Module : Architecture Parallèle

Introduction

L'architecture est l'art ou science de construction, méthode ou style de construction. Donc l'architecte d'un ordinateur développe des spécifications performantes pour une variété de composants de l'ordinateur et définie les interconnexions entre eux. A son tour le concepteur d'ordinateur va raffiner les spécifications de ces composants et l'implémenter par la suite en utilisant le matériel, le logiciel, et les éléments Firmeware.

Le développement des besoins en informatique a conduit à faire évoluer les ordinateurs vers des puissances de plus en plus grandes. L'un des points essentiels de ces développements porte sur l'architecture des machines, c'est-à-dire la manière dont sont agencés les différents composants : processeurs, mémoires, connexions, canaux d'entrées−sorties, etc..
Le mode de fonctionnement classique consiste à faire effectuer, séquentiellement, un programme par la machine suivant la procédure décrite par John Von Neumann (1945), le pionnier de l'informatique telle que nous la connaissons aujourd'hui.
A tout instant du programme, une seule instruction, portant sur une seule variable ou un couple de variables, est exécutée. Si le traitement comporte plusieurs opérations, celles−ci doivent être effectuées séquentiellement, une opération ne pouvant commencer que lorsque la précédente est terminée.

Historique et évolution des différentes architectures

1. **Ordinateurs de la première génération (1938-1953)**

Les machines sont construites avec des tubes (lampes) à vide, des résistances et des relais ; elles occupent une place considérable et tombent facilement en pannes (ENIAC, ENDVAC,..).

2. **Ordinateurs de la deuxième génération (1953-1963)**

Les transistors sont apparus en 1948 et des machines sont commercialisées (par IBM).

3. **Ordinateurs de la troisième génération (1963-1975)**

Les circuits intégrés (conception sur support de silicium) provoquent un saut technologique ; on construit pour un encombrement très réduit des processeurs, des mémoires,… qui sont fiables.

4. **Ordinateurs de la quatrième génération (1975-1990)**

La densité d'intégration des composants augmente encore (VLSI), les puces tiennent moins en moins de place, c'est la naissance de microprocesseur.

5. **Ordinateurs de la cinquième génération (199-…)**

C'est la génération des architectures parallèles (architectures non Von Newman) et la généralisation des réseaux.

Insuffisance des architectures de Von Newman

La structure traditionnelle d'un ordinateur, composée d'un seul processeur envoyant des requêtes successives sur un bus à une mémoire qui répond à une demande à la fois, est souvent appelée goulet d'étranglement de Von Neumann.
Un grand nombre de recherche ont été menées pour éliminer ce goulet de d'étranglement.
Ces recherches ont conduit à de nombreuses architectures parfois très différentes les unes des autres.
Toutes les recherches ont en commun le fait de chercher à faire travailler en parallèle plusieurs unités de commandes, plusieurs UAL et plusieurs modules de mémoire. Elles se diffèrent par le nombre d'unités de chaque catégorie et la façon d'interconnexion.

+Motivations du Parallélisme

Depuis le début de l'informatique s'est posée la question de résoudre rapidement des problèmes (le plus souvent numérique) coûteux en matière de temps de calcul : Simulation numérique, cryptographie, imagerie, SGBD,…etc.

Pour résoudre plus rapidement un problème donné, une idée naturelle consiste à faire coopérer simultanément plusieurs agents à sa résolution, qui travailleront donc en parallèle.

1. **Besoins des applications**

- Performances : Apparition des applications qui demandent plus de puissances : plus de ressources de traitement (CPU). Telles que :
    1. Simulation et calcul numérique.
    2. Comparaison des séquences biologiques (ADN, Protéines,…).
    3. Applications graphiques : traitement et synthèse d'images,…
    4. Algèbre linéaire : manipulation des vecteurs et des matrices de grande taille.
    5. Traitement de signal : Transformé de fourrier,...
    6. Serveurs : BD, Vidéo, moteurs de recherche, serveur WEB, …
    7. Réalité virtuelle, Augmentée, …

2. **Limite des architectures de Von Newman**

La majorité des machines actuelles s'appuient sur le modèle conventionnel de Von Newman. Cependant cette dernière souffre de plusieurs problèmes :

- Performances faibles par rapport aux exigences des nouvelles applications qui demandent de plus en plus de puissance malgré l'évolution rapide des performances.
- Capacité d'accès mémoire limité
- Problème de tolérance aux pannes (résistances aux pannes matérielles)
- Variétés de formats de données et des opérations (inadéquation de ces formats de données vis-à-vis aux caractéristiques de certaines applications telles que le traitement des images ou les comparaisons des séquences biologiques).
- …

3. **Existence des propriétés de parallélismes dans les applications**

Beaucoup d'applications présentent des propriétés de parallélismes, autrement dit elles peuvent être l'objet d'une éventuel exécution parallèle en cas de présences de ressources matérielles suffisantes.

Concept de Parallélisme

Le parallélisme d'une application consiste à faire exécuter simultanément plusieurs parties indépendantes de cette application sur plusieurs ressources matérielles.

Selon les structures des applications, il existe deux formes de parallélismes :

- Parallélisme de contrôle
- Parallélisme de données

- **Parallélisme de données**

C'est un type de parallélisme pour lequel la même opération est réalisée simultanément par de nombreux processeurs sur des données différentes.

Exemple : l'addition de deux matrices A et B. elle consiste, pour tous les éléments de même indices des deux matrices opérandes à les additionner et à stocker le résultat dans l'élément de même indice d'une troisième matrice résultat C.

```
Pour i = 1 à N faire
  Pour J = 1 à M faire
    C[i,j] ← A[i,j] + B[i,j]
  Fin_pour
Fin_pour
```

Les N*M itérations de cette boucle sont indépendantes, on peut donc faire N*M opérations simultanées. L'ampleur du potentiel de parallélisme exploitable dépend directement de la taille des structures de données manipulées.
Exemple d'application de ce parallélisme :

- Applications numériques
- Les traitements de base de données (avec des milliers d'entées)
- Le traitement du signal et de l'images (matrices de plusieurs milliers d'éléments : pixels).

Il s'agit dans ce type de parallélisme, d'utiliser de nombreux processeurs simultanément et de leur faire exécuter la même opération. Généralement le nombre de processeurs est beaucoup plus petit que le parallélisme potentiel et chaque processeur devra donc traiter plusieurs données.

- **Parallélisme de contrôle**

C'est un type de parallélisme pour lequel des opérations différentes sont réalisées simultanément. Ce parallélisme peut provenir de l'existence dans le programme de fonctions indépendantes (parties indépendantes) ou d'opérations indépendantes dans une suite d'opérations. C'est l'absence de dépendance entre différentes parties de programme qui est la source de parallélisme.
Ce parallélisme ne dépend donc pas des données mais de la structure du programme à exécuter.
Les systèmes d'exploitation des ordinateurs modernes sont des exemples de ce type de parallélisme.
Un exemple d'exploitation du parallélisme de contrôle est le pipeline. Il repose sur l'organisation des données à traiter sous la forme d'un flux d'informations sur l'application de plusieurs traitements consécutifs sur chaque élément de flux.

Un autre exemple de ce type de parallélisme, les serveurs :

```
Server
Cycle
  Détecter_demande_client
  Si (demande_client = Vrai) Alors
      Lancer traitement_Client()
  FinSi
Fin_cycle
```

```
Traitement_client()
Début
 Ouvrir_fichier_client
 Faire traitement_client
 Fermer_fichier_client
Fin
```

Supposons que la fonction Lancer retourne immédiatement après avoir lancé le programme (processus) de traitement_client() ; c-a-d sans attendre que celui-ci se termine. Dans ce cas, le programme du serveur continue à exécuter alors que le programme traitement_client() est en

cours d'exécution. S'il y a suffisamment de ressources deux traitements, ces deux programmes seront exécutés simultanément.
Si le serveur reçoit une nouvelle demande_client, il lancera l'exécution d'un nouveau traitement_client() sur une autre ressource de traitement.

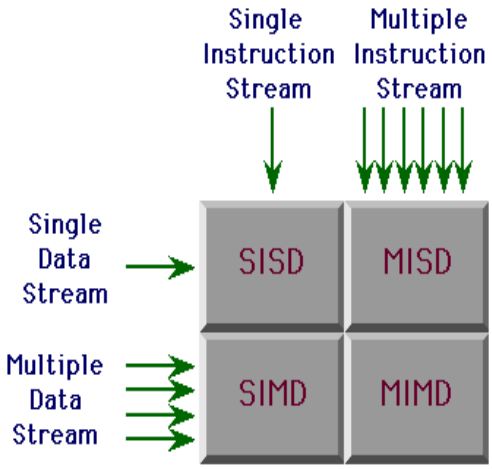
Architecture Parallèle
C'est une architecture composée de plusieurs ressources de traitement (processeurs,…), pas nécessairement identique, coopérant pour l'exécution d'une application.

Classification des architectures parallèles
1. Classification de flynn
La classification des architectures parallèles la plus utilisée est celle de flynn (1966).
Cette classification repose sur deux concepts : Le flux (Stream) des instructions et le flux des données.

- Un flux d'instructions correspond au compteur ordinal. Un système doté de N CPU dispose de N compteurs ordinaux et donc de N flux d'instructions

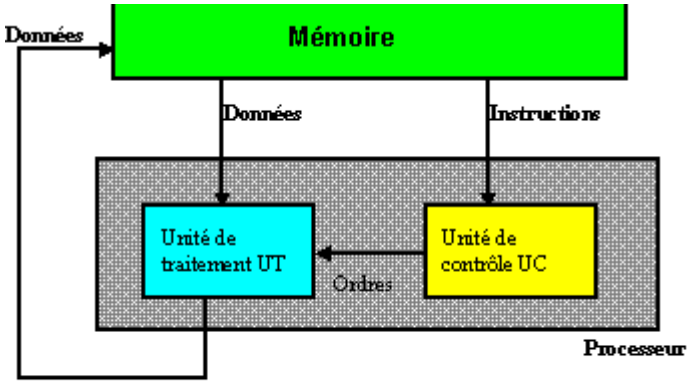
- Un flux de données est un ensemble d'opérandes ; Un programme calculant la moyenne pluviométrique d'une ville a un flux de données, mais un programme calculant les moyennes pluviométriques de N ville a N flux de données.

Ces deux flux (instructions et données) étant indépendants, il y a donc quatre combinaisons possibles :



Flux d'instructions	Flux de Données	Nom d'architecture	Exemple
1	1	SISD	Machine de Von N.
1	Multiple	SIMD	Machine vectorielle
Multiple	1	MISD	Machine Pipeline ?
Multiple	Multiple	MIMD	Multiprocesseur

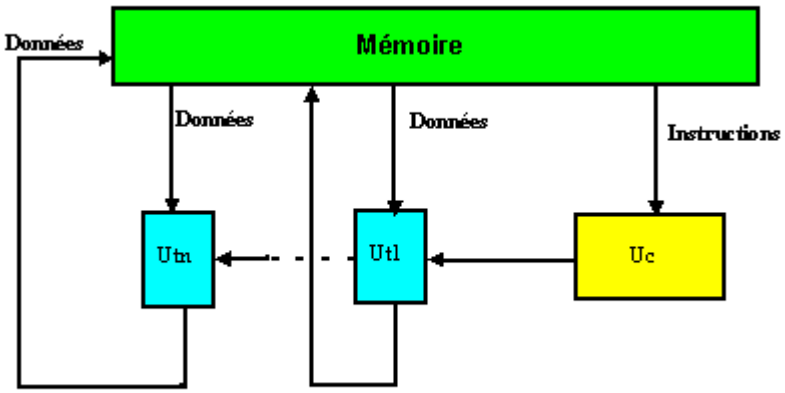
- **SISD** (un seul flux d'instructions et un seul flux de données) (**S**ingle **I**nstruction stream over **S**ingle **D**ata stream) Machine séquentielle classique correspond à la machine de Von Neumann classique. Un flux d'instruction exécute une seule chose à la fois sur un flux de données.



Modèle de Fonctionnement séquentiel

- **SIMD** (une seule instruction sur de multiples données) (**S**ingle **I**nstruction stream over **M**ultiple **D**ata stream) (Système multi-processeur exécutant le même traitement sur des données différentes) ont une seule unité de commande qui n'exécute qu'une seule instruction à la fois mais elles possèdent de nombreuses UAL capables d'appliquer cette instruction à plusieurs jeux de données simultanément. Les ordinateurs vectoriels sont des exemples de machines SIMD.

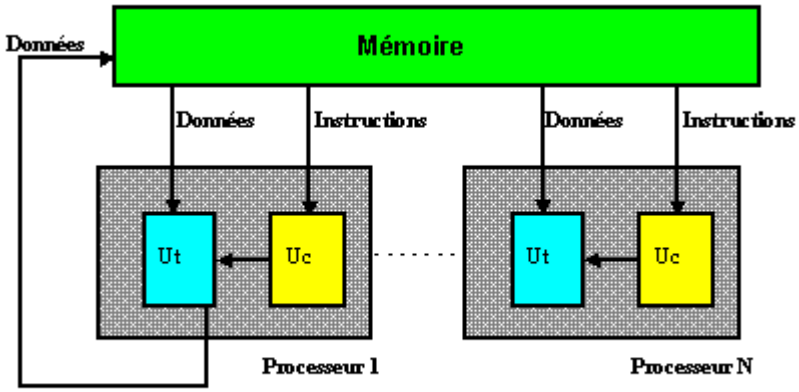
-



Modèle de Fonctionnement S.I.M.D

- **MISD** (multiple flux d'instructions sur un seul flux de données) (Système dans lequel un même flux de donné est passé à travers différents processeurs). Le mode pipeline d'exploitation du parallélisme de contrôle correspond assez bien à cette classe. Cependant, certains disent que les machine MISD n'existent pas.

- **MIMD** (multiples flux d'instructions sur de multiples flux de données) correspondent à des CPU indépendants faisant partie d'un ensemble plus complexe. La plupart des architectures parallèles appartiennent à cette catégorie (à mémoire partagée / passage de message à mémoire privée).



Ce sont des machines multi-processeurs où chaque processeur exécute son code de manière asynchrone et indépendante. Pour assurer la cohérence des données, il est souvent nécessaire de synchroniser les processeurs entre eux, les techniques de synchronisation dépendent de l'organisation de la mémoire
On distingue pour cela 2 types d'architectures :

1. MIMD à mémoire partagée

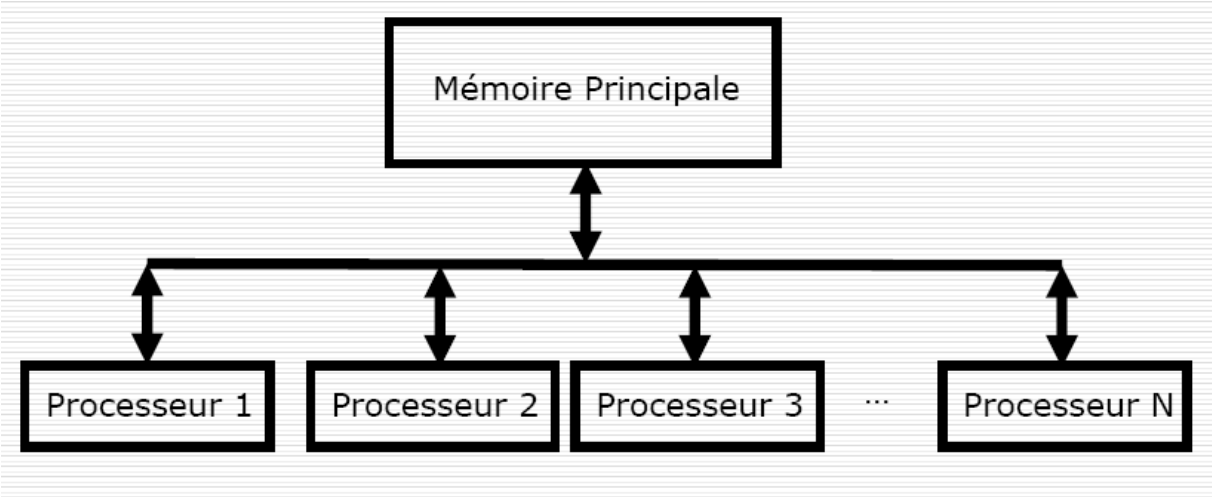
Les processeurs accèdent à une mémoire commune : la synchronisation peut se faire au moyen de:

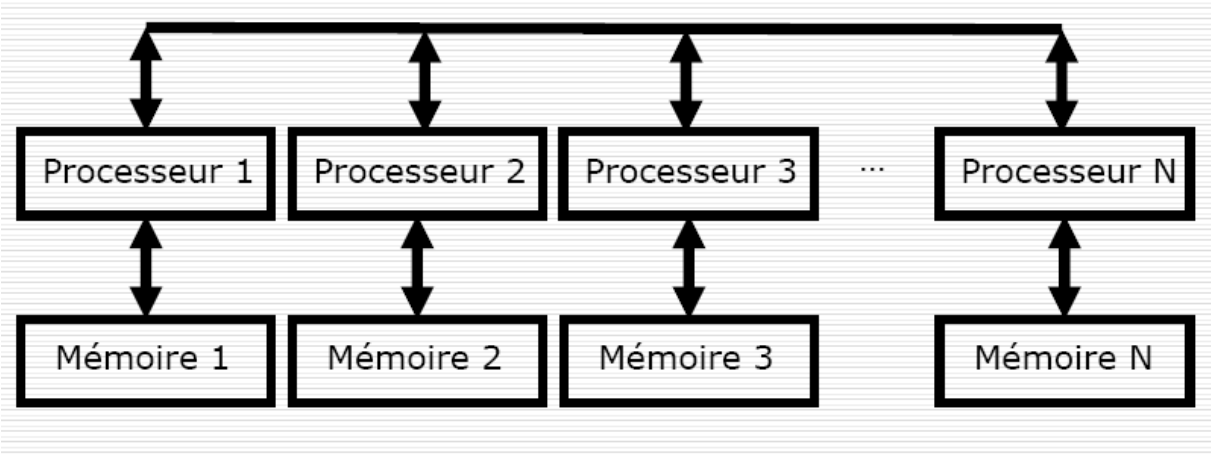
- sémaphores
- verrous, ou Mutex (exclusion mutuelle)
- barrières de synchronisation

2. MIMD à mémoire distribuée

Chaque processeur dispose de sa propre mémoire. Cette architecture nécessite un réseau d'interconnexion pour la synchronisation. On citera pour l'échange d'informations entre les processeurs :

- appel de procédure distante RPC
- envoi / réception de messages synchrone, ou asynchrone





2. Autre Classification (Raina)

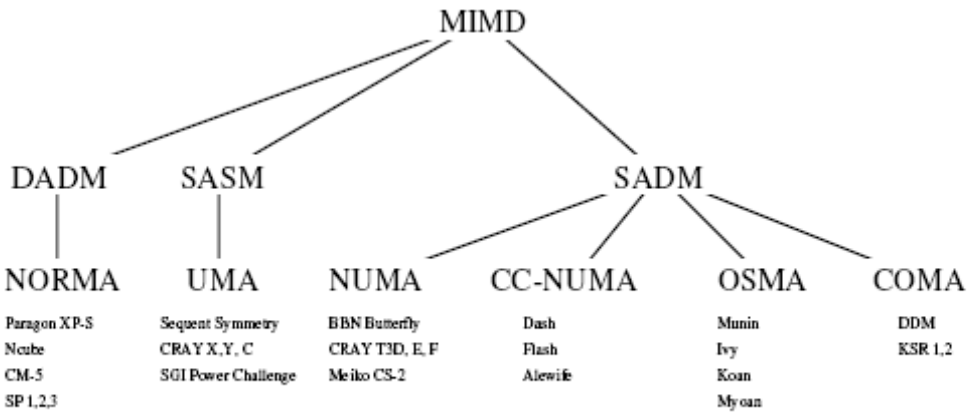
Une sous-classification étendue des machines MIMD, due `a Raina, permet de prendre en compte de manière fine les architectures mémoire, selon deux critères :

– l'organisation de l'espace d'adressage :

- **SASM**(« Single Address space, Shared Memory ») : mémoire partagée ;
- **DADM** (« Distributed Address space, Distributed Memory ») : mémoire distribuée, sans accès aux données distantes. L'échange de données entre processeurs s'effectue nécessairement par passage de messages, au moyen d'un réseau de communication;
- **SADM** (« Single Address space, Distributed Memory ») : mémoire distribuée, avec espace d'adressage global, autorisant éventuellement l'accès aux données situées sur d'autres processeurs.

– le type d'acc`es m´emoire mis en oeuvre :

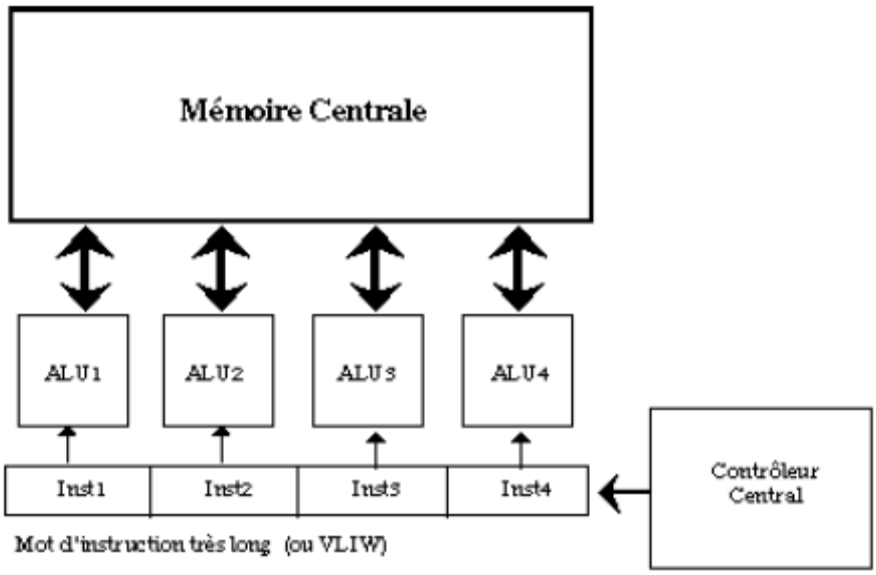
- **NORMA** (« No Remote Memory Access ») : pas de moyen d'accès aux données distantes, nécessitant le passage de messages ;
- **UMA** (« Uniform Memory Access ») : accès symétrique à la mémoire, de coût identique pour tous les processeurs ;
- **NUMA** (« Non-Uniform Memory Access ») : les performances d'accès dépendent de la localisation des données ;
- **CC-NUMA** (« Cache-Coherent NUMA ») : type d'architecture NUMA intégrant des caches ;
- **OSMA** (« Operating System Memory Access ») : les accès aux données distantes sont gérées par le système d'exploitation, qui traite les défauts de page au niveau logiciel et gère les requêtes d'envoi/copie de pages distantes ;
- **COMA** (« Cache Only Memory Access ») : les mémoires locales se comportent comme des caches, de telle sorte qu'une donnée n'a pas de processeur propriétaire ni d'emplacement déterminé en mémoire.



1. **Architecture MISD**

- **Architecture Pipeline (voir le cours architecture des ordinateurs 2 « 3ème ingénieur »**

- **Architecture VLIW**



Le VLIW (Very Long Instruction Word) est une architecture assez proche du modèle MISD. Elle est formée d'instructions très longues, qui contiennent de façon explicite plusieurs opérations (instructions) à effectuer en parallèle sur plusieurs unités indépendantes.

- Cette architecture est déduite directement de l'architecture RISC.
- Elle reprend le principe de recherche de la vitesse du processeur.
- Objectif : augmenter la vitesse d'exécution au maximum en répliquant les unités de traitement (UAL ou FPU) pour exécuter encore plus d'instructions en même cycle.
- En général, le nombre d'UAL est de quelques unités (2, 4 ou 6).
- Une instruction VLIW est une instruction très longue, composée par la concaténation d'instructions destinées aux N UAL constituant le processeur.
- A chaque cycle RISC, le contrôleur centrale délivre une VLIW est alimente ainsi en parallèle le contrôle des UAL d'une manière purement synchrone.

- Le modèle d'exécution VLIW généralise le pipeline et l'étude des dépendances des données, il s'agit de partir d'un programme séquentiel classique, et de générer un code machine qui soit :
    - compatible avec un fonctionnement pipeline (étude des branchement et des données).
    - Compatible avec un fonctionnement en flots parallèles sur N UAL.

+ Ce mode de fonctionnement présente des analogies avec le modèle SIMD, à ceci près que les instructions délivrées aux UAL sont à priori différentes les unes des autres.

+ Le compilateur qui a la responsabilité de générer le code en format VLIW à partir d'un programme source. Il doit faire une analyse complète du graphe de dépendances du programme pour gérer les affectations des registres en long (Pipeline) et en large (UAL en parallèle).

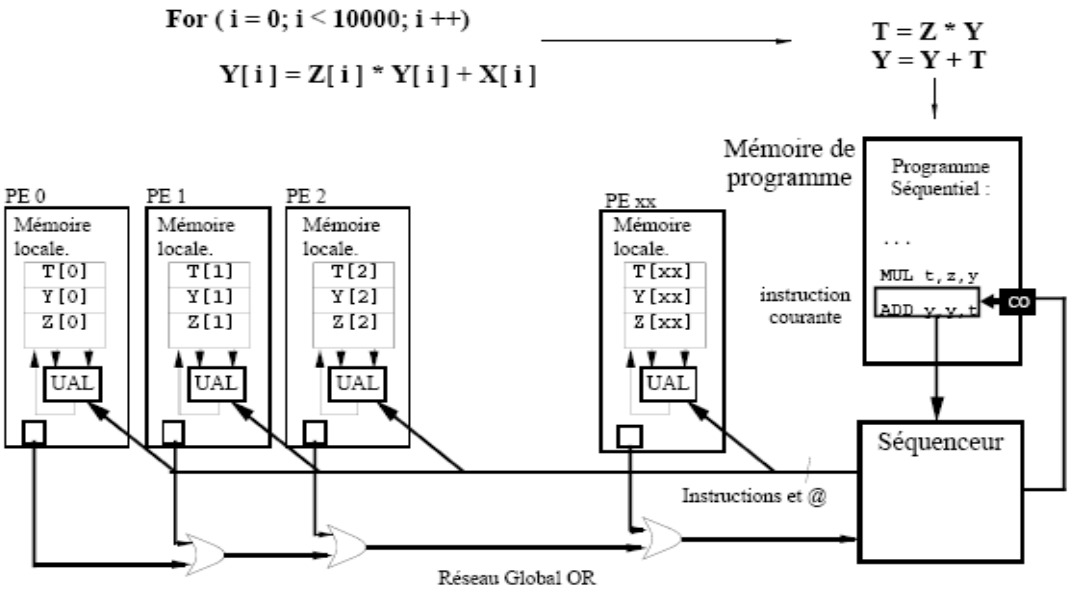
2. Architecture SIMD

Ce modèle est l'un des premiers modèles d'architecture parallèle. Exemple le CRAY 1.
L'architecture SIMD est désormais utilisée essentiellement pour des applications très spécialisées.

- Le modèle SIMD a tiré sa popularité de sa simplicité pour le programmeur et pour l'architecte et le constructeur de machine.
- Le modèle SIMD n'exploite que le parallélisme de données.
- Il s'agit d'appliquer la même opération à une très grande quantité d'informations
- Machines spécialisées dans les traitements des tableaux (ARRAY Processors).
- Modèle d'exécution fortement synchrone.
- L'architecture des machines SIMD est définie directement à partir des caractéristiques du modèle SIMD
- Caractérisée par un très grand nombre d'unités de traitement simples (généralement UAL)
- Une unité de contrôle centralisée :
    - déroulant le programme,
    - exécutant les opérations scalaires
    - indiquant les opérations parallèles à réaliser par les unités de traitement.

Exemple

Il s'agit d'exécuter une multiplication-addition de vecteurs. Les trois tableaux opérandes sont Z, Y et X. Le tableau résultat est Y. Les opérations d'addition et de multiplication doivent être appliquées à tous les indices de tableaux opérandes et résultat. Le programme séquentiel est présenté à gauche. L'expression en notation tableau de boucle est représentée à droite. Il s'agit d'exécuter d'abord le produit qui sera stocké dans un tableau intermédiaire T. Puis, l'addition sera effectuée à partir de T.



Le programme de la machine SIMD comporte donc l'opération de multiplication. Cette opération est lue par le séquenceur et l'instruction correspondante est diffusée à tous les Processeurs Elémentaires (PEi).
Avec l'instruction, le séquenceur diffuse l'adresse des tableaux opérandes. Dans ce cas très simple, chaque PE dispose d'une mémoire privée et il y a autant de PE et de mémoire que d'itérations de boucle et d'indices de tableaux. Les tableaux peuvent donc être distribués sur touts les PEs avec un élément par PE. Les tableaux T, X, Y et Z peuvent donc avoir les même adresses pour tous les PEs, mais les indices sont différents pour chaque PE.

Une seule instruction et seulement trois adresses (T, Z et Y) suffisent pour ordonner le calcul sur 1000 éléments (dans cette boucle). L'instruction suivante (Y := Y + T) se déroule de la même manière. Cette instruction n'est commencée que lorsque l'instruction précédente est terminée pour tous les PEs et le séquenceur.

Une seule instruction et seulement trois adresses (T, Z et Y) suffisent pour ordonner le calcul sur plusieurs éléments (dans cette boucle). L'instruction suivante (Y= Y + T) se déroule de l même manière. Cette instruction n'est commencée que lorsque l'instruction précédente est terminée pour tous les PEs et le séquenceur.

Le rôle du séquenceur est d'exécuter les parties contrôle et scalaire du programme et d'envoyer les instructions parallèles aux PEs. L'état de tous les PEs forme un état global de la machine. Dans certains cas, le séquenceur peut avoir besoin, pour exécuter la partie contrôle, d'obtenir l'état global. Ces cas correspondent au calcul d'un état global, la détection d'un état global ou la détection d'une transition d'état global.

Exemple d'application des architectures SIMD
Le modèle SIMD trouve actuellement un nouveau terrain d'exploitation à l'intérieur des microprocesseurs. Le MMX (Matrix Mach eXtensions) proposé par Intel, est une extension de traitement de matrices pour les microprocesseurs Pentium. L'objectif du MMX est d'accélérer le traitement des données pour les formats 8, 16 et 32 bits. Le principe est d'utiliser un opérateur (unité de traitement) 64 bits sécable capable de réaliser simultanément 8 opérations sur des opérandes 8 bits ou 4 opérations sur des opérandes de 16 bits ou encore 2 opérations

sur des opérations 32 bits. Ces opérations utilisent des registres MMX 64 bits dans lesquels sont regroupées les données 8, 16 et 32 bits.

Limites des architectures SIMD

- Les architectures SIMD ont d'abord été utilisées pour le calcul généraliste.
- Mais, leur principe fondamental (tous les PEs exécutent la même opérations en même temps) ne permet pas d'utiliser les microprocesseurs actuellement commercialisés comme PE.
- Avec des processeurs spécifiques, ces architectures deviennent trop coûteuses ou trop vite dépassées en performance.
- Les architectures SIMD sont utilisées dans le cadre spécifique de machines dédiées

Les machines vectorielles

- les architectures vectorielles sont des machines très coûteuses mais très performantes pour certaines applications.
- Exemple la machine CRAY (1976) est née suite à la nécessité d'exécuter les applications numériques plus rapidement que sur une machine séquentielle de l'époque.
- Architecture dédiée au traitement numérique.
- L'idée de cette architecture (modèle) est issue d'une analyse des applications numériques.
- Ces applications utilisent principalement
    - des algorithmes d'algèbre linéaire,
    - des méthodes de résolution de système d'équations aux dérivés partielles
    - des méthodes de résolution de systèmes d'équations différentielles
    - des transformées (Fourier,…)
    - …
- Ces algorithmes sont caractérisés par :
    - Des calculs en représentation à virgule flottante
    - Un même calcul est appliqué à un très grand nombre de données
- A partir de ces caractéristiques, voici le principe de architectures vectorielles :
    - Les opérateurs de calcul sont tous pipelinés grâce au mode de calcul sur les nombres.
    - Le même calcul est appliqué à un très grand nombre de données.

Exemple : l'addition flottante est décomposée en quatre étapes de pipeline :

- Utilisation de la norme IEEE 754 simples précisions (1 bit de signe, 8 bits d'exposant dont B = 127 et 23 bits de mantisse normalisée avec un bit caché).
- L'addition de deux flottants de signe X(Fx, Ex) et Y(Fy, Ey) correspond au calcul suivant : $X + Y = (Mx * 2^{(Ex-Ey)} + My) * 2^{Ey} (Ey)$, ce calcul est divisé en quatre étapes :
    1. la soustraction des exposants : $A = (Ex – Ey)$
    2. le décalage de la mantisse de X : $B = Mx * 2^{A}$
    3. l'addition des mantisses de X et de Y : $C = B + My$
    4. la normalisation de la mantisse de X + Y = C * 2^{Ey}
- Donc, si on utilise des unités indépendantes pour chaque étape, l'opération est pipelinée.

Principe : l'exécution vectorielle est un mode d'exploitation du parallélisme de données qui repose sur les boucles dans lesquelles le calcul utilise des éléments de tableaux, comme opérandes et comme résultats.
Exemple

```
FOR i :=1 to 10 DO
Begin
  A[i] := B[i] + C[i]
  D[i] := E[i] + F[i]
End
```

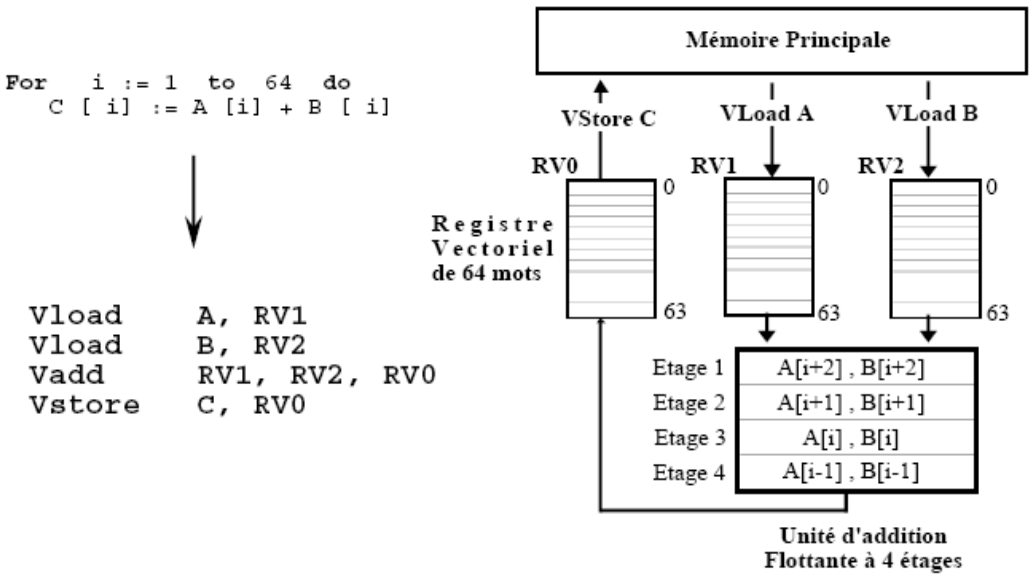
- L'exécution séquentielle consiste à calculer, dans l'ordre : A[1], D[1], A[2], D[2], ..., A[10], D[10]
  Donc, il est impossible de pipeliner ces calculs car les A[i], D[i] demandent des opérateurs différents.

- L'exécution vectorielle consiste à exécuter d'abord toutes les itérations correspondant au calcul de A[i], puis toutes celles de B[i] dans l'ordre : A[1], A[2], … A[10], D[1], D[2], …, D[10]

  - Ce mode de traitement des boucles garantit l'approvisionnement continu les opérateurs pipelinés.
  - Les boucles apparaissent comme des opérations sur des tableaux ou des vecteurs.

Architecture d'une machine vectorielle

- il s'agit de calculer sur des tableaux (vecteurs) et les opérations sont réalisées à partir de registres vectoriels.
- Le jeu d'instructions comprend des instructions vectorielles qui s'appliquent à tous les éléments stockés dans les registres vectoriels opérandes.
- Instructions de sémantique forte (type CISC), c'est l'équivalent d'une boucle.
- Architecture de type LOAD/STORE, uniquement ces instructions référencent la mémoire le reste s'appliquent sur les registres vectoriels.
- L'addition est pipelinée.



Performance

- l'exploitation pipelinée est la source des performances des architectures vectorielles.
    - Si le temps de traversé de chaque étage = 1 cycle, il faut 4 cycles pour exécuter une addition dans le cas séquentiel.
    - En cas de pipeline, un résultat est produit à chaque cycle (4 fois plus rapide).
    - Les 4 premiers cycles correspondent au démarrage de l'instruction vectorielle.
- les instructions LOAD/STORE sont aussi pipelinées

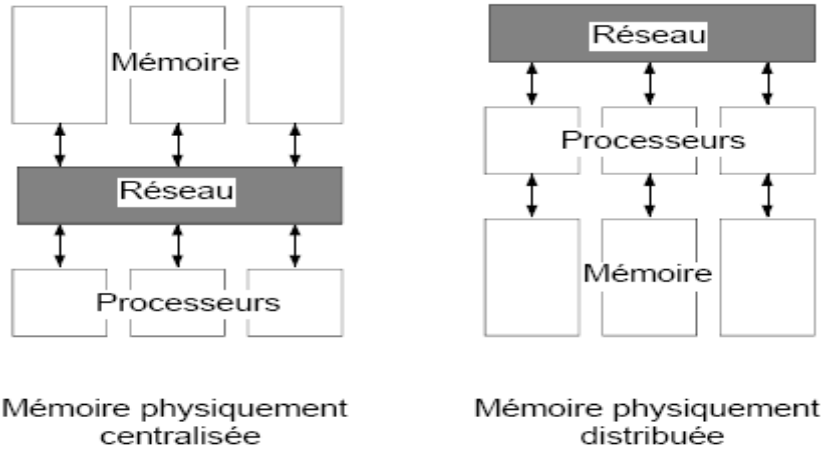
3. Architecture MIMD

- **Modèle à mémoire partagée**

- Modèle adopté par les grands constructeurs de machines (IBM, SGI, CRAY, DEC,…)
- Transformation d'un programme séquentiel en version parallèle (réutilisation des programmes déjà écrit en version séquentielle).
- Ce modèle permet d'exprimer le parallélisme de données comme de parallélisme de contrôle.
- La mémoire est partagée, pas besoin de spécifier la distribution des données sur les processeurs
- Le programmeur doit spécifier le parallélisme et de gérer la synchronisation des processeurs.
- Le programmeur doit utiliser des constructeurs spécifiques ou des directives de programmation pour les régions parallèles de son programme.
- C'est l'architecture et le compilateur qui masquent les mouvements des données et décident de la position des données.

Principe

- Les processeurs d'une architecture à mémoire partagée accèdent à la même mémoire.
- La mémoire doit être dotée de N ports, si N est le nombre de processeurs.
    - Conséquence : mémoire trop coûteuse
    - Solution : mémoire construite à partir de plusieurs composants mémoire.



Mémoire physiquement centralisée

Mémoire physiquement distribuée

Type d'organisation de mémoire

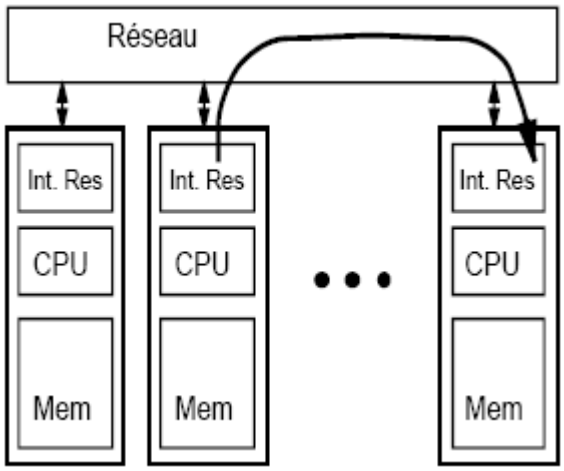
Selon la classification de Raina

- **NORMA** (« No Remote Memory Access ») : pas de moyen d'accès aux données distantes, nécessitant le passage de messages ;
- **UMA** (« Uniform Memory Access ») : accès symétrique à la mémoire, de coût identique pour tous les processeurs ;
- **NUMA** (« Non-Uniform Memory Access ») : les performances d'accès dépendent de la localisation des données ;
- **CC-NUMA** (« Cache-Coherent NUMA ») : type d'architecture NUMA intégrant des caches ;

- **Modèle à passage de messages**

- Modèle conçu pour les architectures parallèles qui ne possèdent pas de mémoire partagée.
- L'absence de mémoire partagée engendre deux conséquences pour le programmeur :
    - Il doit gérer la répartition des données sur les différents processeurs (exemple: découpage et placement des données d'un tableau est géré par le programmeur).
    - L'échange d'information 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 gérer la coordination (synchronisation) des processus (gestion des sections critiques).

- Les machines MIMD à passage de messages appartiennent à la classe NORMA
- La mémoire est distribuée sur les processeurs et pas de mécanisme pour adresser les mémoires non locales.
- Les espaces d'adressage sont différents.



- Pas de possibilité d'échange d'informations à travers la mémoire.
- C'est par le biais des E/S ou les dispositifs de communication (canaux) que l'échange aura lieu.

- Deux processus communiquent (partagent) le même canal de communication