,

CUDA : gestion de mémoire





• Flux de traitement simple

1. Copier les données d'entrée de la mémoire du processeur (CPU) vers la mémoire graphique (GPU)
2. Charger le programme GPU et exécuter, mettre en cache les données pour optimiser les performances
3. Copier les résultats de la mémoire GPU vers la mémoire CPU



Fin