# Aperçu sur le calcul haute performance

Dans ce chapitre subdivisé en trois parties, nous donnons un bref aperçu sur le calcul haute performance. Nous faisons apparaître dans la première partie les motivations pour le calcul haute performance et présentons dans la seconde partie une classification des architectures parallèles et distribuées. Dans la dernière partie, nous nous intéressons aux modèles de programmation parallèle.

## 1.1 Calcul haute performance et parallélisme

Le parallélisme est une stratégie pour accélérer les temps d'exécution de tâches de grande taille. Les tâches sont initialement décomposées en sous-tâches de taille plus petite et celles-çi sont ensuite assignées à plusieurs processeurs qui les exécutent simultanément. Traditionnellement, les programmes informatiques étaient écrits pour des machines monoprocesseurs (machines de von Neumann) qui n'exécutent qu'une seule instruction à la fois et ceci tous les 9ns pour les plus rapides d'entre elles [1]. Cette vitesse d'exécution peut paraître élevée mais pour les applications qui nécessitent beaucoup de puissance de calcul, tels que problèmes de simulation et de modélisation de phénomènes naturels complexes, les problèmes nécessitant de grandes capacités de calcul et la manipulation de grandes quantités de données comme le datamining, le traitement d'image, etc. les performances exigées sont nettement plus élevées. Devant les limites physiques de la taille de la mémoire et de la vitesse de calcul des systèmes monoprocesseurs, le parallélisme apparaît comme une solution compétitive (excellent rapport coût/performance) pouvant répondre aux importants besoins en puissance de calcul (de l'ordre du Gflops et même du Tflops) et

en espace mémoire (de l'ordre du To et même du Po) des classes de problèmes citées plus haut. Les machines parallèles permettent au programmeur de disposer d'un nombre important de processeurs. Ce nombre peut être modulable suivant la puissance de traitement nécessaire. D'autres avantages caractérisent ces systèmes tels que leur grande disponibilité, leur meilleure tolérance aux pannes, leur grande flexibilité et surtout, ils permettent de partager de nombreuses ressources (processeur, mémoire, disque etc.) à travers un réseau. L'utilisation des machines parallèles introduit cependant un certain nombre de problèmes comme celui de l'équilibrage de charge qui sera étudié dans le chapitre suivant, celui de la distribution des données, de la maîtrise du coût des communications entre processeurs, etc. Ces problèmes ont un grand impact sur les performances des systèmes parallèles qu'ils risquent de dégrader s'ils ne sont pas correctement pris en charge.

## 1.2 Taxonomie des architectures parallèles et distribuées

La multiplication des architectures comportant plusieurs processeurs entraîne une certaine confusion dans les terminologies utilisées dans la littérature particulièrement lorsqu'il s'agit de caractériser les systèmes distribués et parallèles. Assez souvent, la frontière entre un système distribué et système parallèle n'est pas très nette. Aussi nous donnons cette caractérisation :

- Un *système parallèle* est un système composé de plusieurs processeurs fortement couplés, localisés sur la même machine, connectés par des liens de communication point à point à base de switch (Hypercube, Crossbar, Omega, grid ...) et qui travaillent à la résolution d'une même problème. Les multiprocesseurs (UMA, NUMA) et les MPPs appartiennent à cette catégorie.
- Un *système distribué* est un système composé de plusieurs processeurs physiquement distribués et faiblement couplés, reliés entre eux par un réseau conventionnel de communication (ATM, FDDI, Token Ring, Ethernet etc.) et qui collaborent à la résolution d'un ou plusieurs problèmes. Internet, les intranets, les systémes embarqués, les systèmes mobiles et les systèmes de téléphonie appartiennent à cette catégorie.

Il existe de nombreuses méthodes de classification des architectures parallèles et distribuées dont celle de Flynn qui est sans doute la plus populaire d'entre elles. Dans cette classification [2] basée sur le flux d'instructions et le flux de données deux principales

classes sont établies. Il s'agit des machines SIMD dont les processeurs exécutent la même opération de façon synchrone sur des données différentes et des machines MIMD dont chaque processeur est autonome et exécute son propre programme sur ses propres données. Aujourd'hui les machines SIMD sont abandonnées au profit des machines MIMD qui offrent la souplesse nécessaire à la résolution de certains problèmes. Une classification plus fine des machines MIMD essentiellement basée sur l'organisation de la mémoire et les types d'interconnexion permet de distinguer deux grandes familles : les multiprocesseurs et les multi-ordinateurs [4].

### 1.2.1 Les multiprocesseurs

Un multiprocesseur est un système informatique dans lequel deux ou plusieurs processeurs partagent l'accès total à une mémoire commune [3]. Les processeurs, la mémoire et les périphériques d'entrée et sortie sont reliés par un réseau d'interconnexion pouvant prendre la forme d'un bus commun, d'un switch crossbar ou d'un réseau multi-étages. Il n'y a qu'une seule copie du système d'exploitation qui tourne sur ce type d'architecture. Les multiprocesseurs sont de deux types : Les machines UMA et les machines NUMA.

#### 1.2.1.1 Les multiprocesseurs UMA

Dans cet archiecture, il existe une mémoire unique, partagée par tous les processeurs. La mémoire est équidistante de tous les processeurs qui ont ainsi le même temps d'accès sur une référence mémoire donnée. Tous les processeurs partagent les ressources globales disponibles (bus, mémoire, E/S) mais peuvent avoir des caches privées (voir Figure 1.1). Toute opération effectuée sur la mémoire est immédiatement visible par l'ensemble des autres processeurs.

Les machines UMA ont l'avantage d'être faciles à programmer. En effet elles supportent le modèle de programmation à mémoire partagée qui est proche du modèle séquentiel, ce qui simplifie le portage du code séquentiel vers le parallèle. De plus, il est plus facile de concevoir un système d'exploitation qui utilise les processeurs de manière symétrique. Cependant ces systèmes sont peu scalables, car au-delà de quelques dizaines de processeurs il devient très difficile de maintenir la cohérence des caches et un temps d'accès uniforme à la mémoire.

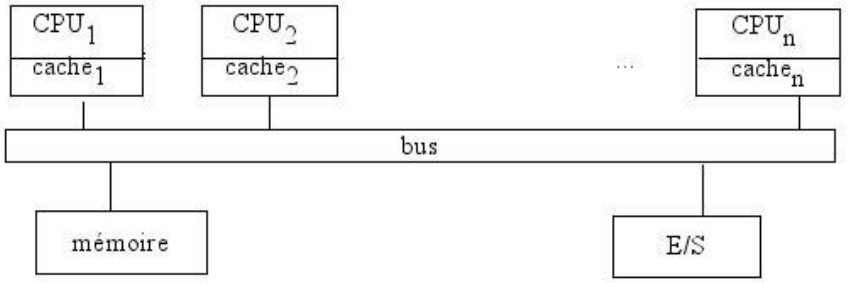


FIG. 1.1 – Machine UMA à bus

#### 1.2.1.2 Les multiprocesseurs NUMA

Les machines NUMA sont des systèmes à mémoire partagée. Cependant ils ont la particularité d'avoir une mémoire physiquement distribuée, ce qui fait varier le temps d'accès en fonction de l'emplacement mémoire. Chaque processeur dispose d'une mémoire locale (voir Figure 1.2). L'ensemble de toutes ces mémoires forme un espace d'adressage global et unique, accessible à tous les processeurs. Il est plus rapide d'accéder à une mémoire locale qu'à celles distantes à cause du délai de latence associé au réseau d'interconnexion.

Les machines NUMA ont l'avantage de supporter efficacement le modèle de programmation à mémoire partagée et d'être facilement scalables. En effet, les performances restent bonnes même avec des centaines de processeurs. Un inconvénient de ces systèmes est la complexité rencontrée au niveau matériel ou logiciel pour garantir la cohérence des données et un espace d'adressage global.

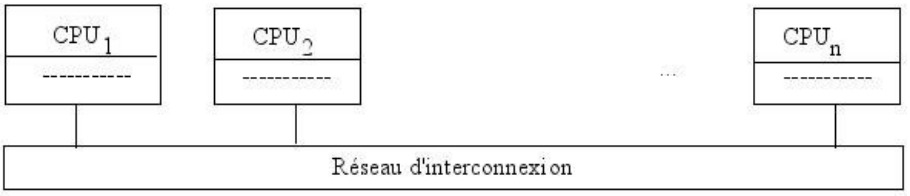


FIG. 1.2 – Machine NUMA

### 1.2.2 Les multi-ordinateurs

Un système multi-ordinateur est constitué par un ensemble de machines indépendantes reliées entre elles par un réseau d'interconnexion. Chaque machine possède une mémoire privée à laquelle elle accède directement (voir Figure 1.3). Les mémoire distantes ne sont pas directement adressables et les données sont partagées en envoyant explicitement ou en

recevant des messages. La nature du réseau d'interconnexion varie d'un multi-ordinateur à un autre. Certains utilisent un bus de type Ethernet, FDDI, ATM alors que d'autres utilisent un switch de type grid, cube ou hypercube. Il y a une copie séparée du système d'exploitation sur chaque nœud du système. Les multi-ordinateurs se repartissent en deux catégories : les multi-ordinateurs homogènes et les multi-ordinateurs hétérogènes.

#### 1.2.2.1 Les multi-ordinateurs homogènes

Parfois appelés MPPs, ils sont constitués de processeurs identiques reliés par un réseau d'interconnexion unique qui utilise la même technologie partout. Ces systèmes sont particulièrement scalables puisqu'ils ne demandent pas un dispositif logiciel ou matériel pour maintenir la cohérence des données. Cependant ils sont relativement difficiles à programmer à cause de la charge supplémentaire dûe au placement et au partitionnement des données entre les différents processeurs, ainsi que les latences élevées dans certains types de réseaux.

#### 1.2.2.2 Les multi-ordinateurs hétérogènes

C'est le cas le plus commun pour les systèmes distribués. Il s'agit de plusieurs ordinateurs aux caractéristiques matérielles et logicielles différentes, connectés à travers des réseaux utilisant des technologies différentes. Il n'y a pas une vision globale du système, les performances et les services offerts ne sont pas les mêmes partout pour les applications (à cause de l'hétérogénéité). Un tel système est relativement facile à construire et à étendre. De plus, la puissance sans cesse croissante des ordinateurs associée à leur faible coût et aux progrès technologiques réalisés dans les réseaux haut débit font que ce type de machine a un excellent rapport coût/performance même si le partage des ressources pose des problèmes de sécurité et nécessite un dispositif logiciel qui masque l'hétérogénéité de l'environnement

## 1.3 Les modèles de programmation parallèle

La programmation parallèle consiste à développer des programmes qui s'exécutent simultanément sur plusieurs processeurs à la fois. Un modèle de programmation parallèle est une abstraction du matériel que le programmeur utilise pour coder et exécuter des programmes parallèles. Il implique les langages, les systèmes de communication, les

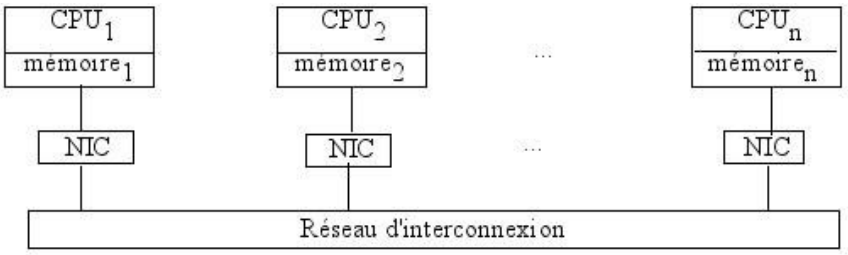


FIG. 1.3 – Exemple d'architecture multi-ordinateurs

bibliothèques et les compilateurs. Les modèles de programmation peuvent être implémentés de différentes manières : comme des librairies qu'on invoque à travers des langages de programmation séquentielle, comme des extensions de langages ou sous d'autres formes complètement nouvelles. Les modèles de programmation parallèle les plus populaires sont le modèle à échange de messages, le modèle à mémoire partagée et le modèle du parallélisme de données.

### 1.3.1 Le modèle à échange de messages

Dans ce modèle, un programme est constitué par un ensemble de processus opérant chacun dans un espace d'adressage privé. Les processus communiquent par échange explicite de messages. Un système d'échange de messages demande un schéma d'adressage qui repère les processus les uns par rapport aux autres et deux primitives de base (send/receive) permettant le transfert de données entre processus. Ce modèle donne un contrôle total de la mise en œuvre du parallélisme au programmeur qui doit cependant gérer explicitement la distribution des données, toutes les communications entre processus et les synchronisations. Le modèle à échange de messages est largement utilisé dans les machines à mémoire distribuée (multi-ordinateurs) mais il est aussi implémenté sur des machines à mémoire physiquement partagée (machines NUMA) avec de bonnes performances. L'ensemble des opérations de communication autorisées par une implémentation du modèle à échange de messages est disponible à travers des bibliothèques et des langages de programmation classiques.

MPI et PVM sont les bibliothèques les plus utilisées dans le modèle de programmation à échange de méssages. Elles supportent les langages C, C++ et Fortran. Elles permettent d'écrire de manière simple et efficace des programmes parallèles s'exécutant sur n'importe quelle plateforme où elles sont implémentées.

PVM donne l'image d'une machine parallèle virtuelle à une collection d'ordinateurs hétérogènes fonctionnant sous UNIX. La bibliothèque PVM est constituée d'un processus résidant dans tous les ordinateurs qui forment une machine virtuelle et d'une bibliothèque de fonctions pour l'échange de messages, la gestion des processus et la synchronisation des tâches [5].

En raison sa grande portabilité et des performances qu'elle offre, MPI est devenue la bibliothèque standard du modèle à échange de messages [45]. Elle fournit des routines pour la communication et la gestion des processus. Les processus utilisent une communication point à point pour l'échange de données entre deux processus. Un groupe de processus peut utiliser une communication collective pour effectuer des opérations de synchronisation globale, de diffusion globale ou sélective, de collecte ou de réduction globale de données [6].

### 1.3.2 Le modèle à mémoire partagée

Le modèle à mémoire partagée permet d'exprimer de manière simple et puissante le parallélisme dans une application. Dans ce modèle, plusieurs processus partagent un espace mémoire commun où ils peuvent lire et écrire des données de manière asynchrone. Pour éviter les conflits d'accès aux données partagées, des mécanismes de synchronisation sont nécessaires. Comme les données ne sont la "propriété" d'aucun processus, il n'y a pas de communication explicite des données selon le modèle producteur/consommateur, ce qui allége le travail du programmeur. Cependant il devient difficile au delà d'une centaine de processeurs de maintenir l'efficacité des applications surtout celles qui exigent un haut degré de parallélisme avec des milliers de processus. Le modèle à mémoire partagée est implémenté dans une large variété de multiprocesseurs (SMP et machines NUMA). Le regain d'intérêt pour les multiprocesseurs a poussé plusieurs constructeurs à définir un standard qui est OpenMP.

OpenMP est une interface de programmation d'applications parallèles sur une architecture SMP. Elle est multi-plateformes (UNIX et Windows NT) et supporte les langages C, C++ et Fortran. OpenMP est constituée de directives de compilation, de sous programmes et de variables d'environnement. Un programme OpenMP est une succession de régions parallèles et de régions séquentielles. Il est constitué par un processus unique (master thread) qui, à l'entrée d'une région parallèle, active des processus légers (thread). Chaque thread exécute une tâche et disparaît à la fin de la région parallèle pendant que

le master thread continue l'exécution du programme jusqu'à la prochaine région parallèle [7].

### 1.3.3 Le modèle du parallélisme de données

Cet autre modèle exploite le parallélisme dérivant de l'application de la même opération à tous les éléments d'un ensemble de données. Les processus exécutent le même code mais ils opèrent sur des données différentes. Le problème est d'abord décomposé en plusieurs tâches de petite taille. Chaque tâche est ensuite assignée à un processeur qui effectue sur les données locales les opérations devant déterminer leur nouvelle valeur. Conceptuellement ce modèle est plus simple que le modèle à échange de messages. Les détails de la distribution des données et de la communication entre les processeurs sont masqués par le compilateur. Le programmeur doit cependant fournir le maximum d'informations sur la distribution des données au compilateur afin de minimiser la communication entre les processus. Le parallélisme de données est utilisé aussi bien dans les systèmes à mémoire partagée que dans les systèmes à mémoire distribuée. Les implémentations de ce modèle sont souvent des extensions des langages de programmation séquentielle existants (Java, C++, Fortran etc.).

HPF est une extension du langage Fortran. Il est le langage le plus utilisé pour écrire des programmes en utilisant le parallélisme de données. Il a été conçu pour réaliser la portabilité des programmes à travers différentes architectures parallèles tout en maintenant élevées leurs performances. HPF comprend des directives qui aident à optimiser le code en indiquant au compilateur la manière de répartir les données entre les processeurs. Il dispose de fonctions qui sont des extensions directes du langage Fortran et qui peuvent affecter les résultats du calcul contrairement aux directives. HPF fournit aussi aux programmeurs une bibliothèque de fonctions particulièrement précieuses pou le calcul haute performance [8].

## 1.4 Conclusion

Dans ce chapitre premier, nous nous sommes intéressés à divers aspects du calcul haute performance. C'est ainsi qu'après avoir donné les motivations pour ce type de calcul, nous avons présenté les principaux modèles de programmation parallèle et une taxonomie des architectures parallèles et distribuées. Comme nous l'avons montré dans la

première partie de de chapitre, le parallélisme offre de multiples avantages (puissance de calcul, disponibilité, tolérance aux pannes, flexibilité etc.) mais il soulève aussi de nouveaux problèmes, qui s'ils ne sont pas correctement traités peuvent influencer négativement les performances obtenues. Dans le chapitre suivant, nous traiterons le problème de l'équilibrage de charge qui représente un aspect très important dans les systèmes parallèles et distribuées.